

Improving a Connectionist Based Syntactical Language Model

Ahmad Emami

Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD 21218, USA
emami@jhu.edu

Abstract

Using a connectionist model as one of the components of the Structured Language Model has lead to significant improvements in perplexity and word error rate, mainly because of the connectionist model's power in using longer contexts and its ability in fighting the data sparseness problem. For its training, the SLM needs the syntactical parses of the word strings in the training data, provided by either humans or an external parser.

In this paper we study the effect of training the connectionist based language model on the *hidden* parses hypothesized by the SLM itself. Since multiple partial parses are constructed for each word position, the model and the log-likelihood function will be in a form that necessitates a specific manner of training of the connectionist model. Experiments on the UPENN section of the Wall Street Journal corpus show significant improvements in perplexity.

1. Introduction

Statistical language models are a key component in many fields dealing with speech and natural language: speech recognition, machine translation, and information retrieval to name a few. The job of a language model is to assign a probability $P(W)$ to any given word string $W = w_1 w_2 \dots w_n$. This is usually done in a left-to-right fashion by factoring the probability:

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

where the sequence of words $w_1 w_2 \dots w_j$ 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, data sparseness is a crippling problem with language models; hence all practical models employ some sort of classification of histories W_1^{i-1} .

The most widely used language models are the so called N-gram models, where the word string W_1^{i-1} is classified into word string W_{i-N+1}^{i-1} . N-grams models perform surprisingly well given their simple structure, but they lack the ability to use longer histories for word prediction (locality problem), and they still suffer from severe data sparseness problems.

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 [1]. In this way 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. The Structured Language Model has shown improvement over N-gram models in perplexity as well as in reducing a speech recognizer's word error rate [1].

Having N-gram type models as its components, the Structured Language Model still suffers from the data sparseness problem. A connectionist model has been used as the word predicting component of the SLM with significant improvements in perplexity and word error rate [2]. In both the baseline [1] and connectionist based SLM [2], only the single best parse for each word string was used for training purposes. In this paper we investigate the case where the training is performed on the partial parses constructed by SLM itself, rather than on the gold standard single parses obtained from an external source. Having multiple partial parses per word changes the form of the log-likelihood function which in turn changes the way the connectionist model is trained.

Section 2 serves as an introduction to the SLM. In Section 3 we explain how a neural network model is used in the SLM, giving a brief introduction to the neural network model and the distributional representation of words that it uses. In the same section we will elaborate on the specific training algorithm necessitated by the new form of the log-likelihood function. Experimental results are presented in Section 4.

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 of n words to which we have prepended

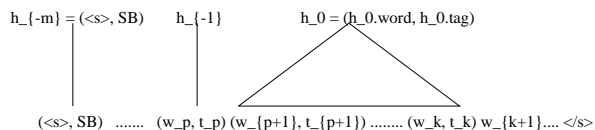


Figure 1: A word-parse k -prefix

the sentence beginning marker $\langle s \rangle$ and appended the sentence end marker $\langle /s \rangle$ so that $w_0 = \langle s \rangle$ and $w_{n+1} = \langle /s \rangle$. 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. Getting the exposed heads from the word-parse k -prefix at a given position k in the input sentence is a deterministic procedure.

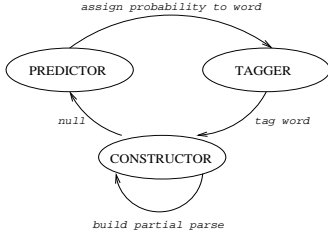


Figure 2: Finite State Representation of the SLM

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 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 types of 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 which can be parameterized (approximated) in a different way.

For any given sentence, there are far too many parses that can be hypothesized (exponential growth with sentence length). Therefore in practice the SLM uses a multi-stack search strategy with pruning to find only the more likely parses in the space of all possible ones.

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

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

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

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 .

As SLM processes a word string from left to right it constructs partial parses as described above, and stores them in its stacks. To each of these partial parses a weight is assigned as given by Equation 3.

We should note here that the probability model $P(w_k | W_{k-1} T_{k-1})$ is used in two capacities in the SLM. One is in constructing and assigning probabilities to the

partial parses (Equation 1), and the other is in computing the probability of the next word based on the constructed partial parses (Equation 2). The models used in these two roles are independent (and can be different) from each other. We tend to distinguish between the two, calling the first model the PREDICTOR and the second one the SCORER.

3. Connectionist Models

In a neural network based language model 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, taking feature vectors as its input, estimates the probability of the next word [3]. The main idea behind this model is to make the estimation process possible by mapping words from the high-dimensional discrete space they exist in, to a low-dimensional continuous one where probability distributions are smooth functions of the points in the space. The network achieves generalization by assigning to an unseen word sequence a probability close to that of a seen word sequence whose words are similar to those of the unseen word sequence. 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.

In summary, this model can be described as follows: a *feature vector* is associated with each token in the *input vocabulary*; i.e. the set of tokens that can be used for prediction. The conditional probability of the next word is then computed by a neural network taking the concatenation of all input feature vectors as its input. This probability is produced for every possible next word belonging to the *output vocabulary*. In general, there need be no relationship between the input and output vocabularies. The feature vectors and the parameters of the neural network are learned simultaneously during training.

3.1. Model Details

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 dimension
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.

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:

$$L = \frac{1}{K} \sum_i \log P(y^i | x_1^i, \dots, x_{n-1}^i; \Phi) - R(\Phi) \quad (4)$$

where $P(y^i | x_1^i, \dots, x_{n-1}^i)$ is the probability of the i^{th} word, K is the training data size and $R(\Phi)$ is a regularization term, L2 norm squared of hidden and output layer parameters in our case.

The model architecture is given in Figure 3. The neural network is a fully connected network with one hidden layer. Following the output layer is a softmax layer [4], making it possible

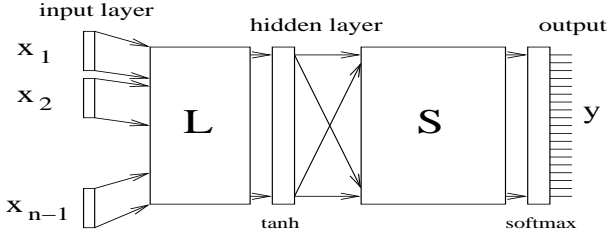


Figure 3: The neural network architecture

to learn a probability distribution at the output. More specifically, the hidden layer is described by:

$$g_k = \tanh(\sum_j f_j L_{kj} + B_k^h) \quad k=1,2,\dots,H$$

where g_k is the k^{th} output of the hidden layer, f_j is the j^{th} input of the network, L_{kj} and B_k^h are weight and biases for the hidden layer respectively, and H is the number of hidden units.

At the final stage of the network we have:

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

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}} \quad k=1,2,\dots,|V_o| \quad (5)$$

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.

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

Stochastic gradient descent is used to train the parameters of the neural network and the feature vectors, using back-propagation to compute the gradient of the log-likelihood function (Equation 4) for every parameter [5].

It should be noted that one of the great advantages of this model is that the number of inputs can be increased resulting in at most linear increase in the model size, as opposed to an exponential growth in the case of N-gram models. In fact this linear increase (reflected in matrix L) leads to insignificant increase in the total number of parameters. This makes the model specially efficient and powerful in handling longer input spans.

3.2. Neural Network Model for SLM

The standard structured language model suffers from severe data sparseness problems. Recent work has shown that increasing the amount of information available to the SLM components leads to a significant improvement on the parsing accuracy as well as to a significant reduction in both perplexity and word error rate [6]. 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's capability in fighting the data sparseness problem, makes the connectionist model a very good candidate for any of the components of the SLM.

We have used a connectionist model as the SCORER component of SLM, i.e. the component responsible for the probability $P(w_k | W_{k-1} T_{k-1})$ in Equation 2. All other components were left unchanged. The main reason was that previous experiments had shown that the probabilities $P(w_k | W_{k-1} T_{k-1})$ are in fact the ones most affected by data sparseness [6], making the choice of using the computationally more complex connectionist model for the other components an inefficient one. In fact the perplexity of both the TAGGER and CONSTRUCTOR

were found to be less than 2.

Furthermore, because of its high computational complexity, a connectionist model is used only for the SCORER and not for the PREDICTOR. That is a neural network is used only for *language model* prediction $P(w_{k+1} | W_k T_k)$ in Equation 2. Using a neural network for the PREDICTOR component makes it involved in constructing the partial parses, a rather involved process, and would increase the time complexity of our model significantly.

3.3. Training on 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 an weighted average of predictions by all the partial parses, is given by Equation 2. Representing the k^{th} partial parse at stage i as history h_i^k and its probability by λ_i^k , the probability of the word string W_1^n is given by:

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

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

$$LL = \sum_{i=1}^K \log \left(\sum_{k=1}^{k(i)} \lambda_i^k P(w_i | h_i^k) \right) \quad (7)$$

In the connectionist based SLM described in [2] the neural network was trained on single best parses (obtained from an external source); hence it was the case that $k(i) = 1$ for all i and the log-likelihood was simply $\sum_{i=1}^K \log P(w_i | h_i)$. Consequently during training, the stochastic gradient descent algorithm would compute the gradient of $\log P(w_i | h_i)$ to all the parameters for each i .

When there are multiple histories for each word the log-likelihood function is given by Equation 7. Again using stochastic gradient descent for training, the gradient of the contribution of each sample to the criteria function (log-likelihood) has to be computed. For every parameter θ of the model we have:

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

This means that for each word, the gradient (and parameter updates) for each parse predicting the word is computed, and the actual update will be a weighted average of the obtained updates, where the weights are the probabilities of the parses. The procedure is shown in detail in Algorithm 1, where the learning rate is denoted by α . It can be said that the algorithm is in essence similar to weighted mini-batch training where the mini-batch is the set of histories preceding each word.

Algorithm 1 Gradient descent for multiple histories

```

for each word  $w_i$  in training data do
  for each parameter  $\theta$  do
     $P \leftarrow 0$ 
     $\Delta \leftarrow 0$ 
    for  $k = 1$  to  $k(i)$  do
       $\Delta \leftarrow \Delta + \lambda_i^k \frac{\partial}{\partial \theta} P(w_i | h_i^k)$ 
       $P \leftarrow P + \lambda_i^k P(w_i | h_i^k)$ 
     $\theta \leftarrow \theta + \frac{\alpha}{P} \Delta$ 

```

4. Experiments

Experiments were carried out on the UPENN section of the Wall Street Journal (WSJ) corpus. The training, held-out, and test data consisted of 929564 words, 73760 words, and 82430 words respectively. A 10,000 words open vocabulary was employed, plus a total of 94 non-terminal and part of speech tags that were used in the parses.

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

We trained the baseline SLM on the human annotated parses of the UPENN corpus. The obtained SLM was then run to construct (hypothesize) partial parses on the same training data. Finally the connectionist based SLM was trained on the 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. The network has 100 hidden units with adaptive learning and a starting learning rate of 0.001, and was trained for a maximum of 30 iterations.

To speed up convergence when training on the generated partial parses, the neural network parameters were initialized from the ones already trained on the human annotated gold standard. This resulted in a significant reduction in training time.

Table 1 shows the perplexity results when the neural network SCORER is trained on the human annotated single best parses [2]. 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 a 5-gram back-off model respectively, with weights found on the held-out set. The column no-intpl shows the perplexities for the case where no interpolation was used. The back-off 3-gram and 5-gram data have test set perplexities of 148 and 141 respectively (trained and tested on the same data as our model). It can be seen that adding more context always helps and in the best case there is significant improvement in perplexity over the baseline model. The interesting point to note is that the neural network model seems to be much more uncorrelated with the 3-gram or 5-gram models than 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 achieve the same improvements when combined with the N-gram back-off models.

Table 2 shows the perplexity results when the neural network SCORER is trained on partial parses generated by the SLM. It can be seen that by training on parses hypothesized by SLM rather than on the single gold standard parses, the performance of the model always improves significantly. This shows the advantage of our approach in using a connectionist model for re-training, previous experiments with re-training the baseline model on partial parses had lead to almost no improvement. Also worth noting is the improvement of the connectionist models before interpolating them with the N-gram models (no-intpl column).

	no-intpl	+slm	+3gm	+5gm
SLM	161	161	137	132
2HW	174	137	127	123
3HW	161	132	123	119
HW-OP	155	129	121	117

Table 1: UPENN perplexities; Gold standard parses train

	no-intpl	+slm	+3gm	+5gm
2HW	141	125	119	115
3HW	136	121	116	112
HW-OP	131	117	113	110
All-3	122	114	110	107

Table 2: UPENN perplexities; Hypothesized parses train

We can further interpolate the 3 models with different context lengths and still get more improvement, as shown in the row All-3. Given that all the information in shorter contexts is available in the longer contexts, this further shows the capability of the connectionist model to exploit different information from different contexts.

5. Conclusions and Future Work

In this paper we have investigated the training and performance of a connectionist based Structured Language Model when it is trained on the hidden partial parses generated by the baseline SLM. The method showed significant improvements in perplexity. Given that, this approach can be used to bootstrap the model, training on a small annotated corpus, and re-training on constructed parses for a bigger unannotated data.

A natural extension of this work is to use connectionist models for all the components of the SLM. Also, given the advantages of hypothesizing and re-training on the hidden parses using the connectionist SLM, a more interesting idea would be to embed this approach completely in the training procedure of the SLM. That can be done in the Expectation Maximization (EM) framework; iteratively hypothesizing the hidden data by the connectionist SLM, and training the components maximizing the EM auxiliary function.

6. References

- [1] Ciprian Chelba and Frederick Jelinek, "Structured language modeling," *Computer Speech and Language*, vol. 14, no. 4, pp. 283–332, October 2000.
- [2] Ahmad Emami, Peng Xu, and Frederick Jelinek, "Using a connectionist model in a syntactical based language model," in *Proc. ICASSP*, 2003.
- [3] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems*, 2001.
- [4] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neuro-computing: Algorithms, Architectures and Applications*, F. Fogelman-Soulie and J. Hérault, Eds., October 1989, pp. 227–236.
- [5] Simon Haykin, *Neural Networks, A Comprehensive Foundation*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.
- [6] 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.