
Machine Learning, 60, 195–227, 2005

A Neural Syntactic Language Model*

AHMAD EMAMI emami@jhu.edu
FREDERICK JELINEK jelinek@jhu.edu
Center for Language and Speech Processing, The Johns Hopkins University, Baltimore, MD

Editors: Dan Roth and Pascale Fung

Abstract. This paper presents a study of using neural probabilistic models in a syntactic based language model. The neural probabilistic model makes use of a distributed representation of the items in the conditioning history, and is powerful in capturing long dependencies. Employing neural network based models in the syntactic based language model enables it to use efficiently the large amount of information available in a syntactic parse in estimating the next word in a string. Several scenarios of integrating neural networks in the syntactic based language model are presented, accompanied by the derivation of the training procedures involved. Experiments on the *UPenn Treebank* and the *Wall Street Journal* corpus show significant improvements in perplexity and word error rate over the baseline SLM. Furthermore, comparisons with the standard and neural net based *N*-gram models with arbitrarily long contexts show that the syntactic information is in fact very helpful in estimating the word string probability. Overall, our neural syntactic based model achieves the best published results in perplexity and WER for the given data sets.

Keywords: statistical language models, neural networks, speech recognition, parsing

1. Introduction

Statistical language models are widely used in fields dealing with speech or natural language, speech recognition, machine translation, and information retrieval to name a few. For example, in the accepted statistical formulation of the speech recognition problem (Jelinek, 1998), the recognizer seeks to find the word string:

$$\hat{W} = \arg\max_{W} P(A|W)\,P(W) \tag{1}$$

where A denotes the observed speech signal, $P(A/W)$ is the probability of producing A when W is spoken, and $P(W)$ is the *prior* probability W was spoken.

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:

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

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

where the sequence of words $w_1 w_2 \dots w_j$ is denoted by W_1^j. Ideally the language model would 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 equivalence classification of histories W_1^{i-1}:

$$P(W) \approx P(w_1) \prod_{i=2}^{n} P\left(w_i | \Phi\left(W_1^{i-1}\right)\right) \tag{2}$$

where $\Phi(W_1^{i-1})$ denotes the class of the word string $(w_1 \dots w_{i-1})$. Research in language modeling is concerned with finding efficient and yet powerful classification schemes.

The most widely used language models are the so called N-gram models, where a 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 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 syntactic parses of a word string and using the information from these partial parses to predict the next word (Chelba & Jelinek, 2000). In this way the SLM also addresses one other shortcoming of the N-gram model (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 speech recognizer's word error rate (Chelba & Jelinek, 2000).

Another approach in tackling data sparseness is the use of a distributed representation for words. It has been shown that this approach, while using a neural network as the probability function estimator, leads to significantly improved results (Bengio et al., 2003). The main advantage of the neural network based model is that unlike the N-gram model, it shows great capability of using long and enriched probabilistic dependencies.

In this paper we investigate neural network models for the Structured Language Model. In a given syntactic parse of a sentence, there is a large amount of information that one would like to use in estimating the probability for the given word string. A neural network model serves as a good probabilistic model for the SLM because of its capability of using long and enriched dependencies. We will present several scenarios of integrating the neural network models in the SLM framework, and derive their corresponding training algorithms. Experiments show that our models achieve a significant reduction in both perplexity and word error rate (WER) over the baseline models.

The paper is organized as follows; Section 2 gives an introduction to the neural probabilistic model. In Section 3 we discuss the Structured Language Model and in Section 4 we present the different scenarios of integrating the neural net model into the SLM. Finally, experimental results are presented in Section 5.

1.1. Relation to previous work

Most of the research in language modeling is focused on studying ways of using information from a longer context span than what is usually captured by N-gram language models. The

idea of using syntactical structure of a sentence in estimating its probability has been investigated in detail (Chelba & Jelinek, 2000; Charniak, 2001; Roark, 2001; Van Uystel, Van Compernolle, & Wambacq, 2001). These models overcome the limitations of the *N*-gram models by using longer contexts as well as the syntactical structure of the word string. Furthermore, efforts have been made to use better smoothing techniques as well as better probabilistic dependencies for the Structured Language Model (Kim, Khudanpur, & Wu 2001; Chelba & Xu, 2001; Xu, Chelba, & Jelinek, 2002).

The idea of learning a distributed representation for symbolic data and concepts has been present in the neural network community for a long time (Hinton, 1986). In fact, distributed representations and neural networks were used in Elman (1991) for finding grammatical structure; however, the results were limited to artificial and over-simplified problems.

The idea of using neural networks for natural language processing has been also studied earlier (e.g. Miikkulainen & Dyer, 1991). Some have argued that the neural networks are not computationally adequate to learn the complexities of natural language (Fodor & Pylyshyn, 1988). In Xu and Rudnicky (2000), neural networks were used for the specific task of language modeling, however the networks didn't contain any hidden units and the input was limited to one word only; therefore the capability of the model in using more complex structures and longer contexts was not examined. Recent work has successfully used the neural network models in large-scale language modeling problems (Bengio, Ducharme, & Vincent, 2001; Bengio et al., 2003).

Neural networks have also been used in learning the grammatical structure in natural language. In Lawrence, Giles, and Fong (1996), neural net models were trained and used to determine the grammatical correctness of a given sentence. Similarly, a connectionist parser was designed and investigated by Ho and Chan (1999). An improved recurrent neural network has been used by Henderson (2000, 2003) to model a left-corner parser and has achieved state-of-the-art results. Similar to our model, the improved recurrent network employs a prediction step and uses an energy function (softmax) for probability estimation. Moreover, the recurrent network makes it possible to use an unbounded parse history by learning a finite representation of the past input.

It should be mentioned here that distributed and vector-space representations have also been used outside the connectionist formalism domain. For example in information retrieval area, feature vectors are learned for words and documents and the distance between these vectors is used as the basis for query search (Deerwester et al. , 1990). The same idea has been successfully applied to the statistical language modeling task, showing improvement over *N*-gram models (Bellegarda, 1997).

The function of the neural network we use in this paper is analogous in form to a Maximum-Entropy model (Berger, Pietra, & Pietra, 1996) (see Eq. (6)). However, in the Maximum-Entropy approach the features are chosen by the designer and are kept unchanged during training. A statistical parser based on maximum-entropy models has been developed by Ratnaparkhi (1997) and has shown state-of-the-art performance in parsing accuracy.

This paper brings together and extends the results from previous work regarding the use of neural network models for the Structured Language Model (Emami, Xu, & Jelinek, 2003; Xu, Emami, & Jelinek, 2003; Emami, 2003; Emami & Jelinek, 2004). The relationship between different previously used approaches are explained in detail and the advantages

and disadvantages, as well as the relative performance of each training method is discussed thoroughly.

2. Neural probabilistic model

The fundamental problem of language modeling, as well as of any other task involving discrete random variables with large sets of allowed values, is the *curse of dimensionality*. For example, in the case of a language model with a vocabulary of size $|V|$, the number of parameters of an N-gram model is $|V|^{N-1}(|V|-1)$; which amounts to an order of 10^{23} free parameters for typical values of $|V| = 50,000$ and $N = 5$. Since there is never enough data to train models of this size, it is necessary to use some equivalence classification of the word strings preceding the word to be predicted (Eq. (2)).

In dealing with the curse of dimensionality for discrete random variables we must recognize that the probability function has to account for every possible combination of the random variables involved. There is no inherent property to help with the probability estimation of unseen events based on the occurrence of observed "similar" samples. Hence, for a discrete probability model a large number of free parameters need to be estimated. In contrast, estimation is easier in the case of a continuous probability density function, due to the smoothness in the conditioning variables. Because of this, the continuous probability models have in general much fewer parameters than their discrete counterparts.

The main idea behind *distributed representation* is to map random variables from the original high-dimensional discrete space into a low-dimensional continuous one. As a result, the estimation problem would not be anymore as severely crippled by the curse of dimensionality associated with high-dimensional discrete random variables. Consequently, if the assumption can be made that the function (probability) to be estimated has local smoothness properties, any standard learning technique (such as a multi-layered neural network) can be used to estimate the function.

The model achieves generalization because "similar" word strings are assumed to have similar (close) representations, and because of the smoothness property, small changes in the representations will induce only a small change in the estimated probability. Thus the presence of any particular event in the training data will increase the probability of not only that event, but also of a combinatorial number of "similar" (neighboring) events.

Clearly the choice of the mapping discussed above is very important since the performance of the function estimation is limited by the mapping used at its input. Furthermore the distributed representations should fulfill the smoothness assumption which is the underlying concept of the approach.

The concept of distributed representation of words, using a neural network as the function estimator, has been successfully used to implement a large scale language model (Bengio, Ducharme, & Vincent, 2001). Their approach uses *simultaneous* learning of the distributed representations (*feature vectors*) and the neural network parameters. The word representations are learned just on the basis of how they can help with the estimation task. In this paper we use the approach and the architecture proposed in Bengio, Ducharme, and Vincent (2001), with modifications to suit our particular application.

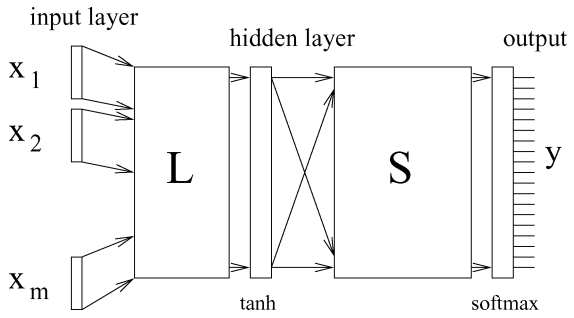

Figure 1. The neural network architecture.

In short, our 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 probability of the next word is then computed by a neural network that takes as its input the concatenation of all input feature vectors. This probability is produced for every possible next word belonging to the *output vocabulary*. In general, the two vocabularies can be separate and different from each other. The feature vectors and the parameters of the neural network are learned simultaneously during training.

2.1. Model detail

The conditional probability function $P(y|x_1, x_2, \ldots, x_m)$ where x_i and y are from the input and output vocabularies V_i and V_o respectively, is estimated in two parts:

1. A mapping that associates with each word in the input vocabulary V_i a real valued vector of fixed dimension.
2. A conditional probability function which takes as its input the concatenation of the feature vectors of the input items $x_1, x_2, \ldots, x_m$. The function estimates 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.

The training data consist of a sequence of events $x_1, x_2, \ldots, x_m \rightarrow y$ that are presented to the network one at a time. In all the experiments in this paper we use a standard fully connected multi-layered neural network with one hidden layer.

2.1.1. Probability computation: Forward pass. The model architecture is given in Figure 1. The weights preceding the hidden and output layer are denoted by L and S respectively. Following the output layer is a *softmax* function ensuring that the final outputs are properly normalized probabilities.

Presented with an event $x_1, x_2, \ldots, x_m \rightarrow y$ the neural network computes the conditional probability $P(y|x_1, x_2, \ldots, x_m)$. As the first step, each of the input variables x_k is mapped to its feature vector $\vec{f}(x_k)$ using a simple table lookup. The input to the network then consists

of the ordered concatenation of the feature vectors:

$$F : V_i \to \mathbb{R}^d, \quad x_k \to^F \vec{f}(x_k) \quad k = 1, 2, \ldots, m \tag{3.1}$$

$$\vec{f} = (\vec{f}(x_1), \quad \vec{f}(x_2), \ldots, \vec{f}(x_m)) \tag{3.1}$$

where d is the dimensionality of the feature vectors and F is the actual mapping.

The hidden layer takes this input and after a linear transformation, passes it through a standard non-linear sigmoid (tanh) function:

$$g_k = \tanh\left(\sum_{j=1}^{m \cdot d} f_j L_{kj} + B_k^1\right) \quad k = 1, 2, \ldots, h \tag{4}$$

where g_k is the kth output of the hidden layer, f_j is the jth input to the network, L_{kj} and B_k^1 are weights and biases of the hidden layer respectively, and h is the number of hidden units.

The output of the hidden layer constitutes the input to the output layer, which transforms it linearly before passing it through the *softmax* layer:

$$z_k = \sum_j g_j S_{kj} + B_k^2 \quad k = 1, 2, \ldots, |V_o| \tag{5}$$

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

where the weights and biases of the output layer are denoted by S_{kj} and B_k^2 respectively. The softmax function (Eq. (6)) ensures that the outputs are properly normalized; and in general is well suited for a network trained to learn a probability distribution (Bridle, 1989). In a sense, the softmax is a generalization of the logistic function to multiple outputs. It also has very convenient mathematical properties; its gradient is easy to compute and hence is readily integrated into the back-propagation algorithm.

The kth output of the neural network, corresponding to the kth element y_k of the output vocabulary, is the model's estimate of the sought conditional probability, that is $p_k = P(y = y_k \mid x_1, \ldots, x_m)$.

The parameters of the model are the feature vectors (table F), weight matrices L and S, and biases B^1 and B^2.

2.1.2. Training: Backward pass. Training is achieved by searching for parameters Θ, the weights and biases of the neural network and the values of feature vectors, that maximize the penalized log-likelihood of the training corpus:

$$L = \frac{1}{n} \sum_t \log P\left(y^t | x_1^t, ..., x_m^t; \Theta\right) - R(\Theta) \tag{7}$$

where $P(y^t|x_1^t, \ldots, x_m^t)$ is the probability of the tth word, n is the training data size, and $R(\Theta)$ is a regularization term, which in our case is the L2 norm squared of hidden and output layer weights (excluding biases) times a factor:

$$R(\Theta) = \omega \cdot \left(\sum_{i,j} L_{ij}^2 + \sum_{i,j} S_{ij}^2 \right) \tag{8}$$

with ω being the *weight decay factor*. Regularization is used to penalize solutions with very large parameters (weights).

Stochastic gradient descent is used to train the model; the training is carried out sequentially with the parameters being updated after presentation of each event to the network. Sequential (as opposed to batch) training is specially helpful in the case of language models where the data is redundant, i.e. the data set contains multiple copies of the same event. The sequential training is better able to take advantage of this redundancy because each of the identical events is presented, and the model's parameters are updated, one at a time. The algorithm is made stochastic by randomizing (in batches) the order of the events at the start of each iteration. This makes the search in weight space stochastic, making the algorithm less likely to be trapped in a local minimum.

For each event $(x_1, \ldots, x_m \rightarrow y)$, each parameter θ is increased by a factor of the gradient of the objective function L to that parameter:

$$\theta \leftarrow \theta + \eta \frac{\partial(\log P(y|x_1, ..., x_m) - \omega \cdot \theta^2)}{\partial \theta} \tag{9}$$

where η is the *learning rate*. The weight decay is used only if θ is a hidden or output layer weight.

Using standard back-propagation (LeCun, 1985; Rumelhart, Hinton, & Williams, 1986; Werbos, 1974) it is straightforward to compute the gradient for every parameter of the model. Starting at the output of the network $\vec{y}$ (where gradient computation is trivial), the gradients in each layer can easily be computed using the gradients in the succeeding layer. Consequently all the parameter updates in the network are found by recursively computing the gradients backwards, starting from the output and going through all the layers, finishing at the feature vectors. Note that for any given event $(x_1, \ldots, x_m \rightarrow y)$, only the feature vectors of the involved variables $(x_1, \ldots, x_m)$ are updated.

2.2. *Model complexity*

The parameters of the neural network model consist of the feature vectors plus the hidden and output layer weights and biases. During training the time taken by each event is the time spent doing a full forward pass followed by a full backward phase (back-propagation). During evaluation though, the time needed for each event is the same as a forward phase pass only.

2.2.1. Number of parameters. The parameters can be broken into three parts:

- *feature vectors* $|V_i|$ vectors, each of dimension d, for a total of $(d \cdot |V_i|)$ parameters.
- *hidden layer* $(m \cdot d \times h)$ matrix and h biases for a total of $((m \cdot d + 1) \cdot h)$ parameters.
- *output layer* $(h \times |V_o|)$ matrix and $|V_o|$ biases for a total number of parameters of $((h + 1) \cdot |V_o|)$.

So the total number of parameters of the network is:

$$(d \cdot |V_i| + (m \cdot d + 1) \cdot h + (h + 1) \cdot |V_o|) \tag{10}$$

We should note here that in a general setting, it is possible for each input variable to have its own separate vocabulary and moreover, its own separate feature vectors (mapping) for the common items in the vocabularies. However, unless there is a large amount of training data available, it is better to tie the feature vectors by using a single mapping for all the input variables. Not doing so would result in a considerable increase in the number of free parameters of the model. Note that even when features are tied, the neural net is still able to distinguish among different input variables by tuning the hidden layer weights accordingly.

2.2.2. Time complexity. *Evaluation* Presenting the network with an event the forward pass is broken into three parts:

- *input layer.* A simple table look-up of m features, each a d-dimensional vector; $(m \cdot d)$ operations
- *hidden layer.* Matrix-vector multiplication plus addition of the biases and passing through the non-linear function (Eq. (4)) for a total of $(h \cdot m \cdot d + 2h)$ operations
- *output layer.* Matrix-vector multiplication plus addition of the biases (Eq. (5)) and passing through the softmax layer – $2|V_o|ops$ (Eq. (6))—for a total of $(|V_o| \cdot h + 3|V_o|)$ operations

So the total number of operations for evaluation of *each* event is:

$$\big(d \cdot m \cdot (h + 1) + 2h + (h + 3)|V_o|\big) \tag{11}$$

Training. During training, both a full forward and a backward pass is needed for each event. The forward phase complexity was derived above. The back-propagation is again segmented into three parts:

- *output layer.* Matrix-vector multiplication – $(h \cdot |V_o|)$ ops, plus weight and bias update – $(h \cdot |V_o| + |V_o|)$ ops
- *hidden layer.* Pointwise vector-vector multiplication – $(3h)$ ops, plus weight and bias update – $(m \cdot d \cdot h)$ ops
- *feature vectors*. Matrix-vector multiplication – $(h \cdot m \cdot d)$ ops, plus feature update – $(m \cdot d)$ ops

which gives us the total number of training operations per event (including the forward pass):

$$(5h + dm(3h + 2) + (3h + 4)|V_o|) \tag{12}$$

It should be noted that the model size increases only *linearly* with vocabulary size and context length. Compare that with *polynomial* and *exponential* increase in standard N-gram model size with vocabulary size and context length respectively, and a great advantage of the neural network model becomes clear: it can handle longer contexts and richer vocabularies without the need to estimate an increasingly large number of extra free parameters.

It should be also noted that the input vocabulary has no 'direct' effect on the model's time complexity. So the context (predicting) vocabulary can be freely extended without affecting either the training or evaluation time. However, the training data time might increase indirectly because a bigger training data, as well as a larger number of hidden units may be required for the model to learn the enriched dependencies.

For typical values of the variables involved, the model size is dominated by the values of V_i and V_o while the time complexity (for both training and evaluation) is dominated by only the terms involving V_o. For this reason the effect of context length in typical applications is negligible on both the model size and complexity.

2.3. Implementation

The neural network model is computationally rather expensive, especially compared to standard N-gram models, mainly because of high dimensionality of the output layer which in turn is due to the required normalization (partition function). For this reason it was necessary to have the model trained and/or evaluated in parallel on multiple CPUs. There are alternate ways to make the model parallel; we chose to do so by splitting the data among the CPUs evenly. All the CPUs have access to and update the same parameters, and this requires us either to have each CPU broadcast its parameter updates to all other CPUs, or alternatively, to use a shared memory structure. The former is impractical due to the high volume of inter-CPU communication involved.

The algorithm was implemented on an IBM RS/6000 SP system using Message Passing Interface (MPI) library for parallel implementation—see (Gropp, Lusk, & Skjellum, 1999) for an introduction to MPI. Each node consisted of 16 CPUs and we restricted each job to run only on one node, thus avoiding the slower inter-node communications. Our implementation didn't employ any synchronization across CPUs (except at the beginning of each iteration); each CPU can read and write freely to the parameters independent of other CPUs. This entails the risk of accessing a parameter during a forward pass by a CPU before another CPU is done updating it in its backward phase. Alternatively, a parameter can get over-written before it is ever used. However, we expected these effects to be minimal because the parameter update for a single event is very small and consequently it should not be a matter of concern as long as not too many parameter updates are overwritten. Furthermore, the random updates of this particular parallel implementation add another

level of randomness to the stochastic gradient descent algorithm. Even though we never tried to analyze or record the risks involved with our implementation, the results showed that there is not much to worry about. The same shared memory implementation was used independently by Bengio et al. (2003). They also used another parallel implementation which worked by splitting the algorithm across the parameters. This implementation has the advantage of not requiring shared memory and hence can be used on any CPU cluster.

Finally, most of the required computations involve matrix and vector operation. We took advantage of the IBM's Engineering and Scientific Subroutine Library (ESSL), which is highly optimized for the particular machine architecture we used. We used the subroutines only for the matrix-vector multiplications in the backward phase—where a matrix has to be transposed before the operation is carried out—and it gave us a total of 4 fold speedup in training time. Using the ESSL subroutines for the other matrix operations didn't lead to any noticeable gain. One could also use the optimized BLAS library (Lawson et al., 1979), especially for machine architectures where proprietary subroutines are not available.

2.4. Vocabulary limitation

The training of the neural network model is a time consuming process. Therefore it would be very useful if the training and evaluation time of the network could be reduced. As pointed out in Section 2.2.2, for typical values of vocabulary size and number of features and hidden units, the bottleneck of the algorithm is at the output unit where most of the calculations are carried out. So one straightforward solution to make the network work faster is to reduce the output vocabulary size. For example in word error rate (WER) experiments the output vocabulary can be limited to a certain number of most frequent words, which would be a fraction of the actual vocabulary (Schwenk & Gauvain, 2002). Both the training and evaluation time are reduced proportionally with the reduction in output vocabulary size. For the words outside the output vocabulary the probabilities from a standard N-gram model can be used. Note that in this case the probabilities need not to be properly normalized as that's not a requirement in WER experiments. We used this approach in most of our WER experiments; the effect on the performance is minimal because the token Out Of Vocabulary (OOV) rate with respect to the output vocabulary is small.

2.5. Preliminary results

Preliminary experiments were carried out using a neural network as a word-based N-gram model (i.e. (N-1) previous words used to predict the next word). In this way we could get an insight for the neural network model and the soundness of our implementation.

The perplexity results where carried out on the UPenn corpus (details in Section 5). The corpus is in fact a treebank, though we use only the words in this experiment. This choice of corpus was made so that we can later compare the word-based results to those of the SLM based ones. Our particular partitioning of the corpus contains 929564, 73760, and 82430 words in the training, held-out, and test set respectively. Table 1 shows the independent test set perplexities of a word-based neural network language model with different context lengths. All the networks have 100 hidden units and use feature vectors

Table 1. UPenn perplexity: Word-based NN.

Model	no-intpl	+3 gm	+5 gm
NN-3 gm	170	132	126
NN-5 gm	157	125	121
NN-7 gm	154	123	119
NN-9 gm	153	122	118

Table 2. UPenn perplexity: *N*-gram models.

	3 gm	5 gm	7 gm	9 gm
PPL	148	141	141	141

of 30 dimensions. The (adaptive) learning rate starts at 0.001 and decreases as the model approaches convergence. This same configuration is going to be used all throughout most of this paper. For comparison, the perplexity results of standard back-off models are shown in Table 2. We used interpolated Kneser-Ney smoothing (Kneser & Ney, 1995; Ney, Essen, & Kneser, 1994) which is considered to produce the best results among currently used smoothing techniques; see Chen & Goodman (1999) and Goodman (2001) for a review.

As can be observed in Table 2 the test set perplexity for standard word-based models saturates at a context length of 4 (5-gram). This has to do with the fact that the chance of encountering the same exact *N*-gram in both training and test set decreases dramatically as *N* becomes larger; and the standard back-off or interpolated smoothings lack the capability to use the statistics of a particular *N*-gram to estimate the probability of a semantically or syntactically similar word string.

The no-intpl column in Table 1 shows the performance of the neural network models by themselves. All the other columns denote linear interpolation of the neural network model with the corresponding standard *N*-gram models with the single interpolation weight found on a separate held-out set (Jelinek & Mercer, 1980). The widely used bucketing scheme (context based interpolation weights) would probably lead to a slight improvement in the results but it was not used on the assumption that it will not have any substantial effect on the performance of the models relative to each other. Interpolation with *N*-gram models with *N* larger than *5* did not make any change in the perplexity, as we might have suspected from the above observation.

It can be seen that the neural network model, when combined with *N*-gram models, improves the perplexity significantly. The best neural net model achieves a 16.3% relative improvement over the best back-off model. Another observation is that the neural net perplexity saturates slower than the *N*-gram model as the context length is increased, which indicates that the neural network model can make better use of longer contexts.

Table 3 shows Word Error Rate (WER) results when the neural net model was used to re-score the *K*-best list output by a speech recognizer. We evaluated our models in the WSJ DARPA'93 HUB1 test setup; more details of which are given in Section 5. The original *K*-best list had a WER of 13.7%. The columns +lattice and +l + 5 gm denote interpolation

Table 3. WSJ WER: word-based NN.

Model	no-intp	+lattice	+5 gm	+1+5 gm
NN-5 gm (full)	14.4	13.4	13.3	12.8
NN-5 gm (4 k)	14.0	13.2	13.3	12.7
NN-8 gm (4 k)	13.7	13.1	13.1	12.6

Table 4. WSJ WER: *N*-gram models.

Model	no-intp	+ lattice
3-gram	13.9	13.4
5-gram	14.0	13.3

with the original lattice, and both the lattice and the back-off 5-gram model respectively. The lattice scores are those of a 3-gram model trained on a larger data set (Section 5). The interpolation weights were found by performing a grid search on the test set itself (the results are almost the same if the fair method of finding the weights on the heldout set is used). Two types of output vocabularies were used; full, and limited to the 4,000 most frequent words, denoted in the table by 'full' and '4 k', and trained for a maximum of 20 and 30 iterations respectively. For comparison reasons, the results for standard *N*-gram models are given in Table 4. Again, as in the case of perplexity, the neural net model shows its capability, reducing the baseline WER significantly. In this case the best results are obtained when the neural net model is integrated with the *N*-gram models.

One observation is that, the number of iterations aside, the limited output vocabulary did not hurt the performance. Consequently, we will employ the limited vocabulary architecture for all the future WER experiments in this paper.

It should be noted that the neural network model does not perform best as a standalone model; rather the best results are achieved when it is used in conjunction with the standard *N*-gram model. A speculative explanation for this behavior is that what the neural network learns from the text is somewhat different from what a regular *N*-gram model does. The neural net shows considerable capacity in using long contexts, but it might not be able to capture some plain localities. On the other hand, *N*-gram models are very capable of using these localities by way of simple counting and memorizing. This can explain why the best perplexity is attained when the two models are combined.

It also seems that the feature vectors obtained after training are only suited for use with the corresponding neural net. We tried clustering the feature vectors using K-means algorithm, but the resulting word classes didn't have any consistent semantic or syntactic similarity. Alternatively (Bengio, Ducharme, & Vincent, 2001) tried initializing the feature vectors using LSA (Deerwester et al., 1990; Bellegarda, 1997) but noticed no improvements in either in perplexity or convergence speed.

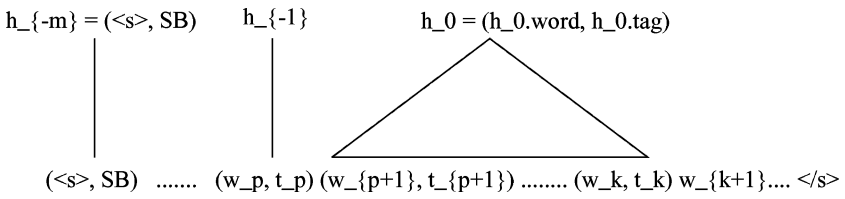

Figure 2. A word-parse k-prefix.

3. Structured language model

An extensive presentation of the SLM can be found in Chelba and Jelinek (2000). Like most other language models it predicts the next word in a string of words based on an equivalence classification of the word prefix (Eq. (2)). In the case of SLM, this classification is in fact a mixture of multiple classifications $\Phi^l(W_{k-1}), l = 1 \ldots N$ weighted by their probabilities $P(\Phi^l(W_{k-1})|W_{k-1})$.

The SLM was designed to be used as the language model in the decoder component of a speech recognition system. This constraints the model to proceed from left to right through the word sequence. A two-pass decoding strategy, such as N-best re-scoring would not bind the SLM to work in this left to right fashion, enabling it to use the whole sentence for predicting its probability. Even though the SLM hasn't been used in a first pass decoding capacity yet—due to the complexity of integrating it in a decoder – the left to right model philosophy of its original design is maintained.

The SLM will attempt to build the syntactic structure incrementally while traversing the sentence left-to-right. It will assign a probability $P(W,T)$ to every word sequence W and parse T, that is every possible POS tag assignment, binary branching parse, non-terminal label, and headword annotation for every constituent of T.

Let W be a sentence of n words to which we have prepended the sentence beginning marker $\langle \mathbf{s} \rangle$ and appended the sentence end marker $\langle \mathbf{/s} \rangle$ so that $w_0 = \langle \mathbf{s} \rangle$ and $w_{n+1} = \langle \mathbf{/s} \rangle$. 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 let $W_k T_k$ be the *word-parse k-prefix*. To stress this point, a word-parse k-prefix contains only those binary subtrees whose span is completely included in the word k-prefix, excluding $w_0 = \langle \mathbf{s} \rangle$. Single words along with their POS tag can be regarded as root-only trees. Figure 2 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, POS tag) in the case of a root-only tree. Determining the exposed heads from the word-parse k-prefix at a given position k in the input sentence is a deterministic procedure.

A *complete parse*—Figure 3—is defined as a binary parse of the **(**$\langle \mathbf{s} \rangle$**, SB)** $(w_1, t_1) \ldots (w_n, t_n)$**(**$\langle \mathbf{/s} \rangle$**,SE)** – **SB/SE** is a distinguished POS tag for $\langle \mathbf{s} \rangle \langle \mathbf{/s} \rangle$, respectively—with the restrictions that:

1. **(**$\langle \mathbf{/s} \rangle$**, TOP)** is the only allowed head.

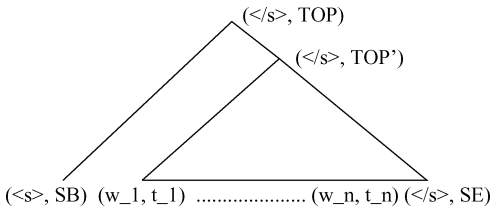

Figure 3. Complete parse.

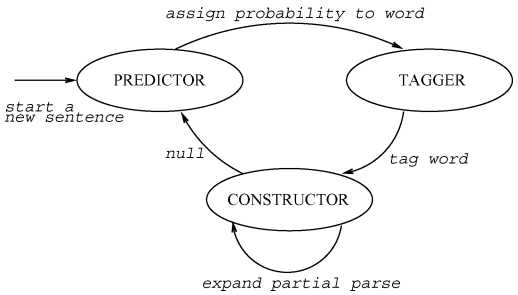

Figure 4. Finite State Representation of the SLM.

2. $(w_1, t_1) \ldots (w_n, t_n)$**(⟨/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**⟩.

The SLM operates by means of three probabilistic components:

- PREDICTOR predicts the next word w_{k+1} given the word-parse k-prefix $W_k T_k$ and then passes control to the TAGGER.
- TAGGER predicts the POS tag t_{k+1} of the next word given the word-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 following 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 which child node the new headword is percolated from.

The finite state machine in Figure 4 presents a simplified operation of the model—it does not illustrate how the model deals with unary transitions.

It is easy to see that any given word sequence with a complete parse (see Figure 3) and headword annotation is generated by a unique sequence of model actions. This will prove very useful in initializing our model parameters from a treebank—see section 3.6.2. Conversely, a generative model running according to the description above can only generate a complete parse.

3.1. Probabilistic model

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

$$P(W,T) = \prod_{k=1}^{n+1}[P(w_k|W_{k-1}T_{k-1}) \cdot P(t_k|W_{k-1}T_{k-1}, w_k) \cdot P(T'_{k-1}|W_{k-1}T_{k-1}, w_k, t_k)]$$

$$P(T'_{k-1}|W_{k-1}T_{k-1}, w_k, t_k) = \prod_{i=1}^{N_k} P\left(p_i^k|W_{k-1}T_{k-1}, w_k, t_k, p_1^k \ldots p_{i-1}^k\right)] \tag{13}$$

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 tag assigned to w_k by the TAGGER
- T'_{k-1} is the incremental parse structure attached to T_{k-1} in order to generate $T_k = T_{k-1} \parallel T'_{k-1}$; 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 sentence position k before passing control to the 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 ith 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 $w_1 \ldots w_k$ word sequence, can be generated. The $p_1^k \ldots p_{N_k}^k$ sequence of CONSTRUCTOR operations at position k grows the word-parse $(k-1)$-prefix into a word-parse k-prefix.

In short, the SLM is based on three types of conditional probabilities, $P(w_k|W_{k-1}T_{k-1})$, $P(t_k|w_k, W_{k-1}T_{k-1})$ and $P(p_i^k|W_kT_k)$, each of which needs to be parameterized and estimated from the training data.

3.2. Model parameterization

To be able to estimate the model components we need to make appropriate equivalence classification of the conditioning part for each component. The equivalence classification should identify the strong predictors in the context and allow reliable estimation from the treebank. The choice in the SLM relies heavily on exposed heads; the experiments in Chelba (1997) show that exposed heads provide good information for the PREDICTOR component of the language model; (Collins, 1996) shows that they are useful for high accuracy parsing, making them also the favorite choice for the CONSTRUCTOR model as well. Experiments have shown that exposed heads are also useful in the TAGGER component model. Since

the word to be tagged is itself a very strong predictor of the POS tag, the equivalence classification of the TAGGER model is limited to include only the non-terminal (NT) labels of the two most recent exposed heads.

$$P(w_k|W_{k-1}T_{k-1}) \approx P(w_k|[W_{k-1}T_{k-1}]) = P(w_k|h_0, h_{-1}) \tag{14}$$

$$P(t_k|w_k, W_{k-1}T_{k-1}) \approx P(t_k|w_k, [W_{k-1}T_{k-1}]) = P(t_k|w_k, h_0.tag, h_{-1}.tag) \tag{15}$$

$$P\left(p_i^k|W_kT_k\right) \approx P\left(p_i^k|[W_kT_k]\right) = P\left(p_i^k|h_0, h_{-1}\right) \tag{16}$$

where $[W_kT_k]$ denotes an equivalence classification of the word-parse prefix W_kT_k suitable for estimating each of the above conditional probabilities.

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

3.3. Pruning strategy

Since the model uses smoothed models, all possible parse trees, each with all possible headword annotations have non-zero probabilities. Consequently the number of possible parses for a given word string of length k grows faster than exponentially with k. Therefore it is necessary to use some pruning scheme to keep only the most likely parses. The pruning strategy used by the model is a synchronous multi-stack search algorithm.

Each stack stores partial parses that have been constructed by the *same number* of PREDICTOR and *same number* of CONSTRUCTOR operations. The partial parses in the stacks are ranked according to their probabilities $P(W_k, T_k)$. The hypotheses (partial parses) in each stack are expanded by first expanding them with all non-**null** CONSTRUCTOR operations (sending the new hypotheses to their appropriate stacks). Subsequently, each hypothesis is extended by a **null** CONSTRUCTOR operation and sent to the stack with *one more* PREDICTOR operation and *same number* of CONSTRUCTOR operations. The pruning is controlled by two parameters:

- maximum stack depth—the maximum number of hypotheses each stack can contain at any given time.
- log-probability threshold—the difference between the log-probability score of the most likely and least likely hypotheses can not be larger than a specified threshold.

For the experiments in this paper we use a maximum stack depth of 10 and a log-probability threshold of 6.91 (= log(1000)).

3.4. Language model

The left-to-right operation constraint on the SLM requires that the probability of the word at position $k + 1$ be estimated using only the information available from W_kT_k—the

preceding words and the partial parses that span them. This gives us the following *word level* probability formulation:

$$P(w_{k+1}|W_k) = \sum_{T_k \in S_k} \rho(W_k, T_k) P(w_{k+1}|W_k T_k) \tag{17.1}$$

$$\rho(W_k, T_k) = P(W_k T_k) \Big/ \sum_{T_k \in S_k} P(W_k T_k), \tag{17.2}$$

where S_k is the set of partial parses contained in the stacks at stage k (all the stacks with exactly k PREDICTOR operations). Note that since partial parse scores $\rho(W_k, T_k)$ sum to 1, the word level probabilities $P(w_{k+1}|W_k)$ are properly normalized assuming that the model $P(w_{k+1}|W_k T_k)$ is normalized as well. This makes it possible to report proper word level perplexities for the language model.

On the other hand, if the left-to-right operation constraint is lifted, the *sentence level* probability can be computed using:

$$\tilde{P}(W) = \sum_{k=1}^{N} P\left(W, T^{(k)}\right) \leq P(W) \tag{18}$$

where $T^{(k)}$ is one of the N-best parses for the entire sentence W, found using the pruning strategy described earlier. This probability assignment is clearly deficient (unless no pruning is used), but it can be used to re-score the N-best list output of a speech recognizer. Moreover, as will be explained later, it is useful to justify the model parameter re-estimation technique employed by the SLM.

3.5. The SCORER

As described above, the SLM is comprised of three components, the PREDICTOR, the CONSTRUCTOR, and the TAGGER. However, the probability model $P(w_k|W_{k-1}T_{k-1})$ associated with the PREDICTOR is used in two capacities in the operation of the SLM. One is in constructing and assigning probabilities to the partial parses (Eq. (13)), and the other is in word level probability assignment (Eq. (17.1)). These two models can in general be parameterized and estimated separately of each other; each using a different context equivalence classification scheme. We shall distinguish between the two, calling the first model the PREDICTOR and the second one the SCORER.

This is desirable because many of the partial parses that were initially used to predict the probability of the next word are not going to survive the pruning and therefore won't participate in the "N-best training" stage of the model (described in the next section). Estimating a separate SCORER enables us to make up partly for this weakness and train a model to achieve a higher likelihood on the training data.

In many of the experiments in this paper only the SCORER component will be modeled by a more complex architecture and the PREDICTOR will remain unchanged from the

baseline SLM. The main reason is a merely practical one; the SCORER model can be easily upgraded, not requiring any changes in the parses used for training.

3.6. Model estimation

The previous sections described the structure of the SLM and the definition of its components. The models need to be estimated from training data which may be in the form of a treebank with complete parses provided for each sentence. Each parse is binarized and headword percolated; both *binarization* and *headword percolation* are rule-based and deterministic procedures (see Chelba & Jelinek, 2000).

The SLM model estimation takes place in three stages:

1. Initialization of the model components' parameters from the training treebank.
2. Increasing the training data likelihood as given by the deficient probability estimation in Eq. (18). This involves the employment of the "N-best training" algorithm (see Section 3.6.2).
3. Estimation of the SCORER component so that the likelihood of the training data as given by Eq. (17.1) is increased. The SCORER is initialized by copying the PREDICTOR estimated in the previous stage.

3.6.1. First stage: Parameter initialization. In the first stage, the complete parses in training set are directly used, after they have undergone binarization and headword percolation.

Each parse (W,T) can be identified by a derivation $d(W,T)$ which is the sequence of steps that the SLM would take to construct that given parse. Each step is either a PREDICTOR, TAGGER, or CONSTRUCTOR operation and is in general in the form of an event $(x_1, \ldots, x_m \rightarrow y)$ where $(x_1, \ldots, x_m)$ is the context information used in taking the step, and y is the actual operation performed by the component. The set of these operations obtained from the parsed data makes up our training set from which it is straightforward to train each of the individual components. The training algorithm would of course depend on the model used for each component. For example, in the case of interpolated or back-off models, the training would consist of a simple counting of the events. Alternatively, in the case of a neural network, the training would be performed by the back-propagation algorithm, presenting $(x_1, \ldots, x_m)$ to its input and maximizing the log-probability of y at its output.

3.6.2. Second stage: N-best EM re-estimation. In this stage each of the three component models are re-estimated with the objective of increasing the likelihood of the training data computed by the probability given in Eq. (18). The approach is in the form of maximum likelihood estimation from incomplete data, with W being the observed and T—the parse structure along with POS and non-terminal tags and headword annotation for a given W—the hidden data. Therefore this stage of the training procedure makes use of the Expectation Maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977).

The EM algorithm requires that all the hidden events—parses T in this case—be considered when computing expected likelihood (EM auxiliary function in the E-step). However,

as mentioned in Section 3.3, this is not feasible due to the large number of parses involved. Therefore a variant of the EM algorithm is used which only considers the "N-best" parses found by the search and pruning strategy described in Section 3.3. Intuitively the "N-best" EM algorithm tries to maximize an *approximation* of the true likelihood and can be thought of as a compromise between full forward-backward (N = all possible parses) and Viterbi ($N = 1$) training of the hidden Markov models.

Even though it maximizes an approximate objective function, it still can be proved that the "N-best" EM algorithm does converge to a local maximum (see Chelba & Jelinek, 2000) for discussion and comments). Also, for a presentations of different variants of the EM algorithm the reader is referred to Byrne, Gunawardana and Khudanpur (1998).

3.6.3. Third stage: SCORER estimation. The previous stage of the training tries to increase an *approximation* of the likelihood of the training data; therefore the trained PREDICTOR model is not optimal. We can partly make up for this by estimating a separate SCORER maximizing the true word level probabilities as given by Eq. (17.1). In this stage of the training, a separate SCORER model is trained using partial parses and their corresponding weights $\rho(W_k, T_k)$. The partial parses and their scores are obtained by using the SLM trained in the previous stages. It should be noted that Eq. (17.1) is analogous to the likelihood of a hidden Markov model with fixed transition probabilities (but dependent on position k) specified by the values $\rho(W_k, T_k)$.

4. Neural net based SLM

As we mentioned in Section 3, all the functionality of the Structured Language Model is governed by its four components, the PREDICTOR, the TAGGER, the CONSTRUCTOR, and the SCORER. By the *baseline* SLM we denote the model where the components are parameterized according to Eqs. (14)–(16), with the SCORER being the same as the PREDICTOR, and where each component is modeled by a bucketed linear interpolation model (Jelinek & Mercer, 1980), characterized by the recursive equations:

$$\begin{aligned} P_m(y|x_1, \ldots, x_m) &= \lambda(x_1, \ldots, x_m) \cdots P_{m-1}(u \mid x_1, \ldots, x_{m-1}) \\ &\quad + (1 - \lambda(x_1, \ldots, x_m)) \cdots f_r(u \mid x_1, \ldots, x_m) \\ P_{-1}(y) &= \mathit{uniform}(\mathcal{Y}) \end{aligned} \tag{19}$$

where λ's are interpolation coefficients and f_r and P_{-1} refer to relative frequency and uniform distributions respectively. This is essentially an N-gram type model – with $N = m + 1$ – and thus comes with the shortcomings mentioned in the earlier sections, mainly the inability to use long or rich dependencies. However, there is a considerable amount of information in a partial parse that one would like to use, and an N-gram type model does not simply prove to be adequate enough for such a purpose.

Previous efforts have tried to improve the Structured Language Model in terms of both language modeling capability and parsing accuracy by using more conditioning information for the model components. In Chelba and Xu (2001), each headword's NT tag was augmented by one or both of its children NT tags; (Xu, Chelba, & Jelinek, 2002) extended

this work by adding to the conditioning context of the CONSTRUCTOR model the NT tag of the second previous headword (h_{-2}) as well. These experiments showed that the SLM performs better in all aspects—PPL, WER, and parsing accuracy—when richer probabilistic dependencies are used for its components.

So it is clearly desirable to use as much conditioning information as possible for the SLM component models. However, as mentioned above, the *N*-gram type models used in the baseline model are not well equipped for this task. In fact, in all the enriching experiments mentioned above, severe data sparseness problems were observed. It is then natural to try a more powerful model for the SLM components. The neural network model is a perfect candidate because of its excellent capability in handling larger vocabularies (enriching) and longer contexts (extending dependencies). As explained in Section 2.2, the neural network model complexity increases negligibly with context length and only linearly with vocabulary size in practical settings.

Ideally, a neural network model would be used for each component of the SLM, using as much conditioning information as possible, and trained by going through the stages outlined in Section 3.6. However, training and experimenting with the neural network models (specially when integrated in the SLM) is a very time consuming task, and thus instead of implementing a rigorous integration we started with more limited settings; sometimes using the neural network model for only one component, other times skipping one or two stages of the SLM model estimation procedure (Section 3.6). Our belief was that if the incomplete (and sometimes ad hoc) combination of neural network and the SLM can be shown to be helpful, then the viability of a complete and rigorous integration will be evident.

Overall, we have used neural networks as SLM component models in three different scenarios which we describe below:

1. *Mismatched SCORER training.* The main goal in this scenario was to integrate the neural net model in the SLM while keeping the task simple. Therefore we decided to have only one of the SLM components modeled by the neural network. The SCORER is a perfect candidate, largely because "upgrading" it will only affect the language model estimation part of the SLM, keeping the parse construction machinery intact. Also, there is a lot of potential in "upgrading" the scorer because it has a high perplexity relative to other components.

   Again, for the sake of simplicity, instead of training the model on the partial parses as required by Eq. (17.1), we used only the annotated (parsed) data set in training the neural net SCORER. In other words, in this scenario only the first stage of the SLM training procedure (Section 3.6) is carried out, training (only) the SCORER as if it were a PREDICTOR. The original PREDICTOR from the baseline model model is kept unchanged.

   Figure 5 shows the schematic of mismatched SCORER training. During evaluation (solid lines), a neural net SCORER that is trained on PREDICTOR events is used to evaluate and combine the test set SCORER events (partial parses) constructed by the baseline SLM. The two streams output by the baseline SLM are the actual partial parses and their corresponding weights respectively.

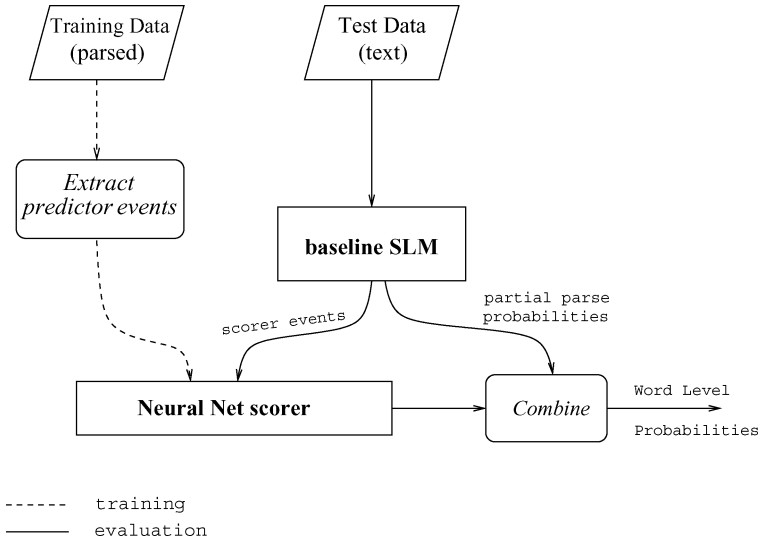

Figure 5. Mismatched Scorer Training.

Clearly, this training procedure is not optimal. The SCORER is trained to maximize the likelihood function involving probabilities as in Eq. (14), but it is used according to Eq. (17.1). On the other hand, one can see that the two sets are equivalent if there was only one partial parse for every word prefix. So the training is sub-optimal in the sense that it is based on the "wrong" (sub-optimal) data set. However, one can assume more or less that the two data sets are close enough; that is, the events encountered by the PREDICTOR are in general similar to the events that the SCORER uses in estimating word level probabilities.

2. *EM training.* In contrast to the previous scenario, this setting involves modeling all the SLM components by a neural network. First, the initial stage of the SLM model estimation procedure (Section 3.6) is carried out, training the neural network based PREDICTOR, TAGGER, and CONSTRUCTOR. In the second stage, the newly estimated neural network models will be re-trained according to the "N-best" EM re-estimation procedure. This would require the SLM model estimated in the first stage to be used to find the N-best parses for each sentence in the training set. Notice that in this scenario, unlike the first one, the neural network models are also involved in parse construction and pruning.

   In the E-step of the EM algorithm, the *expected* likelihood of the training data is calculated using the parameters from the previous iteration of the algorithm. This likelihood expectation—EM auxiliary function—is dependent on the current model parameters:

$$Q(\Theta, \hat{\Theta}) = \sum_{T} P(T|W; \hat{\Theta}) \log P(W, T; \Theta) \tag{20}$$

where $\hat{\Theta}$ and Θ denote the old and new (to be estimated) parameters. In the M-step of the algorithm, those parameters Θ are found that maximize the above expected likelihood:

$$\Theta^{new} = \text{argmax}_{\Theta} Q(\Theta, \hat{\Theta}) = \text{argmax}_{\Theta} \sum_{T} P(T|W; \hat{\Theta}) \log P(W, T; \Theta) \tag{21}$$

where $P(T|W; \hat{\Theta})$ is the fixed probability (weight) assigned to the complete parse T. Using Eq. (13) we can decompose the joint log-probability $\log P(W, T; \Theta)$ into parts, each involving a single SLM component:

$$\begin{aligned} &log P(W, T; \Theta) \\ &= \sum_{k=1}^{n+1} \log P(w_k|W_{k-1}T_{k-1}) + \sum_{k=1}^{n+1} \log P(t_k \mid W_{k-1}T_{k-1}, w_k) \\ &+ \sum_{k=1}^{n+1} \sum_{i=1}^{N_k} \log \mathrm{P}\left(p_i^k \mid W_{k-1}T_{k-1}, w_k, t_k, p_1^k \ldots p_{i-1}^k\right) \end{aligned} \tag{22}$$

As can be seen, the joint log-probability is a summation of SLM component log-probabilities. The summations are over the operations (events) carried out to construct the particular complete parse T. Clearly, the gradient of the EM auxiliary function can in turn be broken up into component-wise constituents, and hence the contribution of each component to the total gradient can be separately computed. This makes the EM training of the neural networks very straightforward, each component neural network can be trained separate and independently of other components, maximizing its likelihood over its own set of operations (events). The N-best EM training for the neural network based SLM is summarized in Algorithm 1. As described in Sections 3.3 and 3.6.2, the re-estimation procedure is limited to the N-best parses for each sentence. Considering all the possible parses for each sentence is simply intractable (exponential growth with sentence length). So the summation in Eqs. (20) and (21) would be over the "N-best" parses rather than all the possible ones. Clearly this is an approximation of the real auxiliary function, with the error depending on the value of N as well as the skewness of the distribution of the partial parses. Note that if $N = 1$, then this algorithm is basically the same as the first stage of the SLM training procedure except that the parses are constructed by the model itself rather than taken from an external source. It should be noted here that the actual weights $P(T \mid W; \hat{\Theta})$ are normalized to ensure they add up to one.

Finally, the SCORER is copied from the PREDICTOR and used to calculate the language model probabilities. So the third stage of the SLM training procedure is skipped in this scenario.

Figure 6 depicts the EM training procedure. The arrows marked by $E0$ refer to the initialization stage (training neural nets on the gathered counts) while $En(n > 1)$ corresponds to later stages of the training were the components are re-trained on the N-best parses with fractional counts.

Algorithm 1 EM training of neural network based SLM

for each iteration **do**
  construct N-best parses for each sentence W in the training data
  normalize weights $P(T|W;\hat{\Theta})$ over the N-best parses
  for each component of the model **do**
    for each parse T obtained above **do**
      for each operation $(x_1, \cdots x_m \rightarrow y) \in d(T)$ **do**
        for each parameter θ **do**
          $\theta \leftarrow \theta + \eta P(T|W;\hat{\Theta}) \cdot \frac{\partial}{\partial\theta} P(y|x_1, \cdots, x_m)$

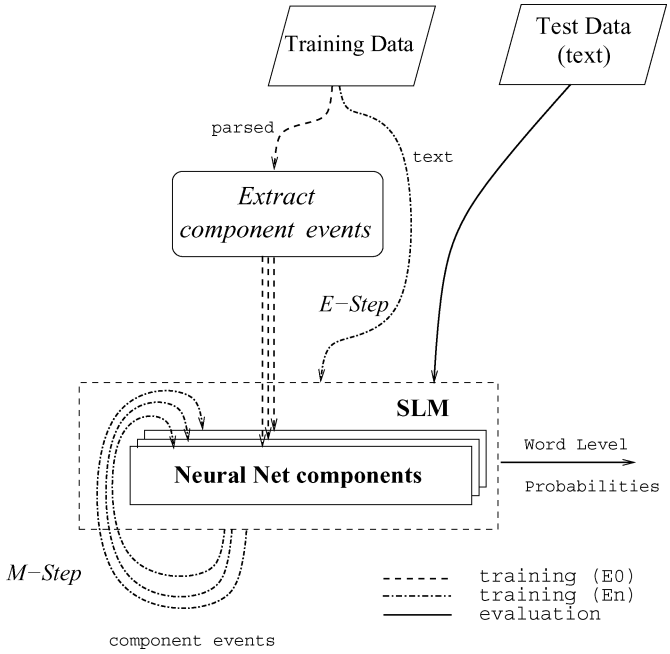

Figure 6. EM Training.

3 *SCORER estimation*. This scenario is exactly the same as the first except that in this case the SCORER is trained to maximize the correct objective function computed using probabilities in Eq. (17.1). In short, in this scenario the first stage of the training is carried out using the baseline model, then the second stage ("N-best") training is skipped altogether, and finally the training is completed by estimating a neural network based SCORER on the partial parses constructed by the baseline model. The gradient descent algorithm is slightly more involved in this case because of the summation over partial parses in Eq. (17.1). If we denote the context in the *ith* event $(x_1^i, \ldots, x_m^i \rightarrow y_i)$ by h_i,

Algorithm 2 Gradient descent for multiple histories

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

and assume that there are $k(i)$ contexts (partial parses) at position i, we have:

$$\begin{aligned}\frac{\partial}{\partial\theta}\log P(y_i) &= \frac{\partial}{\partial\theta}\left(\log\left(\sum_{k=1}^{k(i)} \rho_i^k \cdot P\left(y_i \mid h_i^k\right)\right)\right) \\ &= \frac{1}{\sum_{k=1}^{k(i)} \rho_i^k \cdot P\left(y_i \mid h_i^k\right)} \frac{\partial}{\partial\theta}\left(\sum_{k=1}^{k(i)} \rho_i^k \cdot P\left(y_i \mid h_i^k\right)\right) \\ &= \frac{1}{P(y_i)} \sum_{k=1}^{k(i)} \rho_i^k \cdot \frac{\partial}{\partial\theta} P\left(y_i \mid h_i^k\right) \end{aligned} \tag{23}$$

where ρ_i^k denotes the score $\rho(W_i^k T_i^k)$ of the kth partial parse at ith position.
The gradient descent procedure is accordingly summarized in Algorithm 2. Note that the algorithm is analogous to weighted mini-batch training, with a batch being the set of partial parses at a given position.
The partial parses are simply all the stack entries at each position (equal number of PREDICTOR operations) of the word string. Note that in this scenario, neither of the PREDICTOR, TAGGER, CONSTRUCTOR components are involved. Also the same set of partial parses (built by the baseline model) are used all throughout training, so there is no need re-run the SLM over the training data at the start of each iteration. All said, this scenario is the most time consuming one simply because of the sheer large number of partial parses involved. Even in the "N-best" training case, the number of the events used in training is much smaller because only a few of the partial parses in each position will actually make it to be part of a complete parse.
The training is depicted in Figure 7. As can be seen, this training is similar to that of Figure 5 with the difference that both training and evaluation go through the same path.

In implementing the SCORER training, the inputs to the softmax function (z_k's in Eq. (6) were normalized by substracting from all of them the maximum input value $z_{\max} = \max_k z_k$. In this way, precision related problems that might have occurred due to a very large z_k were avoided.

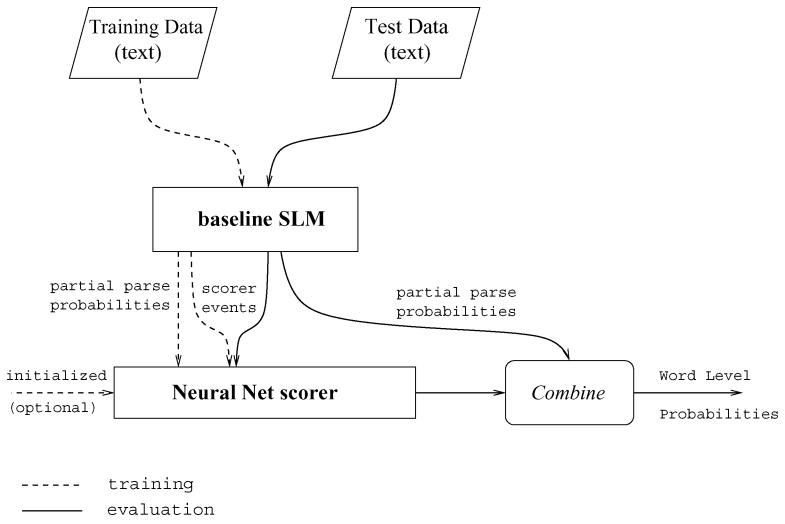

Figure 7. Matched scorer training.

We should note here that if the SCORER training was carried out on the partial parses constructed by the model trained in the second scenario (rather than the baseline), then we would have a full integration of the neural network models in the SLM (see next).

4. *Full training.* It is possible to model all the components of the SLM by neural nets and have them trained on the appropriate training events. Such a comprehensive training will start with EM training (scenario 2) and will later train a separate SCORER by training it on the partial parses generated by the just trained SLM (similar to scenario 3). So basically, a full and comprehensive training of the SLM involving all the four components consists of a scenario 2 training followed by scenario 3 training of the SCORER. This can be thought of as replacing the 'baseline SLM' block in Figure 7 by the fully trained SLM in scenario 2.

5. Experiments

The baseline Structured Language Model uses the model parameterizations formulated in Eqs. (14)–(16). In this section we experiment with using a neural net model in the different scenarios described in the previous section, while extending the conditioning contexts for the neural net component.

Our experimental setup is as follows: for perplexity results we used the UPenn Treebank portion of the WSJ corpus. The UPenn Treebank contains 24 sections of hand-parsed sentences. We used sections 00-20 as the training data, and Sections 21 and 22 and 23 and 24 as the held-out and test sets, respectively. The three sets contained 930 k, 74 k, and 82 k words respectively. We used an open vocabulary consisting of 10 k words. Note that no vocabulary limitation (see Section 2.4) were used in perplexity results since the probabilities are required to be normalized. The text was normalized in the following ways:

numbers in Arabic form are replaced by a single token 'N', punctuations are removed, all words are mapped to lowercase, and extra information in the parses (such as traces) are ignored. There are a total of 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 on the WSJ DARPA'93 HUB1 test setup. The same setup was used in Chelba and Jelinek (2000), Roark (2001), Chelba and Xu (2001), and Emami, Xu, and Jelinek (2003). The test set is a collection of 213 utterances (10 different speakers) for a total of 3446 words. The 20 k words open vocabulary and baseline 3-gram model are standard ones provided by NIST and LDC—see Chelba and Jelinek (2000) for details. The lattice and *K*-best lists were generated using the standard 3-gram trained on 45 M words of the WSJ corpus (Paul & Baker, 1992). The baseline SLM was trained on 19 M words of WSJ text automatically parsed by the parser in Ratnaparkhi (1997). Note that there are memory constraints in using more data for the baseline SLM simply because the size of the *N*-gram type components grow linearly with the number of training data *N*-gram types. In order to be able to compare the results, the neural net models were also limited to the same 19 M words, even though the neural net model is not constrained by the memory limitations.

We used the same neural net configuration for all the experiments, with 100 hidden units and 30 dimensional feature vectors. The learning rate and weight decay factor were set to 10^{-3} and 10^{-4} respectively, with the learning rate decreasing adaptively as more events are presented to the network. Since the inputs to the networks are always a mixture of words and NT/POS tags, while the output probabilities are over words in the PREDICTOR, POS tags in the TAGGER, and adjoin actions in the CONSTRUCTOR, separate input and output vocabularies had to be used. Furthermore, the output vocabulary was limited to the 5 k most frequent words for all the WER experiments (Section 2.4)—the OOV rate with respect to this limited vocabulary was found to be 6.2% on the training data. The parameters of neural nets were randomly initialized with a uniform distribution centered at zero. All the networks were trained for a maximum of 30 iterations unless otherwise stated. The held-out set was used for early stopping, however we didn't observe any overfitting behavior. We observe that the held-out perplexities do increase a few times during training, but in the end the best held-out perplexity is attained at the last or very close to last iteration.

In order to study the behavior of the SLM when longer context is used for conditioning the probabilities, we gradually increased the context of the PREDICTOR/SCORER model. First, the third exposed previous head was added. Since the syntactic head gets the head word from one of the children, either left or right, the child that does not contain the head word (hence called *opposite* child) is never used later on in predicting. This is particularly not appropriate in a prepositional phrase because the preposition is always the head word of the phrase in the UPenn Treebank annotation. Therefore, we also added the opposite child of the first exposed previous head into the context for predicting.

Tables 5 and 6 show the perplexity and the WER results respectively for the mismatched neural net SCORER training (Section 4). For the perplexity results the neural net SCORER was trained on the hand-parsed sentences for 50 iterations. The row SLM denotes the baseline SLM model, while the rows 2 HW, 3 HW, and (3+1)HW refer to conditioning contexts consisting of 2 previous heads, 3 previous heads, and 3 previous heads plus the first

Table 5. Mismatched NN SCORER; UPenn Perplexity.

Model	no-intpl	+slm	+3 gm	+5 gm
SLM	161	161	137	132
2 HW	174	137	127	123
3 HW	161	132	123	119
(3 + 1) HW	155	129	121	**117**
All-3	152	128	120	**117**
2 H W+ 2 w	154	129	122	118

Table 6. Mismatched NN SCORER; WSJ WER.

Model	no-intpl	+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
2 HW	13.5	12.7	12.7	12.5	12.3
3 HW	13.7	12.7	12.9	12.7	12.3
(3 + 1) HW	13.2	12.4	12.8	12.5	12.4

previous opposite head respectively. The columns +3 gm and +5 gm denote interpolation with Kneser-Ney smoothed 3-gram and 5-gram models respectively (trained on the same 19 M words data set as the baseline SLM in WER experiments)—see Tables 2 and 4 for perplexity and WER. All the three neural net models are also interpolated with each other and the results are given in the row All-3. The interpolation weights were found on the held-out data and in most cases were close to 0.5; finding the weights on the test set itself does not improve the performance noticeably. The row 2 HW +2 w in Table 5 refers to the case where the input context is increased from the 2 HW case to include the trigram information as well. In other words, the SCORER uses the 2 previous headwords as well as the 2 immediate previous words in assigning a probability to the next word. In comparing this row to that of 2 HW it can be observed that adding the trigram information to the model indeed boosts the performance considerably. Another observation is that interpolating this model (which already has the 3-gram information) with a regular trigram (+3 gm column) still improves the perplexity. Our explanation for this is that the probability distributions learned by the neural net and the regular trigram are not closely correlated and therefore they have some mutually exclusive information.

In WER tables, the columns marked by +slm and +lattice denote linear interpolation with the baseline SLM and the lattice word language model (3-gram trained on the whole WSJ corpus) scores respectively. The +all notation refers to interpolation with baseline SLM, lattice, and the 5-gram model, all at the same time. Furthermore, in WER experiments, the interpolation weights are found on the test set itself using grid search. A strictly fair experiment should find the weights on some independent and unseen set; however

Table 7. N-best EM training; UPenn Perplexity.

Model,	EM itr.	no-intpl	+slm	+3 gm	+5 gm
2 HW	E0	162.5	130.9	123.8	119.3
2 HW	E1	158.2	129.5	123.0	118.6
(3 + 1) HW	E0	151.2	124.4	119.1	115.2
(3 + 1) HW	E1	147.9	123.2	118.5	114.7
2 HW (full)	E0	137.6	121.8	116.9	113.1
(3 + 1) HW (full)	E0	129.0	114.6	111.3	108.4

optimizing the weights on the test set itself should not be a problem as long as the experiment is for comparison purposes and the approach is applied to all the models.

It can be seen from the tables that a neural net based SCORER—even though trained as a PREDICTOR—leads to significant improvement in perplexity and WER over the baseline SLM. It should be also noted that extending the conditioning dependencies consistently improves the perplexity.

In the second scenario of integrating neural nets into the SLM, all three components—PREDICTOR, TAGGER, and CONSTRUCTOR—are modeled by a neural network (Section 4). In the first step the components are trained on the treebank parses (iteration zero; E0)—similar to the first scenario with the difference that the TAGGER and the CONSTRUCTOR are modeled by a neural net as well. Subsequently, the newly trained SLM is used to obtain "*N*-best" parses for each sentence in the training data, and then the components are re-estimated using the "*N*-best" EM training algorithm. Because of time constraints we performed the EM re-estimation for only one iteration (E1). We also used only the 10 best parses for each sentence ($N = 10$). The results are given in Table 7 (Xu, Emami, & Jelinek, 2003) . The different conditioning context correspond to the PREDICTOR model only. The probabilistic dependencies of the TAGGER and CONSTRUCTOR are the same as in the baseline model. Furthermore, the SCORER is copied from the trained PREDICTOR. By comparing the E0 (iteration 0) rows to the results in Table 5 one can observe that using a neural net model for the components involved in parse construction further decreases the perplexity. This can be attributed to the construction of better partial parses by the neural net based components. It is also clear from the table that the "*N*-best" EM re-estimation reduces the perplexity consistently for all situations; however, the reduction is minimized when the neural net based model is interpolated with baseline or *N*-gram model.

The last 2 rows of the table show the results for the full training of the model (scenario 4), where a separate SCORER is trained on the partial parses constructed by the models in second and fourth rows; 2 HW-E0 and (3 + 1) HW-E0. As can be observed from the table, by training a separate SCORER the perplexity of the model is significantly reduced.

In the third scenario, we trained a neural net SCORER on the partial parses constructed on the training data by the baseline SLM. In order to reach convergence faster we did not randomly initialize the neural net, instead the parameters were copied from the SCORER trained in the first scenario. Subsequently, the network was trained for 30 and 7 iterations

Table 8. Macthed NN SCORER; UPenn perplexity.

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

Table 9. NN SCORER; WSJ WER.

Model	no-intpl	+slm	+lattice	+5 gm	+all
2 HW	12.8	12.3	12.4	12.3	**12.0**
3 HW	12.9	12.7	12.9	12.6	12.4
(3 + 1) HW	12.5	12.3	12.4	12.1	**12.0**

for perplexity and WER experiments respectively. It is worth mentioning that reducing the number of iterations is very important in this case because of the large number of partial parses involved.

The UPenn test set perplexities are given in Table 8 (Emami, 2003). There were a total of 10 M partial parses — compare to 1 M for the first scenario –; an average of 11.12 partial parses per word. Similarly the WER results are presented in Table 9 (Emami & Jelinek, 2004). Here the total number of partial parses was 148 M—compare to 19 M parses of the first scenario.

The best overall results are decidedly achieved—in terms of both perplexity and WER—when we use a neural net SCORER trained to maximize the word level probabilities. With the best models we achieved a perplexity of 107 and a word error rate of 12.0%; which are, as far as we know, better than any published results for the same setup (Emami, 2003; Emami & Jelinek, 2004).

Comparing the results in Tables 7 and 8 (last 2 rows), we observe that small reductions in perplexity are achieved if we train a matched SCORER for an "*N*-best" EM trained model (i.e. full training). However the improvement is practically small, and we conclude that the best (or near best) results are attainable by training *only* a matched SCORER (all other components kept unchanged), and without employing the "*N*-best" the EM training.

It is worth mentioning that in most of the experiments, the neural network by itself (no interpolation) performed worse than the standard *N*-gram models. The exception is when we train a matched SCORER (Tables 8 and 9, and last 2 rows of Table 7), where the neural net based SLM outperforms all the other models by itself.

It is easy to compare the first and third scenarios and explain the significant difference in their performance. Both procedures estimated a SCORER, however in the first scenario the SCORER was trained as a PREDICTOR; therefore there is a mismatch between the objective the model was trained for and its actual use and thus the estimated model is sub-optimal. In contrast, in the third scenario the correct log-likelihood function was optimized,

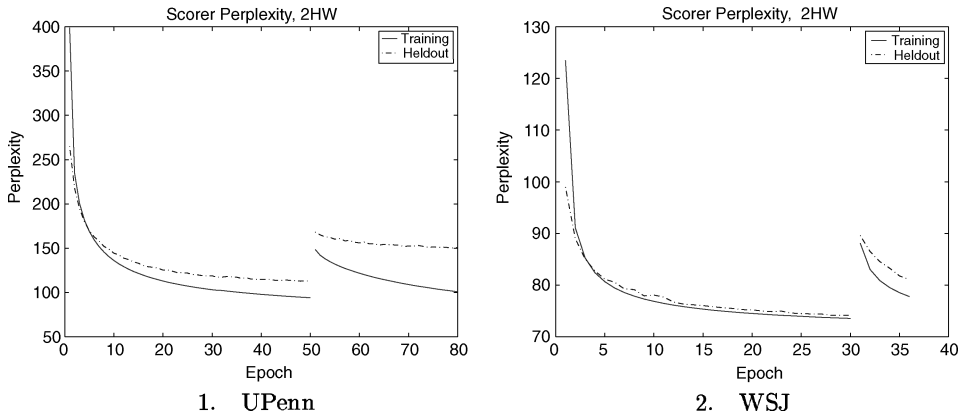

Figure 8. SCORER perplexity with 2 HWs as context.

at the cost of much longer training time, which was due to the large number of partial parses involved.

Figure 8 shows the learning curves of the neural network SCORER when the context used is the 2 previous headwords (2 HW). The curves on the left side of *each figure* correspond to the mismatched training of the SCORER (scenario 1), while the curves on the right show the learning process in the third scenario—when NN SCORER is estimated from the appropriate partial parses. All the *training* perplexities are a slight over-approximation of the actual ones because of the sequential behavior of the stochastic gradient descent algorithm in updating the parameters. On the other hand, the *held-out* perplexities are calculated after the iteration is over, and hence are exact. The curves for the two scenarios are placed one after the other because the parameters in the third scenario are initialized by copying them from the SCORER trained in the first scenario. The discontinuities in the figures are due to the fact that different data is used in the two different scenarios (1 and 3).

In the end, it is interesting to compare the SLM and word based (N-gram) models. By comparing the results in Tables 1 and 3 to those in this section it can be seen that the best word-based model is far inferior compared to the best SLM model. We might conclude from here that the syntactical structure of a sentence is in fact helpful in discriminating among competing hypotheses.

6. Conclusions and future work

By using neural network models in the SLM, we achieved significant improvements in PPL and WER over both the baseline N-gram model and SLM. Three scenarios for integrating neural networks in SLM where presented. Overall, the best studied model gave a 24.1% relative reduction in PPL over the best N-gram model and a 18.9% relative reduction over the baseline SLM. Our experiments also showed that the neural network models enhance the discriminative characteristic of the SLM by achieving a 4.8% relative reduction in WER

over the baseline SLM. The corresponding reductions is 9.8% over the standard *N*-gram model.

Overall, our experiments show that by using the syntactic information from the partial parses generated by the SLM, and by employing a probability estimation function that does not overfit as fast as the *N*-gram models (neural nets in our case), it is possible to improve significantly over the regular word *N*-gram models (either standard or neural net *N*-gram).

In our study the full integration training, where the "*N*-best" EM re-estimation is followed by SCORER training, gave the best perplexity. Therefore in future work the fully integrated model should be use for the WER experiments.

One would also like to experiment with different and extended probabilistic dependencies for the neural net models; given the amount of information available in a syntactic parse and the neural net capability in using it, this would most likely lead to improvements in perplexity. Alternatively, we intend to train our models to minimize specifically the WER, which would entail major modifications to the model's architecture.

We observed in our experiments that the neural net models give a different "view" of the data set than that of the *N*-gram models. Thus a natural extension of our work is to combine the neural net and *N*-gram models at the component level, rather than the current word-level interpolation.

We believe that the results of the "*N*-best" EM training can still be improved. Note that the objective function used was an approximation of the real EM auxiliary function, and that it is possible that the neural network was not optimally trained by learning from the severely pruned data set. As future work we intend to develop and use a different EM training procedure where the training of all model components is carried out on the partial parses stored in the stacks, instead of the last *N*-best parses. Intuitively, the number of partial parses used by this left-to-right EM training is larger than the "*N*-best" one, and the corresponding objective criteria is a better approximation of the true EM auxiliary function. We should note that this EM training would substitute for both the second and third stage of the SLM training procedure outlined in Section 3.6; therefore there would be no longer a need for a separate SCORER component.

Acknowledgments

The authors would like to thank Peng Xu for helpful discussions and for the preparation of the data sets used in this work. We would also like to thank Ciprian Chelba for stimulating discussions and helpful tips. The authors would also 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.

References

Bellegarda, J. R. (1997). A latent semantic analysis framework for large–span language modeling. In *Proceedings of the 5th European Conference on Speech Communication and Technology* (pp. 1451&1454). Vol. 3. Rhodes, Greece.

Bengio, Y., Ducharme, R., & Vincent, P. (2001). A neural probabilistic language model. *Advances in Neural Information Processing Systems, 13*, 933–938.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neuralprobabilistic language model. *Journal of Machine Learning Reseach, 3*, 1137–1155.

Berger, A. L., Pietra, S. A. D., & Pietra, V. J. D. (1996). A maximum entropyapproach to natural language processing. *Computational Linguistics, 22:1*, 39–72.

Bridle, J. S. (1989). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical patternrecognition. In F. Fougelman-Soulie and J. Herault (Eds.), *Neuro-computing: Algorithms, architectures and applicatations* (pp. 227&236).

Byrne, W., Gunawardana, A., & Khudanpur, S. (1998). Information geometry and EMvariants. Technical Report CLSP Research Note (17). Department of Electrical andComputer Engineering, The Johns Hopkins University, Baltimore, MD.

Charniak, E. (2001). Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting and 10th Conference of the European Chapter of ACL* (pp. 116–123). Toulouse, France.

Chelba, C. (1997). A structured language model. In *ACL-EACL, Student Section* (pp. 498&500). Madrid, Spain.

Chelba, C., & Jelinek, F. (2000). Structured language modeling. *Computer Speech and Language, 14:4*, 283–332.

Chelba, C., & Xu, P. (2001). Richer syntactic dependencies for structuredlanguage modeling. In *Proceedings of the Automatic Speech Recognition and Understanding Workshop*. Madonna di Campiglio, Trento-Italy.

Chen, S. F. & Goodman, J. (1999). An empirical study of smoothing techniquesfor language modeling. *Computer Speech and Language, 13*, 359–394.

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

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science, 41:6*, 391–407.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood fromincomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39*, 1–38.

Elman, J. L. (1991). Distributed representations, simple recurrent networks,and grammatical structure. *Machine Learning, 7*, 195–225.

Emami, A. (2003). Improving a connectionist based syntactical language model. In *Proceedings of the 8th European Conference on Speech Communication and Technology* (pp. 413–416), Vol. 1. Geneva, Switzerland.

Emami, A., & Jelinek, F. (2004). Exact training of a neural syntactic languagemodel. In *Proceedings of the IEEE International Conference onAcoustics, Speech and Signal Processing*. Montreal,Quebec.

Emami, A., Xu, P., & Jelinek, F. (2003). Using a connectionist model in asyntactical based language model. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 372–375). Vol. I. Hong Kong.

Fodor, J. A. & Pylyshyn, Z. W. (1988). Connectionism and cognitive structure: A critical analysis. *Cognition, 28*, 3–71.

Goodman, J. (2001). A bit of progress in language modeling. Technical Report MSR-TR-2001-72, Microsoft Research, Redmond, WA.

Gropp, W., Lusk, E., & Skjellum, A. (1999). Using MPI: Portable parallelProgramming with themessage-passing interface. Cambridge: MA: MIT Press.

Henderson, J. (2000). A neural network parser that handles sparse data. In *Proceedings of 6th International Workshop on Parsing Technologies* (pp. 123–134). Trento, Italy.

Henderson, J. (2003). Inducing history representations for broad coveragestatistical parsing. In *Proceedings of the North American Chapter of Association Computational Linguistics and Human Language Technology Conference HLT-NAACL*.

Hinton, G. E. (1986). Learning distributed representations of concepts. In R. G. M. Morris (Ed.), *Parallel distributed processing:Implications for psychology and Neurobiology* (pp. 46–61). Oxford, UK: Oxford University Press.

Ho, E. & Chan, L. (1999). How to design a connectionist holistic parser. *Neural Computation, 11:8*, 1995–2016.

Jelinek, F. (1998). Statistical methods for speech recognition. Cambridge, MA and London: MIT Press.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov sourceparameters from sparse data. In *Proceedings of Workshop on Pattern Recognition in Practice* (pp. 381–397). Amsterdam, The Netherlands: North Holland Publishing Co.

Kim, W., Khudanpur, S., & Wu, J. (2001). Smoothing issues in the structuredlanguage model. In *Proceedings of the 7th European Conference on Speech Communication and Technology* (pp. 717–720). Alborg, Denmark.

Kneser, R., & Ney, H. (1995). Improved backing-off for m-gram languagemodeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 181&184), Vol. I.

Lawrence, S., Giles, C. L., & Fong, S. (1996). Can recurrent neural networkslearn natural language grammars?. In *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1853&1858). Piscataway, NJ: IEEE Press.

Lawson, C. L., Hanson, R. J., Kincaid, D. R., & Krogh, F. T. (1979). Basiclinear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software, 5:3*, 308–323.

LeCun, Y. (1985). A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85* (pp. 599–604). Paris, France.

Miikkulainen, R. & Dyer, M. G. (1991). Natural language processing withmodular neural networks and distributed lexicon. *Cognitive Science, 15*, 343–399.

Ney, H., Essen, U., & Kneser, R. (1994). On structuring probabilisticdependencies in stochastic language modeling.. *Computer Speech and Language, 8*, 1–38.

Paul, D. B., & Baker, J. M. (1992). The design for the wall street journal-based CSR corpus. In *Proceedings of the DARPA SLS Workshop*.

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

Roark, B. (2001). Robust probabilistic predictive syntactic processing: Motivations, models and applications. Ph.D. thesis, Brown University, Providence, RI.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Leaning internalrepresentations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Paralleldistributed processing, I*. Cambridge, MA: MIT Press.

Schwenk, H., & Gauvain, J.-L. (2002). Connectionist language modeling for largevocabulary continuous speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, (pp. 765–768). Vol. II. Orlando, FL.

Van Uystel, D. H., Van Compernolle, D., & Wambacq, P. (2001). Maximum-likelihood training of the PLCG-based language model. In *Proceedings of the Automatic Speech Recognition andUnderstanding Workshop*. Madonna di Campiglio, Trento-Italy.

Werbos, P. J. (1974). Beyond regression: New tools for prediction and analysisin the behavioral sciences. Ph.D. thesis, Harvard University, Cambridge, MA.

Xu, P., Chelba, C., & Jelinek, F. (2002). A study on richer syntacticdependencies for structured language modeling. In *Proceedings of the 40th Annual Meeting of the Associationfor Computational Linguistics*. Philadelphia, PA.

Xu, P., Emami, A., & Jelinek, F. (2003). Training connectionist models for thestructured language model. In M. Collins, & M. Steedman (Eds.), *Proceedings of the 2003conference on empirical methods in natural language processing*. Sapporo, Japan: (pp. 160–167). Association for Computational Linguistics.

Xu, W., & Rudnicky, A. (2000). Can artificial neural networks learn languagemodels? In *Proceedings of 6th International Conference on Spoken Language Processing*. Beijing, China.

Received October 15, 2003
Revised June 26, 2004
Accepted November 15, 2004