Published in International Joint Conference on Neural Networks, pages 3059-3062, 2004

Efficient Training of Large Neural Networks for Language Modeling

Holger Schwenk
LIMSI - CNRS
bat 508, BP 133, 91403 Orsay cedex, FRANCE
E-mail: schwenk@limsi.fr

***Abstract*— Recently there has been increasing interest in using neural networks for language modeling. In contrast to the well known backoff n-gram language models, the neural network approach tries to limit the data sparseness problem by performing the estimation in a continuous space, allowing by this means smooth interpolations. The complexity to train such a model and to calculate one n-gram probability is however several orders of magnitude higher than for the backoff models, making the new approach difficult to use in real applications.**

In this paper several techniques are presented that allow the use of a neural network language model in a large vocabulary speech recognition system, in particular very fast lattice rescoring and efficient training of large neural networks on training corpora of over 10 million words. The described approach achieves significant word error reductions with respect to a carefully tuned 4-gram backoff language model in a state of the art conversational speech recognizer for the DARPA rich transcriptions evaluations.

I. Introduction

Language modeling is known to be a very important aspect of speech recognition. The most prominent approach, at least in state-of-the-art large vocabulary continuous speech recognition (LVCSR) systems, uses statistical language models based on n-grams, i.e. the model predicts the following word based on the previous $n-1$ words, ignoring all other context:

$$P(w_j = i|w_{j-n+1}, ..., w_2, w_1) \approx P(w_j = i|w_{j-n+1}, ..., w_{j-2}, w_{j-1}) \quad (1)$$

Due to data sparseness and computational complexity during speech decoding, n is usually limited to three or four words. Although these statistical language models (LM) perform quite well in practice, there are several drawbacks from a theoretical point of view due to the high dimensionality in the discrete space representation of the words. The vocabulary size in most current LVCSR systems is at least 64k words, which means that many of the $(64k)^3$ trigrams and $(64k)^4$ 4-grams are never observed during training.

"True generalization" is difficult to obtain in a discrete word indice space, since there is no obvious relation between the word indices. The probability distributions are not smooth functions since any change of the word indices can result in an arbitrary change of the LM probability. Various techniques for generalization to new word sequences have been proposed, in particular backing-off and smoothing. These approaches rely on the utilization of probabilities available for shorter contexts. Another approach is to use word classes in order to improve generalization (see for instance [1] for an overview).

Recently, a new approach has been developed that proposes to carry out the estimation task in a *continuous space* [2], [3]. The basic idea is to project the word indices onto a continuous space and to use a probability estimator operating on this space. Since the resulting probability functions are smooth functions of the word representation, better generalization to unknown n-grams can be expected. A neural network can be used to simultaneously learn the projection of the words onto the continuous space and the n-gram probability estimation. This connectionist LM has been evaluated on two text corpora ("Brown": 800K training words, English textbooks; and "Hansard": 32M words, Canadian Parliament proceedings) and achieved perplexity improvements of up to 30% with respect to a standard trigram [2].

The first evaluation of such an approach in a conversational speech recognizer demonstrated that it can be used to reduce the word error [4]. These results were later confirmed with an improved speech recognizer [5]. A neural network LM has also been used with a Syntactical Language Model yielding perplexity and word error improvements on the Wallstreet Journal corpus [6]. In this paper we present new algorithms for fast training and recognition of the neural network LM and discuss convergence properties. Results are presented for a conversational speech recognizer that achieved state-of-the-art performance in the last two NIST rich transcription evaluations.

The remainder of this paper is organized as follows. The next two sections present the basic architecture of the neural network language model and the baseline speech recognizer. Section IV and V describe how to achieve fast recognition and fast training with this approach, providing comparative results.

II. Architecture of the Neural Network LM

The architecture of the neural network n-gram LM is shown in Figure 1. A standard fully-connected multi-layer perceptron is used. The inputs to the neural network are the indices of the $n-1$ previous words in the vocabulary $h_j = w_{j-n+1}, ..., w_{j-2}, w_{j-1}$ and the outputs are the posterior probabilities of *all* words of the vocabulary:

$$P(w_j = i|h_j) \qquad \forall i \in [1, N] \quad (2)$$

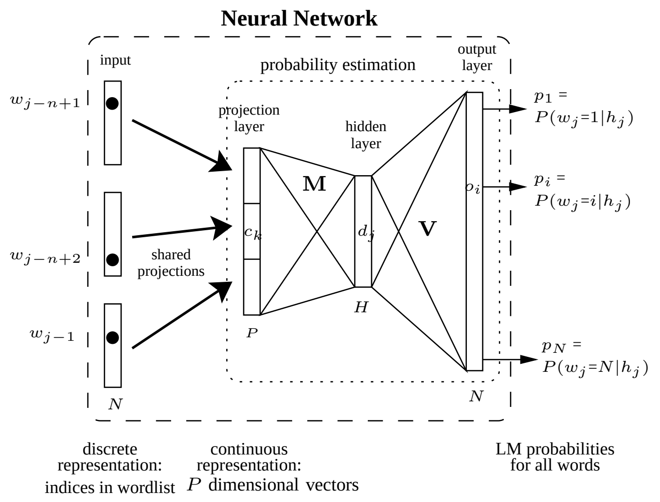

Fig. 1. Architecture of the neural network language model. h_j denotes the context $w_{j-n+1}, ..., w_{j-1}$. P is the size of one projection and H and N is the size of the hidden and output layer respectively. When shortlists are used the size of the output layer is much smaller then the size of the vocabulary.

where N is the size of the vocabulary. This can be contrasted with standard language modeling where each n-gram probability is calculated independently. The input uses the so-called 1-of-n coding, i.e., the *i-th* word of the vocabulary is coded by setting the *i-th* element of the vector to 1 and all the other elements to 0. This coding substantially simplifies the calculation of the projection layer since only the *i-th* line of the $N \times P$ dimensional projection matrix needs to be copied, where N is the size of the vocabulary and P the size of the projection.

Let c_k denote these projections, d_j the hidden layer activities, o_i the outputs, p_i their softmax normalization, m_{jk}, b_j, v_{ij} and k_i the hidden and output layer weights and the corresponding biases. Using matrix/vector notation the neural network performs the following operations:

$$\mathbf{d} = \tanh(\mathbf{M} * \mathbf{c} + \mathbf{b}) \quad (3)$$

$$\mathbf{o} = \mathbf{V} * \mathbf{d} + \mathbf{k} \quad (4)$$

$$\mathbf{p} = \exp(\mathbf{o}) \;/\; \sum_{k=1}^{N} e^{o_k} \quad (5)$$

where lower case bold letters denote vectors and upper case bold letter denote matrices. The tanh- and exp-function as well as the division are performed element wise. The value of the output neuron p_i corresponds directly to the probability $P(w_j = i|h_j)$. Training is performed with the standard back-propagation algorithm using cross-entropy as error function, and a weight decay regularization term. The targets are set to 1.0 for the next word in the training sentence and to 0.0 for all the other ones. It can be shown that the outputs of a neural network trained in this manner converge to the posterior probabilities. Therefore, the neural network minimizes directly the perplexity on the training data. Note also that the gradient is back-propagated through the projection-layer, which means that the neural network learns the projection of the words onto the continuous space that is best for the probability estimation task. The complexity of calculating the probability $P(w_j|h_j)$ of only one n-gram is given by the following equation[1]:

$$((n-1)P \times H) + H + (H \times N) + N \quad (6)$$

Since N is usually much larger than H, the complexity is dominated by the calculation at the output layer. For usual values of n=4, N=51k, P=50 and H=300, about 15 million floating point operations are needed to calculate one LM probability, which is computationally very expensive for full decoding or lattice rescoring. Note that due to the softmax normalization, all of the output activities need to be calculated even if only one probability is needed. The following section describes the baseline speech recognizer, followed by a detailed discussion how to achieve fast recognition and training with the neural network language model.

III. Baseline Speech Recognizer

In this paper the neural network language model is evaluated within a state of the art speech recognizer for conversational telephone speech (CTS). This task is known to be significantly more difficult than the recognition of Wallstreet journal or of broadcast news (BN) data. Based on the NIST speech recognition benchmarks [7], current best BN transcription systems achieve word error rates that approach 10% in 10xRT while the word error rate for the DARPA conversational telephone speech recognition task is about 23% using more computational resources (20–100xRT). A large amount of this difference can of course be attributed to the difficulties in acoustic modeling, but language modeling of conversational speech also faces problems that are much less important in BN data such as unconstrained speaking style, frequent grammatical errors, hesitations, restarts, etc. In addition, language modeling for conversational speech suffers from an extreme lack of adequate training data since the main data source is audio transcriptions, in contrast to the BN task for which other news sources are readily available. If we consider for instance the DARPA Switchboard (SWB) task, there are only about 12.4M words of transcriptions corresponding to the 860h of available transcribed acoustic training data. Unfortunately, collecting large amounts of conversational LM data is very costly. One possibility is to increase the amount of training data by selecting conversational like sentences in BN material and on the Internet, or by transforming other sources to be more conversational-like, see for instance [8], [9]. In this paper, we show that the neural network LM makes better use of the limited amount of data than conventional backoff n-gram models. Results are reported on the official test sets of the last three NIST CTS evaluations.

The LIMSI conversational speech recognizer is derived from the LIMSI broadcast news transcription system [9]. The word recognizer uses continuous density HMMs with Gaussian

[1]The activities of the projection layer are obtained by a simple table look-up and can be neglected in the complexity analysis.

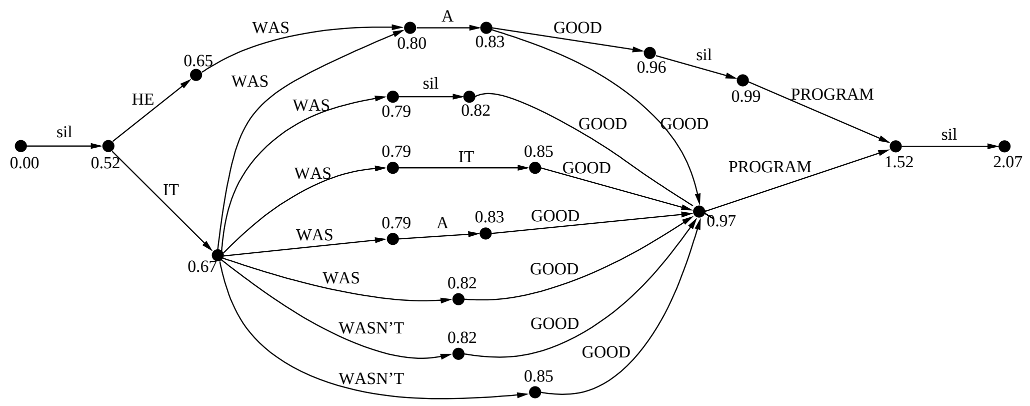

Fig. 2. Example of a lattice produced by the speech recognizer. A lattice is a graph of possible solutions where each edge corresponds to a hypothesized word with its acoustic and language model scores (for clarity these scores are not shown in the figure). [from Gauvain and Lamel, "Large Vocabulary Speech Recognition Based on Statistical Methods", in Pattern Recognition in Speech and Language Processing, CRC Press, 2003]

mixture for acoustic modeling and n-gram statistics estimated on large text corpora for language modeling. Each context-dependent phone model is a tied-state left-to-right CD-HMM with Gaussian mixture observation densities where the tied states are obtained by means of a decision tree. The acoustic feature vector has 39-components including 12 cepstrum coefficients and the log energy, along with their first and second derivatives. Cepstral mean and variance normalization are carried out on each conversation side. The acoustic models are MMIE trained on a total of about 860 hours of data.

Decoding is carried out in 4 passes. In the first pass the speaker gender for each conversation side is identified using Gaussian mixture models, and a fast trigram decode is performed to generate approximate transcriptions. These transcriptions are only used to compute the VTLN warp factors for each conversation side. All of the following passes make use of the VTLN-warped data. Each subsequent decoding pass generates a bigram word lattice per speaker turn which is expanded with the 4-gram baseline backoff LM and converted into a confusion network with posterior probabilities (see Figure 2 for an example of a lattice). The best hypothesis in the confusion network is used in the next decoding pass for unsupervised MLLR adaptation of the acoustic models. Two regression classes are used in the third pass, whereas five phonemic regression classes are used for the fourth pass. The overall run time is about 20xRT.

LM training

The main source for training a LM for conversational speech are transcriptions of the audio training corpora. In addition to this corpus of 12.4M words, the following texts have been used to train the 4-gram backoff LM:

- 240M words of commercially produced BN transcripts,
- 80M words of CNN television BN transcriptions,
- 180M words of conversational like data collected from the Internet.[2]

[2]This data has been provided by the University of Washington

Adding other sources, in particular newspaper text, was not useful. The baseline language model is constructed as follows: Separate backoff n-gram LMs are estimated on each of the above corpora using the modified version of Kneser-Ney smoothing as implemented in the SRI LM toolkit [10]. The LM vocabulary contains 51077 words. A single backoff LM is built by merging these models, estimating the interpolation coefficients with an EM procedure. The resulting LM has 20M bigrams, 31M trigrams and 24M 4-grams. This interpolated model is our baseline 4-gram LM. The perplexity on the Eval01 test data is 56.3 and the word error rate is 21.5%.

IV. Fast Recognition

Language models play an important role during decoding of continuous speech since the information provided about the most probable set of words given the current context is used to limit the exponential search space. Using the neural 4-gram LM directly during decoding imposes an important burden on search space organization since a context of three words must be kept. This lead to long decoding times in our first experiments when the neural LM was used directly during decoding [4]. In order to make the model tractable for LVCSR the following techniques have been applied:

1) *Lattice rescoring*: decoding is done with a standard backoff LM and a lattice is generated. The neural network LM is then used to rescore the lattice.
2) *Shortlists*: the neural network is only used to predict the LM probabilities of a subset of the whole vocabulary.
3) *Regrouping*: all LM probability requests in one lattice are collected and sorted. By these means all LM probability requests with the same context h_j lead to only one forward pass through the neural network.
4) *Block mode*: several examples are propagated at once through the neural network, allowing the use of faster matrix/matrix operations.
5) *CPU optimization*: machine specific libraries BLAS are used for fast matrix and vector operations.

It has been demonstrated in Section II that most calculations are done due to the large size of the output layer. Remember that all outputs need to be calculated in order to perform the softmax normalization even though only one LM probability is needed. Experiments using lattice rescoring with unnormalized LM scores led to much higher word error rates. One may argue that it is not very reasonable to spend a lot of time to get the LM probabilities of words that do not appear very often. Therefore, we chose to limit the output of the neural network to the s most frequent words, $s \ll N$, referred to as a *shortlist* in the following discussion. All words of the word list are still considered at the input of the neural network. The LM probabilities of words in the shortlist ($\hat{P}_N$) are calculated by the neural network and the LM probabilities of the remaining words (P_B) are obtained from a standard 4-gram backoff LM:

$$P(w_j|h_j) = \begin{cases} \hat{P}_N(w_j|h_j) \cdot P_S(h_j) & \text{if } w_j \in \text{shortlist} \\ P_B(w_j|h_j) & \text{else} \end{cases} \tag{7}$$

$$P_S(h_j) = \sum_{w \in shortlist(h_j)} P_B(w|h_j) \tag{8}$$

It can be considered that the neural network redistributes the probability mass of all the words in the shortlist.[3] This probability mass is precalculated and stored in the data structures of the standard 4-gram LM. A backoff technique is used if the probability mass for a requested input context is not directly available. Table I gives the coverage, i.e. the percentage of LM probabilities that are effectively calculated by the neural network when evaluating the perplexity on a development set of 56k words or when rescoring lattices.

TABLE I

COVERAGE FOR DIFFERENT SHORTLIST SIZES, I.E. PERCENTAGE OF 4-GRAMS THAT ARE ACTUALLY CALCULATED BY THE NEURAL LM

shortlist size	1024	2000	4096	8192
Eval01	89.3%	93.6%	96.8%	98.5%
lattice	88.5%	89.9%	90.4%	91.0%

During lattice rescoring LM probabilities with the same context h_j are often requested several times on potentially different nodes in the lattice (for instance the trigram `WAS A GOOD` in the lattice shown in figure 2). Collecting and regrouping all these calls prevents multiple forward passes since all LM predictions for the same context are immediately available (see Eqn. 2). Further improvements can be obtained by propagating several examples at once though the network, also know as bunch mode [11]. This results in using matrix/matrix instead of matrix/vector operations:

$$\mathbf{D} = \tanh(\mathbf{M} * \mathbf{C} + \mathbf{B}) \tag{9}$$

$$\mathbf{O} = \mathbf{V} * \mathbf{D} + \mathbf{K} \tag{10}$$

where $\mathbf{B}$ and $\mathbf{K}$ are obtained by duplicating the bias $\mathbf{b}$ and $\mathbf{k}$ repectiveley for each line of the matrix. These matrix/matrix operations can be aggressively optimized on current CPU architectures, e.g. using SSE2 instructions for Intel processors [12], [13]. Although the number of floating point operations to be performed is strictly identical to single example mode, an up to five times faster execution can be observed in function of the sizes of the matrices.

[3]Note that the sum of the probabilities of the words in the shortlist for a given context is normalized $\sum_{w \in shortlist} \hat{P}_N(w|h_j) = 1$.

The Eval01 test set consists of 6h of speech comprised of 5895 conversations sides. The lattices generated by the speech recognizer for this test set contain on average 511 nodes and 1481 arcs per conversation side. In total 3.8 million 4-gram LM probabilities were requested out of which 3.4 million (89.9%) have been processed by the neural network, i.e. the to be predicted word is among the 2000 most frequent words. After collecting and regrouping all LM calls in each lattice, only 1 million forward passes though the neural network have been performed, giving a cache hit rate of about 70%. Using a bunch size of 128 examples, the total processing time took less than 9 minutes on a Intel Xeon 2.8GHz processor, e.g. in 0.03 times real time. This corresponds to about 1.7 billion floating point operations per second (1.7 GFlops). Lattice rescoring without bunch mode and regrouping of all calls in one lattice is about ten times slower.

V. Fast Training

Language models are usually trained on text corpora of several million words. With a vocabulary size of 51k words, standard back-propagation training would take several weeks. Parallel implementations [3] and resampling techniques [14] that result in important speedups have been proposed. Parallel stochastic back-propagation of neural networks needs connections between the processors with very low latency, which are very costly. Optimized floating point operations are much more efficient if they are applied to data that is stored in continuous locations in memory, making a better use of cache and data prefetch capabilities of processors. This is not the case for resampling techniques. Therefore, a fixed size output layer was used and the words in the shortlist were rearranged in order to occupy continuous locations in memory.

In our initial implementation we used standard stochastic backpropagation and double precision for the floating point operations in order to ensure good convergence. Despite careful coding and optimized BLAS libraries [12], [13] for the matrix/vector operations, one epoch through a training corpus of 12.4M examples took about 47 hours on a Pentium Xeon 2.8 GHz processor. This time was reduced by more than a factor of 30 using:

- Floating point precision (1.5 times faster). Only a slight decrease in performance was observed due to the lower precision.
- Suppression of intermediate calculations when updating the weights (1.3 times faster).
- Bunch mode: forward and back-propagation of several examples at once (up to 10 times faster).
- Multi-processing: use of SMP-capable BLAS libraries for off-the-shelf bi-processor machines (1.5 times faster).

The most of the improvement was obtained by using bunch mode in the forward and backward pass. After calculating the

TABLE II

TRAINING TIMES REFLECTING THE DIFFERENT IMPROVEMENTS (ON A INTEL PENTIUM CPU AT 2.8 GHZ).

size of training data	double prec.	float prec.	bunch mode 2	4	8	16	32	128	SMP 128
1.1M words	2h	1h16	37m	31m	24m	14m	11m	8m18	5m50
12.4M words	47h	30h	10h12	8h18	6h51	4h01	2h51	2h09	1h27

derivatives of the error function $\Delta\mathbf{K}$ at the output layer, the following equations were used (similar to [11]):

$$\begin{aligned}
\mathbf{k} &= \mathbf{k} - \lambda\Delta\mathbf{K} * \mathbf{i} && (11)\\
\Delta\mathbf{B} &= \mathbf{V}^T * \Delta\mathbf{K} && (12)\\
\mathbf{V} &= -\lambda\Delta\mathbf{K} * \mathbf{D}^T + \alpha\mathbf{V} && (13)\\
\Delta\mathbf{B} &= \Delta\mathbf{B}.*(1-\mathbf{D}.*\mathbf{D}) && (14)\\
\mathbf{b} &= \mathbf{b} - \lambda\Delta\mathbf{B} * \mathbf{i} && (15)\\
\Delta\mathbf{C} &= \mathbf{M}^T * \Delta\mathbf{B} && (16)\\
\mathbf{M} &= -\lambda\Delta\mathbf{B} * \mathbf{C}^T + \alpha\mathbf{M} && (17)
\end{aligned}$$

where $\mathbf{i} = (1, 1, ...1)^T$, with a dimension of the bunch size. Note that the backprop and weight update step, including weight decay, is done in one operation using the GEMM function of the BLAS library (eqn. 13 and 17). For this, the weight decay factor ϵ is incorporated into $\alpha = 1 - \lambda\epsilon$. The update step of the projection matrix is not shown for clarity.

Table II summarizes the effect of the different techniques to speed up training. Extensive experiments were first done with a training corpus of 1.1M words and then applied to the full CTS corpus of 12.4M words. Bilmes et al. reported that the number of epochs needed to achieve the same MSE increases with the bunch size [11]. In our experiments the convergence behavior also changed with the bunch size, but after adapting the learning parameters of the neural network only small losses in perplexity were observed, dnd there was no impact on the word error when the neural LM was used in lattice rescoring.

A. Regrouping of training examples

The above described bunch mode optimization is generic and can be applied to any neural network learning problem, although it is most useful for large tasks like this one. In addition we propose new techniques that rely on the particular characteristics of the language model training corpus. A straightforward implementation of stochastic backpropagation is to cycle through the training corpus, in random order, and to perform a forward/backward pass and weight update for each 4-gram. However, in large texts it is frequent to encounter some 4-grams several times. This means that identical examples are trained several times. This is different from other pattern recognition tasks, for instance optical character recognition, for which it is unlikely to encounter twice the exactly same example (same input and targets) since the inputs are usually floating point numbers. In addition, for the LM task, we will find many contexts in the training texts for which several different words should be predicted, of course with potentially different probabilities. In other words, we can say that for each trigram there are usually several corresponding 4-grams. This fact can be used to substantially decrease the number of operations. The idea is to regroup these examples and to perform only one forward and backward pass though the neural network. The only difference is that there are now multiple output targets for each input context. Furthermore, 4-grams appearing multiple times can be learned at once by multiplying the corresponding gradients by the number of occurrences. This is equivalent to using bunch mode where each bunch includes all examples with the same trigram context. Alternatively, the targets can also be set to the estimated posterior probabilities, i.e. the relative frequencies of the 4-grams with a common context.

TABLE III

TRAINING TIMES FOR STOCHASTIC BACKPROPAGATION USING RANDOM PRESENTATION OF ALL 4-GRAMS IN COMPARISON TO REGROUPING ALL 4-GRAMS THAT HAVE A COMMON TRIGRAM CONTEXT (BUNCH SIZE=32).

words in corpus:	229k	1.1M	12.4M
Random presentation:			
# 4-grams	162k	927k	11.0M
training time	144s	11m	171m
Regrouping:			
# distinct 3-grams	124k	507k	3.3M
training time	106s	6m35s	58m

In stochastic backpropagation with random presentation order the number of forward and backward passes corresponds to the total number of 4-grams in the training corpus which is roughly equal to the number of words.[4] With the new algorithm the number of forward and backward passes is equal to the number of *distinct trigrams* in the training corpora. One of the major advantage of this approach is that the expected gain, i.e. the relation between the total number of 4-grams and distinct trigrams, increases with the size of the training corpus. Altough this approach is particularly interesting for large training corpora (see table III), we were not able to achieve the same convergence as with random presentation of individual 4-grams. Overall the perplexities obtained with the regrouping algorithm were slightly higher.

[4]The sentence were surrounded by begin and end of sentence markers. The first two words of a sentence do not form a full 4-gram, i.e. a sentence of 3-words has only two 4-grams.

B. *Neural network capacity*

Another open question is the capacity needed to learn language models with a neural network. Given the new learning algorithm it was possible to train large neural networks (up to 2M parameters) on the full set of 12.4M examples. These experiments are summarized in figure 3 which shows the perplexity when training neural networks with a hidden layer size between 100 and 1024 units. The other parameters were n=4, N=51k, shortlist=2000 and P=50. In all cases no overfitting was observed, neither when performing training for up to fifty epochs, nor when increasing the hidden layer size (the small oscillations are due to a new random order after each epoch). For comparison the perplexity of a backoff 4-gram trained on the same corpus of 12.4M words is 63.0 (stright line in figure 3).

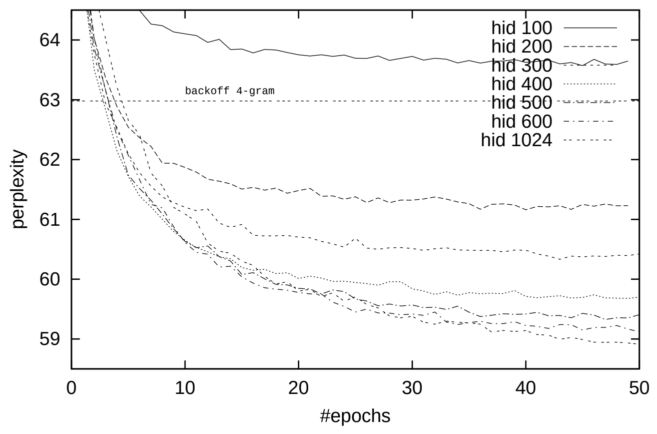

Fig. 3. Perplexity on the development data for different network sizes

The improvements in perplexity obtained by the very large networks did not lead to a notable decrease in word error (see table IV). This is not surprising since it is known that lower perplexity does not necessarily lead to lower word error due to the mismatch in the criteria. For these experiments, the neural network LM was interpolated with the reference backoff LM and the lattices of the last decoding pass were rescored.

TABLE IV

EVAL01 WORD ERROR RATES FOR DIFFERENT NETWORK SIZES

	backoff	number of hidden units of the neural LM						
	LM	100	200	300	400	**500**	600	1024
werr [%]	21.47	21.16	21.15	21.13	21.08	**20.99**	21.03	21.02

Larger short lists (up to 32k) were also tested, but the small gains in perplexity were not worth the increase in complexity. The time to rescore the lattices increasing almost linearly with the size of the hidden layer and the shortlist length, we decided to use a network with a 2k shortlist and with 500 hidden units. This neural network LM has been evaluated on the official test sets of the NIST CTS 2002 and 2003 evaluations. The word error rates decreased from 24.8 to 24.0% and 23.7 to 23.0% respectively compared to the optimized 4-gram backoff LM.

VI. CONCLUSIONS

This paper has described a large scale application of a neural network language model to conversational speech recognition. By using neural network language models we seek to achieve better estimation of the LM probabilities by performing the estimation in a continuous space, allowing by these means "smooth interpolations."

Several algorithms have been developed to reduce the high complexity of the approach during training and recognition. On a standard PC one training epoch with 11M examples takes 1h30m for a neural network with 650k parameters. Decoding of continuous speech is done with help of optimized lattice rescoring in less then 0.05 times real time. This approach achieved consistent word error reductions of about 0.6% with respect to a carefully tuned backoff 4-gram LM. The neural network LM was employed in the LIMSI systems evaluated in the last two NIST benchmarks on conversational speech recognition.

ACKNOWLEDGMENT

The authors would like to recognize the contributions of J.L. Gauvain, L. Lamel, G. Adda, and F. Lefèvre for their involvement in the development of the LIMSI conversational speech recognition system on top of which this work is based.

REFERENCES

[1] S. F. Chen and J. T. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[2] Y. Bengio and R. Ducharme, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems*, vol. 13. Morgan Kaufmann, 2001.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, no. 2, pp. 1137–1155, 2003.

[4] H. Schwenk and J.-L. Gauvain, "Connectionist language modeling for large vocabulary continuous speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing*, 2002, pp. I: 765–768.

[5] ——, "Using Continuous Space Language Models for Conversational Speech Recognition," in *ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition*, Tokyo, April 2003.

[6] A. Emami, P. Xu, and F. Jelinek, "Using a connectionist model in a syntactical based language model," in *International Conference on Acoustics, Speech, and Signal Processing*, 2003, pp. I:272–375.

[7] A. Lee, J. Fiscus, J. Garofolo, M. Przybocki, A. Martin, G. Sanders, and D. Pallett, "Spring speech-to-text transcription evaluation results," in *Rich Transcription Workshop, Boston*, May 19 2003.

[8] R. Iyer and M. Ostendorf, "Relevance weighting for combining multi-domain data for n-gram language modeling," *Computer Speech & Language*, vol. 13, no. 3, pp. 267–282, 1999.

[9] J.-L. Gauvain, L. Lamel, H. Schwenk, G. Adda, L. Chen, and F. Lefèvre, "Conversational telephone speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing*, 2003, pp. I:212–215.

[10] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *International Conference on Speech and Language Processing*, 2002, pp. II: 901–904.

[11] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel, "Using phipac to speed error back-propagation learning," in *International Conference on Acoustics, Speech, and Signal Processing*, 1997, pp. V:4153–4156.

[12] Intel's MKL, "Intel math kernel library, http://www.intel.com/software/products/mkl/."

[13] ATLAS, "Automatically tuned linear algebra software, http://www.netlib.org/atlas."

[14] Y. Bengio and J.-S. Sénécal, "Quick training of probabilistic neural nets by importance sampling," in *AISTATS Conference*, 2003.