Training Neural Network Language Models On Very Large Corpora [*]

Holger Schwenk and **Jean-Luc Gauvain**
LIMSI-CNRS
BP 133, 91436 Orsay cedex, FRANCE
schwenk,gauvain@limsi.fr

Abstract

During the last years there has been growing interest in using neural networks for language modeling. In contrast to the well known back-off n-gram language models, the neural network approach attempts to overcome the data sparseness problem by performing the estimation in a continuous space. This type of language model was mostly used for tasks for which only a very limited amount of in-domain training data is available.

In this paper we present new algorithms to train a neural network language model on very large text corpora. This makes possible the use of the approach in domains where several hundreds of millions words of texts are available. The neural network language model is evaluated in a state-of-the-art real-time continuous speech recognizer for French Broadcast News. Word error reductions of 0.5% absolute are reported using only a very limited amount of additional processing time.

1 Introduction

Language models play an important role in many applications like character and speech recognition, machine translation and information retrieval. Several approaches have been developed during the last decades like n-gram back-off word models (Katz, 1987), class models (Brown et al., 1992), structured language models (Chelba and Jelinek, 2000) or maximum entropy language models (Rosenfeld, 1996). To the best of our knowledge word and class n-gram back-off language models are still the dominant approach, at least in applications like large vocabulary continuous speech recognition or statistical machine translation. In many publications it has been reported that modified Kneser-Ney smoothing (Chen and Goodman, 1999) achieves the best results. All the reference back-off language models (LM) described in this paper are build with this technique, using the SRI LM toolkit (Stolcke, 2002).

The field of natural language processing has recently seen some changes by the introduction of new statistical techniques that are motivated by successful approaches from the machine learning community, in particular continuous space LMs using neural networks (Bengio and Ducharme, 2001; Bengio et al., 2003; Schwenk and Gauvain, 2002; Schwenk and Gauvain, 2004; Emami and Jelinek, 2004), Random Forest LMs (Xu and Jelinek, 2004) and Random cluster LMs (Emami and Jelinek, 2005). Usually new approaches are first verified on small tasks using a limited amount of LM training data. For instance, experiments have been performed using the Brown corpus (1.1M words), parts of the Wallstreet journal corpus (19M words) or transcriptions of acoustic training data (up to 22M words). It is much more challenging to compare the new statistical techniques to carefully optimized back-off LM trained on large amounts of data (several hundred millions words). Training may be difficult and very

[*]This work was partially financed by the European Commission under the FP6 Integrated Project TC-STAR.

time consuming and the algorithms used with several tens of millions examples may be impracticable for larger amounts. Training back-off LMs on large amounts of data is not a problem, as long as powerful machines with enough memory are available in order to calculate the word statistics. Practice has also shown that back-off LMs seem to perform very well when large amounts of training data are available and it is not clear that the above mentioned new approaches are still of benefit in this situation.

In this paper we compare the neural network language model to n-gram model with modified Kneser-Ney smoothing using LM training corpora of up to 600M words. New algorithms are presented to effectively train the neural network on such amounts of data and the necessary capacity is analyzed. The LMs are evaluated in a real-time state-of-the-art speech recognizer for French Broadcast News. Word error reductions of up to 0.5% absolute are reported.

2 Architecture of the neural network LM

The basic idea of the neural network LM is to project the word indices onto a continuous space and to use a probability estimator operating on this space (Bengio and Ducharme, 2001; Bengio et al., 2003). 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 to estimate the n-gram probabilities. This is still a n-gram approach, but the LM posterior probabilities are "interpolated" for any possible context of length n-1 instead of backing-off to shorter contexts.

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:

$$P(w_j = i|h_j) \qquad \forall i \in [1, N] \tag{1}$$

where N is the size of the vocabulary. 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

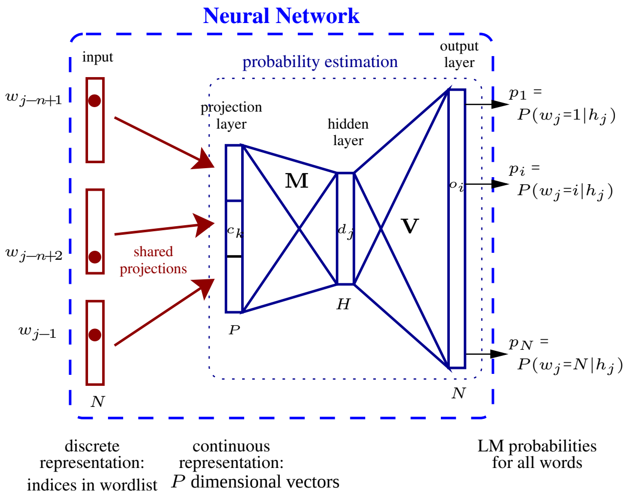

Figure 1: Architecture of the neural network language model. h_j denotes the context $w_{j-n+1}, ..., w_{j-1}$. P is the size of one projection and H and N is the size of the hidden and output layer respectively. When shortlists are used the size of the output layer is much smaller then the size of the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

0. The *i-th* line of the $N \times P$ dimensional projection matrix corresponds to the continuous representation of the i-th word. Let us denote c_k these projections, d_j the hidden layer activities, o_i the outputs, p_i their softmax normalization, and m_{jl}, b_j, v_{ij} and k_i the hidden and output layer weights and the corresponding biases. Using these notations the neural network performs the following operations:

$$d_j = \tanh\left(\sum_l m_{jl}\, c_l + b_j\right) \tag{2}$$

$$o_i = \sum_j v_{ij}\, d_j + k_i \tag{3}$$

$$p_i = e^{o_i} \bigg/ \sum_{k=1}^{N} e^{o_k} \tag{4}$$

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 minimizing the following error function:

$$E = \sum_{i=1}^{N} t_i \, log\, p_i + \beta(\sum_{jl} m_{jl}^2 + \sum_{ij} v_{ij}^2) \tag{5}$$

where t_i denotes the desired output, i.e., the probability should be 1.0 for the next word in the training sentence and 0.0 for all the other ones. The first part of this equation is the cross-entropy between the output and the target probability distributions, and the second part is a regularization term that aims to prevent the neural network from overfitting the training data (weight decay). The parameter β has to be determined experimentally.

It can be shown that the outputs of a neural network trained in this manner converge to the posterior probabilities. Therefore, the neural network directly minimizes 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 to calculate one probability with this basic version of the neural network LM is quite high:

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

where P is the size of one projection and H and N is the size of the hidden and output layer respectively. Usual values are n=4, P=50 to 200, H=400 to 1000 and N=40k to 200k. The complexity is dominated by the large size of the output layer. In this paper the improvements described in (Schwenk, 2004) have been used:

1. *Lattice rescoring*: speech recognition is done with a standard back-off LM and a word 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 probabilities needed for one lattice are collected and sorted. By these means all LM probability requests with the same context h_t 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 BLAS libraries are used for fast matrix and vector operations.

The idea behind shortlists is to use the neural network only to predict the s most frequent words, $s \ll |V|$, reducing by these means drastically the complexity. 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 ($\hat{P}_B$) are obtained from a standard 4-gram back-off LM:

$$\hat{P}(w_t|h_t) = \begin{cases} \hat{P}_N(w_t|h_t) P_S(h_t) & \text{if } w_t \in \text{shortlist} \\ \hat{P}_B(w_t|h_t) & \text{else} \end{cases} \quad (7)$$

$$P_S(h_t) = \sum_{w \in shortlist(h_t)} \hat{P}_B(w|h_t) \quad (8)$$

It can be considered that the neural network redistributes the probability mass of all the words in the shortlist. This probability mass is precalculated and stored in the data structures of the back-off LM. A back-off technique is used if the probability mass for a requested input context is not directly available.

Normally, the output of a speech recognition system is the most likely word sequence given the acoustic signal, but it is often advantageous to preserve more information for subsequent processing steps. This is usually done by generating a lattice, a graph of possible solutions where each arc corresponds to a hypothesized word with its acoustic and language model scores. In the context of this work LIMSI's standard large vocabulary continuous speech recognition decoder is used to generate lattices using a n-gram back-off LM. These lattices are then processed by a separate tool and all the LM probabilities on the arcs are replaced by those calculated by the neural network LM. During this lattice rescoring LM probabilities with the same context h_t are often requested several times on potentially different nodes in the lattice. Collecting and regrouping all these calls prevents multiple forward passes since all LM predictions for the same context are immediately available at the output.

Further improvements can be obtained by propagating several examples at once though the network, also known as bunch mode (Bilmes et al., 1997; Schwenk, 2004). In comparison to equation 2 and 3, this results in using matrix/matrix instead of matrix/vector operations which can be aggressively optimized on current CPU architectures. The Intel

Math Kernel Library was used.[1] Bunch mode is also used for training the neural network. Training of a typical network with a hidden layer with 500 nodes and a shortlist of length 2000 (about 1M parameters) take less than one hour for one epoch through four million examples on a standard PC.

3 Application to Speech Recognition

In this paper the neural network LM is evaluated in a real-time speech recognizer for French Broadcast News. This is a very challenging task since the incorporation of the neural network LM into the speech recognizer must be very effective due to the time constraints. The speech recognizer itself runs in 0.95xRT[2] and the neural network in less than 0.05xRT. The compute platform is an Intel Pentium 4 extreme (3.2GHz, 4GB RAM) running Fedora Core 2 with hyper-threading.

The acoustic model uses tied-state position-dependent triphones trained on about 190 hours of Broadcast News data. The speech features consist of 39 cepstral parameters derived from a Mel frequency spectrum estimated on the 0-8kHz band (or 0-3.8kHz for telephone data) every 10ms. These cepstral coefficients are normalized on a segment cluster basis using cepstral mean removal and variance normalization. The feature vectors are linearly transformed (MLLT) to better fit the diagonal covariance Gaussians used for acoustic modeling.

Decoding is performed in two passes. The first fast pass generates an initial hypothesis, followed by acoustic model adaptation (CMLLR and MLLR) and a second decode pass using the adapted models. Each pass generates a word lattice which is expanded with a 4-gram LM. The best solution is then extracted using pronunciation probabilities and consensus decoding. Both passes use very tight pruning thresholds, especially for the first pass, and fast Gaussian computation based on Gaussian short lists. For the final decoding pass, the acoustic models include 23k position-dependent triphones with 12k tied states, obtained using a divisive decision tree based clustering algorithm with a 35 base phone set.

[1]http://www.intel.com/software/products/mkl/

[2]In speech recognition, processing time is measured in multiples of the length of the speech signal, the real time factor xRT. For a speech signal of 2h, a processing time of 0.5xRT corresponds to 1h of calculation.

The system is described in more detail in (Gauvain et al., 2005).

The neural network LM is used in the last pass to rescore the lattices. A short-list of length 8192 was used in order to fulfill the constraints on the processing time (the complexity of the neural network to calculate a LM probability is almost linear with the length of the short-list). This gives a coverage of about 85% when rescoring the lattices, i.e. the percentage of LM requests that are actually performed by the neural network.

3.1 Language model training data

The following resources have been used for language modeling:

- Transcriptions of the acoustic training data (4.0M words)
- Commercial transcriptions (88.5M words)
- Newspaper texts (508M words)
- WEB data (13.6M words)

First a language model was built for each corpus using modified Kneser-Ney smoothing as implemented in the SRI LM toolkit (Stolcke, 2002). The individual LMs were then interpolated and merged together. An EM procedure was used to determine the coefficients that minimize the perplexity on the development data. Table 1 summarizes the characteristics of the individual text corpora.

corpus	**#words**	**Perpl.**	**Coeffs.**
Acoustic transcr.	4M	107.4	0.43
Commercial transcr.	88.5M	137.8	0.14
Newspaper texts	508M	103.0	0.35
WEB texts	13.6M	136.7	0.08
All interpolated	614M	70.2	-

Table 1: Characteristics of the text corpora (number of words, perplexity on the development corpus and interpolation coefficients)

Although the detailed transcriptions of the audio data represent only a small fraction of the available data, they get an interpolation coefficient of 0.43. This shows clearly that they are the most appropriate text source for the task. The commercial transcripts, the newspaper and WEB texts reflect less well the speaking style of broadcast news, but this is to some extent counterbalanced by the large amount of data. One could say that these texts are helpful to learn the general grammar of the language. The word list includes 65301 words and the OOV rate is 0.95% on a development set of 158k words.

3.2 Training on in-domain data only

Following the above discussion, it seems natural to first train a neural network LM on the transcriptions of the acoustic data only. The architecture of the neural network is as follows: a continuous word representation of dimension 50, one hidden layer with 500 neurons and an output layer limited to the 8192 most frequent words. This results in 3.2M parameters for the continuous representation of the words and about 4.2M parameters for the second part of the neural network that estimates the probabilities. The network is trained using standard stochastic back-propagation.[3] The learning rate was set to 0.005 with an exponential decay and the regularization term is weighted with 0.00003. Note that fast training of neural networks with more than 4M parameters on 4M examples is already a challenge. The same fast algorithms as described in (Schwenk, 2004) were used. Apparent convergence is obtained after about 40 epochs though the training data, each one taking 2h40 on standard PC equipped with two Intel Xeon 2.8GHz CPUs.

The neural network LM alone achieves a perplexity of 103.0 which is only a 4% relative reduction with respect to the back-off LM (107.4, see Table 1). If this neural network LM is interpolated with the back-off LM trained on the whole training set the perplexity decreases from 70.2 to 67.6. Despite this small improvements in perplexity a notable word error reduction was obtained from 14.24% to 14.02%, with the lattice rescoring taking less than 0.05xRT. In the following sections, it is shown that larger improvements can be obtained by training the neural network on more data.

3.3 Adding selected data

Training the neural network LM with stochastic back-propagation on all the available text corpora would take quite a long time. The estimated time for one training epoch with the 88M words of commercial transcriptions is 58h, and more than 12 days if all the 508M words of newspaper texts were used. This is of course not very practicable. One solution to this problem is to select a subset of the data that seems to be most useful for the task. This was done by selecting six month of the commercial transcriptions that minimize the perplexity on the development set. This gives a total of 22M words and the training time is about 14h per epoch.

One can ask if the capacity of the neural network should be augmented in order to deal with the increased number of examples. Experiments with hidden layer sizes from 400 to 1000 neurons have been performed (see Table 2).

size	**400**	**500**	**600**	**1000***
Tr. time	11h20	13h50	16h15	11+16h
Px alone	100.5	100.1	99.5	94.5
interpol.	68.3	68.3	68.2	68.0
Werr	13.99%	13.97%	13.96%	13.92%

* Interpolation of networks with 400 and 600 hidden units.

Table 2: Performance for a neural network LM and training time per epoch as a function of the size of the hidden layer (fixed 6 months subset of commercial transcripts).

Although there is a small decrease in perplexity and word error when increasing the dimension of the hidden layer, this is at the expense of a higher processing time. The training and recognition time are in fact almost linear to the size of the hidden layer. An alternative approach to augment the capacity of the neural network is to modify the dimension of the continuous representation of the words (in the range 50 to 150). The idea behind this is that the probability estimation may be easier in a higher dimensional space (instead of augmenting the capacity of the non-linear probability estimator itself). This is similar in spirit to the theory behind support vector machines (Vapnik, 1998).

Increasing the dimension of the projection layer has several advantages as can be seen from the Figure 2. First, the perplexity and word error rates are lower than those obtained when the size of the

[3]The weights are updated after each example.

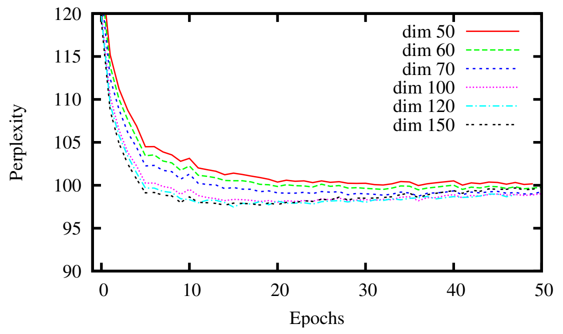

Figure 2: Perplexity in function of the size of the continuous word representation (500 hidden units, fixed 6 months subset of commercial transcripts).

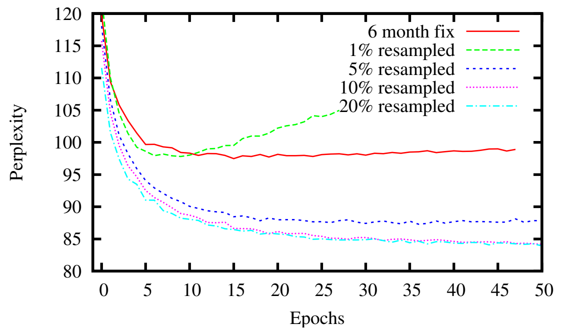

Figure 3: Perplexity when resampling different random subsets of the commercial transcriptions. (word representation of dimension 120, 500 hidden units)

hidden layer is increased. Second, convergence is faster: the best result is obtained after about 15 epochs while up to 40 are needed with large hidden layers. Finally, increasing the size of the continuous word representation has only a small effect on the training and recognition complexity of the neural network[4] since most of the calculation is done to propagate and learn the connections between the hidden and the output layer (see equation 6). The best result was obtained with a 120 dimensional continuous word representation. The perplexity is 67.9 after interpolation with the back-off LM and the word error rate is 13.88%.

3.4 Training on all available data

In this section an algorithm is proposed for training the neural network on arbitrary large training corpora. The basic idea is quite simple: instead of performing several epochs over the whole training data, a different small random subset is used at each epoch. This procedure has several advantages:

- There is no limit on the amount of training data,
- After some epochs, it is likely that all the training examples have been seen at least once,
- Changing the examples after each epoch adds noise to the training procedure. This potentially increases the generalization performance.

This algorithm is summarized in figure 4. The parameters of this algorithm are the size of the random subsets that are used at each epoch. We chose to always use the full corpus of transcriptions of the acoustic data since this is the most appropriate data for the task. Experiments with different random subsets of the commercial transcriptions and the newspaper texts have been performed (see Figure 3 and 5). In all cases the same neural network architecture was used, i.e a 120 dimensional continuous word representation and 500 hidden units. Some experiments with larger hidden units showed basically the same convergence behavior. The learning rate was again set to 0.005, but with a slower exponential decay.

First of all it can be seen from Figure 3 that the results are better when using random subsets instead of a fixed selection of 6 months, although each random subset is actually smaller (for instance a total of 12.5M examples for a subset of 10%). Best results were obtained when taking 10% of the commercial

```
Repeat
    Select training data:
        – Use all acoustic transcriptions (4M words)
        – Extract random subset of examples
          from the large corpora
        – Shuffle data
  + Train network for one epoch
    (performing weight updates after each example)
  + Test performance on development data
Until convergence
```

Figure 4: Training algorithm for large corpora

[4]14h20 for P=120 and H=500.

	Back-off LM	**Neural Network LM**				
Training data [#words]	600M	4M	22M	92.5M*	600M*	
Training time [h/epoch]	-	2h40	14h	9h40	12h	3 × 12h
Perplexity (NN LM alone)	-	103.0	97.5	84.0	80.0	76.5
Perplexity (interpolated LMs)	70.2	67.6	67.9	66.7	66.5	65.9
Word error rate (interpolated LMs)	**14.24%**	14.02%	13.88%	13.81%	**13.75%**	**13.61%**

* By resampling different random parts at the beginning of each epoch.

Table 3: Comparison of the back-off and the neural network LM using different amounts of training data. The perplexities are given for the neural network LM alone and interpolated with the back-off LM trained on all the data. The last column corresponds to three interpolated neural network LMs.

transcriptions. The perplexity is 66.7 after interpolation with the back-off LM and the word error rate is 13.81% (see summary in Table 3). Larger subsets of the commercial transcriptions lead to slower training, but don't give better results.

Encouraged by these results, we also included the 508M words of newspaper texts in the training data. The size of the random subsets were chosen in order to use between 4 and 9M words of each corpus. Figure 5 summarizes the results. There seems to be no obvious benefit from resampling large subsets of the individual corpora. We choose to resample 10% of the commercial transcriptions and 1% of the newspaper texts.

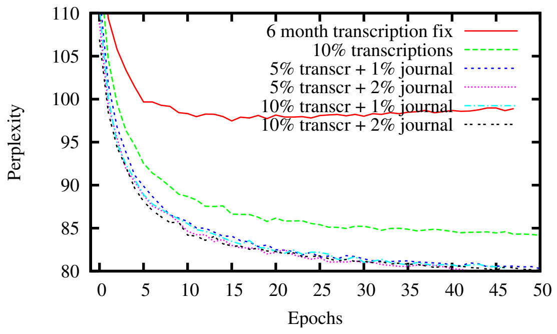

Figure 5: Perplexity when resampling different random subsets of the commercial transcriptions and the newspaper texts.

Table 3 summarizes the results of the different neural network LMs. It can be clearly seen that the perplexity of the neural network LM alone decreases significantly with the amount of training data used. The perplexity after interpolation with the back-off LM changes only by a small amount, but there is a notable improvement in word error rate. This is another experimental evidence that the perplexity of a LM is not directly related to the word error rate.

The best neural network LM achieves a word error reduction of 0.5% absolute with respect to the carefully tuned back-off LM (14.24% → 13.75%). The additional processing time needed to rescore the lattices is less than 0.05xRT. This is a significant improvement, in particular for a fast real-time continuous speech recognition system. When more processing time is available a word error rate of 13.61% can be achieved by interpolating three neural networks together (in 0.14xRT).

3.5 Using a better speech recognizer

The experimental results have also been validated using a second speech recognizer running in about 7xRT. This systems differs from the real-time recognizer by a larger 200k word-list, additional acoustic model adaptation passes and less pruning. Details are described in (Gauvain et al., 2005). The word error rate of the reference system using a back-off LM is 10.74%. This can be reduced to 10.51% using a neural network LM trained on the fine transcriptions only and to 10.20% when the neural network LM is trained on all data using the described resampling approach. Lattice rescoring takes about 0.2xRT.

4 Conclusions and future work

Neural network language models are becoming a serious alternative to the widely used back-off language models. Consistent improvements in perplexity and word error rate have been reported (Bengio et al., 2003; Schwenk and Gauvain, 2004; Schwenk and Gauvain, 2005; Emami and Jelinek, 2004). In these works, the amount of training data was however limited to a maximum of 20M words due to the high complexity of the training algorithm.

In this paper new techniques have been described to train neural network language models on large amounts of text corpora (up to 600M words). The evaluation with a state-of-the-art speech recognition system for French Broadcast News showed a significant word error reduction of 0.5% absolute. The neural network LMs is incorporated into the speech recognizer by rescoring lattices. This is done in less than 0.05xRT.

Several extensions of the learning algorithm itself are promising. We are in particular interested in smarter ways to select different subsets from the large corpus at each epoch (instead of a random choice). One possibility would be to use active learning, i.e. focusing on examples that are most useful to decrease the perplexity. One could also imagine to associate a probability to each training example and to use these probabilities to weight the random sampling. These probabilities would be updated after each epoch. This is similar to boosting techniques (Freund, 1995) which build sequentially classifiers that focus on examples wrongly classified by the preceding one.

5 Acknowledgment

The authors would like to thank Yoshua Bengio for fruitful discussions and helpful comments. The authors would like to recognize the contributions of G. Adda, M. Adda and L. Lamel for their involvement in the development of the speech recognition systems on top of which this work is based.

References

Yoshua Bengio and Rejean Ducharme. 2001. A neural probabilistic language model. In *NIPS*, volume 13.

Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(2):1137–1155.

Jeff Bilmes, Krste Asanovic, Chee whye Chin, and Jim Demmel. 1997. Using phipac to speed error backpropagation learning. In *ICASSP*, pages V:4153–4156.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–470.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech & Language*, 13(4):283–332.

Stanley F. Chen and Joshua T. Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394.

Ahmad Emami and Frederick Jelinek. 2004. Exact training of a neural syntactic language model. In *ICASSP*, pages I:245–248.

Ahmad Emami and Frederick Jelinek. 2005. Random clusterings for language modeling. In *ICASSP*, pages I:581–584.

Yoav Freund. 1995. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285.

Jean-Luc Gauvain, Gilles Adda, Martine Adda-Decker, Alexandre Allauzen, Veronique Gendner, Lori Lamel, and Holger Schwenk. 2005. Where are we in transcribing BN french? In *Eurospeech*.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on ASSP*, 35(3):400–401.

Ronald Rosenfeld. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech & Language*, 10(3):187–228.

Holger Schwenk and Jean-Luc Gauvain. 2002. Connectionist language modeling for large vocabulary continuous speech recognition. In *ICASSP*, pages I: 765–768.

Holger Schwenk and Jean-Luc Gauvain. 2004. Neural network language models for conversational speech recognition. In *ICSLP*, pages 1215–1218.

Holger Schwenk and Jean-Luc Gauvain. 2005. Building continuous space language models for transcribing european languages. In *Eurospeech*.

Holger Schwenk. 2004. Efficient training of large neural networks for language modeling. In *IJCNN*, pages 3059–3062.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *ICSLP*, pages II: 901–904.

Vladimir Vapnik. 1998. *Statistical Learning Theory*. Wiley, New York.

Peng Xu and Frederick Jelinek. 2004. Random forest in language modeling. In *EMNLP*, pages 325–332.