# EMPIRICAL STUDY OF NEURAL NETWORK LANGUAGE MODELS FOR ARABIC SPEECH RECOGNITION

*Ahmad Emami and Lidia Mangu*

IBM T J Watson Research Center
Yorktown Heights, NY 10598
{*emami, mangu*}*@us.ibm.com*

## ABSTRACT

In this paper we investigate the use of neural network language models for Arabic speech recognition. By using a distributed representation of words, the neural network model allows for more robust generalization and is better able to fight the data sparseness problem. We investigate different configurations of the neural probabilistic model, experimenting with such parameters as $N$-gram order, output vocabulary, normalization method, and model size and parameters. Experiments were carried out on Arabic broadcast news and broadcast conversations data and the optimized neural network language models showed significant improvements over the baseline $N$-gram model.

***Index Terms***— Language Modeling, Speech Recognition, Neural Networks.

## 1. INTRODUCTION

Statistical language models are widely used in fields dealing with speech or natural language. For example, in the commonly used statistical formulation of the speech recognition problem, the recognizer seeks to find the word string:

$$\hat{W}=\arg\max_W P(A|W)P(W), \quad (1)$$

where A denotes the observed speech signal, $P(A|W)$ is the probability of producing $A$ when $W$ is spoken, and $P(W)$, called the language model (LM), is the *prior* probability that $W$ was spoken.

The role of a statistical language model is to assign a probability $P(W)$ to any given word string $W = w_1 w_2 \dots w_n$. This is usually done in a left-to-right manner by factoring the probability:

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

where the sequence of words $w_1 w_2 \dots w_j$ is denoted by $W_1^j$. Ideally, the language model should use the entire history $W_1^{i-1}$ to make its prediction for word $w_i$. However, because of data sparseness some equivalence classification of histories $W_1^{i-1}$ should be employed. The popular $N$-gram models which classify the word string $W_1^{i-1}$ into $W_{i-N+1}^{i-1}$ perform surprisingly well given their simple structure. Nevertheless, they lack the ability to use longer histories, and still suffer from severe data sparseness even for small values of $N$.

Most of the language modeling research has been carried out for English. The data sparseness problem, which is the biggest issue in language modeling, gets only worse when we are dealing with a highly inflectional language such as Arabic. This is mainly due to the increased vocabulary size in the inflectional languages which increases the space of possible word combinations in an $N$-gram. There have been efforts addressing this issue, for example breaking each inflected word into parts (eg. by way of morphological analysis) and using these parts, which have a smaller vocabulary, for language modeling [1, 2, 7, 3]. However breaking words into parts will increase the effective $N$-gram order which in turn will require a model that is better than regular $N$-gram models in capturing long dependencies.

*Distributed representation* of words, combined with a neural network for probability estimation, has been shown to be a powerful smoothing method and has enabled the use of longer and richer probabilistic dependencies [4, 5]. Since these models work in a distributed space, and since function estimation is better understood and solved in distributed (continuous) spaces, it can be assumed that the neural network models are better in generalizing to unseen data. A great advantage of this approach is its ability to fight data sparseness. The model size grows at most linearly with the $N$-gram order or the vocabulary size, compared to exponential and polynomial growth respectively for regular $N$-gram models. It has been shown that this method improves in both perplexity and word error rate over state-of-the-art smoothing methods when it is used with standard $N$-gram history (i.e. $N-1$ previous words) [4, 6, 7] as well as when it is used in the context of a syntactic based language model [8, 9, 10].

In this paper we investigate the use of distributed representations and neural network language models for Ara-

We would like to acknowledge the support of DARPA under Grant HR0011-06-2-0001 for funding part of this work.

bic speech recognition. There are many parameters of the neural network model that need to be optimized. We experiment with different configurations of the neural network model such as $N$-gram order, output vocabulary selection, and model size and parameters.

Section 2 gives a brief overview of the neural network language model. In Section 3 we describe the automatic speech recognition (ASR) system and the experimental setup. In Section 4 the various configurations and parameters that we investigated are described and the experimental results are shown.

## 2. NEURAL NETWORK MODEL

In a neural network based language model words are represented by points in a continuous multi-dimensional feature space and the probability of a sequence of words is computed by means of a neural network. The feature vectors of the preceding words make up the input to the neural network, which then will produce a probability distribution over a given vocabulary [5]. The main idea behind this model is to make the estimation task easier by mapping words from the original high-dimensional discrete space to a low-dimensional continuous space where probability distributions are smooth functions in their variables. The network achieves generalization by assigning to an unseen word sequence a probability close to that of a "similar" word string seen in the training data. The similarity is defined as being close in the multi-dimensional feature space. Since the probability function is a smooth function of the feature vectors, a small change in the features leads to only a small change in the probability.

### 2.1. Model Details

Suppose the goal is to compute the probability of a certain event $Y = y$ given the values $x_1, x_2, \cdots, x_m$ of $m$ conditioning variables. The conditional probability function $P(y|x_1, x_2, \cdots, x_m)$ is determined in two parts:

1. A mapping that associates a real vector of fixed dimension with each token in the *input vocabulary* $V_i$: the set of all tokens that can be used for prediction.
2. A function which takes as the input the concatenation of the feature vectors of the input items $x_1, x_2, \cdots, x_m$. The function produces a conditional probability distribution (a vector) over the *output vocabulary* $V_o$: the set of all tokens to be predicted.

Note that the input and output vocabularies $V_i$ and $V_o$ are independent of each other and can be completely different. Training is achieved by searching for parameters $\Phi$ of the neural network and the values of feature vectors that maximize the penalized log-likelihood of the training corpus:

$$L=\frac{1}{T}\sum_t logP(y^t|x_1^t,...,x_m^t;\Phi)-R(\Phi) \quad (2)$$

where superscript $t$ denotes the $t^{th}$ event in the training data, $T$ is the training data size and $R(\Phi)$ is a regularization term, which in our case is a factor of the L2 norm squared of the hidden and output layer weights.

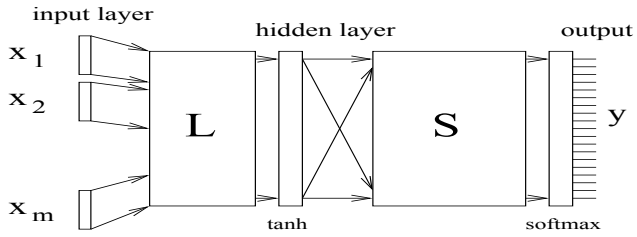


**Fig. 1**. The neural network architecture

The model architecture is given in Figure 1 [5]. The neural network is fully connected and contains one hidden layer. The operations of the input and hidden layers are given by:

$$\vec{f}=(f_1,...,f_{d\cdot m})=(\vec{f}(x_1),\vec{f}(x_2),\cdots,\vec{f}(x_m))$$
$$g_k=\tanh\left(\sum_j f_j L_{kj}+B_k^1\right) \qquad k=1,2,...,h$$

where $\vec{f}(x)$ is the $d$-dimensional feature vector for token $x$. The weights and biases of the hidden layer are denoted by $L_{kj}$ and $B_k^1$ respectively, and $h$ is the number of hidden units.

At the output layer of the network we have:

$$z_k=\sum_j g_j S_{kj}+B_k^2 \qquad k=1,2,...,|V_o|$$
$$p_k=\frac{e^{z_k}}{\sum_j e^{z_j}} \qquad k=1,2,...,|V_o| \quad (3)$$

with the weights and biases of the output layer denoted by $S_{kj}$ and $B_k^2$ respectively. The softmax layer (Equation 3) ensures that the outputs are valid probabilities and provides a suitable framework for learning a probability distribution.

The $k^{th}$ output of the neural network, corresponding to the $k^{th}$ item $y_k$ of the output vocabulary, is the desired conditional probability: $p_k = P(y^t = y_k|x_1^t, ..., x_m^t)$.

The neural network weights and biases, as well as the input feature vectors, are learned simultaneously using stochastic gradient descent training via back-propagation algorithm, with the objective function being the one given in Equation 2.

One great advantage of this model is that context length (number of inputs) can be increased resulting in at most linear increase in model size, in contrast to exponential growth for regular $N$-gram models. This makes the neural network a very suitable model for capturing longer and richer probabilistic dependencies.

#### *2.1.1. Implementation and Speed-ups*

The computational complexity of the neural network language model is very high since it requires normalization over

all the words in the output vocabulary (Equation 3). In contrast, in the regular $N$-gram model the same probability computation consists of few (up to $N$) table lookup operations.

For the conventional model configurations ($N < 10, d < 200, h < 400$, and thousands of words in the output vocabulary $V_o$) the bulk of computation is carried out in the output layer. Therefore, reducing the output vocabulary size (or the number of hidden units) has an almost linear effect on reducing the training or decoding time with the neural network model. There have been attempts at reducing the effective output vocabulary size by using either sampling methods or hierarchical architectures, or by simply limiting the output to a small subset of the original language model vocabulary [11, 12]. In the latter case the model can not produce probabilities for words outside the output vocabulary and substitute probabilities need to be used. Two such substitution methods are described and investigated in Section 4.3.

Another speed-up method is the bunch mode training where several instances are propagated through the network at once. This results in matrix-by-matrix (instead of matrix-by-vector) operations which are much better optimized on the current CPU architectures [13, 6]. We further reduced the effective output vocabulary size by parallelizing the output layer computations over multiple machines (usually 8-10 CPUs). The computations before the output layer are carried out by all the processes separately, but the output layer itself is divided among the processes. All that is needed to be communicated between the processes is the normalization denominator in Equation 3 and that is performed by a single reduce operation using the Message Passing Interface (MPI) library [10, 5]. Therefore, as long as the number of processes is kept small relative to the output vocabulary size, the overall time spent for communications between processes is negligible compared to the CPU time spent on computations. Furthermore, using the bunch mode helps reduce the communication latency since there is only one message sent for every bunch of examples.

## 3. EXPERIMENTAL SETUP

All the experiments are carried out in the context of lattice rescoring. We first present the speech recognition system that produced the lattices used in this study, then we describe the baseline language model, the training data used for building neural network (NN) language models and the test sets used in the experiments.

### 3.1. Description of the ASR system

The speech recognition system has a cross-adapted architecture between unvowelized and vowelized speaker-adaptive trained (SAT) acoustic models. The distinction between the two comes from the explicit modeling of short vowels which are pronounced in Arabic but almost never transcribed. Both sets of models are trained discriminatively on approx. 500 hours of supervised data and 2000 hours of unsupervised data. More details about the training of the Arabic models can be found in [14]. We generated a set of lattices with an average link density of 256. The ASR decoder that was used to generate the lattices is described in [15].

### 3.2. Baseline Language Model

The following corpora are used in the baseline language model:

- Transcripts of audio data from a variety of sources released by LDC (7M words)
- Arabic Gigaword corpus, 5 parts (approx. 400M words)
- Web downloaded data from CMU (95M words)
- Web downloaded data from Cambridge University (200M words)
- Web text, namely newsgroups and weblogs, collected by LDC (28M words)

From all these sources a vocabulary of 737K words was extracted. We built 4-gram language models with modified Kneser-Ney smoothing on each of these sources. These models were then linearly interpolated and the result was pruned down, generating a final language model consisting of 65M $N$-gram entries. A heldout set of 80K words was extracted from the acoustic transcripts released by LDC and was used for optimizing the interpolation weights.

In all the experiments in this paper, the NN LM was linearly interpolated with the baseline 4-gram language model. The interpolation weights were optimized on the heldout set using the EM algorithm. We did not observe much variability in the value of the interpolation weight for the NN LM; in most of the cases it was found to be approximately 0.2.

### 3.3. Training and Test Sets

For the purpose of this study we used two test sets totaling 4.5 hours of Arabic broadcast news and broadcast conversations speech: *Dev07* set distributed by LDC in March 2007 and consisting of 2.5 hours of speech (18186 words), and *Eval06* consisting of 2 hours of speech (12286 words) which was extracted from the test set used in DARPA's GALE 2006 evaluation. The baseline word error rates (WER) on these two test sets are 12.40% and 20.84% respectively. The training data used to build NN language models was the 7M words of broadcast news and broadcast conversations acoustic transcripts released by LDC.

### 3.4. Baseline Neural Network Model Setup

All the NN language models used in this paper were trained on the 7M words training data mentioned above. The network

contained 100 hidden units and used 30 dimensional feature vectors unless otherwise specified. The initial learning rate was set to $10^{-3}$ and this was decayed during training according to the formula $\frac{10^{-3}}{1+10^{-8}*n_t}$ where $n_t$ is the total number of examples propagated through the network until time $t$.

## 4. CONFIGURATION OF THE NEURAL NETWORK MODEL

There are many aspects of the configuration of the NN model that need to be considered and optimized for a given language modeling task. They range from general language modeling variables such as $N$-gram order and vocabulary, to the NN model specific parameters such as number of hidden units, dimensionality of the feature vectors, and learning rate. In the following sections, we will discuss some of these configuration variables and show their effect on the performance of the NN language model.

### 4.1. $N$-gram Order

It was mentioned earlier that a great advantage of the NN model is that its complexity increases at most linearly as the number of inputs ($m$) to the model is increased. Therefore the NN model is theoretically less susceptible to over-training when the $N$-gram order in incremented. Given this fact, an obvious experiment is to try neural network $N$-gram models of different orders to see if we can get improvements using longer contexts in word probability estimation. How much the neural network can learn from longer $N$-grams is obviously dependent on the size of the training data. Under the assumption of better generalization from distributed representation of words, the NN model should be able to better utilize longer contexts than regular $N$-gram models for the same amount of training data.

Table 1 shows the heldout data perplexities and test set word error rates (WER) for models of different history lengths. The output vocabulary is set to the 10K most frequent words. All results are reported after training the model for 20 iterations. The $N$-gram order has been increased from 4 to 8. It can be seen that changing the history length does not affect the WER in any significant manner. On the other hand, as we increase the history length the number of unique $N$-grams in the lattices to be rescored is increased. We have chosen the 6-gram model for all the subsequent experiments as it strikes a good balance between model performance and the practicality of lattice rescoring.

### 4.2. Output Vocabulary Selection

As mentioned in Section 2, the neural network model makes use of two separate and independent word lists, namely the input and output vocabularies. The size of the output vocabulary should be kept as small as possible to keep the computational complexity of the model at a practical level. Since the neural network model produces probability distributions on and only on the words in the output vocabulary, the selection of output vocabulary can have a significant effect on the performance of the model.

| model | heldout PPL | eval06 | dev07 |
|---|---|---|---|
| baseline | 901.5 | 20.84% | 12.40% |
| 4-gram | 853.6 | 20.33% | 12.25% |
| 6-gram | 844.6 | 20.35% | 12.19% |
| 8-gram | 838.7 | 20.42% | 12.13% |

**Table 1**. Neural network LMs perplexity and WER for different N-gram orders

| output vocabulary | vocab size | train | eval06 | dev07 |
|---|---|---|---|---|
| ov10k | 10000 | 21.43% | 18.59% | 17.06% |
| ov20k | 20000 | 15.84% | 12.22% | 10.58% |
| ov40k | 40000 | 11.60% | 7.92% | 6.58% |
| eval06 CN voc branch | 7792 | 38.37% | 27.07% | - |
| eval06 CN voc all | 9073 | 33.49% | 17.58% | - |
| eval06 LAT voc | 23046 | 28.67% | 12.38% | - |
| dev07 CN voc branch | 8763 | 35.70% | - | 19.82% |
| dev07 CN voc all | 10737 | 30.07% | - | 5.11% |
| dev07 LAT voc | 19659 | 25.50% | - | 3.5% |

**Table 2**. Out of vocabulary rates for different vocabs

One can choose the top $M$ most frequent words from the training data, expecting that this will ensure a low out of vocabulary (OOV) rate on the test set as well. This also warrants, for a fixed $M$, the highest coverage of the training data and thus results in a better convergence in training. On the other hand, since we are using the NN model in a lattice rescoring framework, we have some information about the words likely to occur in the test set. Therefore, one can build the output vocabulary exclusively from the words in the lattices, or for better discrimination, from the words found in the confusion networks ( [16]) which have fewer spurious words.

Table 2 shows the OOV rates on the training data and on the reference transcriptions of the test sets for different output vocabularies. Table 3 shows the corresponding WER results. Entries ov$M$k refer to the model with output vocabularies consisting of the top most $M$ words. There are separate entries for confusion network (CN) and lattice (LAT) vocabs. Two types of CN vocabs can be built: one which contains only the words that are confusable, i.e. words for which alternative hypotheses exist (CN voc branch), and the other where all the words in the confusion network are used (CN voc all). Note that LAT and CN vocabularies are test set specific, so separate vocabs and NN models need to be built for each test set.

As can be expected and as evidenced in Table 2, for eval06 test set where the baseline WER is higher, confusion network and lattice extracted vocabularies do not have as good coverage of the reference transcriptions as is the case for the dev07

| output vocabulary | eval06 | dev07 |
|---|---|---|
| ov10k | 20.33% | 12.16% |
| ov20k | 20.11% | 12.05% |
| ov40k | 20.27% | 12.03% |
| eval06 CN voc branch | 20.23% | - |
| eval06 CN voc all | 20.10% | - |
| eval06 LAT voc | 20.27% | - |
| dev07 CN voc branch | - | 12.01% |
| dev07 CN voc all | - | 11.99% |
| dev07 LAT voc | - | 12.16% |

**Table 3**. Word Error Rates for NN LMs with different output vocabs

test set. One can observe from Table 3 that confusion network (CN) vocabs perform consistently well, especially when the vocabulary consists of all the words in the confusion network ('CN voc all' entries). It should be noted that the CN vocab models achieve good results with relatively small output vocabulary sizes. This is of great practical importance since one can build CN vocab models which train faster and perform as good or better than models with output vocabs consisting of the top $M$ most frequent words. However, the disadvantage of CN or LAT vocabularies is that a new NN model, with a new output vocab, needs to be trained for every new test set.

Another interesting observation is that output vocabularies extracted from lattices do not perform as well as the ones built from confusion networks. One explanation is that using lattice extracted vocabularies forces the neural network to focus on and discriminate among more irrelevant words than using CN vocabularies.

### 4.3. Normalization

As mentioned in Section 2, it is not practical to produce complete probability distributions over the very large vocabularies typically used in ASR systems. For this reason the output vocabulary is usually limited to a comparatively small subset of the input vocabulary. However such a model is not able to produce probabilities for words outside its output vocab and substitute probabilities need to be used to cover for the probabilities that are missing.

We have investigated two methods for extending the NN model output to the words outside the output vocabulary while keeping the distribution normalized. The first approach, which we refer to as *norm*, is to use a back-off language model as a substitute for the probabilities of $N$-grams where the predicted word is not in the output vocabulary:

$$P(w_i|W_{i-N+1}^{i-1})=\begin{cases} P_{NN}(w_i|W_{i-N+1}^{i-1})\cdot\alpha(W_{i-N+1}^{i-1}), & \text{if } w_i \in V_o; \\ P_{bo}(w_i|W_{i-N+1}^{i-1}), & \text{otherwise.} \end{cases}$$

where $P_{NN}(w_i|W_{i-N+1}^{i-1})$ and $P_{bo}(w_i|W_{i-N+1}^{i-1})$ are the neural network and the back-off (substitute) distributions respectively, and $\alpha(W_{i-N+1}^{i-1})$ is a normalization factor ensuring that the probabilities add up to 1 for a given history $W_{i-N+1}^{i-1}$ [11, 6]. Note that the back-off language model can be of any order equal or less than $N$. This approach however requires computing and storing the constants $\alpha(W_{i-N+1}^{i-1})$ for every history $W_{i-N+1}^{i-1}$ that is found in the lattices. An easier option is to use the non-smooth distribution:

$$P(w_i|W_{i-N+1}^{i-1})=\begin{cases} P_{NN}(w_i|W_{i-N+1}^{i-1}), & \text{if } w_i \in V_o; \\ 0, & \text{otherwise.} \end{cases}$$

which we refer to as *znorm*. It is possible to use zero probabilities for words outside the output vocabulary since the neural network model is interpolated with smooth regular $N$-gram models in all the experiments.

| | eval06 | dev07 |
|---|---|---|
| ov10K znorm | 20.35% | 12.19% |
| ov10K norm | 20.30% | 12.14% |
| ov20K znorm | 20.11% | 12.05% |
| ov20K norm | 20.04% | 12.02% |
| eval06 CN voc branch znorm | 20.22% | - |
| eval06 CN voc branch norm | 20.2% | - |
| dev07 CN voc branch znorm | - | 11.99% |
| dev07 CN voc branch norm | - | 11.99% |

**Table 4**. Effect of normalization on neural network model performance

Table 4 compares the results of the two different normalization methods. As can be seen, using zeros (znorm) instead of probabilities from a back-off language model (norm) does not change the performance of the model in any significant manner. This is especially true for the CN vocab models where the best results were obtained. For this reason, and to avoid the trouble of pre-computing constants $\alpha(W_{i-N+1}^{i-1})$, we use the znorm method in all the experiments in this paper unless otherwise specified.

### 4.4. Model Size and Parameters

All the previous experiments were carried out with a fixed number of hidden units (100) and word feature vector dimension (30). These parameters are by no means optimal and were chosen solely based on our earlier experiences with the neural network LMs. Intuitively, one would need more hidden units when there are more patterns to be learned by the model, for example when using more training data or when working on a language when there is a higher variation of permissible $N$-gram combinations (eg. Arabic). Another approach in increasing the model complexity is to use higher dimensional feature vector for word representations, thus forcing the model to learn the probability distributions in a higher dimensional space.

We experimented with NN LMs with both more hidden units and higher dimensional feature vectors. The results are given in Table 5. The h$x$ and d$y$ entries refer to models with $x$ hidden units and $y$-dimensional feature vectors respectively.

| model | eval06 | dev07 |
|---|---|---|
| ov20k, h100 | 20.11% | 12.05% |
| ov20k, h200 | 20.17% | 12.06% |
| ov20k, h400 | 20.16% | 12.14% |
| dev07 CN voc branch, d30 | - | 12.01% |
| dev07 CN voc branch, d120 | - | 12.03% |

**Table 5**. Effect of model size on performance

The results from the table show that increasing the model size by either adding more hidden units or using higher dimensional feature vectors does not have any significant effect on the model performance. Based on these results and given that increasing the model size increases the memory requirements and/or the computational complexity of the model significantly,we have decided to use the baseline configuration of 100 hidden units and 30 dimensional feature vectors for our experiments.

## 5. CONCLUSIONS AND FUTURE WORK

In this paper we studied the use of neural network language models for Arabic broadcast news and broadcast conversations speech recognition. The NN models improved considerably over the baseline 4-gram model, resulting in reductions of up to 0.8% absolute and 3.8% relative in WER. We investigated the choice of different parameters and configurations for the NN LMs. In most of the cases the choice of the parameters did not have a significant effect on the performance of the model. However some interesting patterns were observed. For example, choosing the output vocabulary from the confusion network seems to work as well or better than choosing the most frequent words, even though the resulting vocabulary is usually smaller in size. Also, using the simple znorm method instead of the more elaborate normalization based on substitute language models does not seem to have any noticeable effect on the performance. This is especially the case for CN vocab models where the two normalizatoin methods are almost identical in performance. On the other hand, using larger neural networks did not seem to improve the performance, which is counter intuitive.

We plan to extend this work by taking advantage of the morphological analysis of the Arabic language. We also plan to investigate methods for discriminative training of the NN LM so as to better disambiguate between word choices in a lattice or confusion network. Another future direction is to study the convergence of the neural network model for large amounts of training data.

## 6. REFERENCES


[1] Dimitra Vergyri, Katrin Kirchhoff, Kevin Duh, and Andreas Stolcke, "Morphology-based language modeling for arabic speech recognition," in *ICSLP*, 2004.

[2] Katrin Kirchhoff, Dimitra Vergyri, Jeff Bilmes, Kevin Duh, and Andreas Stolcke, "Morphology-based language modeling for conversational arabic speech recognition.," *Computer Speech and Language*, vol. 20, no. 4, 2006.

[3] Ruhi Sarikaya and Yonggang Deng, "Joint morphological-lexical language modeling for machine translation," in *Human Language Technology conference*, 2007.

[4] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems*, 2001.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Reseach*, vol. 3, 2003.

[6] Holger Schwenk, "Continuous space language models," *Comput. Speech Lang.*, vol. 21, no. 3, 2007.

[7] Andrei Alexandrescu and Katrin Kirchhoff, "Factored neural language models," in *NAACL*, June 2006.

[8] Ahmad Emami, Peng Xu, and Frederick Jelinek, "Using a connectionist model in a syntactical based language model," in *ICASSP*, 2003.

[9] Peng Xu, Ahmad Emami, and Frederick Jelinek, "Training connectionist models for the structured language model," in *Proceedings of EMNLP*, 2003.

[10] Ahmad Emami and Frederick Jelinek, "A neural syntactic language model," *Machine Learning*, vol. 60, no. 1-3, pp. 195–227, 2005.

[11] Holger Schwenk and Jean-Luc Gauvain, "Connectionist language modeling for large vocabulary continuous speech recognition," in *Proc. ICASSP*, 2002.

[12] Holger Schwenk and Jean-Luc Gauvain, "Training neural network language models on very large corpora," in *Human Language Technology conference*, 2005.

[13] J. Bilmes, K. Asanovi, c Chin, and J. Demmel, "Using phipac to speed error back-propagation learning," in *Proceedings of ICASSP*, 1997.

[14] H. Soltau, G. Saon, B. Kingsbury, J. Kuo, L. Mangu, D. Povey, and G. Zweig, "The IBM 2006 GALE Arabic ASR system," in *ICASSP*, 2007.

[15] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast LVCSR decoder," in *Interspeech-05*, 2005.

[16] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus among words: lattice-based word error minimization," in *Eurospeech*, 1999.