Strategies for Training Large Scale Neural Network Language Models

Tomáš Mikolov [#1], Anoop Deoras [*2], Daniel Povey [†3], Lukáš Burget [#4], Jan "Honza" Černocký [#5]

[#] *Brno University of Technology, Speech@FIT, Brno, Czech Republic*
[1] imikolov@fit.vutbr.cz, [4] burget@fit.vutbr.cz, [5] cernocky@fit.vutbr.cz

[*] *HLT-COE, CLSP, Johns Hopkins University, Baltimore, MD, USA*
[2] adeoras@jhu.edu

[†] *Microsoft Research, Redmond, WA, USA*
[3] dpovey@microsoft.com

***Abstract*—We describe how to effectively train neural network based language models on large data sets. Fast convergence during training and better overall performance is observed when the training data are sorted by their relevance. We introduce hash-based implementation of a maximum entropy model, that can be trained as a part of the neural network model. This leads to significant reduction of computational complexity. We achieved around 10% relative reduction of word error rate on English Broadcast News speech recognition task, against large 4-gram model trained on 400M tokens.**

I. Introduction

Statistical language models are important part of many applications that work with natural language, such as machine translation and automatic speech recognition. Backoff models based on n-gram statistics have been dominating the language modeling field for almost three decades, mainly due to their simplicity and low computational complexity.

In recent years, some other types of language models have also attracted a lot of attention. The most successful examples of such models are Maximum entropy language model [1] (ME LM) and Neural network based language model [2] (NN LM). The main reason for their attractiveness is their ability to model natural language more precisely than the backoff models. Their important drawback is their huge computational complexity. Thus, a lot of research effort has been devoted towards making these models more computationally efficient.

In this paper, we briefly mention existing approaches for reducing the computational complexity of NN LMs (most of these approaches are applicable to ME LMs). We propose new simple techniques that can be used to reduce computational cost of the training and test phases. We show that these new techniques are complementary to existing approaches.

Most interestingly, we show that a standard neural network language model can be trained together with a maximum entropy model, which can be seen as a part of the the neural network. We introduce a hash-based implementation of a class-based maximum entropy model, that allows us to easily control the trade-off between memory complexity and computational complexity.

We report results on the NIST RT04 Broadcast News speech recognition task. We use lattices generated from IBM Attila decoder [3] that uses state-of-the-art discriminatively trained acoustic models. The language models for this task are trained on about 400M tokens. This highly competitive setup has been used in the recent summer workshop at Johns Hopkins University [4]. In our experiments, we work with the Recurrent neural network language model [5] (RNN LM), as we have recently shown that RNN LMs can outperform standard feedforward NN LMs [6].

II. Model Description

A maximum entropy model has the following form:

$$P(w|h) = \frac{e^{\sum_{i=1}^{N} \lambda_i f_i(h,w)}}{\sum_w e^{\sum_{i=1}^{N} \lambda_i f_i(h,w)}}, \quad (1)$$

where $\mathbf{f}$ is a set of features, $\boldsymbol{\lambda}$ is a set of weights and h is a history. Training maximum entropy model consists of learning the set of weights $\boldsymbol{\lambda}$. Usual features are n-grams, but it is easy to integrate any information source, for example triggers or syntactic features [1]. The choice of features is usually done manually, and significantly affects the overall performance of the model.

The standard neural network language model has a very similar form. The main difference is that the features for this model are automatically learned as a function of the history. Also, the usual features for the ME model are binary, while NN models use continuous-valued features. We can describe the NN LM as follows:

$$P(w|h) = \frac{e^{\sum_{i=1}^{N} \lambda_i f_i(s,w)}}{\sum_w e^{\sum_{i=1}^{N} \lambda_i f_i(s,w)}}, \quad (2)$$

where s is a state of the hidden layer. For the feedforward NN LM architecture introduced by Bengio et al. in [2], the state of the hidden layer depends on a projection layer, that is formed as a projection of $N-1$ recent words into low-dimensional space. After the model is trained, similar words have similar low-dimensional representations.

Alternatively, the state of hidden layer can depend on the most recent word and the state in the previous time step. Thus, the time is not represented explicitly. This recurrence allows the hidden layer to represent low-dimensional representation

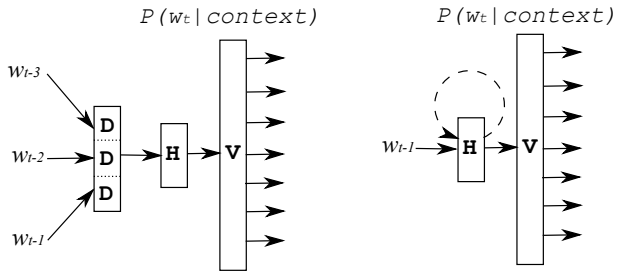

Fig. 1. Feedforward neural network 4-gram model (on the left) and Recurrent neural network language model (on the right).

of the entire history (or in other words, it provides the model with a memory). The architecture is called the Recurrent neural network based language model (RNN LM), and we have described it in [7] and [5]. We have recently shown that RNN LM achieves state of the art performance on the well-known Penn Treebank Corpus, and that it outperforms standard feedforward NN LM architectures, as well as many other advanced language modeling techniques [6].

It is interesting to see that ME models trained with just n-gram features have almost the same performance as usual backoff models with modified Kneser-Ney smoothing [8]. On the other hand, neural network models, due to their ability to cluster similar words (or similar histories), outperform the state-of-the-art backoff models. Moreover, NN LMs are complementary to backoff models, and further gains can be obtained by linearly interpolating them.

We can view ME models as NN models with no hidden layer, with the input layer directly connected to the output layer. Such a model has been already described in [9], where it was shown that it can be trained to perform similarly to a Kneser-Ney smoothed n-gram model.

III. Computational Complexity

The computational complexity of a neural network language model is high for several reasons, and there have been attempts to deal with almost all of them. The training time of N-gram neural network LM is proportional to

$$I \times W \times \Big((N-1) \times D \times H + H \times V \Big), \tag{3}$$

where I is the number of training epochs before convergence is achieved, W is the number of tokens in the training set, N is the N-gram order, D is the dimensionality of words in the low-dimensional space, H is size of the hidden layer and V size of the vocabulary (see Figure 1). The term $(N-1) \times D$ is equal to size of the projection layer. The recurrent NN LM has computational complexity:

$$I \times W \times \Big(H \times H + H \times V \Big). \tag{4}$$

It can be seen that for increasing order N, the complexity of the feedforward architecture increases linearly, while it remains constant for the recurrent one (actually, N has no meaning in RNN LM). Assuming that the maximum entropy model uses feature set $\mathbf{f}$ with full N-gram features and that it is trained using on-line stochastic gradient descent in the same way as neural network models, its computational complexity is

$$I \times W \times \Big(N \times V \Big). \tag{5}$$

A. Reduction of Training Epochs

Training of NN LMs is usually performed by stochastic gradient descent with on-line update of weights. Usually, it is reported that 10-50 training epochs are needed to obtain convergence, although there are exceptions (in [9], it is reported that thousands of epochs were needed). In the next section, we will show that good performance can be achieved while performing as few as 7 training epochs, if the training data are sorted by their complexity.

B. Reduction of Number of Training Tokens

In usual circumstances, backoff n-gram language models are trained on as much data as are available. However, for common speech recognition tasks, only small part of this data is in-domain. Out-of-domain data usually occupy more than 90% size of the training corpora, but their weight in the final model is relatively low. Thus, NN LMs are usually trained only on the in-domain corpora. In [10], NN LMs are trained on in-domain data plus some randomly subsampled part of the out-of-domain data that is randomly chosen at the start of each training epoch.

In a vast majority of cases, NN LMs for LVCSR tasks are trained on 5-30M tokens. Although the subsampling trick can be used to claim that the neural network model has seen all training data at least once, simple subsampling techniques lead to severe performance degradation, against a model that is trained on all data - a more advanced subsampling technique has been recently introduced in [11].

C. Reduction of Vocabulary Size

It can be seen that most of the computational complexity of NN LM in Eq. 3 is caused by the huge term $H \times V$. For LVCSR tasks, the size of the hidden layer H is usually between 100 and 500 neurons, and the size of the vocabulary V is between 50k and 300k words. Thus, many attempts have been made to reduce the size of the vocabulary. The most simple technique is to compute probability distribution only for the top M words in the neural network model; the rest of the words use backoff n-gram probabilities. The list of top M words is then called a *shortlist*. However, it was shown in [12] that this technique causes severe degradation of performance for small values of M, and even with $M = 2000$, the complexity of the $H \times V$ term is still significant.

More successful approaches are based on Goodman's trick for speeding up maximum entropy models [13]. Each word from the vocabulary is assigned to a class, and only the probability distribution over classes is computed. In the second step, we compute the probability distribution over words that are members of a particular class (we know this class from the predicted word whose probability we are trying to compute). As the number of classes can be very small (several hundreds),

this is a more effective solution than using shortlists, and the performance degradation is smaller. We have recently shown that meaningful classes can be formed very easily, by considering only unigram frequencies of words [5]. Similar approaches have been described in [12] and [14].

D. Reduction of Size of the Hidden Layer

Another way to reduce $H \times V$ is to choose a small value of H. For example, in [15] $H = 100$ is used when the amount of the training data is over 600M words. However, we will show that $H = 100$ is insufficient to obtain good performance when the amount of training data is large, as long as usual NN LM architecture is used.

In Section VII, we show what is to our knowledge the first technique that allows small hidden layers to be used for models that are trained on huge amounts of data, while having good final performance[1]: we train the computationally complex neural network model together with maximum entropy model.

E. Parallelization

Artificial neural networks are naturally simple to parallelize. It is possible to either divide the matrix times vector computation between several CPUs, or to process several examples at once, which allows going to matrix times matrix computation that can be optimized by existing libraries such as BLAS. In the context of NN LMs, Schwenk [16] has reported a speedup of several times from parallelization.

It might seem that recurrent networks are much harder to parallelize, as the state of the hidden layer depends on the previous state. However, one can parallelize just the computation between hidden and output layers. It is also possible to parallelize the whole network by training from multiple points in the training data simultaneously. However, parallelization is highly architecture-specific optimization problem, and we will deal in this paper only with algorithmic approaches for reducing computational complexity.

Another approach is to divide the training data into K subsets and train a single NN model on each of them. However, neural network models profit from clustering of similar events, thus such an approach would lead to suboptimal results. Also, in the test phase, we would end up with K models, instead of one.

IV. Experimental Setup

We performed recognition on the English Broadcast News (BN) NIST RT04 task using state-of-the-art acoustic models trained on the English Broadcast News (BN) corpus (430 hours of audio) provided to us by IBM [17]. IBM also provided us its state-of-the-art speech recognizer, Attila [3] and two Kneser-Ney smoothed backoff 4-gram LMs containing 4.7M N-grams and 54M N-grams, both trained on about 400M word tokens. We refer readers to [17] for more details of the recognizer and corpora used for training the models.

[1]By good performance, we mean that the resulting NN LM has better perplexity than normal n-gram model.

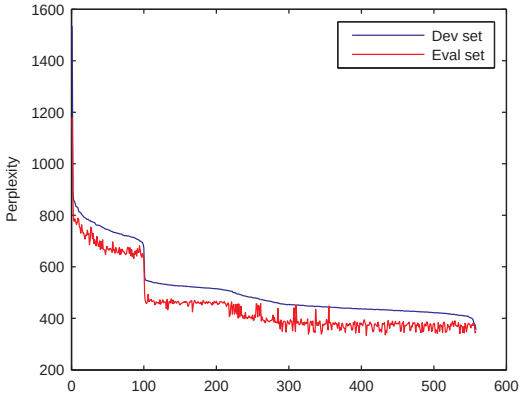

Fig. 2. Perplexity of sorted data chunks.

We followed IBM's multi-pass decoding recipe using the 4.7M n-gram LM in the first pass followed by rescoring using the larger LM. The development data consisted of DEV04f+RT03 data (25K tokens). For evaluation, we used the RT04 evaluation set (47K tokens). The size of the vocabulary is 84K words.

V. Automatic Data Selection and Sorting

Usually, stochastic gradient descent is used for training neural networks. This assumes randomization of the order of the training data before start of each epoch. In the context of NN LMs, the randomization is usually performed on the level of sentences. However, an alternative view can be taken when it comes to training deep neural network architectures, such as recurrent neural networks: we hope that the model will be able to find complex patterns in the data, that are based on simpler patterns. These simple patterns need to be learned before complex patterns can be learned. Such concept is usually called 'incremental learning'. In the context of simple RNN based language models, it has previously been investigated by Elman [18].

In the context of NN LMs, it was described and formalized in [15]. It is assumed that the training should start with simple patterns, so that the network can initialize its internal representation, and the additional knowledge should be added during training, in the form of gradually more complex patterns.

Inspired by these approaches, we have decided to change the order of the training data, so that the training starts with out-of-domain data, and ends with the most important in-domain data. Another motivation for this approach is even simpler: if the most useful data are processed at the end of the training, they will have higher weights, as the update of parameters is done on-line.

We divided the full training set into 560 equally-sized chunks (each containing 40K sentences). Next, we computed perplexity on the development data given a 2-gram model trained on each chunk. We sorted all chunks by their performance on the development set. We observed that although we use very standard LDC data, some chunks contain noisy data or repeating articles, resulting in high perplexity of models based on these parts of the training data. In Figure 2, we

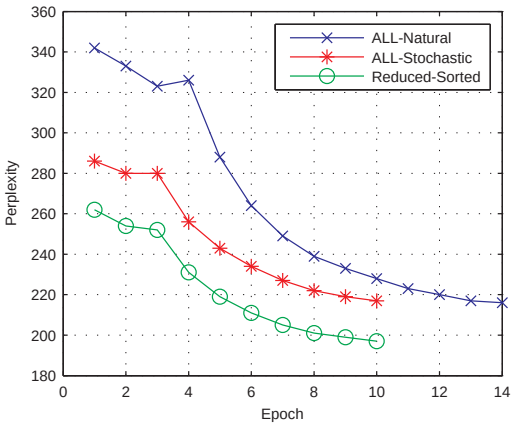

Fig. 3. Perplexity on the dev set during training RNN models with 80 neurons.

TABLE I
PERPLEXITY ON THE EVALUATION SET WITH DIFFERENTLY ORDERED TRAINING DATA, FOR RNN MODELS WITH VARIOUS SIZES OF THE HIDDEN LAYER.

Model	Hidden layer size			
	10	20	40	80
ALL-Natural	357	285	237	193
ALL-Stochastic	371	297	247	204
Reduced-Sorted	347	280	228	183

plot the performance on the development set, as well as performance on the evaluation set, to show that the correlation of performance on different, but similar test sets is very high.

We decided to discard the data chunks with perplexity above 600 to obtain the Reduced-Sorted training set, that is ordered as shown in Figure 2. This set contains about 318M tokens[2]. In Figure 3, we show three variants of training the RNN LM: training on all concatenated training corpora with the natural order, standard stochastic gradient descent using all data (randomized order of sentences), and training with the reduced and sorted set.

We can conclude that stochastic gradient descent helps to reduce the number of required training epochs before convergence is achieved, against training on all data with the natural order of sentences. However, sorting the data results in significantly lower final perplexity on the dev set - we observe around 10% reduction of perplexity. In Table I, we show that these improvements carry over to the evaluation set.

VI. EXPERIMENTS WITH LARGE RNN MODELS

The perplexity of the large 4-gram model with Kneser-Ney smoothing (denoted later as KN4) is 144 on the development set, and 140 on the evaluation set. We can see in Table I that RNN models with 80 neurons are still far away from this performance. In further experiments, we have used RNN models with increasing size of the hidden layer, trained on the Reduced-Sorted data set. We have used just 7 training epochs, as with sorted data, the convergence is achieved quickly (for the first three epochs, we used constant learning rate 0.1, and halved it at start of each new epoch). These results are summarized in Table II. We denote RNN model with 80 neurons as RNN-80 etc.

[2]Training a standard n-gram model on the reduced set results in about the same perplexity as with the n-gram model that is trained on all data.

TABLE II
PERPLEXITY OF MODELS WITH INCREASING SIZE OF THE HIDDEN LAYER.

Model	Dev PPL		Eval PPL	
		+KN4		+KN4
backoff 4-gram	144	144	140	140
RNN-10	394	140	347	140
RNN-20	311	137	280	137
RNN-40	247	133	228	134
RNN-80	197	126	183	127
RNN-160	163	119	160	122
RNN-240	148	114	149	118
RNN-320	138	110	138	113
RNN-480	122	103	125	107
RNN-640	114	99	116	102

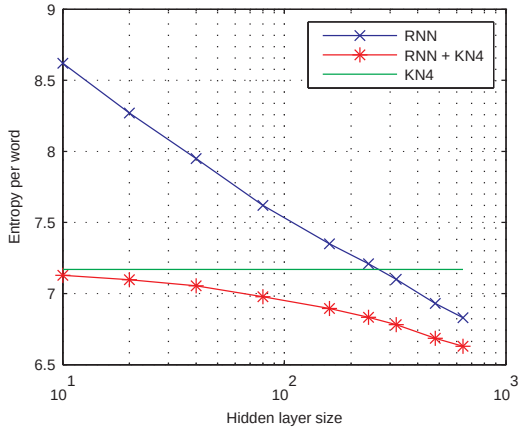

Fig. 4. Entropy per word on the dev set with increasing size of hidden layer.

In Figure 4, we show that the performance of RNN models is strongly correlated with the size of the hidden layer. We needed about 320 neurons for the RNN model to match the performance of the baseline backoff model. However, even small models are useful when linearly interpolated with the baseline KN4 model. It should be noted that training models with more than 500 neurons on this data set becomes computationally very complex: the computational complexity of RNN model as given by Eq. 4 depends on the recurrent part of the network with complexity $H \times H$, and the output part with complexity $H \times C + H \times W$, where C is the number of classes (in our case 400) and W is the number of words that belong to a specific class. Thus, the increase is quadratic with the size of the hidden layer for the first term, and linear for the second term.

We rescore the word lattices using the iterative decoding method previously described in [19], as it is much less computationally intensive than basic N-best list rescoring. As shown in Figure 5, RNN models perform very well for lattice rescoring; we can see that even the stand-alone RNN-80 model is better than the baseline 4-gram model. As the weights of individual models are tuned on the development set, we

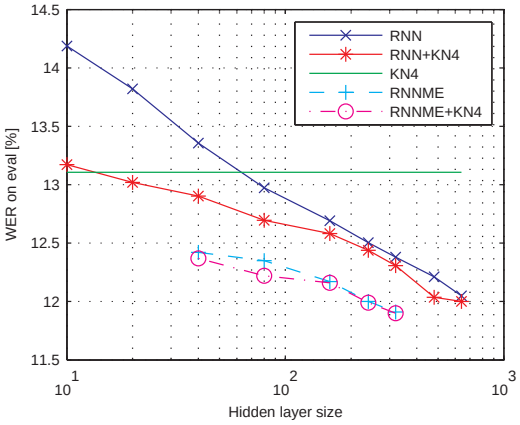

Fig. 5. WER on the eval set with increasing size of hidden layer.

have observed a small degradation for the RNN-10 model interpolated with baseline 4-gram model. On the other hand, the RNN-640 model provides quite an impressive reduction of WER, from 13.11% to 12.0%.

It is intersting to notice that the RNN-80 model outpeforms already the large backoff model in terms of WER, although its perplexity is much higher, as shown in Figure 4. This suggests that even limited RNN models are able to discriminate well between correct and ambiguous sentences.

By using three large RNN models and a backoff model combined together, we achieved the best result so far on this data set - 11.70% WER. The model combination was carried out using the technique described in [20]. The models in combination were: RNN-480, RNN-640 and RNN-640 model trained on 58M subset of the training data.

VII. Hash-based Implementation of Class-based Maximum Entropy Model

We have already mentioned in the introduction that maximum entropy models are very close to neural network models with no hidden layer. In fact, it has previously been shown that a neural network model with no hidden layer can learn a bigram language model [9], which is similar to what was shown for the maximum entropy models. However, in [9], the memory complexity for bigram model was V^2, where V is size of the vocabulary. For a trigram model and V around 100k, it would be infeasible to train such model.

The maximum entropy model can be seen in the context of neural network models as a weight matrix that directly connects the input and output layers. Direct connections between projection layer and the output layer were previously investigated in the context of NNLMs in [2], with no improvements reported on a small task.

In our work, we have added direct connections to the class-based RNN architecture that we have proposed earlier [5]. We use direct parameters that connect input and output layers, and input and class layers. We learn direct parameters as part of the whole network - the update of weights is performed online. We found that it is important to use regularization when learning the direct parameters on small data sets (we currently use L2 regularization). Using classes also helps to avoid overfitting, as the direct parameters that connect input and class layers already perform ad-hoc clustering.

TABLE III
Perplexity on the evaluation set with increasing size of hash array, for RNN model with 80 neurons.

Model	Hash size				
	0	10^6	10^7	10^8	10^9
RNNME-80	183	176	160	136	123
RNNME-80 + KN4	127	126	125	118	113

We have used bigram and trigram features for the direct part of the RNN model, and we denote this architecture as RNNME. As only two features are active in the input layer at any given time, the computational complexity of the model increases about the same as if two neurons were added to the hidden layer.

Representing the direct connections in memory is however a problem: on our setup with $V = 84$K, the full set of all possible trigram features would be impractically large. Thus, instead of having full weight matrix, we use hash function to map the huge sparse matrix into one-dimensional array. This exploits the fact that most of the weights are almost never used. This approach has an obvious advantage: we can easily control size of the model by choosing a single parameter, the size of the hash array. In Table III, we show how the hash size affects the overall performance of the model. In the following experiments, we have used a hash size of 10^9, as it gives reasonable performance (using a larger hash would lead only to marginal improvements and would increase memory demands considerably). If we train just the ME part of the model (RNNME-0) with hash size 10^9, we obtain perplexity 157 on the evaluation set.

The achieved perplexity of 123 on the evaluation set with RNNME-80 model is significantly better than the baseline perplexity 140 of the KN4 model. After interpolation of both models, the perplexity drops further to 113. This is significantly better than the interpolation of RNN-80 and KN4, which gives perplexity 127. Such result already proves the usefulness of training the RNN model with direct parameters.

Most importantly, we have observed good performance when we used the RNNME model for rescoring experiments. Reductions of word error rate on the RT04 evaluation set are summarized in Table IV. The model with direct parameters with 40 neurons in the hidden layer performs almost as well as model without direct parameters and with 320 neurons

VIII. Conclusion and Future Work

We have shown that neural network models, in our case with recurrent architecture, can provide significant improvements on state-of-the-art setup for Broadcast News speech recognition. The models we built are probably the largest neural network based language models ever trained. We have used about 400M tokens (318M in the reduced version). Size of the hidden layer of the largest model was 640 neurons, and the vocabulary size 84K words.

TABLE IV
WORD ERROR RATE ON THE RT04 EVALUATION SET AFTER LATTICE RESCORING WITH VARIOUS MODELS, WITH AND WITHOUT INTERPOLATION WITH THE BASELINE 4-GRAM MODEL.

Model	WER[%]	
	Single	Interpolated
KN4 (baseline)	13.11	13.11
RNN-40	13.36	12.90
RNN-80	12.98	12.70
RNN-160	12.69	12.58
RNN-320	12.38	12.31
RNN-480	12.21	12.04
RNN-640	12.05	**12.00**
RNNME-0	13.21	12.99
RNNME-40	12.42	12.37
RNNME-80	12.35	12.22
RNNME-160	12.17	12.16
RNNME-320	11.91	**11.90**
3xRNN	-	**11.70**

We have shown that by discarding parts of the training data and by sorting them, we can achieve about 10% reduction of perplexity, against classical stochastic gradient descent. This improvement would be probably even larger for tasks where only some training corpora can be considered as in-domain.

The relative word error rate reduction was almost 11%, over a large 4-gram model with Kneser-Ney smoothing - absolutely, we reduced WER from 13.11% to 11.70%. We are aware of slightly better baseline for this setup - in [17], 13.0% was reported as baseline, and 12.3% after rescoring with "model M" [21] (while in our experiments, rescoring with model M resulted in 12.5% WER). We suspect that if we used wider lattices, we would be able to observe further improvements in our experiments.

We have shown that training RNN model with direct connections can lead to good performance both on perplexity and word error rate, even if very small hidden layers are used. The model RNNME-40 with only 40 neurons has achieved almost as good performance as RNN-320 model that uses 320 neurons. We have shown that direct connections in NN model can be seen as a maximum entropy model, and we have also verified that it is important to train the RNN and ME models jointly. Roughly speaking, we can reduce training times from weeks to days by using the novel RNN architecture. We have performed additional experiments on a Wall Street Journal setup that we have used previously [6], and we were able to reduce WER from 17.2% to 14.9% with RNNME-100 model that was trained in a single day, on 36M training tokens.

In the future, we plan to extend our results even further. It would be interesting to see if the RNN model would benefit from training together with a more refined maximum entropy model, for example with model M.

The presented techniques can also be easily applied to more traditional feedforward NN LMs. We extended our publicly available toolkit[3] to support training of RNNME models.

[3]Available at `http://www.fit.vutbr.cz/~imikolov/rnnlm/`

ACKNOWLEDGMENTS

We are grateful to Bhuvana Ramabhadran and Brian Kingsbury for sharing with us state-of-the-art IBM speech recognition system (Attila) and relevant statistical models. We thank Karel Veselý and Stefan Kombrink for discussions about parallel training of neural networks. This work was partly supported by Technology Agency of the Czech Republic grant No. TA01011328, Czech Ministry of Education project No. MSM0021630528, Grant Agency of Czech Republic project No. 102/08/0707, and by Czech Ministry of Trade and Commerce project No. FR-TI1/034. Anoop Deoras was partly funded by the HLT-COE, Johns Hopkins University.

REFERENCES

[1] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modeling," *Computer, Speech and Language*, vol. 10, pp. 187–228, 1996.

[2] Y. Bengio, R. Ducharme, P. Vincent *et al.*, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[3] H. Soltau, G. Saon, and B. Kingsbury, "The IBM Attila speech recognition toolkit," in *Proc. IEEE Workshop on Spoken Language Technology*, 2010.

[4] G. Zweig, P. Nguyen *et al.*, "Speech recognition with segmental conditional random fields: A summary of the JHU CLSP summer workshop," in *Proceedings of ICASSP*, 2011.

[5] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Proceedings of ICASSP*, 2011.

[6] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Černocký, "Empirical evaluation and combination of advanced language modeling techniques," in *Proceedings of Interspeech*, 2011.

[7] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of Interspeech*, 2010.

[8] T. Alumae and M. Kurimo, "Efficient estimation of maximum entropy language models with N-gram features: an SRILM extension," in *Proceedings of Interspeech*, 2010.

[9] W. Xu and A. Rudnicky, "Can artificial neural networks learn language models?" in *International Conference on Statistical Language Processing*, 2000.

[10] H. Schwenk and J.-L. Gauvain, "Training neural network language models on very large corpora," in *Proceedings of EMNLP*, 2005.

[11] P. Xu, A. Gunawardana, and S. Khudanpur, "Efficient subsampling for training complex language models," in *Proceedings of EMNLP*, 2011.

[12] H.-S. Le, I. Oparin, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Structured output layer neural network language model," in *Proceedings of ICASSP*, 2011.

[13] J. Goodman, "Classes for fast maximum entropy training," in *Proceedings of ICASSP*, 2001.

[14] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *AISTATS*, 2005, pp. 246–252.

[15] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of ICML*, 2009.

[16] H. Schwenk, "Continuous space language models," *Comput. Speech Lang.*, vol. 21, pp. 492–518, July 2007.

[17] S. F. Chen, L. Mangu, B. Ramabhadran, R. Sarikaya, and A. Sethy, "Scaling shrinkage-based language models," in *Proceedings of ASRU*, 2009.

[18] J. L. Elman, "Learning and development in neural networks: The importance of starting small," *Cognition*, vol. 48, pp. 71–99, 1993.

[19] A. Deoras, T. Mikolov, and K. Church, "A fast re-scoring strategy to capture long-distance dependencies," in *Proceedings of EMNLP*, 2011.

[20] A. Deoras, D. Filimonov, M. Harper, and F. Jelinek, "Model combination for speech recognition using Empirical Bayes risk minimization," in *Proc. of IEEE Workshop on Spoken Language Technology (SLT)*, 2010.

[21] S. F. Chen, "Shrinking exponential language models," in *Proc. NAACL HLT*, 2009.