Efficient GPU-based Training of Recurrent Neural Network Language Models Using Spliced Sentence Bunch

X. Chen, Y. Wang, X. Liu, M.J.F. Gales and *P. C. Woodland*

University of Cambridge Engineering Dept,
Trumpington St., Cambridge, CB2 1PZ, U.K.
Email: {*xc257,yw293,xl207,mjfg,pcw*}*@eng.cam.ac.uk*

Abstract

Recurrent neural network language models (RNNLMs) are becoming increasingly popular for a range of applications including speech recognition. However, an important issue that limits the quantity of data, and hence their possible application areas, is the computational cost in training. A standard approach to handle this problem is to use class-based outputs, allowing systems to be trained on CPUs. This paper describes an alternative approach that allows RNNLMs to be efficiently trained on GPUs. This enables larger quantities of data to be used, and networks with an unclustered, full output layer to be trained. To improve efficiency on GPUs, multiple sentences are "spliced" together for each mini-batch or "bunch" in training. On a large vocabulary conversational telephone speech recognition task, the training time was reduced by a factor of 27 over the standard CPU-based RNNLM toolkit. The use of an unclustered, full output layer also improves perplexity and recognition performance over class-based RNNLMs.

Index Terms: language models, recurrent neural network, speech recognition, GPU

1. Introduction

Statistical language models are crucial components in automatic speech recognition (ASR) systems. In order to handle the data sparsity problem associated with conventional back-off n-gram language models (LMs), language modelling techniques that represent history contexts in a continuous vector space, such as neural network language models (NNLMs) [1, 2, 3, 4, 5, 6], can be used. Depending on the underlying network architecture being used, they can be categorised into two major categories: feedforward NNLMs [1, 2, 3, 6], which use a vector representation of preceding contexts of a finite number of words, and recurrent NNLMs (RNNLMs) [4, 7], which use a recurrent vector representation of longer and potentially variable length histories. In recent years RNNLMs have been shown to give significant improvements over back-off n-gram LMs and feedforward NNLMs, thus becoming an increasingly popular choice for state-of-the-art ASR systems [4, 7, 8, 9, 5, 10, 11, 12].

One important practical issue associated with RNNLMs is the computational cost incurred in model training. As a major part of the computation is required at the output layer, most of existing techniques use an RNNLM architecture with a class based factorized output layer [7], known as class based RNNLMs (C-RNNLM). In common with a similar architecture based feedforward NNLMs [13], as the number of classes is normally significantly smaller than the full output layer size, training time speed-ups can be achieved [7]. Combined with further parallelized model training [14] and multi-stage classing at the output layer [15], training time speed-ups up to 10 fold were reported in previous research for C-RNNLMs. However, there are several issues associated with these approaches. First, the use of class base output layer limits the potential speed up from bunch[1] mode parallelization [11]. Second, the underlying word to class assignment scheme at the output layer can also affect the resulting C-RNNLM's performance [7, 18]. Finally, CPU based speed up implementations were studied in previous research [7, 14, 15, 11]. Hence, it is preferrable to also exploit the parallelization power of GPUs.

To address these issues, a modified RNNLM architecture with a non-class based, full output layer structure (F-RNNLM) is used in this paper. This F-RNNLM architecture not only removes the performance sensitivity to word classing, but also facilitates a novel spliced sentence bunch mode parallelization of F-RNNLM training. This efficient parallelization algorithm is implemented on GPUs to fully exploit their parallel computing power. Experimental results on a large vocabulary conversational telephone speech transcription task show that a 27 fold training speed-up was obtained over the standard CPU based C-RNNLM training using the RNNLM toolkit [19]. Consistent improvements in both recognition accuracy and perplexity were also obtained.

The rest of this paper is organized as follows. In Section 2 recurrent neural network LMs are reviewed and the two RNNLM architectures are presented. A novel spliced sentence bunch mode parallelization algorithm for F-RNNLM training is proposed in Section 3. Details of a GPU implementation of this algorithm in Section 4. In Section 5 the proposed RNNLM training parallelization method is evaluated on a state-of-the-art conversational telephone speech transcription task. Section 6 draws the conclusions and discusses possible future work.

2. Recurrent Neural Network LMs

In contrast to feedforward NNLMs, recurrent NNLMs [4] represent the full, non-truncated history $h_1^{i-1} =< w_{i-1}, \ldots, w_1 >$ for word w_i using the 1-of-k encoding of the most recent preceding word w_{i-1} and a continuous vector v_{i-2} for the remaining context. For an empty history, this is initialized,

Xie Chen is supported by Toshiba Research Europe Ltd, Cambridge Research Lab. The research leading to these results was also supported by EPSRC grant EP/I031022/1 (Natural Speech Technology) and DARPA under the Broad Operational Language Translation (BOLT) and RATS programs. The paper does not necessarily reflect the position or the policy of US Government and no official endorsement should be inferred.

[1] It is also sometimes referred as minibatch in literature [16, 17]. For clarity, the term "bunch" is used throughout this paper.

for example, to a vector of all ones. The topology of the recurrent neural network used to compute LM probabilities $P_{\text{RNN}}(w_i|w_{i-1}, v_{i-2})$ consists of three layers. The full history vector, obtained by concatenating w_{i-1} and v_{i-2}, is fed into the input layer. The hidden layer compresses the information of these two inputs and computes a new representation v_{i-1} using a sigmoid activation to achieve non-linearity. This is then passed to the output layer to produce normalized RNNLM probabilities using a softmax activation, as well as recursively fed back into the input layer as the "future" remaining history to compute the LM probability for the following word $P_{\text{RNN}}(w_{i+1}|w_i, v_{i-1})$. As RNNLMs use a vector representation of full histories, they are mostly used for N-best list rescoring. For more efficient lattice rescoring using RNNLMs, appropriate approximation schemes, for example, based on clustering among complete histories [20] can be used.

2.1. Full output layer based RNNLMs (F-RNNLMs)

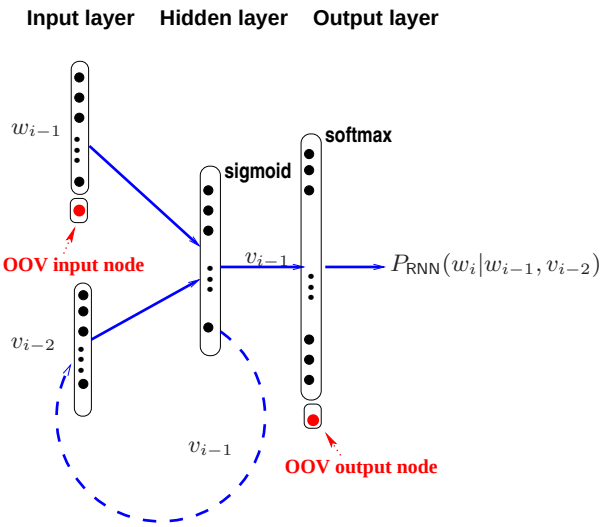

Figure 1: *A full output layer RNNLM with OOS nodes.*

A traditional RNNLM architecture with an unclustered, full output layer (F-RNNLM) is shown in Figure 1. RNNLMs can be trained using an extended form of the standard back propagation algorithm, back propagation through time (BPTT) [21], where the error is propagated through recurrent connections back in time for a specific number of time steps, for example, 4 or 5 [7]. This allows the recurrent network to record information for several time steps in the hidden layer. To reduce the computational cost, a shortlist [2, 22] based output layer vocabulary limited to the most frequent words can also be used for class based RNNLMs. A similar approach may also be used at the input layer when a large vocabulary is used. To reduce the bias to in-shortlist words during NNLM training and improve robustness, an additional node is added at the output layer to model the probability mass of out-of-shortlist (OOS) words [3, 6, 20].

2.2. Class Based RNNLMs (C-RNNLMs)

Training F-RNNLMs is computationally expensive. As a major part of the cost is incurred at the output layer, existing techniques have been centered around class based RNNLMs (C-RNNLMs), an RNNLM architecture with a class based factorized output layer [7]. An example C-RNNLM is illustrated in Figure 2. Each word in the output layer vocabulary is attributed to a unique class based on frequency counts. The LM probability assigned to a word is factorized into two individual terms

$$P_{\text{RNN}}(w_i|w_{i-1}, v_{i-2}) = P(w_i|c_i, v_{i-1})P(c_i|v_{i-1}). \quad (1)$$

The calculation of word probability is based on a small subset of words from the same class, and the number of classes is normally significantly smaller than the full output layer size. Hence, training time speed-ups can be achieved. A special case of C-RNNLM using a single class is equavalent to a traditional, full output layer based F-RNNLM introduced in Section 2.1. A modified version of the RNNLM toolkit [19] supporting the above architecture is used.

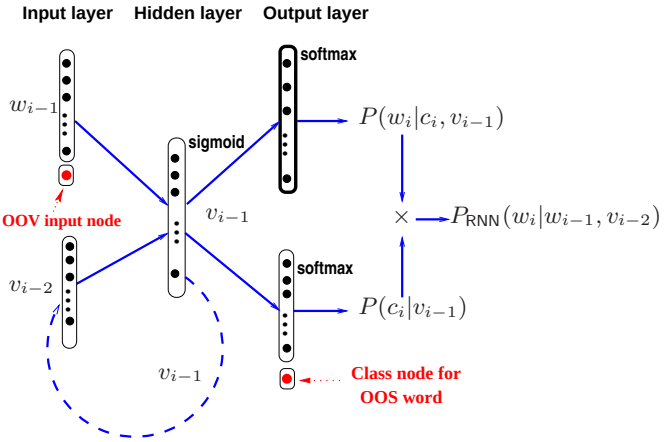

Figure 2: *A class based RNNLM with OOS nodes.*

In state-of-the-art ASR systems, NNLMs are often linearly interpolated with n-gram LMs to obtain both a good context coverage and strong generalisation [2, 22, 3, 4, 11, 6]. The interpolated LM probability is given by

$$P(w_i|h_1^{i-1}) = \lambda P_{\text{NG}}(w_i|h_1^{i-1}) + (1-\lambda) P_{\text{RNN}}(w_i|h_1^{i-1}) \quad (2)$$

λ is the weight assigned to the n-gram LM distribution $P_{\text{NG}}(\cdot)$, and kept fixed as 0.5 in all experiments of this paper for all RNNLMs. In the above interpolation, the probability mass of OOS words assigned by the RNNLM component needs to be re-distributed among all OOS words [3, 6].

3. Efficient Recurrent Neural Network LM Training Using Spliced Sentence Bunch

In order to reduce the computational cost in model training, a bunch mode parallelization can be applied to RNNLMs. This technique was previously proposed for feedforward NNLMs [2, 23]. A fixed number of n-grams from the training data is propagated through the network without updating its weights to independently generate intermediate gradient statistics. These were then accumulated and used for updating the weight parameters. As RNNLMs use a vector representation of full history contexts, a necessary modification of the data structure used by this algorithm is required. Instead of operating at the n-gram level, a sentence level bunch should be used [24, 11]. By definition, this form of parallelization requires each sentence to be regarded as independent in RNNLM training by re-initializing the hidden history vector at the start of every sentence.

The basic idea of bunch mode training is shown in Figure 3. For a N size bunch, a total of N sentences are aligned from

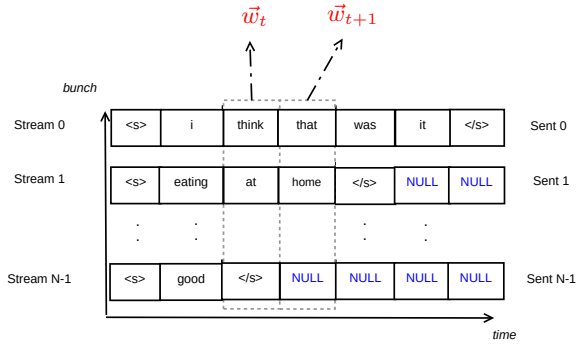

Figure 3: *Bunched RNNLM training without sentence splicing*

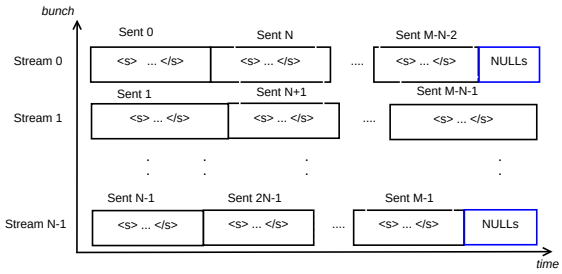

Figure 4: *Bunched RNNLM training with sentence splicing*

left to right. During parallelization, a regular structured input matrix is formed. The element at the ith row and t^{th} column in the input matrix, associated with time $t+1$ and an output word $w_{t+1}^{(i)}$, represents a vector $[w_t^{(i)}, v_{t-1}^{(i)}]^\top$, where $w_t^{(i)}$ and $v_{t-1}^{(i)}$ are the 1-of-k vector encoding of the t^{th} word of the i^{th} sentence in the bunch, and the corresponding recurrent history vector at word $w_t^{(i)}$ respectively.

Two issues arise when directly using the above sentence bunch mode training. First, the variation of sentence length in the training data requires setting the number of columns of the input matrix to the maximum sentence length in the training corpus. NULL words are then inserted at the end of other shorter sentences in the bunch, as is shown in Figure 3. These redundant NULL words are ignored during BPTT. As the ratio between the maximum and average sentence length of the training data increases, and more NULL tokens are inserted, the potential speed up from parallelization is increasingly limited. Second, the standard sentence bunch mode training also interacts with the use of class based RNNLMs [7, 11]. As words aligned at the same position across different sentences can belong to different classes, the associated output layer submatrices of irregular sizes will be used at the same time instance. This can also result in inefficiency during training.

In order to handle these issues, an efficient bunch mode parallelization based on spliced sentences is used. Instead of a single sentence, each stream in the bunch now contains a sequence of concatenated sentences, as is illustrated in Figure 5. Sentences in the training corpus are joined into streams that are more comparable in length. Individual sentence boundaries within each stream are marked in order to appropriately reset the recurrent history vector as required. As the streams are more comparable in length, the insertion of NULL tokens at the stream end is minimized. This approach can thus significantly reduce the synchronization overhead and improve the efficiency in parallelization. The non-class based, full output layer RNNLMs (F-RNNLMs) introduced in Section 2.1 are also used to address the second issue as mentioned above. F-RNNLMs use the entire output layer both in training and LM probability calculation, therefore allow the speed improvements from parallelization techniques to be fully exploited.

4. GPU Implementation

To improve efficiency, graphics processing units (GPUs), which have been previously employed to train neural network based acoustic models in speech recognition [16, 17], are used to train RNNLMs used in this paper. CUBLAS from CUDA 5.0, the basic linear algebra subprograms (BLAS) library optimized for Nvidia GPUs, is used for fast matrix operation. As discussed in Sections 1 and 2.2, when a large number of output layer nodes are used, the softmax computation during gradient calculation is very expensive. To handle this problem, a fast GPU implementation of the softmax function is used. Instead of summing the sufficient statistics sequentially over all output layer nodes, they are processed in one block. Shared memory is also used and to facilitate rapid address access time. An array in each block with a fixed length (1024 used in this work) is allocated in the shared memory. Partial accumulates are stored in the array elements. A binary tree structured summation performed over the array reduces the execution time from N to $\log N$, for example, from 1024 down to 10 parallelized GPU cycles.

Table 1: Initial learning rate for different bunch size

bunch size	1	8	32	64	128	256
learning rate	0.1	0.3	0.8	1.0	2.0	2.0

In order to obtain a fast and stable convergence during RNNLM training, the appropriate setting and scheduling of the learning rate parameter is necessary. For the F-RNNLMs that are trained in bunch mode, the initial learning rate is empirically adjusted in proportion to the underlying bunch size. When the bunch size is set to 1 and no form of parallelization is used, the initial learning rate is set to 0.1, in common with the default setting used in the RNNLM toolkit [19]. The initial learning rate settings used for various other bunch sizes are also shown in Table 1. When the bunch size is increased to 128, the initial learning rate is set to 2.0.

5. Experiments

In this section, RNNLMs are evaluated on the CU-HTK LVCSR system for conversational telephone speech (CTS) used in the 2004 DARPA EARS evaluation. The acoustic models were trained on approximately 2000 hours of Fisher conversational speech released by the LDC. A 59k recognition word list was used in decoding. The system uses a multi-pass recognition framework. A detailed description of the baseline system can be found in [25]. The 3 hour **dev04** data, which includes 72 Fisher conversations, was used as a test set. The baseline 4-gram LM was trained using a total of 545 million words from 2 text sources: the LDC Fisher acoustic transcriptions, **Fisher**, of 20 million words (weight 0.75), and the University Washington conversational web data [26], **UWWeb**, of 525 million words (weight 0.25). The **Fisher** data was used to train various RNNLMs. A 38k word input layer vocabulary and 20k word output layer shortlist were used. RNNLMs were interpolated with the baseline 4-gram LM using a fixed weight 0.5. This

baseline LM gave a WER of 16.7% on **dev04** measured using lattice rescoring.

The baseline class based RNNLMs were trained on CPU with the modified RNNLM toolkit [19] compiled with g++ [2]. The number of BPTT steps was set as 5. A computer with dual Intel Xeon E5-2670 2.6GHz processors with a total of 16 physical cores was used for CPU-based training. The number of classes was fixed as 200. The number of hidden layer nodes was varied from 100 to 800. The 100-best hypotheses extracted from the baseline 4-gram LM lattices were rescored for performance evaluation. The perplexity and error rates of various RNLLMs are shown in Table 2. The C-RNNLM with 512 hidden layer nodes gave the lowest WER of 15.3% and serves as the baseline C-RNNLM in the experiments.

Table 2: Speed, PPL and WER results on class based RNNLMs trained on CPU with different hidden nodes (class size is 200)

hidden nodes	speed (w/s)	train time (hours)*	valid	
			PPL	WER
100	7.6k	9.8	50.7	16.1
200	2.1k	35.6	48.6	15.8
512	0.37k	202.1	46.5	**15.3**
800	0.11k	679.9	45.8	15.4

The Nvidia GeForce GTX TITAN GPU was used to train various F-RNNLMs. The spliced sentence bunch mode parallelization algorithm and its GPU implementation described in Sections 3 and 4 were used. A range of different bunch size settings from 8 to 256 were used. Consistent with the above C-RNNLM baseline, all F-RNNLMs have 512 hidden layer nodes and a comparable number of weight parameters. Their performance measured in terms of training speed, perplexity and WER are shown in the 2nd Section of Table 3. The performance of the baseline C-RNNLM with 512 hidden nodes (previously shown in 3rd line in Table 2) is again shown in the 1st line of Table 3. Setting the bunch size to 8, a 4 times speed up is achieved. Improvements in perplexity and WER over the C-RNNLM baseline are also obtained. Further improvements in training speed can be consistently achieved by increasing the bunch size to 128 without performance degradation. The best performance in terms of speed and WER was obtained by using a bunch size of 128. This gives 27 times speed up and a 0.1% absolute reduction in WER over the C-RNNLM baseline.

Examining the breakdown of the training time suggests the output and hidden layers account for the majority of computation during BPTT (44.8% and 39.4% respectively), due to the heavy matrix multiplications required. The cost of the softmax function at the output layer is comparatively minimal at 2.4%. The remaining 13.4% is shared by other operations such as resetting F-RNNLM hidden vectors at the sentence start, and data transfer between CPU and GPU. A further speed up is possible, for example, by increasing the bunch size to 256. However, the convergence becomes less unstable and leads to performance degradation.

As a major contribution factor to the above speed improvements, the importance of using sentence splicing in bunch mode based GPU implementation is shown in Figure 5, where a contrast in speed with and without sentence splicing is drawn.

[2] A speedup of 1.7 times for CPU based training could be obtained by the Intel MKL CBLAS implementation with multi-threading (compiled with icc version 14.0.2) over the baseline RNNLM toolkit for C-RNNLMs with 512 hidden layer nodes and 200 classes.

Table 3: Speed, PPL and WER results on RNNLMs trained on GPU with different bunch sizes (hidden layer size is 512)

model type	bunch size	#parameter	speed (w/s)	time (hours)	PPL	WER
C-RNN	-	26.9M	0.37k	202.1	46.5	15.3
F-RNN	8	26.8M	1.4k	53.4	45.7	15.2
	32		4.6k	16.3	45.6	15.2
	64		7.6k	9.8	45.7	15.2
	128		9.9k	7.5	46.3	15.2
	256		12.9k	5.7	46.5	15.4

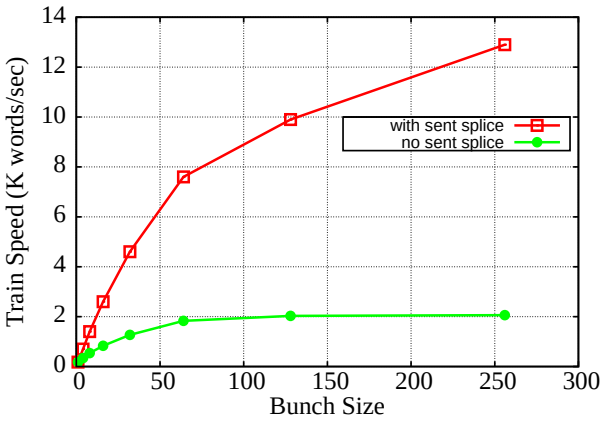

Figure 5: *Bunched RNNLM training with sentence splicing*

When using the standard bunch model training with no sentence splicing, only limited speed improvements are obtained by increasing the bunch size. This is due to the large number of inserted NULL tokens and the resulting inefficiency, as discussed in Section 3.

The spliced sentence bunch based parallelization can also be used for RNNLM performance evaluation on GPU. Table 4 shows the speed information measured for N-best rescoring using the baseline C-RNNLM and the F-RNNLM of Tables 3. As expected, it is very expensive to use F-RNNLM on CPU. C-RNNLMs can improve the speed by 43 times. A further speed up of 9 times over the CPU C-RNNLM baseline was obtained using the bunch mode (bunch size 512) parallelized F-RNNLM.

Table 4: Comparison of speed for N-Best scoring

model type	device	bunch	speed (words/sec)
F-RNN	CPU	N/A	0.14k
C-RNN			5.9k
F-RNN	GPU	1	1.1k
		64	41.3k
		512	56.3k

6. Conclusions

In this paper an efficient GPU implementation of bunch mode RNNLM training is investigated. On a large vocabulary conversational telephone speech recognition task, the proposed method yields a 27 times reduction in the training time compared to the standard CPU-based RNNLM toolkit [19]. Improvements in both perplexity and recognition performance were also obtained over the standard class based RNNLMs using the full output RNNLM architecture.

7. References

[1] Y. Bengiom, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," Journal of Machine Learning Research, vol. 3, pp. 1137–1155, 2003.

[2] Holger Schwenk, "Continuous space language models," *Computer Speech & Language*, vol. 21, no. 3, pp. 492–518, 2007.

[3] Junho Park, Xunying Liu, Mark J. F. Gales, and P. C. Woodland, "Improved neural network based language modelling and adaptation," in *Proc. ISCA Interspeech*, 2010, pp. 1041–1044.

[4] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur, "Recurrent neural network based language model.," in *Proc. ISCA Interspeech*, 2010, pp. 1045–1048.

[5] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney, "LSTM neural networks for language modeling," in *Proc. ISCA Interspeech*, 2012.

[6] Hai-Son Le, Ilya Oparin, Alexandre Allauzen, J Gauvain, and François Yvon, "Structured output layer neural network language models for speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 21, no. 1, pp. 197–206, 2013.

[7] Tomas Mikolov, Stefan Kombrink, Lukas Burget, J.H. Cernocky, and Sanjeev Khudanpur, "Extensions of recurrent neural network language model," in *Proc. ICASSP*. IEEE, 2011, pp. 5528–5531.

[8] Anoop Deoras, Tomas Mikolov, Stefan Kombrink, Martin Karafiát, and Sanjeev Khudanpur, "Variational approximation of long-span language models for LVCSR," in *Proc. ICASSP*. IEEE, 2011, pp. 5532–5535.

[9] Gwénolé Lecorvé and Petr Motlicek, "Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition," Tech. Rep., Idiap, 2012.

[10] Anoop Deoras, Tomas Mikolov, Stefan Kombrink, and Kenneth Church, "Approximate inference: A sampling based modeling technique to capture complex dependencies in a language model," *Speech Communication*, vol. 55, no. 1, pp. 162–177, 2013.

[11] Martin Sundermeyer, Ilya Oparin, Jean-Luc Gauvain, Ben Freiberg, Ralf Schluter, and Hermann Ney, "Comparison of feedforward and recurrent neural network language models," in *Proc. ICASSP*, Vancouver, Canada, May 2013, pp. 8430–8434.

[12] Yujing Si, Ta Li, Jielin Pan, and Yonghong Yan, "Prefix tree based n-best list re-scoring for recurrent neural netowrk language model used in speech recognition system.," in *Proc. ISCA Interspeech*, 2013.

[13] Frederic Morin and Yoshua Bengio, "Hierarchical probabilistic neural network language model," in *Proceedings of the international workshop on artificial intelligence and statistics*, 2005, pp. 246–252.

[14] Yangyang Shi, Mei-Yuh Hwang, Kaisheng Yao, and Martha Larson, "Speed up of recurrent neural network language models with sentence independent subsampling stochastic gradient descent," in *Proc. ISCA Interspeech*, 2013.

[15] Zhiheng Huang, Geoffrey Zweig, Michael Levit, Benoit Dumoulin, Barlas Oguz, and Shawn Chang, "Accelerating recurrent neural network training via two stage classes and parallelization," in *ASRU, IEEE Workshop on*. IEEE, 2013.

[16] Xie Chen, Adam Eversole, Gang Li, Dong Yu, and Frank Seide, "Pipelined back-propagation for context-dependent deep neural networks.," in *Proc. ISCA Interspeech*, 2012.

[17] Frank Seide, Gang Li, and Dong Yu, "Conversational speech transcription using context-dependent deep neural networks.," in *Proc. ISCA Interspeech*, 2011, pp. 437–440.

[18] Geoffrey Zweig and Konstantin Makarychev, "Speed regularization and optimality in word classing," in *Proc. ICASSP*. IEEE, 2013, pp. 8237–8241.

[19] Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukas Burget, and Jan Cernocky, "Recurrent neural network language modeling toolkit," in *ASRU, IEEE Workshop on*. IEEE, 2011.

[20] Xunying Liu, Yongqiang Wang, Xie Chen, Mark Gales, and P. C. Woodland, "Efficent lattice rescoring using recurrent neural network language models," in *Proc. ICASSP*. IEEE, 2014.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning representations by back-propagating errors*, MIT Press, Cambridge, MA, USA, 1988.

[22] Ahmad Emami and Lidia Mangu, "Empirical study of neural network language models for arabic speech recognition," in *ASRU, IEEE Workshop on*. IEEE, 2007, pp. 147–152.

[23] Holger Schwenk, "Efficient training of large neural networks for language modelling," in *in Proceedings of IEEE International Joint Conference on Neural Networds*. IEEE, 2002, pp. 3059–3064.

[24] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu, "Temporal kernel neural network language modeling," in *Proc. ICASSP*. IEEE, 2013.

[25] G. Evermann, H. Y. Chan, M. J. F. Gales, B. Jia, D. Mrva, P. C. Woodland, and K. Yu, "Training LVCSR systems on thousands of hours of data," in *Proc. of ICASSP*, 2005, vol. 1, pp. 209–212.

[26] Ivan Bulyko, Mari Ostendorf, and Andreas Stolcke, "Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures," in *In Proc. HLT*. ACL, 2003, pp. 7–9.