IMPROVING THE TRAINING AND EVALUATION EFFICIENCY OF RECURRENT NEURAL NETWORK LANGUAGE MODELS

X. Chen, X. Liu, M.J.F. Gales & P. C. Woodland

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

ABSTRACT

Recurrent neural network language models (RNNLMs) are becoming increasingly popular for speech recognition. Previously, we have shown that RNNLMs with a full (non-classed) output layer (F-RNNLMs) can be trained efficiently using a GPU giving a large reduction in training time over conventional class-based models (C-RNNLMs) on a standard CPU. However, since test-time RNNLM evaluation is often performed entirely on a CPU, standard F-RNNLMs are inefficient since the entire output layer needs to be calculated for normalisation. In this paper, it is demonstrated that C-RNNLMs can be efficiently trained on a GPU, using our spliced sentence bunch technique which allows good CPU test-time performance (42x speedup over F-RNNLM). Furthermore, the performance of different classing approaches is investigated. We also examine the use of variance regularisation of the softmax denominator for F-RNNLMs and show that it allows F-RNNLMs to be efficiently used in test (56x speedup on CPU). Finally the use of two GPUs for F-RNNLM training using pipelining is described and shown to give a reduction in training time over a single GPU by a factor of 1.6.

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

1. INTRODUCTION

Recurrent neural network language models (RNNLMs) have shown promising performance improvements in many applications, such as speech recognition [1, 2, 3, 4, 5], spoken language understanding [6, 7, 8], and machine translation [9, 10] .

One key practical issue is slow training speed of standard RNNLMs on standard CPUs. Previously we showed that using the "spliced sentence bunch" technique, which processes many sentences in parallel and performs mini-batch parameter updates, RNNLMs with a full output layer (F-RNNLMs) could be trained efficiently on a GPU [11], resulting in a 27$\times$ speed-up over a CPU with a class-based factorised output layer. However, F-RNNLMs are very time-consuming to evaluate (e.g. for lattice-rescoring) on CPUs, and hence techniques that allow fast GPU-based training and efficient CPU-based evaluation are of great practical value.

In this paper we extend our previous work on GPU-based RNNLMs training with spliced sentence bunch [11] and present two methods to improve CPU-based evaluation efficiency. First a simple modification is introduced to allow class based RNNLMs to be trained on GPUs with the same method. Furthermore, different word clustering algorithms are investigated and compared. The second method allows the RNNLM to be used without softmax normalisation during testing, by training with an extra variance regularisation term in the training objective function. This approach was applied on feedforward NNLMs and class based RNNLM in previous work [12, 10, 13]. It can also be applied to full output layer RNNLMs. Finally, to further improve training speed, pipelined training using multiple GPUs is explored.

The rest of this paper is structured as follows. Section 2, reviews RNNLMs. Efficient training of class based RNNLMs is described in Section 3, and variance regularisation in Section 4. Pipelined training of RNNLMs is described in Section 5. Experimental results on a conversational telephone speech transcription task are given in Section 6 and conclusions presented in Section 7.

2. RECURRENT NEURAL NETWORK LMS

In contrast to feedforward NNLMs, recurrent NNLMs [1] 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 previous word w_{i-1} and a continuous vector v_{i-2} for the remaining context. For an empty history, this is initialised, 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 normalised 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 [14] can be used.

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

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) [15], where the error is propagated through recurrent connections back in time for a specific

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.

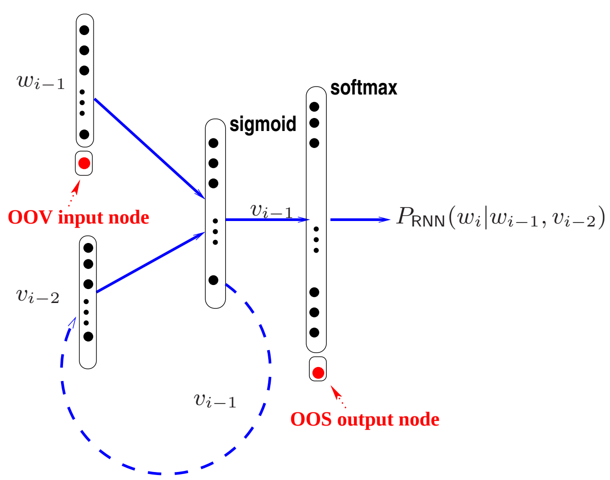

Fig. 1. *A full output layer RNNLM with OOS nodes.*

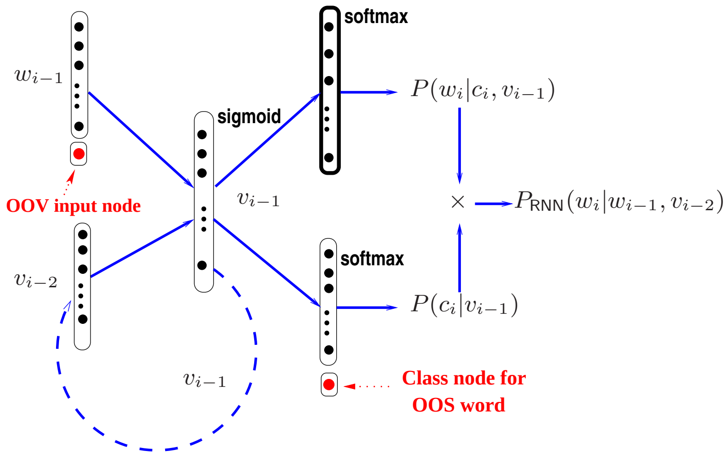

Fig. 2. *A class based RNNLM with OOS nodes.*

number of time steps, for example, 4 or 5 [2]. This allows the recurrent network to record information for several time steps in the hidden layer. To reduce the computational cost, a shortlist [16, 17] based output layer vocabulary limited to the most frequent words can be 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 [18, 19, 14].

2.2. Class Based RNNLMs (C-RNNLMs)

Although F-RNNLMs could be trained and evaluated efficiently using GPUs in [11], it is computationally expensive on CPUs due to the normalisation on output layer. Class based RNNLMs (C-RNNLMs) provides an alternative choice to speedup training and evaluation on CPUs, which adopts a modified RNNLM architecture with a class based factorised output layer [2]. An illustration of C-RNNLM is given in Figure 2. Each word in the output layer is assigned to a unique class. The LM probability assigned to a word is factorised 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, computation is able to be reduced. It is worth noting that a special case of C-RNNLMs using a single class is equivalent to a traditional, full output layer based F-RNNLM introduced in Section 2.1.

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 [16, 17, 18, 1, 5, 19]. 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 [18, 19].

3. CLASS BASED RNNLMS TRAINING WITH SPLICED SENTENCE BUNCH

Spliced sentence bunch training operates on many sentences in parallel and performs a mini-batch update. For F-RNNLMs this uses the same output layer matrix irrespective of the next word to be predicted and can be performed very efficiently on a GPU due to the large number of computational units. A very efficient implementation of C-RNNLMs training with bunch mode is not easy since the data samples in one bunch may belong to different classes, which requires a different sub-matrix to be used and greatly complicates implementation. However, here the aim is to train a C-RNNLM for efficient CPU-evaluation, rather than to provide a speed-up over GPU-based F-RNNLM training. During training, only the outputs for each parallel stream that belong to the subset of words that belongs to the target class for that stream are kept from the forward pass, and the outputs for other words are set to zero. By applying this modification C-RNNLMs can be trained on a GPU with bunch mode with a similar cost to F-RNNLMs.

It has been shown that the training accuracy and speed are sensitive to word clustering for RNNLM training. In [2], frequency based class was adopted to speedup training. However, it degraded perplexity on the Penn Tree Bank corpus [2, 20]. Word clustering using Brown's classing method [21] was investigated in [20, 22, 23] and improved perplexity were reported compared to frequency based classes. As well as frequency-based and Brown-like word clustering[1], word clustering derived from a vector-based word representation is also explored. Each word can be represented by a vector in a low-dimensional space [25] obtained from the matrix associated input word and hidden nodes. The similarity of words could be measured by the distance of vectors in the continuous space. For F-RNNLMs, the weight matrix between the hidden nodes could also be used to represent words[2]. A k-means approach is used to cluster words into a specific number of classes in this work and the input and output matrices are obtained from a well-trained F-RNNLM.

[1]We adopted the Brown-like classing method from [24], which is slightly different to the original version in [21].

[2]Most previous work on vector word representation used an hierarchical output layer.

4. F-RNNLM WITH VARIANCE REGULARISATION

Another type of solution to speedup evaluation for NNLMs has been proposed both in [12] (variance regularisation) and [10] (self-norm). The variance of the softmax log normalisation is added into the objective function for optimisation. If the normalisation term can be regarded as constant at test time, a large speedup can be achieved by only computing the outputs needed a and avoiding the calculation of the time-consuming softmax function. The use of variance regularisation was also explored for RNNLMs training in [13], where C-RNNLMs were used and trained sample by sample. In this work, we investigate the use of variance regularisation on F-RNNLMs and train using GPU-based sentence-splice bunch mode. The objective function to be minimised is

$$O^{vr} = O^{ce} + \frac{1}{T}\sum_{i=1}^{N}\sum_{j=1}^{M}(\frac{\gamma}{2}(log(Z_j^{(i)}) - (Log\bar{Z}_i))^2) \quad (3)$$

where O^{ce} is the cross-entropy based loss function,

$$O^{ce} = -\frac{1}{T}\sum_{i=1}^{N}(\sum_{j=1}^{M}(log(P(w_j^{(i)}|h_j^{(i)})) \quad (4)$$

T is the number of training samples and N is the number of bunches in the training corpus and M is the bunch size. Here $Z_j^{(i)}$ is the normalisation term for word w_j in ith bunch, $Log\bar{Z}_i = \frac{1}{M}\sum_{j=1}^{M} log(Z_j^{(i)})$ is the mean of log normalisation (Log-Norm) term in the ith bunch. It is worth mentioning that in C-RNNLMs training with variance regularisation in [13], the mean of log normalisation is set to zero, which works well for C-RNNLMs. However, it doesn't work well for F-RNNLMs training where the number of classes equals one. Hence, It is important to calculate the mean and variance of Log-Norm for every bunch.

In a well-trained F-RNNLM, the mean of the normalisation term on a validation set, denoted $\bar{Z}$, is calculated. It is used to compute the probability of predicted words at test time as

$$P(w_j|h_j) = \tilde{P}(w_j|h_j)/\bar{Z} \quad (5)$$

where $\tilde{P}(w_j|h_j)$ is the unnormalised probability that can be used in evaluation time. This significantly reduces the computation at the output layer as the normalisation is no longer required.

5. PIPELINED TRAINING OF RNNLMS

The parallel structure of neural network training can be classified into two categories: model parallelism and data parallelism [26]. The difference lies in whether the model or data is split across multiple machines or cores. Pipelined training is a type of model parallelism. It was proposed to speedup the training of deep neural network for acoustic models in [27]. Here, we extend it to the training of RNNLMs. Layers of the network are distributed across different GPUs, and operations on these layers (e.g. forward-pass, BPTT) are executed on their own GPUs. It allows each GPU to proceed independently and simultaneously, and communication between layers happens after a parameter update step.

The data flow of pipelined training is shown in Figure 3. Two weight matrices (W0 and W1) are kept in two GPUs (denoted as GPU 0 and GPU 1). For the first bunch in each epoch, the input is forwarded to the hidden layer and the output of hidden layer is copied from GPU 0 to GPU 1. For the 2nd bunch, the input is forwarded again. Simultaneously, GPU 1 forwards the previous bunch obtained from hidden layer to output layer, followed by error back propagation and parameter update. The communication (i.e. copy operation) between GPUs happens afterwards. For the following bunches, GPU 0 updates model parameters using corresponding error signal and input with BPTT, then forwards the new input data for the next bunch. GPU 1 performs successively a forward pass, error back propagation and update . Although there is one bunch update delay for the update of W0, pipelined training can guarantee that the update direction is correct and deterministic for every update.

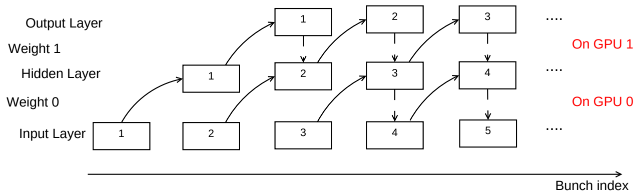

Fig. 3. *Data flow in pipelined training using two GPUs*

6. EXPERIMENTS

In the main part of 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 [28]. 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 [29], **UWWeb**, of 525 million words (weight 0.25). This baseline LM gave a perplexity of 51.8 and word error rate (WER) of 16.7% on **dev04** measured using lattice rescoring. The **Fisher** data was used to train RNNLMs. A 32k word input layer vocabulary and 20k word output layer shortlist were used. All RNNLMs are trained in a sentence independent mode. The size of hidden layer is set as 512, the BPTT step as 5 and the bunch size set to 128. For the C-RNNLMs, the number of class is 200. The NVidia GTX Titan GPU is used in RNNLM training. The CPU used in this paper is the dual Intel Xeon E5-2670 2.6GHz processors with a total of 16 physical cores. All RNNLMs are interpolated with the baseline 4-gram LM using a fixed weight 0.5. The 100-best hypotheses extracted from the baseline 4-gram LM lattices were then rescored for performance evaluation. A detailed description of the baseline RNNLM can be found in [11].

6.1. Experiments on C-RNNLMs training

The performance of the bunch mode trained C-RNNLMs described in section 3 are evaluated first. The performance of the three types of word clustering schemes presented in section 3 based on frequency, Brown classing or K-Means based classing are compared in an initial experiment conducted on the Penn Tree Bank (PTB) corpus. In common with previous research reported in [2, 30, 20, 22], section 0-20 were used as the training data (about 930K words), while sections 21-22 kept as the validation data (74K) and section 23-24 as the test data (82K). The size of vocabulary is 10K. RNNLMs modelling cross-sentence dependency were trained using various word clustering methods with 200 hidden layer nodes, 100 classes and a

BPPT step of 5. In practice, the GPU-based bunch mode training speed of C-RNNLMs was found close to that of F-RNNLMs. Their respective perplexities (PPLs) are then evaluated. As shown in Table 1, the performance of C-RNNLMs were found to be sensitive to the underlying word clustering scheme being used at the output layer. The C-RNNLM trained with Brown classing gave the lowest perplexity of 127.4 among all C-RNNLMs, though slightly higher than the F-RNNLM. Frequency based C-RNNLMs gave the highest PPL score of 135.3.

Table 1. PPL using different word clustering on Penn Corpus

Word clustering type	PPL
Frequency	135.3
Brown	127.4
K-means on input matrix	132.2
K-means on output matrix	130.6
none	126.1

Table 2 shows a comparable set of PPL and WER results obtained on the CTS task described above. As is shown in the table, the K-Means based clustering on the output layer matrix parameters gave the best performance in terms of both PPL and WER, though is slightly outperformed by the F-RNNLM in terms of WER. The other three word clustering methods gave comparable error rates. This indicates that using a larger amount of training data, the performance of C-RNNLMs become less sensitive to the word clustering algorithm being used.

Table 2. PPL and WER results using different word clustering

Word clustering	CTS	
	PPL	WER
Frequency	47.4	15.36
Brown	46.3	15.36
K-means on Input matrix	47.1	15.40
K-means on Output matrix	46.2	15.28
none	46.3	15.22

6.2. Experiments on F-RNNLMs with variance regularisation

In this section, the performance of F-RNNLMs trained with variance regularisation are evaluated. These experimental results are shown in table 3. In practice the training of F-RNNLMs with variance regularisation normally requires one more epoch than CE based training for good convergence. The error rates marked as "WER" in the table are the WER scores measured using normalised RNNLM probabilities, while "WER*" in the last column are the WERs obtained using a more efficient, and unnormalised RNNLM probability given in equation (5). The first row of the table gives results without variance regularisation by setting γ to 0. As expected, the WER increases from 15.22 to 16.24 without normalisation. This confirms that the normalisation term computation for the softmax function is crucial for using cross entropy (CE) trained RNNLMs in decoding. When the variance regularisation term is applied in RNNLM training, the difference between the "WER" and "WER*" metrics are quite small. As expected, when the setting of γ is the increased, the variance of the log normalisation term is decreasing. When γ is set as 0.4, it gives a WER of 15.28 comparable to that of the baseline CE trained RNNLM, and a much faster speed in evaluation time.

Table 4 shows the CPU based evaluation speed of a CE-trained C-RNNLM, F-RNNLM and a CE-trained F-RNNLM using variance regularisation. As is shown in the table, the C-RNNLM gave a speed up of 42 times over the CE trained F-RNNLM baseline. Using variance regularisation during F-RNNLM training, a 56 time acceleration in evaluation speed was obtained compared to the baseline CE-based F-RNNLM.

Table 3. PPL and WER results with variance regularisation

γ	log(norm)		PPL	WER	WER* [1]
	mean	var			
0.0	15.4	1.67	46.3	15.22	16.24
0.1	14.2	0.12	46.5	15.21	15.34
0.2	13.9	0.08	46.6	15.33	15.35
0.3	14.0	0.06	46.5	15.40	15.30
0.4	14.2	0.05	46.6	15.29	15.28
0.5	14.4	0.04	46.5	15.40	15.42

[1] WER* denotes WER using unnormalised RNNLM probability from Eqn (5).

Table 4. Evaluation speed of RNNLMs on CPUs

RNNLMs	Train Crit	Speed (w/s)
F-RNNLM	CE	0.14k
C-RNNLM		5.9k
F-RNNLM	+VR	7.9k

6.3. Experiments on dual GPU pipelined training of F-RNNLMs

In this section, the performance of a dual GPU based pipelined F-RNNLM training algorithm is evaluated. In the previous experiments, a single NVidia GeForce GTX TITAN GPU (designed for a workstation) was used. For multi-GPU work, two slightly slower NVidia Tesla K20m GPUs housed in the same server were used. Table 5 gives the training speed, PPL scores and WER results of the pipelined training algorithm. According to these results, pipelined training gave a speed up of 1.6 times and performance comparable to a single GPU based training.

Table 5. Train Speed, PPL and WER results for pipelined training of F-RNNLMs

Model Type	GPU	Train Speed (w/s)	PPL	WER
C-RNN	-	0.37k	46.5	15.31
F-RNN	1xTITAN	9.9k	46.3	15.22
	1xK20m	6.9k		
	2xK20m	11.0k	46.3	15.23

7. CONCLUSION

Following our previous research on efficient parallelised training of full output layer RNNLMs [11], several approaches are investigated in this paper to further improve their efficiency during training and evaluation time: class based RNNLMs were efficiently trained on GPU in a modified spliced sentence bunch mode and gave a 42 time speed up in evaluation time; the variance normalised form of RNNLM training scheme produced a 56 time speed up in test time; a pipelined RNNLM training algorithm using two GPUs also gave an additional 1.6 time acceleration of training speed.

8. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding.," in *INTERSPEECH*, 2013, pp. 3771–3775.

[7] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu, "Recurrent neural networks for language understanding.," in *INTERSPEECH*, 2013, pp. 2524–2528.

[8] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding.," .

[9] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig, "Joint language and translation modeling with recurrent neural networks.," in *EMNLP*, 2013, pp. 1044–1054.

[10] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul, "Fast and robust neural network joint models for statistical machine translation," in *Association for Computational Linguistics*, 2014.

[11] Xie Chen, Yongqiang Wang, Xunying Liu, Mark Gales, and P. C. Woodland, "Efficient training of recurrent neural network language models using spliced sentence bunch," in *Proc. ISCA Interspeech*, 2014.

[12] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu, "Efficient one-pass decoding with nnlm for speech recognition," *Signal Processing Letterss on*, vol. 21, no. 4, pp. 377–381, 2014.

[13] Yongzhe Shi, Wei-Qiang Zhang, Meng Cai, and Jia Liu, "Variance regularization of rnnlm for speech recognition.," in *Proc. of ICASSP*, 2014.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

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

[21] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[22] Diamanino Caeiro and Andrej Ljolje, "Multiple parallel hidden layers and other improvements to recurrent neural network language modeling," in *Proc. ICASSP*. IEEE, 2013, pp. 8426–8429.

[23] Hong-Kwang Kuo, Ebru Arisoy, Ahmad Emami, and Paul Vozila, "Large scale hierarchical neural network language models.," in *INTERSPEECH*, 2012.

[24] Reinhard Kneser and Hermann Ney, "Improved clustering techniques for class-based statistical language modelling," in *Proc. EUROSPEECH*, p. 1993.

[25] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of NAACL-HLT*, 2013, pp. 746–751.

[26] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu, "On parallelizability of stochastic gradient descent for speech dnns," in *Proc. ICASSP*, 2014.

[27] 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.

[28] 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.

[29] 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.

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