CUED-RNNLM – AN OPEN-SOURCE TOOLKIT FOR EFFICIENT TRAINING AND EVALUATION OF RECURRENT NEURAL NETWORK LANGUAGE MODELS

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

Cambridge University Engineering Dept, Trumpington St., Cambridge, CB2 1PZ, U.K.
Email: {xc257,xl207,yq236,mjfg,pcw}@eng.cam.ac.uk

ABSTRACT

In recent years, recurrent neural network language models (RNNLMs) have become increasingly popular for a range of applications including speech recognition. However, the training of RNNLMs is computationally expensive, which limits the quantity of data, and size of network, that can be used. In order to fully exploit the power of RNNLMs, efficient training implementations are required. This paper introduces an open-source toolkit, the `CUED-RNNLM` toolkit, which supports efficient GPU-based training of RNNLMs. RNNLM training with a large number of word level output targets is supported, in contrast to existing tools which used class-based output-targets. Support fot N-best and lattice-based rescoring of both HTK and Kaldi format lattices is included. An example of building and evaluating RNNLMs with this toolkit is presented for a Kaldi based speech recognition system using the AMI corpus. All necessary resources including the source code, documentation and recipe are available online[1].

Index Terms: language model, recurrent neural network, speech recognition, GPU, open-source toolkit

1. INTRODUCTION

Language models are crucial components in many speech and language processing applications, such as speech recognition and machine translation. Due to the good performance and efficient implementation of n-gram LMs, they have been the dominant language modelling approach for several decades. However, there are two well know issues associated with n-gram LMs. The first issue is data sparsity. Sophisticated smoothing techniques are required for robust parameter estimation [1, 2]. The second issue lies in the n^{th} order Markov assumption. The predicted word probability is only dependent on the preceding $n-1$ words, while longer range context dependency is ignored.

Recurrent neural network language models (RNNLMs) project each word into a compact continuous vector space which uses a relatively small set of parameters, and uses recurrent connections to model long range context dependencies. Hence, RNNLMs provide a solution for the two key n-gram issues. Furthermore, RNNLMs have been shown to produce significant improvements over n-gram LMs for speech recognition tasks, and this has resulted in their use for a wide range of applications [3, 4, 5, 6, 7].

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). Supporting data for this paper is available at the https://www.repository.cam.ac.uk/handle/1810/253374 data repository.

[1] http://mi.eng.cam.ac.uk/projects/cued-rnnlm/

However, training RNNLMs is computationally very demanding and the resulting slow training speed limits the use of RNNLMs when processing large amounts of data. In our previous work, RNNLMs were efficiently trained on GPUs using a novel sentence splicing bunch mode parallelisation algorithm. A significant speedup of 27 times compared to Mikolov's RNNLM toolkit [8] running on a CPU was obtained [9]. In order to reduce the performance sensitivity of word to class assignment [10] and improve the efficiency of bunch mode training, RNNLMs with a full output layer (FRNNLM) were employed [9], in contrast to the widely used class based RNNLM (CRNNLM) [3].

The softmax normalisation at the output layer heavily impacts the evaluation speed of FRNNLMs, especially when modelling large vocabularies, that are perhaps number hundreds of thousands of words. In order to solve this problem, two improved RNNLM training criteria have been proposed: variance regularisation[11]; and noise contrastive estimation[12]. Both of these methods allow a constant, history independent normalisation term to be used, and therefore considerably increase the RNNLM evaluation speed on a CPU.

This paper presents the **CUED-RNNLM toolkit**, which includes the above efficient implementation of GPU based RNNLM training and improved training criteria. This rest of this paper is organised as follows. Section 2 gives a brief review of RNNLMs. Section 3 discusses three existing open-source toolkits for training neural network language models. The efficient training, evaluation and implementation of the `CUED-RNNLM` toolkit are presented in Sections 4, 5 and 6 respectively. Experiments on a Kaldi speech recognition system constructed on the AMI corpus are presented in Section 7. Section 8 draws conclusions and discusses future work.

2. RECURRENT NEURAL NETWORK LMS

RNNLMs [13] represent the full, non-truncated history $h_i = < w_{i-1}, \ldots, w_1 >$ for word w_i using a 1-of-k encoding of the previous word w_{i-1} and a continuous vector v_{i-2} for the remaining context. An out-of-vocabulary (OOV) input node can also be used to represent any input word not in the chosen recognition vocabulary. The topology of the recurrent neural network used to compute LM probabilities $P_{\mathsf{RNN}}(w_i|w_{i-1}, v_{i-2})$ 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 from 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 $P_{\mathsf{RNN}}(w_{i+1}|w_i, v_{i-1})$.

An example RNNLM architecture with an unclustered, full out-

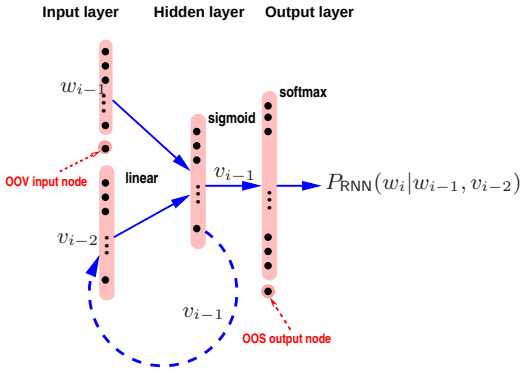

Fig. 1. *A full output layer RNNLM with OOV and OOS nodes.*

put layer is shown in Fig. 1. RNNLMs can be trained using back propagation through time (BPTT) [14], where the error is propagated through the recurrent connections back for a specific number of time steps, for example, 4 or 5 [3]. To reduce the computational cost, a shortlist [15, 16] can be used to limit the output layer to the most frequent words. To reduce the bias to in-shortlist words during RNNLM training and improve robustness, an additional node is added at the output layer to model the probability mass of out-of-shortlist (OOS) words [17, 18, 19].

Training of full output layer RNNLMs is computationally expensive. One popular solution is to use class based RNNLMs [3]. Individual words in the output layer vocabulary are assigned to classes. As the number of classes and number of words within each classes are both significantly smaller than the output layer vocabulary, class based RNNLMs can be efficiently trained on CPUs. However, the use of class based RNNLM not only introduces performance sensitivity to word classing, it also difficult to parallel the training of irregular sized class specific weight matrices for further acceleration. In our previous work, full output layer RNNLMs were adopted and trained efficiently on GPUs in bunch (or minibatch) mode [11, 12, 9] that processes multiple training sentences in parallel. The spliced sentence bunch technique was used to minimise the synchronisation overhead between bunch streams introduced by sentence length variation. The idea of spliced sentence bunch is illustrated in Figure 2. Multiple sentences are spliced into one of the N streams (N is the bunch size). During training, an input word vector of dimension N is formed by taking one word from each stream. The target word vector is formed by taking the next word in each stream.

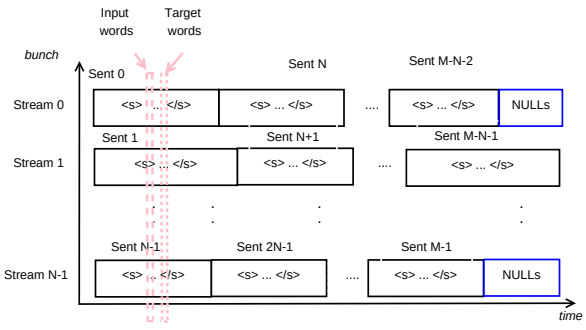

Fig. 2. *RNNLM training with spliced sentence bunch*

In state-of-the-art ASR systems, RNNLMs are often linearly interpolated with n-gram LMs to obtain both good context coverage and strong generalisation [13, 15, 17, 18]. The interpolation weight λ could be optimised via EM algorithm on a held-out set. For simplicity, λ is kept fixed at 0.5 in this paper. In the above interpolation, the probability mass of OOS words assigned by the RNNLM component is re-distributed with equal probability over all OOS words to guarantee a valid probability.

3. TOOLKITS FOR NNLM/RNNLM TRAINING

There are various toolkits which implement recurrent neural networks, such as Theano[2], Torch[3], TensorFlow[4], Chainer[5], and CNTK [20]. For language modelling, there are also several open-source toolkits specifically for neural network based language model training. Three popular toolkits are presented in this section.

CSLM [21] is an efficient implementation of feedforward neural network language models. It includes CPU and GPU versions that allow large quantities of training data to be processed using efficient parallelisation algorithms and resampling techniques.

RNNLM [8] is probably the first RNNLM toolkit designed specifically for language modelling. It allows class based RNNLMs to be efficiently trained on CPUs using small amounts of data. The training speed rapidly reduces as the size of the hidden layer increases [9]. The `RNNLM` toolkit also provided recipes for various functions including perplexity evaluation, N-best rescoring and text generation. The `CUED-RNNLM` toolkit was initially based on the `RNNLM` toolkit and provides similar debugging information and model format.

RWTHLM [22] provides an implementation of long short-term memory (LSTM) based RNNLMs and class based output layers are also used. The toolkit features a more efficient implementation of CPU based RNNLM training. However, similar issues exist in reduced training speed with the increase of hidden layer size.

In this paper, we describe our GPU-based implementation for efficient RNNLM training of full output layer RNNLMs: `CUED-RNNLM`. The `CUED-RNNLM` toolkit is implemented in C++, it is freely available under the BSD license and copyright from the `RNNLM` toolkit.

4. TRAINING IN CUED-RNNLM

`CUED-RNNLM` includes FRNNLM training, evaluation and text generation via sampling. Both training and sampling require the use of a GPU, while model evaluation is performed on a CPU.

4.1. Cross Entropy (CE) Training

The conventional objective function used in RNNLM training is based on cross entropy (CE),

$$J^{\text{CE}}(\theta) = -\frac{1}{N_w}\sum_{i=1}^{N_w} \ln P_{\text{RNN}}(w_i|h_i) \quad (1)$$

where N_w is the number of words in training corpus. RNNLMs with full output layer are trained on a GPU efficiently using bunch (i.e. minibatch) mode[9]. However, the softmax layer in output layer

[2]https://github.com/gwtaylor/theano-rnn
[3]https://github.com/tomsercu/lstm
[4]https://github.com/tensorflow/tensorflow/tree/master/tensorflow/models/rnn/ptb
[5]https://github.com/pfnet/chainer/tree/master/examples/ptb

requires the computation of normalisation term, as shown in Eqn (2), where a_i is the weight vector associated with word w_i. The computation of normalisation term Z_i is very expensive during both training and test time.

$$P_{\mathsf{RNN}}(w_i|h_i) = \frac{e^{v_{i-1}^T a_i}}{\sum_j e^{v_{i-1}^T a_j}} = \frac{e^{v_{i-1}^T a_i}}{Z_i} \quad (2)$$

One solution to the above problem is to learn a constant, history independent softmax normalisation term during RNNLM training. If the normalisation term Z_i could be approximated as constant D, un-nomalised RNNLM probabilities in Eqn (3) could be used to provide a large speed up at test time.

$$P_{\mathsf{RNN}}(w_i|h_i) \approx \frac{e^{v_{i-1}^T a_i}}{D} \quad (3)$$

Using this idea, two improved training criteria have been implemented in `CUED-RNNLM`: variance regularisation (VR) and noise contrastive estimation (NCE).

4.2. Variance Regularisation (VR)

Variance regularisation explicitly adds the variance of the normalisation term into the standard CE objective function [11, 23]. The associated objective function is given by

$$J^{\mathsf{VR}}(\theta) \;=\; J^{\mathsf{CE}}(\theta) + \frac{\gamma}{2}\frac{1}{N_w}\sum_{i=1}^{N_w}(\ln(Z_i) - (\ln Z))^2 \quad (4)$$

where $\ln Z$ is the mean of log normalisation term. The second term added to the CE objective function given in Eqn. (1) models the variance of the log normalisation term. The parameter γ is used to tune the effect of the variance term against the standard CE criterion. At test time, the RNNLM output probabilities can be approximated by the un-nomalised probabilities in Eqn (3).

4.3. Noise Contrastive Estimation (NCE)

In NCE training, each word in the training corpus is assumed to be generated by two different distributions [24]. One is data distribution, which is the RNNLM, and the other is noise distribution, where unigram is normally used. The objective function is to discriminate these two distributions over the training data and a group of randomly generated noise samples. This is given by

$$\begin{aligned} J^{\mathsf{NCE}}(\theta) \quad = \quad & -\frac{1}{N_w}\sum_{i=1}^{N_w}\Big(\ln P(C_{w_i}^{\mathsf{RNN}} = 1|w_i, h_i) \\ & + \sum_{j=1}^{k}\ln P(C_{\check{w}_{i,j}}^{n} = 1|\check{w}_{i,j}, h_i)\Big) \end{aligned} \quad (5)$$

where w_i is the ith target word, $\check{w}_{i,j}$ is the jth noise word generated for word w_i, and k is the number of noise samples. $P(C_{w_i}^{\mathsf{RNN}} = 1|w_i, h_i)$ is the posterior probability of word w_i is generated by the RNNLM, and $P(C_{\check{w}_{i,j}}^{n} = 1|\check{w}_{i,j}, h_i)$ the posterior probability of word $\check{w}_{i,j}$ is generated by a noise distribution.

During NCE training, the variance of the normalisation term Z_i can be implicitly constrained to be constant. The training procedure only relates to the target word and k samples in output layer, instead of the whole output layer. Hence, the output layer computational cost is no longer sensitive to vocabulary size and is reduced significantly. In common with variance regularisation, the un-nomalised probabilities in Eqn. (3) can be used at test time. Implementation details can be found in [12].

5. MODEL EVALUATION IN CUED-RNNLM

The test set perplexity (PPL) and word error rate (WER) are used to evaluate the performance of language models for speech recognition tasks. The `CUED-RNNLM` toolkit provides functions for computing perplexity, N-best rescoring and lattice rescoring.

5.1. Perplexity Calculation

Perplexity can be calculated using RNNLMs alone, or linearly interpolated with n-gram LMs. Calculating the exact perplexity for full output RNNLMs using normalised probabilities is computationally inefficient using a CPU: adding GPU perplexity calculations is future work.

5.2. N-best Rescoring

N-best lists can be generated from word lattices using e.g. the SRILM toolkit [28]. Then sentence level log likelihoods with the language model score computed from an RNNLM can be calculated and the N-best lists reranked. Note that the un-nomalised probability in Eqn 3 can be applied when using VR or NCE trained RNNLMs.

5.3. Lattice Rescoring

Lattice rescoring using RNNLMs is also supported. Models trained using CUED-RNNLMs can be applied using an extension added to the HTK 3.5 [25] lattice processing tool HLRescore. The lattice rescoring extension is also available as a patch to HTK 3.4.1. Via a conversion tool, Kaldi [26] format lattices are also supported.

6. OTHER FEATURES

An RNNLM can be used to generate a large quantities of texts by sampling [27, 29]. An n-gram LM can be then trained on the generated text which is interpolated with a baseline LM. The resulting LM can be applied directly for first-pass decoding and/or lattice rescoring as an approximation to the original RNNLM.

There are several other features in `CUED-RNNLM`. RNNLMs with more than one hidden layer are supported. Currently, only the first hidden layer is allowed to have recurrent connections. Additional values can be appended to the input layer, such as topic features [30]. Both sentence independent and dependent mode training of RNNLMs are implemented. In sentence independent mode RNNLM training, the history vector v_{i-2} in input layer is reset to an initial value (e.g. 0.1) at the beginning of each sentence. For sentence dependent RNNLM training, the sentence boundary is processed as a normal word without resetting the history. Sentence independent RNNLM training is used by default.

In many applications, the training data is from several sources. The ordering of training data presented in RNNLM training can significantly impact performance [31]. For good performance on in-domain test data, it is advisable is to present the out-of-domain data to the network first during RNNLM training, before the more important in domain training data is processed. For this reason, the training data is not randomly shuffled during training. It is therefore recommended that the sentence order is randomised for each source of data as a pre-processing step, while keeping the order of data sources.

The toolkit requires no other third-party libraries except the standard NVIDIA CUDA library for GPU based computation. The debugging information and output is similar to those used by the `RNNLM` toolkit [8]. A detailed description of the command options can be found online in the toolkit documentation.

7. EXPERIMENTS

Experiments are conducted on the AMI meeting corpus [32] to evaluated the performance of different types of RNNLM in a speech recognition context. In total 78 hours of speech was used in acoustic model training consisting of about 1M word of acoustic transcription (including sentence start and end). In addition eight meetings were kept from the training set and used as the development and test sets.

A Kaldi acoustic model training recipe featuring sequence training [33] was applied for deep neural network (DNN) training. A FMLLR transformed MFCC feature was used as input and 4000 clustered states (senones) were used as clustered as target. The DNN was trained with 6 hidden layers, each layer with 2048 hidden nodes.

The first part of the Fisher corpus of 13M words was also used to further improve language modelling performance. A 49k word decoding vocabulary was used. A 33k RNNLM input vocabulary was constructed from the intersection between the decoding vocabulary and all words present in the LM training data. The 22k most frequent words were then selected as output vocabulary. The BPTT algorithm is used in RNNLM training with the error back-propagated for 5 previous words. All RNNLMs in this paper use one hidden layer. On average 10 epochs of training are required to reach convergence. All LMs presented in this section were trained on the combined (AMI+Fisher) 14M word training set. Experimental results using only the 1M word AMI transcriptions can be found in the documentation online.

The first experiment is to evaluate the performance of RNNLMs. A pruned 3-gram LM was used in the first-pass decoding and followed by lattice rescoring using an un-pruned 4-gram LM. During RNNLM training, the AMI corpus (1M) was presented after the Fisher data (13M). RNNLMs with 512 hidden nodes were trained using the cross entropy criterion. Table 1 shows the performance of RNNLMs trained by both the `RNNLM` and `CUED-RNNLM` toolkits. RNNLMs give significant perplexity and word error rate (WER) improvements over the baseline 4-gram LM. The full output layer RNNLM trained by `CUED-RNNLM` toolkit slightly outperformed the class based model trained with `RNNLM`. Rescoring lattices and 50-best lists gave comparable 1-best (Viterbi) performance. An additional WER reduction of 0.2% absolute was obtained by confusion network (CN) [34] decoding using RNNLM rescored lattices, while CN decoding using the rescored 50-best lists gave no improvement.

LM Type	Re score	PPL		WER	
		dev	eval	dev	eval
3g	-	84.5	79.6	24.2	24.7
4g	lattice	80.3	76.3	23.7	24.1
+CRNN	lattice	70.5	67.5	22.4	22.5
	50 best			22.4	22.6
+FRNN	lattice	69.8	67.0	22.0	22.3
	50 best			22.2	22.5

Table 1. Performance of CRNNLMs (trained with `RNNLM`) and FRNNLMs (trained with `CUED-RNNLM`).

The next experiment investigates the performance of FRNNLMs trained using various criteria when using 50-best rescoring. The Fisher and AMI corpora were shuffled separately before being concatenated into single training data file. Shuffling gave a small reduction of WER [6]. The performance of VR and NCE trained RNNLMs are shown in Table 2. RNNLMs trained using CE, VR and NCE respectively were found to give comparable performance. In order to obtain stable convergence, the NCE based training required two more epochs than the CE baseline.

[6]No improvements obtained on CRNNLMs using data shuffling.

Train Crit	PPL		WER	
	dev	eval	dev	eval
CE	67.5	63.9	22.1	22.4
VR	68.0	64.4	22.1	22.4
NCE	68.5	65.1	22.1	22.4

Table 2. FRNNLMs trained with various criteria

Table 3 presents the training and evaluation speed of RNNLMs. A single process on a computer with dual Intel Xeon E5-2680 2.5GHz processors was used for CPU-based CRNNLM training and evaluation. The NVIDIA GeForce GTX TITAN GPU was used for training FRNNLMs. As expected, FRNNLM training on GPU is much faster than CRNNLM training on CPU and NCE training provided a further speedup. FRNNLMs trained using the VR and NCE criteria were also found to be more than 2.5 times faster than CRNNLMs at test time.

RNN Type	Train Crit	Train(GPU) Speed(kw/s)	Test (CPU) Speed(kw/s)
CRNN	CE	0.45	6.0
FRNN	CE	11.5	0.32
	VR	11.5	15.3
	NCE	20.5	15.3

Table 3. Training and testing speed of RNNLMs

The training speed heavily depends on the hidden layer size. Table 4 compares the training speed using a varying number of hidden nodes with `RNNLM` and `CUED-RNNLM`. It can be seen that CRNNLMs are efficient when a small sized hidden layer is used. However, the training speed decreases rapidly as the hidden layer size increases. When the hidden layer size is increased from 128 to 2048 nodes, the number of words processed per second is decreased by a factor of 340 to 12 words for CRNNLM. In contrast, the training speed of FRNNLMs were found less sensitive to such increase in hidden layer size. This shows the superior scaling to network size of `CUED-RNNLM`.

Toolkit	# Hidden node				
	128	256	512	1024	2048
RNNLM	4.1	1.7	0.45	0.095	0.012
CUED-RNNLM	19.8	14.2	11.5	6.6	3.7

Table 4. Train Speed (kw/s) against number of hidden nodes

8. CONCLUSION AND FUTURE WORK

We have introduced the `CUED-RNNLM` toolkit which provides an efficient GPU-based implementation for training RNNLMs. RNNLMs with full output layers are trained using a variance regularisation or noise contrastive estimation approach on a GPU and then efficiently evaluated on CPU. There are several features that are planned to be added in future. These include long short term memory (LSTM) based RNNLM [5], and also supporting more flexible RNNLM model structures. All resources related to this toolkit can be downloaded from http://mi.eng.cam.ac.uk/projects/cued-rnnlm/.

9. REFERENCES

[1] R. Kneser and H. Ney, "Improved backing-off for n-gram language modeling", *Proc. ICASSP*, 1995.

[2] S. F. Chen and J. T. Goodman. "An empirical study of smoothing techniques for language modeling", *Computer Speech and Language*, Vol. 13, Issue 4, pp. 359-394, 1999.

[3] T. Mikolov, S. Kombrink, L. Burget, J.H. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model", *Proc. ICASSP*, 2011.

[4] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul, "Fast and robust neural network joint models for statistical machine translation," *Association for Computational Linguistics*, 2014.

[5] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling", *Proc. Interspeech*, 2012.

[6] W.D. Mulder, S. Bethard, and M.F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling", *Computer Speech & Language*, vol. 30, no. 1, pp. 61–98, 2015.

[7] S. Kombrink, T. Mikolov, M. Karafiat, and L. Burget, "Recurrent neural network based language modeling in meeting recognition.," *Proc. Interspeech*, 2011.

[8] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Cernocky, "Recurrent neural network language modeling toolkit", *Proc. ASRU Workshop*, 2011.

[9] X. Chen, Y. Wang, X. Liu, M.J.F. Gales, and P.C. Woodland, "Efficient training of recurrent neural network language models using spliced sentence bunch", *Proc. Interspeech*, 2014.

[10] Geoffrey Zweig and Konstantin Makarychev, "Speed regularization and optimality in word classing", *Proc. ICASSP*, 2013.

[11] X. Chen, X. Liu, M.J.F. Gales, and P.C. Woodland, "Improving the training and evaluation efficiency of recurrent neural network language models", *Proc. ICASSP*, 2015.

[12] X. Chen, X. Liu, M.J.F. Gales, and P.C. Woodland, "Recurrent neural network language model training with noise contrastive estimation for speech recognition", *Proc. ICASSP*, 2015.

[13] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model", *Proc. ISCA Interspeech*, 2010.

[14] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, MIT Press, Cambridge, MA, USA, 1988.

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

[16] A. Emami and L. Mangu, "Empirical study of neural network language models for Arabic speech recognition", *Proc. ASRU Workshop*, 2007.

[17] J. Park, X. Liu, M.J.F. Gales, and P.C. Woodland, "Improved neural network based language modelling and adaptation", *Proc. Interspeech*, 2010.

[18] H. Le, I. Oparin, A. Allauzen, J. Gauvain, and F. Yvon, "Structured output layer neural network language models for speech recognition", *IEEE Trans Audio, Speech, and Language Processing*, vol. 21, no. 1, pp. 197–206, 2013.

[19] X. Liu, Y. Wang, X. Chen, M.J.F. Gales, and P.C. Woodland, "Efficient lattice rescoring using recurrent neural network language models", *Proc. ICASSP*, 2014.

[20] D. Yu, A. Eversole, M. Seltzer, K. Yao, Z. Huang, B. Guenter, O. Kuchaiev, Y. Zhang, F. Seide, H. Wang, et al., "An introduction to computational networks and the computational network toolkit", *Tech. Rep. MSR, http://codebox/cntk*, 2014.

[21] H. Schwenk, "CSLM-a modular open-source continuous space language modeling toolkit.," *Proc. Interspeech*, 2013.

[22] M. Sundermeyer, R. Schlüter, and H. Ney, "rwthlm the RWTH Aachen university neural network language modeling toolkit", *Proc. Interspeech*, 2014.

[23] Y. Shi, M.-Y. Hwang, K. Yao, and M. Larson. "Speed up of recurrent neural network language models with sentence independent subsampling stochastic gradient descent", *Proc. Interspeech*, 2013.

[24] A. Mnih and Y.W. Teh, "A fast and simple algorithm for training neural probabilistic language models," *Proc. International Conference on Machine Learning*, 2012.

[25] S. Young, G. Evermann, M.J.F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, A. Ragni, V. Valtchev, P.C. Woodland and C. Zhang, "The HTK book (for HTK 3.5)", *Cambridge University Engineering Department*, 2015.

[26] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsk, G. Stemmer and K. Vesel "The Kaldi speech recognition toolkit," *Proc. ASRU Workshop*, 2011.

[27] A. Deoras, T. Mikolov, S. Kombrink, and K. 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.

[28] A. Stolcke, "SRILM-an extensible language modeling toolkit," *Proc. Interspeech*, 2002.

[29] E. Arisoy, S.F. Chen, B. Ramabhadran, and A. Sethy, "Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition," *IEEE/ACM Trans. Audio, Speech, and Language Processing*, vol. 22, no. 1, pp. 184–192, 2014.

[30] X. Chen, T. Tan, X. Liu, P. Lanchantin, M. Wan, M.J.F. Gales and P.C. Woodland, "Recurrent Neural Network Language Model Adaptation for Multi-Genre Broadcast Speech Recognition," *Proc. Interspeech*, 2015.

[31] Y. Shi, M. Larson, and C.M. Jonker, "K-component recurrent neural network language models using curriculum learning," *Proc. ASRU Workshop*, 2013.

[32] I. McCowan, J. Carletta, W. Kraaij, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, M. Kronenthal, G. Lathoud, M. Lincoln, A. Lisowska, W. Post, D. Reidsma, and P. Wellner, "The AMI meeting corpus: A pre-announcement," *Machine learning for multimodal interaction*, pp. 28–39. Springer, 2006.

[33] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks.," *Proc. Interspeech*, 2013.

[34] L. Mangu, E. Brill, and A. Stolcke. "Finding consensus in speech recognition: word error minimization and other applications of confusion networks," *Computer Speech and Language*, 14(4): 373-400, 2000.