Efficient Training and Evaluation of Recurrent Neural Network Language Models for Automatic Speech Recognition

Xie Chen, *Member, IEEE*, Xunying Liu, *Member, IEEE*, Yongqiang Wang, *Member, IEEE*,
Mark J. F. Gales, *Fellow, IEEE*, and Philip C. Woodland, *Fellow, IEEE*

***Abstract*—Recurrent neural network language models (RNNLMs) are becoming increasingly popular for a range of applications including automatic speech recognition. An important issue that limits their possible application areas is the computational cost incurred in training and evaluation. This paper describes a series of new efficiency improving approaches that allows RNNLMs to be more efficiently trained on graphics processing units (GPUs) and evaluated on CPUs. First, a modified RNNLM architecture with a nonclass-based, full output layer structure (F-RNNLM) is proposed. This modified architecture facilitates a novel spliced sentence bunch mode parallelization of F-RNNLM training using large quantities of data on a GPU. Second, two efficient RNNLM training criteria based on variance regularization and noise contrastive estimation are explored to specifically reduce the computation associated with the RNNLM output layer softmax normalisation term. Finally, a pipelined training algorithm utilizing multiple GPUs is also used to further improve the training speed. Initially, RNNLMs were trained on a moderate dataset with 20M words from a large vocabulary conversational telephone speech recognition task. The training time of RNNLM is reduced by up to a factor of 53 on a single GPU over the standard CPU-based RNNLM toolkit. A 56 times speed up in test time evaluation on a CPU was obtained over the baseline F-RNNLMs. Consistent improvements in both recognition accuracy and perplexity were also obtained over C-RNNLMs. Experiments on Google's one billion corpus also reveals that the training of RNNLM scales well.**

***Index Terms*—Estimation, GPU, language models, noise contrastive, pipelined training, recurrent neural network, speech recognition, variance regularisation.**

I. Introduction

Statistical language models (LMs) are crucial components in a wide-range of applications, including automatic speech recognition (ASR) systems. A central part of language modelling, is how to appropriately model long-distance context dependencies in natural languages. This generally leads to a data sparsity problem for conventional back-off n-gram LMs given limited training data. In order to address this issue, LMs that can represent history contexts in a continuous vector space, for example, neural network LMs (NNLMs), can be used [5]–[10]. Depending on the underlying network architecture, these models can be splitted into two major categories: feedforward NNLMs [5]–[7], [10], which use a vector representation of preceding contexts of a finite number of words, and recurrent NNLMs (RNNLMs) [8], [11], which use a recurrent vector representation of longer and potentially variable length histories. LSTM RNNLMs [9] use the long short term memory unit which allows longer history be modelled. 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 [2], [3], [12]–[25], as well as other related applications including spoken language understanding [26], [27], and statistical machine translation [28]–[30].

A key issue that limits the possible application areas of RNNLMs, and standard recurrent neural networks (RNNs) in general, is the computational cost incurred in model training and evaluation. In order to address this issue, there has been increasing research interest in deriving efficient parallel training algorithms for RNNs based acoustic models and machine translation systems. Many of the these approaches including the Baidu's deep speech and Microsoft's CNTK systems [31], [32] use a synchronous stochastic gradient descent update algorithm with data parallelization and optionally pipelining between multiple processors [33], [34].

In contrast to previous research which has largely focused on RNN based acoustic modelling and machine translation tasks, this paper aims to improve the training and evaluation efficiency of RNN based LMs, with a particular focus on deriving novel training criteria to explicitly reduce the computational cost incurred at the RNNLM output layer. As a normalisation term is required for the softmax function, the output layer accounts for a major part of the overall computation. This has a significant impact on both RNNLM training and evaluation, when a large output layer vocabulary is used. In order to improve efficiency,

Manuscript received November 20, 2015; revised June 3, 2016 and July 13, 2016; accepted July 22, 2016. Date of publication August 4, 2016; date of current version September 2, 2016. This work was supported by the Engineering and Physical Sciences Research Council under Grant EP/I031022/1 (Natural Speech Technology). The work of X. Chen was supported by Toshiba Research Europe Ltd, Cambridge Research Lab. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Yang Liu.

X. Chen, P. C. Woodland, and M. J. F. Gales are with the Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, U.K. (e-mail: xc257@cam.ac.uk; pcw@eng.cam.ac.uk; mjfg@eng.cam.ac.uk).

X. Liu is with the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: xyliu@se.cuhk.edu.hk).

Y. Wang was with the Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, U.K. He is now with the Microsoft Corporation, Redmond, WA 98052 USA (e-mail: yw293@cam.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASLP.2016.2598304

most of existing techniques use an RNNLM architecture with a class based factorized output layer [11], known as class based RNNLMs (C-RNNLM). A similar architecture has been used for feedforward NNLMs [35]. As the number of classes is normally significantly smaller than the full output layer size, a speed up of both training and evaluation can be achieved [11]. When further combined with parallelized model training [36] and multi-stage classing at the output layer [37], training time speed 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 speedup from bunch[1] mode training parallelization [15]. Second, the underlying word to class assignment schemes can impact the resulting C-RNNLM's performance [2], [11], [39], [40]. Finally, only CPU based speed up techniques were studied in previous research [11], [15], [36], [37]. Hence, it is preferable 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 proposed 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 can be implemented on GPUs to fully exploit their parallel computing power.

Several techniques are also explored to further improve the training and evaluation speed of F-RNNLMs. These include two efficient RNNLM training algorithms that attempt to significantly reduce the computation associated with the output layer softmax normalisation term. In variance regularisation (VR) based RNNLM training, the variance of the output normalisation term is introduced into the conventional cross entropy (CE) based training objective function [2], [30], [41]–[43], and explicitly minimized. This allows this term to be ignored during testing time thus gaining significant speed up, while retaining the performance comparable to conventional CE training. A second more efficient training algorithm based on noise contrastive estimation (NCE) [44] is also investigated. NCE training implicitly minimizes the variance of the output layer softmax normalisation term, and allows this normalisation term to be ignored during both training and evaluation time.

The rest of this paper is organized as follows. In Section II RNNLMs are reviewed and the two RNNLM architectures are presented. A novel spliced sentence bunch mode parallelization algorithm for F-RNNLM training and a GPU based implementation of this algorithm are proposed in Section IV. A detailed GPU based implementation of this algorithm is described in Section IV-C. VR and NCE based RNNLM training methods are presented in Sections III-B and III-C. Pipelined RNNLM training is described in Section V. In Section VI the performance of the proposed F-RNNLMs and a range of efficiency improving techniques are evaluated on a large vocabulary conversational telephone speech (CTS) transcription system and the Google's one billion word benchmark task. Section VII draws the conclusions and discusses possible future work.

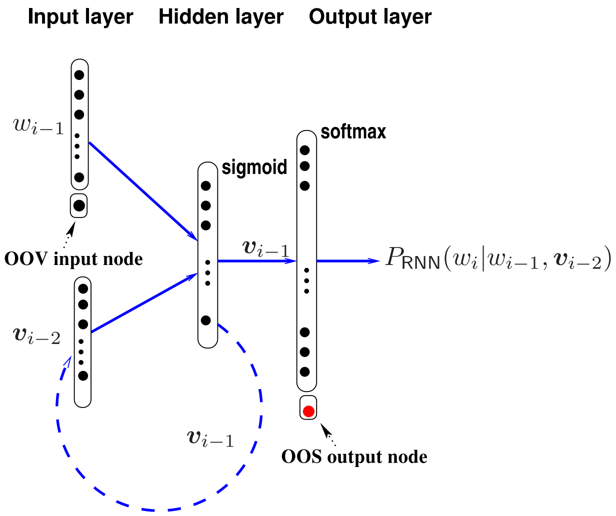


Fig. 1. An example RNNLM with an full output layer and OOS nodes.

II. Recurrent Neural Network LMs

There are two types of neural network structures used for language modelling. They are feedforward NNLMs [5] and recurrent NNLMs [8] respectively. n gram Feedforward NNLMs could be constructed using a standard multilayer perceptron neural network. In contrast to feedforward NNLMs, recurrent NNLMs represent the full, non-truncated history $h_1^{i-1} = \langle w_{i-1}, \ldots, w_1\rangle$ for word w_i using a 1-of-k encoding of the most recent preceding word w_{i-1} and a continuous vector $\boldsymbol{v}_{i-2}$ for the remaining context. For an empty history, this is initialized, for example, to a vector of all ones. The topology of the RNN used to compute LM probabilities $P_{\textsf{RNN}}(w_i|w_{i-1},\boldsymbol{v}_{i-2})$ consists of three layers. The full history vector, obtained by concatenating w_{i-1} and $\boldsymbol{v}_{i-2}$, is fed into the input layer. The hidden layer compresses the information of these two inputs and computes a new representation $\boldsymbol{v}_{i-1}$ 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 $P_{\textsf{RNN}}(w_{i+1}|w_i,\boldsymbol{v}_{i-1})$. 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 [17] can be used.

A. Full Output Layer Based RNNLMs (F-RNNLMs)

A standard RNNLM architecture with an unclustered, full output layer (F-RNNLM) is shown in Fig. 1. RNNLMs can be trained using an extended form of the standard back propagation algorithm, back propagation through time (BPTT) [45], where the error is propagated through recurrent connections back in time for a specific number of time steps, for example, 4 or

[1] It is also sometimes referred as "minibatch" in the literature [34], [38]. For clarity, the term "bunch" is used throughout this paper.

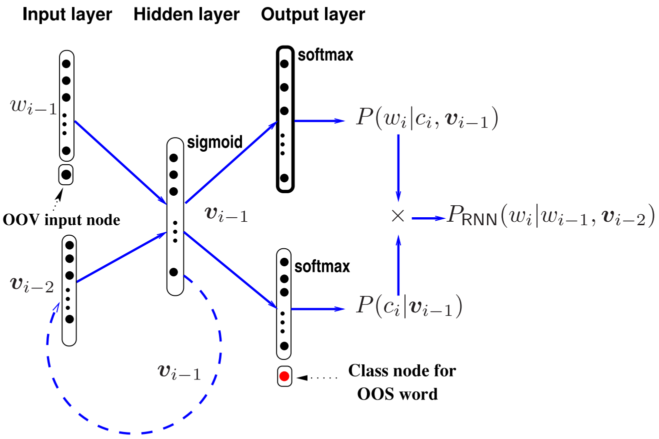


Fig. 2. An example RNNLM with a class-based output layer and OOS nodes.

5 [11]. This allows the recurrent network to record information for several time steps in the hidden layer. To reduce the computational cost, a shortlist [6], [46] 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 [7], [10], [17].

B. 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 C-RNNLMs, an RNNLM architecture with a class based factorized output layer [11]. An example C-RNNLM is illustrated in Fig. 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

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

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, speed up in both training and evaluation time can be achieved. A special case of C-RNNLM using a single class is equivalent to a traditional, full output layer based F-RNNLM introduced in Section II-A. A modified version of the RNNLM toolkit [47] supporting the above architecture is used.

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 [6]–[8], [10], [15], [46]. The interpolated LM probability is given by

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

where 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 [7], [10].

III. RNNLM Training Criteria

RNNLMs were normally trained with CE based objective function. In this section, two other training criteria will also be introduced to improve train and evaluation efficiency.

A. Cross Entropy

Conventional RNNLM training aims to maximise the log-likelihood, or equivalently minimize the CE measure of the training data. For a given sequence containing a total of N_w words, the objective function is given by

$$J^{\text{CE}}(\boldsymbol{\theta}) = -\frac{1}{N_w}\sum_{i=1}^{N_w} \ln P_{\text{RNN}}(w_i|h_i) \quad (3)$$

where

$$P_{\text{RNN}}(w_i|h_i) = \frac{\exp\left(\boldsymbol{\theta}_i^\top \boldsymbol{v}_{i-1}\right)}{\sum_{j=1}^{|V|}\exp\left(\boldsymbol{\theta}_j^\top \boldsymbol{v}_{i-1}\right)} = \frac{\exp\left(\boldsymbol{\theta}_i^\top \boldsymbol{v}_{i-1}\right)}{Z(h_i)} \quad (4)$$

is the probability of word w_i given history h_i. $\boldsymbol{\theta}_i$ is the weight vector associated with word i at the output layer. $\boldsymbol{v}_{i-1}$ is the hidden history vector computed at the hidden layer, and $|V|$ is the size of output layer vocabulary. The gradient used in the conventional CE based training for RNNLMs is

$$\frac{\partial J^{\text{CE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{1}{N_w}\sum_{i=1}^{N_w}\left(\frac{\partial\left(\boldsymbol{\theta}_i^\top \boldsymbol{v}_{i-1}\right)}{\partial \boldsymbol{\theta}} - \sum_{j=1}^{|V|} P_{\text{RNN}}(w_j|h_i)\frac{\partial\left(\boldsymbol{\theta}_j^\top \boldsymbol{v}_{i-1}\right)}{\partial \boldsymbol{\theta}}\right). \quad (5)$$

The denominator term $Z(h_i) = \sum_{j=1}^{|V|}\exp\left(\boldsymbol{\theta}_j^\top \boldsymbol{v}_{i-1}\right)$ in (4) performs a normalisation over the full output layer. As discussed in Section I, this operation is computationally highly expensive when computing the RNNLM probabilities during both test time and CE based training when the gradient information of (5) is calculated. As discussed in Section IV, the efficient bunch mode GPU based parallelization with sentence splicing is used to improve the speed of conventional CE training.

B. Variance Regularisation

One technique that can be used to improve the testing speed is introducing the variance of the normalisation term into the conventional CE based objective function of (4). In previous research VR has been applied to training of feedforward NNLMs and class based RNNLMs [30], [41], [42]. By explicitly minimizing the variance of the softmax normalisation term during variance training, the normalisation term at the output layer can be ignored during testing time thus gaining significant improvements in speed. The conventional CE objective function of (3) is modified as

$$J^{\text{VR}}(\boldsymbol{\theta}) = J^{\text{CE}}(\boldsymbol{\theta}) + \frac{\gamma}{2N_w}\sum_{i=1}^{N_w}\left(\ln Z(h_i) - \ln \bar{Z}\right)^2 \quad (6)$$

where $J^{\mathsf{CE}}(\boldsymbol{\theta})$ and $Z(h_i)$ are the CE based training criterion and the softmax normalisation term associated with a history h_i in (4) respectively, and $\ln Z$ is the mean of the log scale normalisation term computed over the N_w words in the train data

$$\ln Z = \frac{1}{N_w} \sum_{i=1}^{N_w} \ln\left(Z(h_i)\right). \tag{7}$$

γ is a tunable parameter that adjusts the contribution from the VR term. Directly maximising the above objective function in (6) during VR based RNNLM training explicitly minimises the variance of the softmax normalisation term. The gradient used in the VR based training is given by

$$\frac{\partial J^{\mathsf{VR}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\gamma}{N_w} \sum_{i=1}^{N_w} (\ln Z(h_i) - \ln Z) \times \sum_{j=1}^{|V|} P_{\mathsf{RNN}}(w_j|h_i) \frac{\partial\left(\boldsymbol{\theta}_j^{\top} \boldsymbol{v}_{i-1}\right)}{\partial \boldsymbol{\theta}} \tag{8}$$

where $\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ is the CE gradient given in (5), and $P_{\mathsf{RNN}}(\cdot|h_i)$ the standard RNNLM probabilities computed using (4) with normalisation. VR allows a history independent, constant softmax normalisation term to be used in evaluation time. The RNNLM probabilities are thus approximated as

$$P_{\mathsf{RNN}}^{\mathsf{VR}}(w_i|h_i) \approx \frac{\exp\left(\boldsymbol{\theta}_i^{\top} \boldsymbol{v}_{i-1}\right)}{Z}. \tag{9}$$

This significantly reduces the computation at the output layer as the normalisation is no longer required. As such computation is no longer sensitive to the size of output layer vocabulary, $|V|$, a maximum $|V|$ times speed up at the output layer can be achieved in test time.

In this paper VR based training is used to improve the evaluation time efficiency for F-RNNLMs [2], [43] and integrated with the bunch mode parallel training algorithm in Section IV. In contrast to setting the mean of log normalisation term $\ln Z$ to zero used in the previous research for C-RNNLMs [42], it is found that calculating $\ln Z$ separately for individual bunches gave improved convergence speed and stability in F-RNNLM training. During test time, this term is computed on a validation set first, remains fixed and applied on unnormalized probability as Equation (9). [2]

C. Noise Contrastive Estimation

As discussed in Section I, the calculation of the the softmax normalisation term required at the output layer significantly impacts both the training and testing speed of RNNLMs. VR can significantly reduce the associated computation during evaluation time. However, the calculation of this normalisation term is still required in training and used to compute the variance regularised gradient information in (8). A more general solution to this problem is to use techniques that can remove the need to compute such normalisation term in both training and testing. One such technique investigated in this paper is based on NCE [3], [20], [43], [44].

NCE provides an alternative solution to estimate normalized statistical models when the exact calculation of the normalisation term is either not possible or highly expensive to perform, for example, in feedforward and recurrent NNLMs, when a large output layer vocabulary is used. The central idea of NCE is to perform a nonlinear logistic regression to discriminate between the observed data and some artificially generated noise data. The variance of normalisation term is minimized implicitly during training. Hence, it allows normalized statistical models, for example, NNLMs, to use "unnormalized" probabilities without explicitly computing the normalisation term during both training and decoding. In common with the use of a class based output layer, the NCE algorithm presents a dual purpose solution to improve both the training and evaluation efficiency for RNNLMs.

For NCE based training of RNNLMs, it is assumed that for a given full history context h_i, data samples are generated from a mixture of two distributions: the NCE estimated RNNLM distribution $\tilde{P}_{\mathsf{RNN}}^{\mathsf{NCE}}(\cdot|h_i)$, and some noise distribution $P_n(\cdot|h_i)$ that satisfies the desired sum-to-one constraint. Assuming that the noise samples are k times more frequent than true RNNLM data samples, the distribution of all data can be described as $\frac{1}{k+1}\tilde{P}_{\mathsf{RNN}}^{\mathsf{NCE}}(\cdot|h_i) + \frac{k}{k+1}P_n(\cdot|h_i)$. The posterior probabilities of a word $\tilde{w}$ being generated either from the RNNLM or noise distribution, are

$$\begin{aligned} P(C_{\tilde{w}}^{\mathsf{RNN}} = 1|\tilde{w}, h_i) &= \frac{\tilde{P}_{\mathsf{RNN}}^{\mathsf{NCE}}(\tilde{w}|h_i)}{\tilde{P}_{\mathsf{RNN}}^{\mathsf{NCE}}(\tilde{w}|h_i) + kP_n(\tilde{w}|h_i)} \\ P(C_{\tilde{w}}^{n} = 1|\tilde{w}, h_i) &= \frac{kP_n(\tilde{w}|h_i)}{\tilde{P}_{\mathsf{RNN}}^{\mathsf{NCE}}(\tilde{w}|h_i) + kP_n(\tilde{w}|h_i)} \end{aligned} \tag{10}$$

where $C_{\tilde{w}}^{\mathsf{RNN}}$ and $C_{\tilde{w}}^{n}$ are the binary labels indicating which of the two distributions that generated word $\tilde{w}$. The following objective function is minimized during NCE based training:

$$J^{\mathsf{NCE}}(\boldsymbol{\theta}) = -\frac{1}{N_w} \sum_{i=1}^{N_w} \Big(\ln P(C_{w_i}^{\mathsf{RNN}} = 1|w_i, h_i) + \sum_{j=1}^{k} \ln P(C_{\check{w}_{i,j}}^{n} = 1|\check{w}_{i,j}, h_i)\Big) \tag{11}$$

where a total of k noise samples $\{\check{w}_{i,j}\}$ are drawn from the noise distribution $P_n(\cdot|h_i)$ for the current training word sample w_i and its history context h_i. The gradient of the above NCE objective function in (11) is then computed as

$$\frac{\partial J^{\mathsf{NCE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{1}{N_w} \sum_{i=1}^{N_w} \Big(P(C_{w_i}^{n} = 1|w_i, h_i) \frac{\partial}{\partial \boldsymbol{\theta}} \ln \tilde{P}_{\mathsf{RNN}}^{\mathsf{NCE}}(w_i|h_i) - \sum_{j=1}^{k} P(C_{\check{w}_{i,j}}^{\mathsf{RNN}} = 1|\check{w}_{i,j}, h_i) \frac{\partial}{\partial \boldsymbol{\theta}} \ln \tilde{P}_{\mathsf{RNN}}^{\mathsf{NCE}}(\check{w}_{i,j}|h_i)\Big) \tag{12}$$

[2] The fixed normalisation term Z is also equivalent to changing the word insertion penalty.

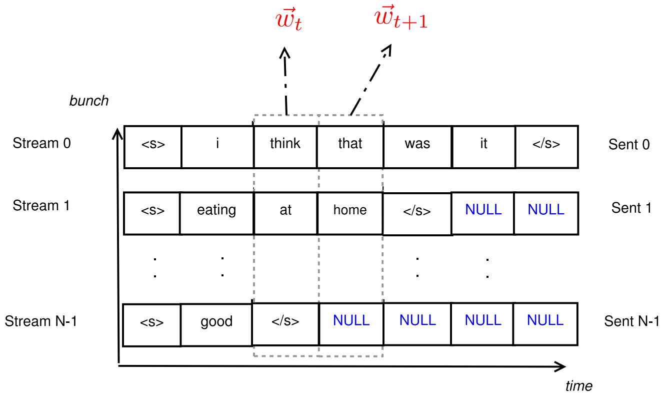
where the NCE trained RNNLM distribution is given by

$$\tilde{P}_{\text{RNN}}^{\text{NCE}}(w_i|h_i) = \frac{\exp\left(\boldsymbol{\theta}_i^\top \boldsymbol{v}_{i-1}\right)}{Z} \tag{13}$$

and constrained during NCE training by setting a constant, history context independent normalisation term Z, in contrast to fully normalized RNNLM distribution that is used to compute both the CE and variance regularised gradient information given in (3) and (8).[3] This crucial feature not only allows the resulting RNNLM to learn the desired sum-to-one constraint of standard CE estimated RNNLMs, but also to be efficiently computed during both training and test time without requiring explicitly computing the softmax normalisation term at the output layer.

In this paper, NCE training of RNNLMs is implemented on GPU and integrated with the sentence splicing based bunch mode training of Section IV. A bunch size of 128 was used in all experiments. CUBLAS is used for matrix operation. The NCE objective function shown in (11) is optimized on the training set. The CE measure on the validation set computed using the conventional RNNLM probabilities with normalisation in (4) is used to control the learning rate. During NCE training, a number of parameters need to be appropriately set. First, a noise distribution is required in NCE training to provide a valid sum-to-one constraint for the NCE estimated RNNLM to learn. As suggested in earlier research presented in [48], [49], a history independent unigram LM distribution is used to draw the noise samples during NCE training in this paper. Second, the setting of k controls the bias towards the characteristics of the noise distribution. It also balances the trade-off between training efficiency and performance. In this paper, for each target word w, a total of $k = 10$ noise samples are independently generated from the noise distribution. It is worth noting that the noise sample could be the predicted word and the same noise sample may appear more than once. Finally, NCE training also requires a constant normalisation term Z in equation (13) to be set. In previous research on NCE training of log-bilinear LMs [48] and feedforward NNLMs [49], the constant normalisation term was set as $\ln Z = 0$. In this paper for RNNLMs an empirically adjusted setting of $\ln Z = 9$ was used. This was close to the mean of the log scale normalisation term computed using a randomly initialized RNNLM on the whole training corpus. This setting was found to give a good balance between convergence speed and performance and used in all experiments.

The main advantages of RNNLMs training with NCE are summarized below. First, the computation on output layer is reduced dramatically as it only needs to consider k noise samples and the target word, instead of the whole output layer. Compared with the CE based training gradient given in equation (5), the computation of NCE gradient in equation (12) gives a total speed up of $\frac{|V|}{k+1}$ times at the output layer. Second, the train speed is insensitive to output layer size, which allows RNNLMs with larger vocabulary to be trained. Finally, the normalisation term is constrained to be a constant during NCE training. This can avoid the re-computation of the normalisation term for different histories, therefore allows the normalized RNNLM probabilities to calculated in test time with the same efficiency as unnormalized probabilities. In common with VR based training, a $|V|$ times speed up at the output layer during test time can thus be achieved.



Stream 0	<s>	i	think	that	was	it	</s>	Sent 0
Stream 1	<s>	eating	at	home	</s>	NULL	NULL	Sent 1
:	:	:			:	:		
Stream N-1	<s>	good	</s>	NULL	NULL	NULL	NULL	Sent N-1

Fig. 3. An example of bunched RNNLM training without sentence splicing (where N denotes the bunch size).

IV. Spliced Sentence Bunch Training

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 [6], [50]. A fixed number of n-grams from the training data are formed as a bunch. This bunch of data is propagated through the network for the computation of gradients. These gradients over the bunch were then accumulated and used for updating the weight parameters.

A. Standard Bunch Mode RNNLM Training

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 [15], [51]. This form of parallelization requires each sentence to be regarded as independent in RNNLM training by re-initializing the recurrent hidden history vector at the start of every sentence.

The basic idea of bunch mode training is shown in Fig. 3. Given the bunch size N, N sentences are aligned from left to right. During parallelization, a regular structured input matrix is formed. The element at the jth row and tth column in the input matrix, associated with time $t+1$ and an output word $w_{t+1}^{(j)}$, represents a vector $[w_t^{(j)}, v_{t-1}^{(j)}]^\top$, where $w_t^{(j)}$ and $v_{t-1}^{(j)}$ are the 1-of-k vector encoding of the t^{th} word of the j^{th} sentence in the bunch, and the corresponding recurrent history vector at word $w_t^{(j)}$ 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

[3] A more general case of NCE training also allows the normalisation term to vary across different histories, thus incurring the same cost as in conventional CE based training [44].

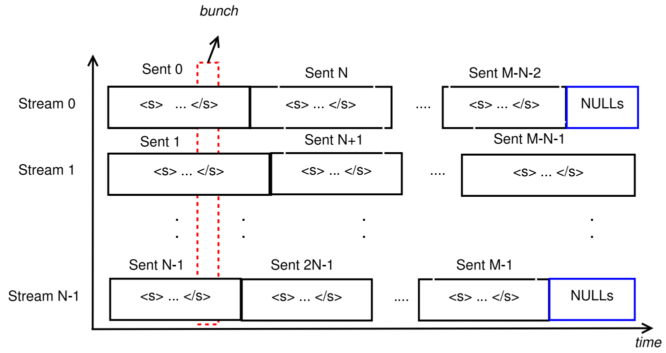


Fig. 4. An example of bunched RNNLM training with sentence splicing (N denotes the bunch size and M is the number of sentences in the whole train corpus). The bunch is initially filled top-down from stream 0 to stream $N-1$ with a total of N randomly sorted training data sentences, and then from left to right repeat the same process until the N stream bunch is filled with a minimum number of NULL tokens at the stream end.

corpus. NULL words are then inserted at the end of other shorter sentences in the bunch, as is shown in Fig. 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 [11], [15]. 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.

B. Bunch Mode RNNLM Training With Sentence Splicing

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 Fig. 4. Sentences in the training corpus are joined into streams that are more comparable in length. 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, F-RNNLMs introduced in Section II-A are also chosen 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.

C. GPU Implementation of Bunch Mode RNNLM Training

To improve efficiency, graphics processing units (GPUs), which have been previously employed to train deep neural network (DNN) based acoustic models in speech recognition [34], [38], are used to train RNNLMs used in this paper. CUBLAS from CUDA 5.0, the basic linear algebra subprograms library optimized for NVidia GPUs, is used for fast matrix operation. As discussed in Sections I and II-B, 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 I
INITIAL LEARNING RATE PER SAMPLE WITH DIFFERENT BUNCH SIZE

Bunch size	1	8	32	64	128	256
Learning rate	0.1	0.0375	0.025	0.0156	0.0156	0.0078

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 trained with bunch mode, the initial learning rate per sample 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 0.1, in line with the default setting used in the RNNLM toolkit [47]. The initial learning rate settings used for various other bunch sizes are also shown in Table I. When the bunch size increases to 128, the initial learning rate is set to 0.0078 per sample.

V. PIPELINED TRAINING OF RNNLMs

The parallel structure of neural network training can be classified into two categories: model parallelism and data parallelism [52]. 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 first proposed to speedup the training of DNN based acoustic models in [34]. In this paper pipelined training of RNNLMs is used. Layers of the network are distributed across different GPUs. Operations on these layers such as the forward pass and error back propagation are executed on their own GPUs. It allows each GPU to proceed independently and simultaneously. The communication between different layers is performed after each parameter update step.

The data flow of pipelined training is shown in Fig. 5. Two weight matrices (denoted by Weight 0 and Weight 1) are kept in two GPUs (denoted by 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 the output layer. This is followed in sequence by error back propagation, parameter update, and the communication between GPUs in the form of a copying operation. For the following

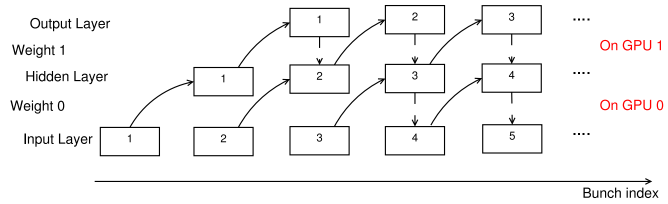


Fig. 5. An example of data flow in pipelined RNNLM training using 2 GPUs.

bunches, GPU 0 updates the model parameters using the corresponding error signal and input with BPTT, before forwarding the new input data for the next bunch. GPU 1 performs successively a forward pass, error back propagation and parameter update again.

TABLE II
TRAINING SPEED, PERPLEXITY AND WER RESULTS OF CPU TRAINED C-RNNLMS ON CONVERSATIONAL ENGLISH FISHER **DEV04** WITH VARYING HIDDEN NODES AND A FIXED NUMBER OF CLASSES OF 200

Hidden	Speed	Train time	dev04	
nodes	(w/s)	(hours)*	PPL	WER
100	7.6 k	9.8	50.7	16.13
200	2.1 k	35.6	48.6	15.82
512	0.37 k	202.1	46.5	**15.32**
800	0.11 k	679.9	45.8	15.40

TABLE III
TRAINING SPEED, PERPLEXITY AND WER RESULTS OF GPU TRAINED F-RNNLMS ON **DEV04** WITH VARYING BUNCH SIZES AND A FIXED HIDDEN LAYER SIZE OF 500

Model type	Bunch size	#Parameter	Speed (w/s)	Time (hours)	dev04	
					PPL	WER
C-RNN	–	26.9M	0.37 k	202.1	46.5	15.32
F-RNN	1	26.8M	0.17	435.3	45.9	15.26
	8		1.4 k	53.4	45.7	15.22
	32		4.6 k	16.3	45.6	15.25
	64		7.6 k	9.8	45.7	15.16
	128		10.1 k	7.4	46.3	15.22
	256		12.9 k	5.7	46.5	15.38
	512		13.3 k	5.6	47.5	15.55
	1024		14.2 k	5.2	48.2	15.63

VI. EXPERIMENTS

In this section the performance of the proposed techniques to improve RNNLM training and evaluation efficiency are evaluated using two tasks: a HTK-based large vocabulary speech recognition system developed for English CTS used in the 2004 DARPA EARS evaluation [53]; and Google's one billion word benchmark corpus [54] for language modelling.

A. CTS-English ASR Experiment

In this section, RNNLMs are evaluated on the CU-HTK LVCSR system for 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 59 k 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 [53]. 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 [55], **UWWeb**, of 525 million words (weight 0.25). The **Fisher** data was used to train various RNNLMs. A 38k word input layer vocabulary and 20 k 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[4] with the modified RNNLM toolkit [47] compiled with g++. The number of BPTT steps was set as 5. A computer with dual Intel Xeon E5-2670 2.6 GHz 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. 12 epochs were required for RNNLMs training to get convergence in this task. The 100-best hypotheses extracted from the baseline 4-gram LM lattices were rescored for performance evaluation. The perplexity and error rates of various RNNLMs are shown in Table II. The C-RNNLM with 512 hidden layer nodes gave the lowest WER of 15.32% and serves as the baseline C-RNNLM in all the experiments.

1) Experiment on Bunch Mode GPU Training: The next experiment is to examine the efficiency of bunch mode GPU based RNNLM training with sentence splicing. 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 IV and IV-C 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 III. The performance of the baseline C-RNNLM with 512 hidden nodes (previously shown in 3rd line in Table II) is again shown in the 1st line of Table III. 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 training speed and WER was obtained by using a bunch size of 128. This gives 27 times speed up and a 0.1% absolute reduction (significant at alpha = 0.05, obtained with MASSSWE test) in

[4] A speedup of 1.7 times for CPU based training could be obtained by the Intel MKL CUBLAS 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.

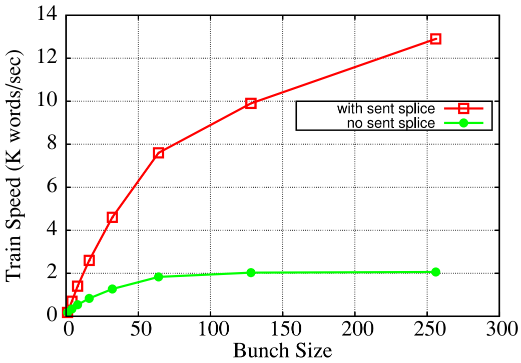


Fig. 6. F-RNNLM training speed with and without sentence splicing.

TABLE IV
EVALUATION SPEED OF RNNLMs FOR N-BEST SCORING ON DEV04

Model type	Device	Bunch	Test speed (words/sec)
F-RNN	CPU	N/A	0.14 k
C-RNN			5.9 k
F-RNN	GPU	1	1.1 k
		64	41.3 k
		512	56.3 k

The performance is the same for F-RNNLM using GPU with various bunch size.

WER over the C-RNNLM baseline. 128 is chosen as the default bunch size for the following experiments.

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 multiplication required. The remaining computation is shared by other operations such as resetting F-RNNLM hidden vectors at the sentence start, and data transfer between the CPU and GPU. This breakdown of training time suggests that further speed up is also possible via pipelined training by allocating the computation of the hidden layer and output layer into different GPUs, as will be later shown in Section VI-A4. 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 Fig. 6, where a contrast in speed with and without sentence splicing is drawn. When using the standard bunch model training with no sentence splicing as shown in Fig. 3, only limited speed improvements were 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 IV. These results suggest that the proposed sentence splicing technique is important for improving the efficiency of bunch mode RNNLM training.

The spliced sentence bunch based parallelization can also be used for RNNLM performance evaluation on GPU. Table IV shows the speed information measured for N-best rescoring using the baseline C-RNNLM and the F-RNNLM of Table III. 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 V
PERPLEXITY AND WER PERFORMANCE OF F-RNNLMs TRAINED WITH VARIANCE REGULARISATION ON DEV04

γ	Log-norm		PPL	WER	
	Mean	Var		Z(h)	Z
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

The mean and variance of log normalisation term were computed over the validation data at the end of each epoch. The two columns under WER (Z(h) and Z) denote word error rates using normalised or approximated RNNLM probabilities computed using (4) and (9).

2) Experiment on VR: In this section, the performance of F-RNNLMs trained with VR are evaluated. These experimental results with various settings of the regularisation constant γ in (6) are shown in Table V. The word error rates with $Z(h)$ in the table are the WER scores measured using the standard normalised RNNLM probabilities computed using (4). WERs with Z in the last column are obtained using the more efficiently approximated RNNLM probabilities given in (9). The first row of Table V shows the results without VR by setting γ to 0, the same to the standard CE based training. As expected, the WER was increased from 15.22% (standard fully normalized F-RNNLMs) to 16.24% without performing the normalisation. This confirms that the normalisation term computation for the softmax function is crucial for using CE trained RNNLMs in decoding.

When the VR term is applied in F-RNNLM training, there is only a small difference in terms of WER between using the accurate normalisation term $Z(h)$ or approximate normalisation term Z. 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 gave a WER of 15.28%, insignificant to the WER of the baseline CE trained F-RNNLM (1st line in Table V and also 5th line in Table III). At the same time, significantly improvements in evaluation speed were also obtained. This is shown in Table VI. The CPU based F-RNNLM evaluation speed was increased by a factor of 56 over the CE trained F-RNNLM baseline using VR, while retaining the same training speed.

3) Experiment on NCE Training: As discussed in Sections I and III-C, an important attribute of NCE based RNNLM training is that the variance of the RNNLM output layer normalisation term Z of (13) can be implicitly constrained to be minimum during parameter estimation. This effect is illustrated in Fig. 7 on log scale over a total of 12 epochs on the validation data

TABLE VI
TRAINING AND EVALUATION SPEED OF F-RNNLMS TRAINED WITH VARIANCE REGULARISATION ON **DEV04**

Model type	Train crit	Train speed(w/s)	Train time (hours)	Test speed(w/s)
C-RNN	CE	0.37 k	202.1	5.9 k
F-RNN	CE	10.1 k	7.4	0.14 k
	VR	10.1 k	7.4	7.9 k

C-RNNLMs were trained on CPU and F-RNNLMs on GPU. Both were evaluated on CPU.

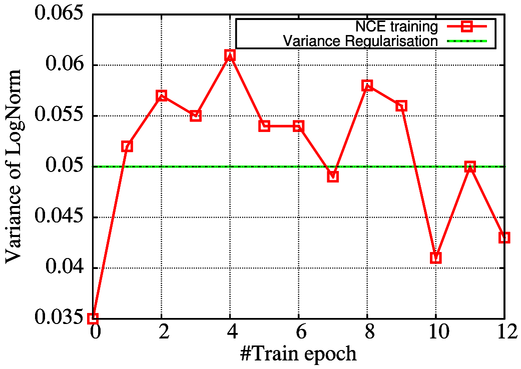


Fig. 7. Variance of the output layer log-normalisation term on validation data at different epochs during NCE based RNNLM training.

TABLE VII
PERPLEXITY AND WER PERFORMANCE, TRAINING AND EVALUATION SPEED OF NCE TRAINED F-RNNLMS ON **DEV04**

Model type	Train crit	Train speed(w/s)	Train time(hr)	Test speed(w/s)	dev04	
					PPL	WER
C-RNN	CE	0.37 k	202.1	5.9 K	46.5	15.32
F-RNN	CE	10.1 k	7.4	0.14 k	46.3	15.22
	VR	10.1 k	7.4	7.9 k	46.6	15.28
	NCE	19.7 k	3.8	7.9 k	46.8	15.37

C-RNNLMs trained on CPU and F-RNNLMs on GPU. Both evaluated on CPU.

TABLE VIII
TRAINING SPEED AGAINST THE SIZE OF F-RNNLM OUTPUT LAYER

#Output	Train speed (w/s)	
layer nodes	CE	NCE
20 k	10.1k	19.7 k
25 k	9.1 k	
30 k	8.0 k	

TABLE IX
TRAINING SPEED, PERPLEXITY AND WER PERFORMANCE OF F-RNNLMS ON **DEV04** USING PIPELINED CE TRAINING

Model type	GPU	Train speed(w/s)	dev04	
			PPL	WER
C-RNN	–	0.37 k	46.5	15.32
F-RNN	1 × TITAN	10.1 k	46.3	15.22
	1 × K20m	6.9 k	46.3	15.22
	2 × K20m	11.0 k	46.3	15.23

set. The variance of the normalisation term is slightly increased from 0.035 to 0.06 from the beginning to the fourth epoch, then gradually reduced to 0.043 at the last epoch.

The WER and PPL performance of an NCE trained RNNLM are shown in Table VII. 12 epochs were required for both the conventional CE and NCE based training to converge. As discussed in Section III-C, the log normalisation term ln Z in Equation (13) was fixed as 9. The perplexity scores in Table VII were obtained by explicitly computing the output layer normalisation term. During N-best rescoring, normalized RNNLM probabilities were used for the CE trained RNNLM baseline, while unnormalized probabilities were used for the NCE trained RNNLM. As expected, when unnormalized probabilities were used by the CE trained RNNLM, a large degradation in performance was found. As is shown in Table VII, the NCE trained RNNLM gave slightly worse performance to the CE trained baseline. At the same time, the training speed was doubled. This is expected as the time consumed on output layer is approximately half of the total training time required for conventional CE training.

Similarly a large testing time speed up of 56 times over the CE trained RNNLM on CPUs was also obtained, as is shown in Table VII. This improvement is comparable to the speed up obtained using VR based RNNLM training previously shown in Table VI. As the computation of the normalisation term is no longer necessary for NCE trained RNNLMs, the computational cost incurred at the output layer can be significantly reduced. The statistical significance test was also carried out for various training criteria. It reveals that the WER performance difference between CE and VR is insignificant, while the difference between CE and NCE is significant.

As expected, the NCE training speed is also largely invariant to the size of the output layer, thus improves the scalability of RNNLM training when a very large output vocabulary is used. This highly useful feature is clearly shown in Table VIII, where CE training speed is decreased rapidly when the output layer size increases. In contrast, the NCE training speed remains constant against different output layer vocabulary sizes.

4) Experiment on Dual GPU Pipelined Training: 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 the pipelined training experiments, two slightly slower NVidia Tesla K20m GPUs housed in the same server were used. Table IX shows the training speed, perplexity and WER results of pipelined CE training for F-RNNLMs. As is shown in the table, Pipelined training gave a speed up of a factor of 1.6 times and performance comparable to a single GPU based training.

TABLE X
PERPLEXITY PERFORMANCE OF RNNLMS ON GOOGLE'S ONE BILLION (FOR ASR (PROVIDED BY CANTAB RESEARCH) CORPUS) WORD CORPUS

LMs	Train crit	Train speed(w/s)	PPL	
				+NG5
NG5	–	–	–	83.7
+F-RNN	CE	6.7 k	104.4	65.8
	NCE	11.3 k	107.3	66.0

TABLE XI
PERPLEXITY PERFORMANCE OF RNNLMS ON GOOGLE'S ONE BILLION WORD CORPUS USING 793 K VOCABULARY

LMs	Train crit	Train speed(w/s)	Test speed(w/s)	PPL	
					+NG5
NG5	–	–	–	70.9	–
F-RNN	NCE	6.5 k	37.1	77.3	52.6

The train is run on GPU and test is on CPU

B. Google's One Billion Word Experiment

A new benchmark corpus was released by Google for measuring performance of statistical LMs [54]. Two categories of text normalisation are provided. One is for machine translation (StatMT) and the other is for ASR (by Cantab Research).[5] The later was used to further evaluate the performance and scalability of NCE based RNNLM training in this experiment for training RNNLMs on large corpus. The former will be used to build LM in the next experiment. A total of 800 million words were used in LM training. A test set of 160 k words (obtained from the first split from held-out data) was used for perplexity evaluation. A modified KN smoothed 5-gram LM was trained using the SRILM toolkit [56] with zero cut-offs and no pruning. In order to reduce the computational cost in training, an input layer vocabulary of 60 k most frequent words and a 20 k word output layer shortlist were used. RNNLMs with 1024 hidden layer nodes were either CE or NCE trained on a GPU using a bunch size of 128. The other training configurations were the same as the experiments presented in Section VI-A3. A total of 10 epochs were required to reach convergence for both CE and NCE based training. The perplexity performance of these two RNNLMs are shown in Table X. Consistent with the trend found in Table VII, the CE and NCE trained RNNLMs gave comparable perplexity when interpolated with the 5-gram LM. A large perplexity reduction of 21% relative over the 5-gram LM was obtained.

In order to further investigate the scalability of NCE based RNNLM training, an additional set of experiments comparable to those presented in Table X were conducted using a much larger, full output layer vocabulary of 793 k words as used in previous research [52]. It is worth mentioning that the corpus used in this experiment is for machine translation, which is different to the corpus used for ASR provided by Cantab Research in previous experiment. Due to the large size of such output layer vocabulary, the speed of standard CE training of a full output layer based RNNLM is as slow as 200 words per second. It is therefore computationally infeasible in practice to train such a baseline RNNLM. The training speed and perplexity score of a NCE trained RNNLM with a 793 k vocabulary are presented in Table XI, together with the baseline 5-gram LM's perplexity performance. A total of 1024 hidden nodes and a bunch size of 128 were used. The stand alone NCE trained RNNLM gave a perplexity score of 77.3. This was further reduced to 52.6 after an interpolation with the 5-gram LM.[6] Note that in order to ensure stable convergence during NCE training, additional gradient clipping step was also applied. In combination with drawing noise samples over a much larger output layer, this additional operation led to only a moderate decrease in the training speed from 11.3 k to 6.5 k words per second, when compared with the NCE trained 20 k vocabulary RNNLM in Table X. The relative increase in training time is much lower than that of the output layer vocabulary size, by approximately 40 times.

VII. CONCLUSION

RNNLMs are becoming increasingly important for a range of speech and language processing applications. Several new approaches are presented in this paper to improve both the training and evaluation efficiency for RNNLMs. These include a modified F-RNNLM architecture with a non-class based, full output layer structure; a novel spliced sentence bunch mode parallelization of F-RNNLM training on GPU; VR and NCE based training to specifically reduce the computation associated with the RNNLM output layer softmax normalisation term; and a pipelined training algorithm using multiple GPUs to further improve the training speed. The training time of F-RNNLMs is reduced by up to a factor of 53 on a single GPU over the standard CPU based RNNLM toolkit on a large vocabulary CTS recognition task. A 56 times speed up in test time evaluation on a CPU was obtained over the baseline F-RNNLMs. A further 1.6 times increase of training speed was achieved using pipelined training on 2 GPUs. Improved scalability to larger data sets was also obtained using NCE based RNNLM training. The code and related resources for RNNLM training can be downloaded from http://mi.eng.cam.ac.uk/projects/cued-rnnlm/.

REFERENCES

[1] X. Chen, Y. Wang, X. Liu, M. Gales, and P. C. Woodland, "Efficient training of recurrent neural network language models using spliced sentence bunch," in *Proc. ISCA Interspeech*, 2014, pp. 641–645.

[2] X. Chen, X. Liu, M. Gales, and P. C. Woodland, "Improving the training and evaluation efficiency of recurrent neural network language models," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2015, pp. 5401–5405.

[3] X. Chen, X. Liu, M. Gales, and P. C. Woodland, "Recurrent neural network language model training with noise contrastive estimation for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2015, pp. 5411–5415.

[5] All sources are available in https://code.google.com/p/1-billion-word-language-modeling-benchmark/. The machine translation normalized version of this data was previously used in [54] for RNNLM training.

[6] The log file for perplexity computation is also available in http://mi.eng.cam.ac.uk/projects/cued-rnnlm/ppl.h1024.log

[4] X. Chen, X. Liu, M. Gales, and P. Woodland, “CUED-RNNLM an open-source toolkit for efficient training and evaluation of recurrent neural network language models,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2015, pp. 6000–6004.

[5] Y. Bengiom, R. Ducharme, P. Vincent, and C. Jauvin, “A neural probabilistic language model,” *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, 2003.

[6] H. Schwenk, “Continuous space language models,” *Comput. Speech Lang.*, vol. 21, no. 3, pp. 492–518, 2007.

[7] J. Park, X. Liu, M. J. F. Gales, and P. C. Woodland, “Improved neural network based language modelling and adaptation,” in *Proc. ISCA Interspeech*, 2010, pp. 1041–1044.

[8] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, “Recurrent neural network based language model.” in *Proc. ISCA Interspeech*, 2010, pp. 1045–1048.

[9] M. Sundermeyer, R. Schlüter, and H. Ney, “LSTM neural networks for language modeling,” in *Proc. ISCA Interspeech*, 2012, pp. 194–197.

[10] H.-S. Le, I. Oparin, A. Allauzen, J. Gauvain, and F. Yvon, “Structured output layer neural network language models for speech recognition,” *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 21, no. 1, pp. 197–206, Jan. 2013.

[11] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, “Extensions of recurrent neural network language model,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2011, pp. 5528–5531.

[12] A. Deoras, T. Mikolov, S. Kombrink, M. Karafiát, and S. Khudanpur, “Variational approximation of long-span language models for LVCSR,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2011, pp. 5532–5535.

[13] G. Lecorvé and P. Motlicek, “Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition,” Idiap, Martigny, Switzerland, Tech. Rep. Idiap-RR-21-2012, 2012.

[14] 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 Commun.*, vol. 55, no. 1, pp. 162–177, 2013.

[15] M. Sundermeyer, I. Oparin, J.-L. Gauvain, B. Freiberg, R. Schluter, and H. Ney, “Comparison of feedforward and recurrent neural network language models,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Vancouver, BC, Canada, May 2013, pp. 8430–8434.

[16] Y. Si, T. Li, J. Pan, and Y. 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, pp. 3419–3423.

[17] X. Liu, Y. Wang, X. Chen, M. Gales, and P. C. Woodland, “Efficient lattice rescoring using recurrent neural network language models,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2014, pp. 4908–4912.

[18] Z. Huang, G. Zweig, and B. Dumoulin, “Cache based recurrent neural network language model inference for first pass speech recognition,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2014, pp. 6354–6358.

[19] T. Hori, Y. Kubo, and A. Nakamura, “Real-time one-pass decoding with recurrent neural network language model for speech recognition,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2014, pp. 6364–6368.

[20] W. Williams, N. Prasad, D. Mrva, T. Ash, and T. Robinson, “Scaling recurrent neural network language models,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2015, pp. 5391–5395.

[21] W. De Mulder, S. Bethard, and M.-F. Moens, “A survey on the application of recurrent neural networks to statistical language modeling,” *Comput. Speech and Lang.*, vol. 30, no. 1, pp. 61–98, 2015.

[22] M. Sundermeyer, R. Schlüter, and H. Ney, “From feedforward to recurrent LSTM neural networks for language modeling,” *IEEE/ACM Trans. Audio, Speech Lang. Process.*, vol. 23, no. 3, pp. 517–529, Mar. 2015.

[23] X. Liu, X. Chen, M. Gales, and P. C. Woodland, “Paraphrastic recurrent neural network language models,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2015, pp. 5406–5410.

[24] X. Chen *et al.*, “Recurrent neural network language model adaptation for multigenre broadcast speech recognition,” in *Proc. ISCA Interspeech*, 2015, pp. 6151–6155.

[25] X. Liu, F. Flego, L. Wang, C. Zhang, M. Gales, and P. C. Woodland, “The Cambridge University 2014 BOLT conversational telephone mandarin chinese LVCSR system for speech translation,” in *Proc. ISCA Interspeech*, 2015, pp. 3145–3149.

[26] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu, “Recurrent neural networks for language understanding,” in *Proc. ISCA Interspeech*, 2013, pp. 2524–2528.

[27] G. Mesnil, X. He, L. Deng, and Y. Bengio, “Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding,” in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, 2013, pp. 3771–3775.

[28] M. Auli, M. Galley, C. Quirk, and G. Zweig, “Joint language and translation modeling with recurrent neural networks,” in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2013, pp. 1044–1054.

[29] N. Kalchbrenner and P. Blunsom, “Recurrent continuous translation models,” in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2013, pp. 1700–1709.

[30] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul, “Fast and robust neural network joint models for statistical machine translation,” in *Proc. Assoc. Comput. Linguistics*, 2014, pp. 1370–1380.

[31] A. Hannun *et al.*, “Deep speech: Scaling up end-to-end speech recognition,” 2014, arXiv:1412.5567.

[32] D. Yu *et al.*, “An introduction to computational networks and the computational network toolkit,” *Microsoft Research*, Redmond, WA, USA, Tech. Rep. MSR, 2014. [Online]. Available: http://codebox/cntk

[33] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[34] X. Chen, A. Eversole, G. Li, D. Yu, and F. Seide, “Pipelined back-propagation for context-dependent deep neural networks,” in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, 2012, pp. 26–29.

[35] F. Morin and Y. Bengio, “Hierarchical probabilistic neural network language model,” in *Proc. Int. Workshop Artif. Intell. Stat.*, 2005, pp. 246–252.

[36] 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,” in *Proc. Interspeech*, 2013, pp. 1203–1207.

[37] Z. Huang, G. Zweig, M. Levit, B. Dumoulin, B. Oguz, and S. Chang, “Accelerating recurrent neural network training via two stage classes and parallelization,” in *Proc. IEEE Workshop Automat. Speech Recog. Understanding*, 2013, pp. 326–331.

[38] F. Seide, G. Li, and D. Yu, “Conversational speech transcription using context-dependent deep neural networks,” in *Proc. Interspeech*, 2011, pp. 437–440.

[39] G. Zweig and K. Makarychev, “Speed regularization and optimality in word classing,” in *Proc. IEEE Int. Conf. Spoken Lang. Process.*, 2013, pp. 8237–8241.

[40] H.-K. Kuo, E. Arisoy, A. Emami, and P. Vozila, “Large scale hierarchical neural network language models,” in *Proc. Interspeech*, 2012, pp. 1081–1084.

[41] Y. Shi, W.-Q. Zhang, M. Cai, and J. Liu, “Efficient one-pass decoding with NNLM for speech recognition,” *Signal Process. Lett.*, vol. 21, no. 4, pp. 377–381, 2014.

[42] Y. Shi, W.-Q. Zhang, M. Cai, and J. Liu, “Variance regularization of RNNLM for speech recognition,” in *Proc. IEEE Int. Conf. Spoken Lang. Process.*, 2014, pp. 4893–4897.

[43] A. Sethy, S. Chen, E. Arisoy, and B. Ramabhadran, “Unnormalized exponential and neural network language models,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2015, pp. 5416–5420.

[44] M. U. Gutmann and A. Hyvärinen, “Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics,” *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 307–361, 2012.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-Propagating Errors*. Cambridge, MA, USA: MIT Press, 1988.

[46] A. Emami and L. Mangu, “Empirical study of neural network language models for Arabic speech recognition,” in *Proc. IEEE Workshop Automat. Speech Recog. Understanding*, 2007, pp. 147–152.

[47] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Cernocky, “Recurrent neural network language modeling toolkit,” in *Proc. IEEE Automat. Speech Recog. Understanding Workshop*, 2011, pp. 1–4.

[48] A. Mnih and Y. W. Teh, “A fast and simple algorithm for training neural probabilistic language models,” in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1751–1758.

[49] A. Vaswani, Y. Zhao, V. Fossum, and D. Chiang, “Decoding with large-scale neural language models improves translation,” in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2013, pp. 1387–1392.

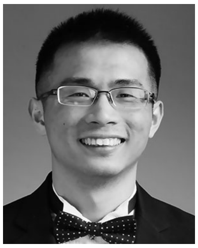
[50] H. Schwenk, “Efficient training of large neural networks for language modelling,” in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2002, pp. 3059–3064.

[51] Y. Shi, W.-Q. Zhang, M. Cai, and J. Liu, “Temporal kernel neural network language modeling,” in *Proc. IEEE Int. Conf. Spoken Lang. Process.*, 2013, pp. 8247–8251.

[52] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, “On parallelizability of stochastic gradient descent for speech DNNs,” in *Proc. IEEE Int. Conf. Spoken Lang. Process.*, 2014, pp. 235–239.
[53] G. Evermann *et al.*, “Training LVCSR systems on thousands of hours of data,” in *Proc. Int. Conf. Spoken Lang. Process.*, 2005, vol. 1, pp. 209–212.
[54] C. Chelba *et al.*, “One billion word benchmark for measuring progress in statistical language modeling,” Google, Mountain View, CA, USA, Tech. Rep., 2013. [Online]. Available: http://arxiv.org/abs/1312.3005
[55] I. Bulyko, M. Ostendorf, and A. Stolcke, “Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures,” in *Proc. North American Chapter Assoc. Comput. Linguistics, Human Language Technol.*, 2003, pp. 7–9.
[56] A. Stolcke *et al.*,“SRILM-an extensible language modeling toolkit,” in *Proc. Int. Conf. Spoken Lang. Process.*, 2002, pp. 901–904.

Xie Chen (M’12) received the Bachelor’s degree from Xiamen University, Xiamen, China, in 2009, and the M.Phil. degree in electronic engineering from Tsinghua University, Beijing, China, in 2012. He is currently working toward the Ph.D. degree on automatic speech recognition supervised by Prof. M. Galesat the Machine Intelligence Laboratory. He joined the Cambridge University Engineering Department in 2012. His current research interests include large vocabulary continuous speech recognition, meeting transcription, and language modeling. He is a Student Member of ISCA.

Xunying Liu (M’06) received the Bachelor’s degree from Shanghai Jiao Tong University, Shanghai, China, the M.Phil. degree in computer speech and language processing, and the Ph.D. degree in speech recognition, both from the University of Cambridge, Cambridge, U.K. He has been a Senior Research Associate at the Machine Intelligence Laboratory, Cambridge University Engineering Department, and from 2016 an Associate Professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong, Shatin, Hong Kong. He received the best paper award at the ISCA Interspeech 2010. His current research interests include large vocabulary continuous speech recognition, language modelling, noise robust speech recognition, speech synthesis, speech and language processing. He is a Member of ISCA.



Yongqiang Wang (M’08) received the B.Eng. degree in electronic engineering from the University of Science and Technology of China, Hefei, China, the M.Phil. degree in computer science from the University of Hong Kong, Pok Fu Lam, Hong Kong, and the Ph.D. degree in engineering from the University of Cambridge, Cambridge, U.K. He is currently working as a Senior Speech Scientist in Microsoft, Redmond, WA, USA. His research interest includes robust speech recognition and large-scale deep learning.

Mark J. F. Gales (F’11) received the B.A. degree in electrical and information sciences from the University of Cambridge, Cambridge, U.K., in 1988. Following graduation he worked as a Consultant at Roke Manor Research Ltd. In 1991, he worked as a Research Associate in the Speech Vision and Robotics group in the Engineering Department, Cambridge University. In 1995, he completed his doctoral thesis: Model-Based Techniques for Robust Speech Recognition supervised by Prof. S. Young. From 1995 to 1997, he was a Research Fellow at Emmanuel College Cambridge. He was then a Research Staff Member in the Speech group at the IBM T.J. Watson Research Center until 1999 when he returned to Cambridge University, Engineering Department as a University Lecturer. He was appointed a Reader in Information Engineering in 2004. He is currently a Professor of Information Engineering and a College Lecturer and Official Fellow of Emmanuel College. He is a Member of the Speech and Language Processing Technical Committee (2015–2017, previously a member from 2001–2004). He was an Associate Editor for IEEE SIGNAL PROCESSING LETTERS from 2008 to 2011 and IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSINGfrom 2009 to 2013. He is currently on the Editorial Board of *Computer Speech and Language*. He has been awarded a number of paper awards, including the 1997 IEEE Young Author Paper Award for his paper on Parallel Model Combination and the 2002 IEEE Paper Award for his paper on Semi-Tied Covariance Matrices.

Philip C. Woodland (F’13) is a Professor of Information Engineering at the University of Cambridge Engineering Department, where he leads the Speech Group, and a Professorial Fellow of Peterhouse. He has published almost 200 papers in speech technology, including the most cited paper in *Computer Speech and Language*. He developed techniques for speaker adaptation and discriminative training that have now become standard in speech recognition. His research team developed speech recognition systems which have frequently been the most accurate in international research evaluations organised by the U.S. Government. He is well known as one of the original coauthors of the widely used HTK toolkit and has continued to play a major role in its development. He has been a Member of the editorial board of *Computer Speech and Language* (1994–2009), and a current editorial board member of *Speech Communication*. One of his current major interests is developing flexible systems that can adapt to a wide range of speakers, acoustic conditions, and speaking styles with relatively limited training resources. His current work also includes techniques for improved language modelling and confidence estimation. An increasing trend in his work is the use of deep neural networks for both acoustic models and language models. He is a Fellow of ISCA.