LIMITED-MEMORY BFGS OPTIMIZATION OF RECURRENT NEURAL NETWORK LANGUAGE MODELS FOR SPEECH RECOGNITION

Xunying Liu[1]*, Shansong Liu*[1]*, Jinze Sha*[2]*, Jianwei Yu*[1]*, Zhiyuan Xu*[2]*, Xie Chen*[2] *& Helen Meng*[1]

[1]Department of Systems Engineering and Engineering Management,
The Chinese University of Hong Kong, Hong Kong SAR, China
[2]Cambridge University Engineering Department
Email: {*xyliu,ssliu,jwyu,hmmeng*}*@se.cuhk.edu.hk* {*js2294,zyx22,xc257*}*@cam.ac.uk*

ABSTRACT

Recurrent neural network language models (RNNLM) have become an increasingly popular choice for state-of-the-art speech recognition systems. RNNLMs are normally trained by minimizing the cross entropy (CE) using the stochastic gradient descent (SGD) algorithm. The SGD method only uses first-order derivatives and no higher order gradient information is used to consider the correlation between model parameters. It is unable to fully capture the curvature of the error cost function. This can lead to slow convergence in model training. In this paper, a limited-memory Broyden Fletcher Goldfarb Shannon (L-BFGS) based second order optimization technique is proposed for RNNLMs. This method efficiently approximates the matrix-vector product between the inverse Hessian and gradient vector via a recursion over past gradients with a compact memory requirement. Consistent perplexity and error rate reductions are obtained over the SGD method on two speech recognition tasks: Switchboard English and Babel Cantonese. A faster convergence and speed up in RNNLM training time was also obtained.

Index Terms: recurrent neural network, language model, second order optimization, limited-memory BFGS, speech recognition

1. INTRODUCTION

In order to handle the data sparsity problem associated with back-off n-gram language models (LMs), language modelling techniques that represent preceding history contexts in a continuous and lower dimensional vector space, such as neural network language models (NNLMs) [1, 28, 24, 21, 29, 16, 30], can be used. Depending on the network architecture, NNLMs can be categorized into two major types: feedforward NNLMs [1, 28, 24, 16], which use a vector representation of preceding contexts of a fixed number of words, and recurrent NNLMs (RNNLMs) [21, 22, 29, 30], which use a recurrent vector representation of variable length full histories. In recent years RNNLMs and long-short term memory (LSTM) variants [29, 30] have been shown to define state-of-the-art language modelling performance on a wide range of tasks and give significant improvements over conventional back-off n-gram LMs, thus drawing increasing research interest [21, 22, 29, 10, 14].

Standard RNNLM training is normally based on minimize the cross entropy (CE) of the training data using the stochastic gradient descent (SGD) algorithm [2]. Like other gradient descent based techniques, the SGD method only uses first-order derivatives. No higher order gradient information [12, 17, 3, 20, 15, 8] is used to consider the correlation between model parameters and therefore can not fully capture the curvature of the error cost function. This can leads to unstable and slow convergence in model training. One general approach to address this issue is to use a quadratic approximation to the error cost function using Newton methods. However, for tasks with a large number of parameters to estimate, explicitly computing the Hessian matrix and its inverse for Newton methods is problematic.

In order to address this issue, truncated Newton methods based on Hessian-free optimization [20, 15] has been proposed and successfully applied to speech recognition tasks [15, 8]. Instead of directly computing the Hessian matrix and its inverse, the product between the inverse Hessian and the gradient vector is approximated using an iterative conjugate gradient (CG) based approach. The Gauss-Newton matrix based curvature approximate is guaranteed to positive semidefinite when additional damping is applied [20].

In this paper, an alternative efficient second order optimization method based on the limited-memory Broyden Fletcher Goldfarb Shannon (L-BFGS) [17, 3] algorithm is proposed for RNNLM training. The L-BFGS method efficiently approximates the matrix-vector product between the inverse Hessian and gradient vector via a recursion over past gradients instead. It only requires a few vectors representing a history of the past m updates of this matrix-vector product to be stored. Due to its compact memory requirement, the L-BFGS method is well suited for optimization problems such as RNNLMs, where a large number of parameters are estimated.

The rest of the paper is organized as follows. Sections 2 and 3 reviews the RNNLM model architecture and SGD based training algorithm. Section 4 proposes the L-BFGS based second order optimization scheme for RNNLMs. Experimental results on large vocabulary speech recognition tasks are presented in section 5. Section 6 is the conclusion and future work.

2. RECURRENT NEURAL NETWORK LMS

In contrast to feedforward NNLMs, recurrent NNLMs [21] represent the full, non-truncated history $h_1^{i-1} = <w_{i-1}, \ldots, w_1>$ for word w_i using the 1-of-k encoding of the most recent preceding word w_{i-1} and a continuous vector $\boldsymbol{v}_{i-2}$ for the remaining context. For an empty history, this is initialized, for example, to a vector of all ones. The topology of the recurrent neural network used to compute LM probabilities $P_{\mathsf{RNN}}(w_i|w_{i-1}, \boldsymbol{v}_{i-2})$ consists of three layers, as is shown in figure 1. The full history vector, obtained by concatenating the those of 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 activa-

This research was supported by MSRA grant no. 6904412 and Chinese University of Hong Kong (CUHK) grant no. 4055065. Jinze Sha and Zhiyuan Xu were under an Industrial Placement scheme hosted by CUHK.

tion to achieve non-linearity. In order to address vanishing gradient during RNNLM training, more complicated forms of recurrent layer activation based on gated recurrent units (GRU) [7] and long short-term memory (LSTM) units [13, 29] can also be used. This hidden layer output vector 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 $P_{\mathsf{RNN}}(w_{i+1}|w_i, \boldsymbol{v}_{i-1})$.

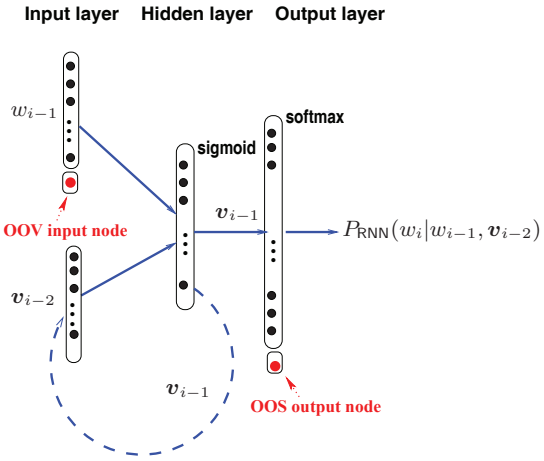

Fig. 1. *A full output layer RNNLM with OOS nodes.*

To reduce computational cost, a shortlist based output layer vocabulary limited to the most frequent words can be used. In order to reduce the bias to in-shortlist words during NNLM training, It is necessary to explicitly model the probability mass of out-of-shortlist (OOS) words using an additional output node [24, 16]. This ensures that all training data are used in training, and the probabilities of in-shortlist words are smoothed by the OOS probability mass to obtain a more robust parameter estimation. The full output RNNLMs with OOS nodes is shown in figure 1 is used in this paper.

In state-of-the-art speech recognition systems, NNLMs are often linearly interpolated with n-gram LMs to obtain both a good coverage of contexts and strong generalization ability [28, 11, 24, 21, 29, 16]. The interpolated LM probability is given by

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

λ is the weight assigned to the back-off n-gram LM $P_{\mathsf{NG}}(\cdot)$, and kept fixed as 0.5 in all experiments of this paper. In the above interpolation, the probability mass of OOS words assigned by the RNNLM component needs to be re-distributed among all OOS words [24, 16].

3. RNNLM TRAINING USING SGD

Conventional RNNLM training maximizes the log-likelihood, or equivalently minimize the cross entropy (CE) of the training data. For a given sequence containing a total of N_w words, the objective function is given by

$$J^{\mathsf{CE}}(\boldsymbol{\theta}) \;=\; -\frac{1}{N_w}\sum_{i=1}^{N_w} \ln P_{\mathsf{RNN}}(w_i|h_i) \quad (2)$$

where $P_{\mathsf{RNN}}(w_i|h_i) \;=\; f_{\mathsf{softmax}}(\boldsymbol{v}_{i-1};\boldsymbol{\theta})$ is the probability of word w_i given history h_i. $\boldsymbol{\theta}$ is the output layer weight matrix, and $\boldsymbol{v}_{i-1}$ is the hidden history vector computed at the hidden layer.

The conventional stochastic gradient descent (SGD) algorithm is normally used in CE training. For the $(t+1)$th randomly selected minibatch of N_w words in total within each training epoch, the gradient statistics accumulated over the minibatch are scaled by a tunable learning rate η before being used to update the weight parameters, for example, at the output layer as,

$$\boldsymbol{\theta}[t+1] \;=\; \boldsymbol{\theta}[t] - \eta \left.\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[t]} \quad (3)$$

The gradient statistics in the above are computed as

$$\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \;=\; -\frac{1}{N_w}\sum_{i=1}^{N_w} \boldsymbol{v}_i \boldsymbol{\xi}_i^{\top} \quad (4)$$

where the jth element of the output layer error cost vector $\boldsymbol{\xi}_i$ is $\boldsymbol{\xi}_{i,j} = \delta(w_j|h_i) - P_{\mathsf{RNN}}(w_j|h_i)$, and $\delta(w_j|h_i)$ are the binary coded target probabilities $\delta(w_i|h_i) = 1$ and $\delta(w_j|h_i) = 0, \forall j \neq i$.

Further propagating back the errors to the recurrent layers and input layers lead to their respective gradient statistics. For Sigmoid activation based recurrent layer weight matrix $\boldsymbol{\zeta}$, these are

$$\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\zeta})}{\partial \boldsymbol{\zeta}} \;=\; -\frac{1}{N_w}\sum_{i=1}^{N_w} \boldsymbol{v}_{i-1} \left(\boldsymbol{\xi}_i \odot \boldsymbol{u}_i\right)^{\top} \quad (5)$$

where $\boldsymbol{u}_{i,j} = \boldsymbol{v}_{i,j}(1-\boldsymbol{v}_{i,j})$ and $\odot$ denotes an element wise multiplication. Similarly the gradient statistics associated with the input layer weight matrix $\boldsymbol{\phi}$ are computed as

$$\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \;=\; -\frac{1}{N_w}\sum_{i=1}^{N_w} \tilde{\boldsymbol{w}}_i \left(\boldsymbol{\xi}_i \odot \boldsymbol{u}_i\right)^{\top} \quad (6)$$

where $\tilde{\boldsymbol{w}}_i$ is the vector encoding of the input word w_i. For the recurrent and input layers weight parameters $\boldsymbol{\zeta}$ and $\boldsymbol{\phi}$, the same above SGD update in equation (3) is also used where the comparable gradient statistics are computed respectively using equations (5) and (6).

RNNLMs can be trained using an extended form of the standard back propagation algorithm, back propagation through time (BPTT) [27]. This requires the above gradient statistics for the recurrent and input layers to be accumulated further over a finite number of N_τ steps back in time, equivalent to unfolding the RNNLM as a feed forward neural network with N_τ hidden layers. For example, the BPTT gradient statistics for the recurrent layer weights are

$$\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\zeta})}{\partial \boldsymbol{\zeta}} \;=\; -\frac{1}{N_w}\sum_{i=1,\tau=1}^{N_w,N_\tau} \boldsymbol{v}_{i-\tau-1} \left(\boldsymbol{\xi}_{i-\tau} \odot \boldsymbol{u}_{i-\tau}\right)^{\top} \quad (7)$$

4. RNNLM TRAINING USING L-BFGS

4.1. Newton methods

In contrast to Gradient descent, Newton methods explicitly consider higher order gradient information to model the correlation between model parameters using a quadratic approximation to the error cost function. The derived ***Newton direction*** is then used in parameter update. For example, the CE objective function used in RNNLM

training at the kth minibatch can be approximated via a second order Taylor series with respect to the output layer weight parameters as

$$J^{\mathsf{CE}}(\boldsymbol{\theta}[t]+\Delta\boldsymbol{\theta}) \approx J^{\mathsf{CE}}(\boldsymbol{\theta}[t]) + \Delta\boldsymbol{\theta}^{\top} \left.\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[t]} + \frac{1}{2}\Delta\boldsymbol{\theta}^{\top}\mathbf{H}_t\Delta\boldsymbol{\theta} \quad (8)$$

The gradient of the above is given by

$$\nabla_{\boldsymbol{\theta}} J^{\mathsf{CE}}(\boldsymbol{\theta}[t]+\Delta\boldsymbol{\theta}) = \left.\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[t]} + \mathbf{H}_t\Delta\boldsymbol{\theta} \quad (9)$$

and setting it to zero leads to the Newton direction

$$\Delta\boldsymbol{\theta} = -\mathbf{H}_t^{-1} \left.\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[t]} \quad (10)$$

where the Hessian matrix is computed as $\mathbf{H}_{t,i,j} = \left.\frac{\partial^2 J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}_i\partial\boldsymbol{\theta}_j}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[t]}$.

4.2. Quasi-Newton methods

In practice directly applying Newton methods to tasks with a large number of model parameters can be problematic. First, the evaluation of the Hessian matrix of $\mathcal{O}(N^2)$ parameters and its inversion of $\mathcal{O}(N^3)$ complexity are both computationally expensive. Second, when the Hessian is not positive-definite, the resulting Newton direction may not head towards a desired minimum, but rather towards a opposite direction or saddle point.

In order to address these issues, two types of techniques that do not require direct calculation of the Hessian can be used. The first type are based on truncated Newton methods based on Hessian-free optimization [20, 15]. Instead of directly computing the Hessian matrix and its inverse, the matrix-vector product in equation (10) is is approximated using an iterative conjugate gradient (CG) based approach to access the parameter curvature. The conjugate gradient search is truncated and based on the relative improvement in the approximate error cost function. The parameter curvature is based on the Gauss-Newton matrix [20], which is guaranteed positive semidefinite when additional damping is applied. Hessian-free optimization can be parallelized for neural networks and have been successfully applied to speech recognition tasks [15, 8].

The second category of techniques are based on Quasi-Newton methods. These methods allow the inverse Hessian matrix to be approximated instead by recursively analyzing past gradient vectors. An earlier form of these techniques was based on the Davidon Fletcher Powell (DFP) [12] algorithm, before being further developed into more advanced forms including the symmetric rank one (SR1) [9] method, and the widely used Broyden Fletcher Goldfarb Shannon (BFGS) [12] algorithm.

4.3. L-BFGS method for RNNLM training

The standard BFGS algorithm requires a full matrix approximation to the inverse Hessian of $\mathcal{O}(N^2)$ parameters. For optimization problems with a large number of model parameters to train, this approach becomes computationally highly expensive. In order to address this issue, a low memory extension to the standard BFGS algorithm, limited-memory BFGS (L-BFGS) method, can be used. In contrast to the standard BFGS algorithm that approximates the inverse Hessian directly via a recursion over past gradients, the L-BFGS method approximates the matrix-vector product between the inverse Hessian and gradient vector in equation (10). Only a few vectors representing a history of the past m updates of such matrix-vector product need to be stored. The gradient history size m can be set small as $m < 10$. In this paper, the setting $m = 5$ is used throughout the experiments.

Algorithm 1 For RNNLM output layer weights $\boldsymbol{\theta}$, L-BFGS algorithm approximates inverse Hessian gradient matrix-vector product

$\boldsymbol{q}_t \leftarrow \left.\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[t]}$
for i = t-1, t-2, ..., t-m **do**
  $\boldsymbol{s}_i \leftarrow \boldsymbol{\theta}[i+1] - \boldsymbol{\theta}[i]$, $\boldsymbol{y}_i \leftarrow \boldsymbol{q}_{i+1} - \boldsymbol{q}_i$
  $\rho_i \leftarrow \frac{1}{\boldsymbol{y}_i^{\top}\boldsymbol{s}_i}$, $\alpha_i \leftarrow \rho_i \boldsymbol{s}_i^{\top}\boldsymbol{q}_t$
end for
$\boldsymbol{B}_t^0 \leftarrow \frac{\boldsymbol{y}_{t-m}\boldsymbol{s}_{t-m}^{\top}}{\boldsymbol{y}_{t-m}^{\top}\boldsymbol{y}_{t-m}}$, $\boldsymbol{z} \leftarrow \boldsymbol{B}_t^0\boldsymbol{q}_t$
for i = t-m, t-m+1, ..., t-1 **do**
  $\beta_i \leftarrow \rho_i \boldsymbol{y}_i^{\top}\boldsymbol{z}$, $\boldsymbol{z} \leftarrow \boldsymbol{z} + (\alpha_i - \beta_i)\boldsymbol{s}_i$
end for
$\mathbf{H}_t^{-1}\left.\frac{\partial J^{\mathsf{CE}}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}[t]} \leftarrow \boldsymbol{z}$

For example, When updating the RNNLM output layer weight parameters $\boldsymbol{\theta}$ using the L-BFGS method at the $(t+1)$th step, the inverse Hessian and gradient vector in equation (10) is recursively approximated using the above algorithm shown in (1). The inverse Hessian and gradient matrix-vector product obtained in the final step of the above algorithm is then used to replace the standard gradient used in the SGD update of equation (3).

4.4. Efficient GPU based training parallelization

The L-BFGS RNNLM training algorithm proposed in this paper is implemented as an extension to the publicly available **CUED-RNNLM** toolkit [5]. In order to improve training efficiency, the above L-BFGS RNNLM training scheme is integrated into an efficient bunch mode GPU parallelization algorithm based on spliced sentences [4, 6]. This GPU based parallelization approach was originally for the SGD method and modified in this work by replacing the standard gradient information used in SGD in equation (3) by the inverse Hessian and gradient matrix-vector product produced by the above L-BFGS algorithm. Each stream in the bunch contains a sequence of concatenated sentences to minimize synchronization overhead among parallel streams. Sentences in the training corpus are joined into streams that are more comparable in length. Individual sentence boundaries within each stream are marked to appropriately reset the recurrent history vector as required. CUBLAS from CUDA 8.0, the basic linear algebra subprograms (BLAS) library optimized for Nvidia GPUs, is used for fast matrix operation.

5. EXPERIMENTS AND RESULTS

In this section the performance of RNNLMs trained using the proposed L-BFGS algorithm are evaluated on two large vocabulary speech recognition tasks: the Switchboard English system with 300 hour of conversational telephone speech from Switchboard I for acoustic modelling, 3.6M words of acoustic transcripts for language modelling and a 30k words lexicon; the Babel Cantonese system with 175 hours of telephony speech from the full language pack

LDC release, 1.1M words of transcripts for language modelling and a 25k words vocabulary.

For both systems, MPE trained stacked hybrid DNN-HMM acoustic models constructed using the HTK toolkit version 3.5 [31] are used. Both the top and bottom level DNNs contain 6 Sigmoid activation based hidden layers of 2000 nodes of each, except the 2nd last bottleneck (BN) layer in the bottom level DNN contains 39 nodes used to produce BN features. Both systems' bottom level DNNs were CE trained using 40 dimensional log Mel-filter bank (FBK) features spliced over a 9 frame widow covering 4 frames to left and right as the input, and decision tree clustered triphone state labels as the output layer targets. A total of 12k tied states were used in the Switchboard system, and 6k tied states were used in the Cantonese system. The resulting 39 dimensional BN features extracted were then diagonalized using a global STC transform before augmented with 39 dimensional HLDA project PLP features. For the Cantonese system, these were further augmented with static and differential pitch parameters derived using the Kaldi toolkit [25] up to the 2nd order. These tandem features were then used as the inputs to the top level DNN network with the same input feature context window, hidden layer architecture, and output layer targets as the bottom level DNNs except all its hidden layers have 2000 nodes. The top level DNNs were initially CE trained before being further MPE [26] trained. All RNNLMs used 512 hidden layer nodes and Sigmoid activation. These were trained using either SGD with newbob scheduling or L-BFGS optimization via the extended **CUED-RNNLM** toolkit on NVidia K40 GPUs. These were used for perplexity and error rate evaluation by rescoring lattices using a n-gram truncated history based approximation [18].

During training, the proposed L-BFGS optimization was found to converge much faster than the SGD approach. On the Cantonese setup as is shown in Fig. 2, for example, the SGD based training required 16 epochs of 1453 seconds in total, while the L-BFGS training converged much faster after 6 epochs using only 767 seconds, thus almost halving the SGD training time. L-BFGS also produced validation data entropy consistently lower than SGD at every epoch.

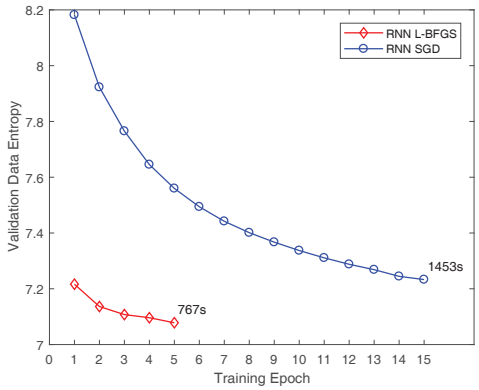

Fig. 2. *Validation entropy, total training time on Babel Cantonese.*

The perplexity and WER performance of various RNNLMs are shown in table 1 for the Switchboard English task. In the first section of the table, using the L-BFGS optimization technique in RNNLM training, consistent perplexity reductions of 3.5 to 8.5 points and WER reductions of 0.7% absolute were obtained using the L-BFGS trained RNNLM (line 2) alone or its equal weighted interpolation

LM	PPL	WER% swbd	WER% chm
rnn.SGD	104.3	13.9	26.1
rnn.LBFGS	100.8	13.2	25.9
rnn.SGD+rnn.LBFGS	**95.8**	**13.2**	**25.4**
4-gram	97.2	12.9	25.4
4-gram+rnn.SGD	87.0	12.6	24.8
4-gram+rnn.LBFGS	87.7	12.4	24.7
4-gram+rnn.SGD+rnn.LBFGS	**85.9**	**12.4**	**24.6**

Table 1. Perplexity and WER performance of RNNLMs on 2hr **swbd** and 1.5hr **callhm** subsets of NIST **eval00** test set.

with the SGD trained baseline RNNLM (line 3) over the SGD method (line 1). In the second section of table 1, further combining the 2-way interpolated "rnn.SGD+rnn.LBFGS" model (line 3) with the 4-gram back-off LM (line 7) using an equal weighted interpolation, reduced but consistent perplexity improvements and WER reductions of 0.2% absolute were obtained across both test sets over the baseline "4-gram+rnn.SGD" interpolated model (line 5) constructed by an equal weighted interpolation between the 4-gram back-off LM and the SGD trained baseline RNNLM.

The same trends can also be found on a comparable set of experiments conducted on the Babel Cantonese system, as are shown in in table 2. Before the interpolation with 4-gram LM, the L-BFGS RNNLMs (line 2 and 3) outperformed the SGD trained baseline RNNLM by 9 to 17 points of perplexity improvements, and character error rate (CER) reductions of 0.8%-0.9% absolute. After the equal weighted interpolation with the 4-gram LM, small but consistent perplexity improvements of 2 points and CER reductions of 0.2%-0.3% were obtained on both test sets over the baseline "4-gram+rnn.SGD" interpolated model (comparing lines 5 and 7).

LM	PPL	CER% devsub1	CER% devsub2
rnn.SGD	136.5	43.5	44.1
rnn.LBFGS	127.9	42.7	43.3
rnn.SGD+rnn.LBFGS	**119.7**	**42.6**	**43.2**
4-gram	113.7	42.1	42.7
4-gram+rnn.SGD	106.8	42.0	42.6
4-gram+rnn.LBFGS	106.2	41.8	42.4
4-gram+rnn.SGD+rnn.LBFGS	**104.8**	**41.8**	**42.3**

Table 2. Perplexity and CER performance of RNNLMs on 3hr **devsub1** and 7hr **devsub2** subsets of Babel Cantonese development set.

6. CONCLUSION AND RELATION TO PRIOR WORK

In this paper, an efficient limited-memory Broyden Fletcher Goldfarb Shannon (L-BFGS) based 2nd order optimization method is proposed for training recurrent neural network language models. Experimental results on two speech recognition tasks suggest the proposed technique is useful to improve both the performance and training speed for RNNLMs. To the best of our knowledge, this is the first work using L-BFGS optimization for RNNLMs. Future work will focus on L-BFGS training of more complicated forms of RNNLMs and improving the parallelization method.

7. REFERENCES

[1] Y. Bengio and R. Ducharme (2003), "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[2] L. Bottou (1991). "Stochastic gradient learning in neural networks", in *Proc. Neuro-Nimes*.

[3] R. .H. Byrd, P. Lu, J. Nocedal, and C. Zhou (1995). "A limited memory algorithm for bound constrained optimization", *SIAM Journal on Scientific and Statistical Computing*, 16(5).

[4] X. Chen, Y. Wang, X. Liu, M. J. F. Gales and P. C. Woodland (2014). "Efficient GPU-based training of recurrent neural network language models using spliced sentence bunch", in *Proc. ISCA Interspeech2014*, Singapore, pp. 641–645.

[5] X. Chen, X. Liu, Y. Qian, M. J. F. Gales, and P. C. Woodland (2016), "CUED-RNNLM - an open source toolkit for efficient training and evaluation of recurrent neural network language models," in *Proc. IEEE ICASSP*, Shanghai, pp. 6000–6004.

[6] X. Chen, X. Liu, Y. Wang, M. J. F. Gales, and P. C. Woodland (2016), "Efficient Training and Evaluation of Recurrent Neural Network Language Models for Automatic Speech Recognition," *IEEE/ACM Trans. Audio, Speech & Lang. Proc.* 24(11): 2146-2157.

[7] K. Cho, B. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014), "Learning phrase representations using RNN encoder-decoder for statistical machine translation", in *Proc. EMNLP*, pp. 1724–1734.

[8] I.-H. Chung, T. N. Sainath, B. Ramabhadran, M. Picheny, J. A. Gunnels, V. Austel, U. Chauhari, and B. Kingsbury (2017). "Parallel deep neural network training for big data on Blue Gene/Q", *IEEE Trans. Parallel Distrib. Syst.* 28(6): 1703–1714.

[9] A. R. Conn, N. I. M. Gould, and P. .L. Toint (1991). "Convergence of quasi-Newton matrices generated by the symmetric rank one update", *Mathematical Programming*, Springer Berlin/Heidelberg. 50 (1): 177–195.

[10] A. Deoras, T. Mikolov, S. Kombrink and K. Church (2013), "Approximate inference: A sampling based modeling technique to capture complex dependencies in a language model," *Speech Communication*, vol. 55, no. 1, pp. 162–177, January.

[11] A. Emami and L. Mangu (2007), "Empirical study of neural network language models for Arabic speech recognition," in *Proc. ASRU*, Kyoto, Japan, 2007, pp. 147–152.

[12] R. Fletcher (1987). "Practical methods of optimization (2nd edition)", New York: John Wiley & Sons, ISBN 978-0-471-91547-8.

[13] S. Hochreiter, and J. Schmidhuber (1997), "Long short-term memory," *Neural Computation* 9(8): pp. 1735–1780.

[14] Z. Huang, G. Zweig, and B. Dumoulin (2014). " Cache based recurrent neural network language model inference for first pass speech recognition", in *Proc. IEEE ICASSP2014*, Florence, Italy, pp. 6404–6408.

[15] B. Kingsbury, T. N. Sainath, and H. Soltau (2012), "Scalable minimum bayes risk training of deep neural network acoustic models using distributed Hessian-free optimization", in *Proc. ISCA Interspeech*, Portland, Oregon, USA, pp. 10–13.

[16] H.-S. Le, I. Oparin, A. Allauzen, J. Gauvain, and F. Yvon (2013), "Structured output layer neural network language models for speech recognition," *IEEE Trans. on Audio, Speech & Lang. Proc.*, vol. 21, no. 1, pp. 197–206, 2013.

[17] D. C. Liu, and J. Nocedal (1989). "On the mimited memory method for large scale optimization", *Mathematical Programming B*, 45 (3): 503–528.

[18] X. Liu, Y. Wang, X. Chen, M. J. F. Gales and P. C. Woodland (2014). "Efficient lattice rescoring using recurrent neural network language models", in *Proc. IEEE ICASSP2014*, Florence, Italy.

[19] X. Liu, X. Chen, Y. Wang, M. J. F. Gales, and P. C. Woodland (2016). "Two efficient lattice rescoring methods using recurrent neural network language models," *IEEE/ACM Trans. Audio, Speech & Lang. Proc.* 24(8): 1438-1449.

[20] J. Martens (2010), "Deep learning via Hessian-free optimization," in *Proc. ICML*, Haifa, Israel, pp. 735–742.

[21] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur (2010), "Recurrent neural network based language model," in *Proc. ISCA Interspeech*, Makuhari, Japan, 2010, pp. 1045–1048.

[22] T. Mikolov, S. Kombrink, L. Burget, J. H. Cernocky, and S. Khudanpur (2011), "Extensions of recurrent neural network language model," in *Proc. ICASSP*, Prague, Czech Republic, 2011, pp. 5528–5531.

[23] F. Morin and Y. Bengio (2005), "Hierarchical probabilistic neural network language model," in *Proc. International workshop on artificial intelligence and statistics*, Barbados, 2005, pp. 246–252.

[24] J. Park, X. Liu, M. J. F. Gales, and P. C. Woodland (2010), "Improved neural network based language modelling and adaptation," in *Proc. ISCA Interspeech*, Makuhari, Japan, 2010, pp. 1041–1044.

[25] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz et al. (2011). "The kaldi speech recognition toolkit", in *IEEE ASRU*.

[26] D. Povey and P. C. Woodland (2002). "Minimum phone error and I-smoothing for improved discriminative training", in *Proc. IEEE ICASSP*, Orlando, Florida, USA, vol. 1 105–108.

[27] D. E. Rumelhart, G. E. Hintont, and R. J. Williams (1986), "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536.

[28] H. Schwenk (2007) , "Continuous space language models," *Computer Speech & Language*, 21(3): 492-518.

[29] M. Sundermeyer, R. Schlüter, and H. Ney (2012), "LSTM neural networks for language modeling," in *Proc. ISCA Interspeech*, Portland, OR, 2012.

[30] M. Sundermeyer, H. Ney, and R. Schlüter (2015), "From feedforward to recurrent LSTM neural networks for language modeling," *IEEE/ACM Transactions on Audio, Speech & Language Processing*, vol. 23, no. 3, pp. 517–529.

[31] 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 Version 3.5a", http://htk.eng.cam.ac.uk, 2015.