SCALABLE TRAINING OF DEEP LEARNING MACHINES BY INCREMENTAL BLOCK TRAINING WITH INTRA-BLOCK PARALLEL OPTIMIZATION AND BLOCKWISE MODEL-UPDATE FILTERING

Kai Chen[1,2*], *Qiang Huo*[2]

[1]University of Science and Technology of China, Hefei, China
[2]Microsoft Research, Beijing, China
{v-kachen, qianghuo}@microsoft.com

ABSTRACT

We present a new approach to scalable training of deep learning machines by incremental block training with intra-block parallel optimization to leverage data parallelism and blockwise model-update filtering to stabilize learning process. By using an implementation on a distributed GPU cluster with an MPI-based HPC machine learning framework to coordinate parallel job scheduling and collective communication, we have trained successfully deep bidirectional long short-term memory (LSTM) recurrent neural networks (RNNs) and fully-connected feed-forward deep neural networks (DNNs) for large vocabulary continuous speech recognition on two benchmark tasks, namely 309-hour Switchboard-I task and 1,860-hour "Switchboard+Fisher" task. We achieve almost linear speedup up to 16 GPU cards on LSTM task and 64 GPU cards on DNN task, with either no degradation or improved recognition accuracy in comparison with that of running a traditional mini-batch based stochastic gradient descent training on a single GPU.

Index Terms— Incremental block training, parallel optimization, blockwise model-update filtering, deep learning, LVCSR

1. INTRODUCTION

In the past several years, deep learning machines, including fully-connected feed-forward deep neural networks (DNNs), convolutional neural networks (CNNs), deep (bidirectional) long short-term memory (LSTM) recurrent neural networks (RNNs) and their variants, have become new state-of-the-art solutions for applications such as large vocabulary continuous speech recognition (LVCSR) (e.g., [1–11] and the references therein), image classification (e.g., [12, 13]) and handwriting recognition (HWR) (e.g., [14–16]). So far, the most popular method to train deep learning machines remains mini-batch based stochastic gradient descent (SGD) with error back propagation [17] and momentum tricks [18], which is known to be difficult to parallelize over large-scale CPU or GPU cluster for its serial nature. Given the exciting success of deep learning machines, one of the most important research problems is how to scale out deep learning to leverage big data and further improve recognition accuracy, which is also the topic of this paper.

In addition to our work reported in this study, there are several successful efforts in scaling out deep learning (e.g., [19–23]). The most visible one is a software framework called DistBelief developed by Google [19] that can utilize computing clusters with thousands of machines to train different models with an asynchronous SGD (ASGD) procedure (a.k.a., Hogwild [24]). "ASGD in DistBelief" has been used successfully to train DNNs and deep LSTMs for LVCSR with both frame-level maximum cross-entropy (CE) criterion and sequence-level maximum mutual information (MMI) criterion (e.g., [8, 9, 19, 25, 26]). However there is no comparison with the standard mini-batch SGD, therefore it is not clear yet whether "ASGD in DistBelief" incurs any loss of recognition accuracy. In [27], ASGD has also been used to parallelize DNN training on multiple GPU cards in a single computing server, which achieves a 3.2 times speedup on 4 GPUs than on 1 GPU without the degradation of recognition accuracy.

Another promising approach to scaling out deep learning is to use the idea of model averaging (MA, e.g., [28, 29]) that first solves the learning problem independently on each worker using the portion of data stored on that worker, and then averages the independent local solutions to obtain a global solution. Although almost linear speedup can be achieved in terms of the throughput of processing training data, this approach incurs recognition accuracy degradation compared with single-worker scheme, especially when the number of workers increases (e.g., [21, 30]).

There are also several recent successful efforts to parallelize standard mini-batch SGD on multiple GPU cards. In [31], very large DNN acoustic models are trained by leveraging model parallelism on a GPU cluster using a software framework described in [20]. In [23], by using a 1-bit quantization technique to compress gradients aggressively to reduce significantly data-exchange bandwidth, the feasibility of running SGD on a GPU cluster by exploiting data parallelism within a mini-batch is demonstrated. For example, the parallel training solution in [23] achieves 6.9 times speedup with 20 GPUs than on a single GPU with little degradation of recognition accuracy. A follow-up work in [32] improves scalability for training DNNs by applying 1-bit quantization on those significant enough gradients. Yet a large mini-batch SGD method with tied scalar regularization is proposed in [33] to train DNNs with rectified linear units (ReLUs) and achieves promising results.

In this paper, we propose a new approach to scaling out training of deep learning machines by incremental block training with intra-block parallel optimization to leverage data parallelism and blockwise model-update filtering (BMUF) to stabilize learning process, which will be described in Section 2. By using this approach, we have trained successfully deep bidirectional LSTMs (DBLSTMs) and DNNs for LVCSR on two benchmark tasks, namely 309-hour Switchboard-I task and 1,860-hour "Switchboard+Fisher" task. In Section 3, we will present experimental results and the findings of this study. Finally, we conclude the paper in Section 4.

[*] Kai Chen contributed to this work when he worked as an intern with the Speech Group, Microsoft Research Asia. We thank our former colleague, Dr. Zhi-Jie Yan, for his contributions made in the early stage of this study.

978-1-4799-9988-0/16/$31.00 ©2016 IEEE

2. OUR APPROACH

2.1. Training Procedure

In this section, we present the training procedure of our proposed approach, which can be used to train DNN, CNN, LSTM and other types of deep learning machines.

2.1.1. Data Partition

Our approach takes an incremental block training procedure. Conceptually, during training, full training set $\mathbf{D}$ is partitioned into M non-overlapping blocks and each block is partitioned into N non-overlapping splits, which can be represented formally as follows:

$$\begin{aligned} \mathbf{D} &= \{\mathbf{D}_j | j = 1, 2, \cdots, M\} \\ \mathbf{D}_j &= \{\mathbf{D}_{jk} | k = 1, 2, \cdots, N\} \\ \text{for} \quad \forall j, k, l, m & \quad \mathbf{D}_{jk} \cap \mathbf{D}_{lm} = \emptyset \ . \end{aligned} \tag{1}$$

We denote the above partition configuration as $N \times M$. Depending on the nature of the training strategy used, the data partition can be done randomly at frame, chunk or sequence level as appropriate.

2.1.2. Intra-block Parallel Optimization

Select randomly a block of unprocessed data denoted as $\mathbf{D}_t$, which generates the t-th blockwise update, and distribute N splits of this block to N workers (e.g., GPU cards in a GPU cluster). These workers run in parallel to optimize local models with its portion of data.

By leveraging data parallelism within the block, the intra-block optimization can be conducted with different parallel algorithms. For example, if the problem is formulated as a global consensus problem, it can be solved by the alternating direction method of multipliers (ADMM) (e.g., [34, 35]). If the problem is formulated as an elastic forced problem, it can be solved by an elastic averaging SGD (EASGD) method [36].

In this paper, we just broadcast a global model $\mathbf{W}_g(t-1)$ to all workers to initialize local models and then update these local models by each worker independently with 1-sweep mini-batch SGD with classical momentum trick. It is noted that local models can also be optimized by other algorithms such as natural gradient SGD [30] and even ASGD for workers with multiple GPUs or CPUs. Finally, an aggregated model denoted as $\overline{\mathbf{W}}(t)$ can be obtained by averaging N optimized local models.

2.1.3. Blockwise Model-Update Filtering

After intra-block parallel optimization is completed, global model need be updated. Instead of treating $\overline{\mathbf{W}}(t)$ as updated global model directly, which is the strategy of MA, we treat global model update as a block-level stochastic optimization process and propose a ***B****lockwise* ***M****odel-****U****pdate* ***F****iltering* (BMUF) technique to stabilize the learning process, which works as follows:

- Firstly, we use $\mathbf{G}(t)$ to denote the model-update resulting from block $\mathbf{D}_t$:

$$\mathbf{G}(t) = \overline{\mathbf{W}}(t) - \mathbf{W}_g(t-1) \ . \tag{2}$$

- Then, we calculate the global-model update $\mathbf{\Delta}(t)$ as

$$\mathbf{\Delta}(t) = \eta_t \mathbf{\Delta}(t-1) + \zeta_t \mathbf{G}(t), 0 \leq \eta_t < 1, \zeta_t > 0. \tag{3}$$

- Consequently, the global model is updated as

$$\mathbf{W}(t) = \mathbf{W}(t-1) + \mathbf{\Delta}(t) \ . \tag{4}$$

Equations (3) and (4) look similar to the SGD with momentum trick, where $\mathbf{G}(t)$ plays the role of negative gradient. We call Equation (3) as doing BMUF since it generates model-update vector by filtering $\mathbf{G}(t)$ with the previous update vector. It is noted that when $\eta_t = 0$ and $\zeta_t = 1$, the above procedure becomes MA.

Because η_t plays the role of a block-level momentum, we call it "*block momentum*" (BM). Similarly, we call ζ_t "*block learning rate*" (BLR). Akin to SGD with momentum trick [37], BM can also work with a classical momentum scheme [17] or a Nesterov momentum scheme [38], which are referred to hereinafter as CBM and NBM, respectively. Before processing next data block $\mathbf{D}_{t+1}$, a global model $\mathbf{W}_g(t)$ need be broadcast to all workers to initialize their local models, which is calculated as follows:

- For η_t in CBM scheme,

$$\mathbf{W}_g(t) = \mathbf{W}(t), \tag{5}$$

- For η_t in NBM scheme,

$$\mathbf{W}_g(t) = \mathbf{W}(t) + \eta_{t+1} \mathbf{\Delta}(t) \ . \tag{6}$$

Repeat Steps 2.1.2 and 2.1.3 until all M data blocks are processed, which is called one sweep. We can fine-tune the model by several sweeps until a stopping criterion is satisfied and obtain the final global model $\mathbf{W}(T)$.

2.2. Implementation

As discussed in Section 2.1, intra-block parallel optimization can be run on a computer cluster, while local model aggregation, BMUF and global model update rely on collective communication among computational nodes. We implement our approach on an HPC GPU cluster with multiple computing nodes, each equipped with 2-8 Nvidia Telsa K40xm GPUs. A 56 Gbps private InfiniBand network is configured to connect all GPU nodes. The GPU cluster is connected to a shared storage with Hadoop Distributed File System (HDFS) via several spine switches. The total throughput of the spine switches to HDFS and to HPC GPU cluster are 8 Tbps and 320 Gbps, respectively. The HDFS serves as a high-performance distributed data storage in our experiments.

In our implementation, an MPI-based HPC machine learning platform [39] is used to coordinate parallel job scheduling and collective communication. It implements a master-slave model among computing nodes, where the master is responsible for job scheduling, load balancing, BMUF and global model update, and the slaves are workers for intra-block parallel optimization. The peer-to-peer and collective communications among master and slaves are very efficient through MPI. Using such a machine learning platform facilitates a scenario when the number of free GPUs in a shared cluster is less than the number of data splits N. In this case, the platform could for example only use $N/2$ GPUs and assign 2 data splits to each of them for processing. As a result, the actual number of GPUs only affects the speedup factor, while the learning behavior of our approach is unaltered.

To reduce the overhead of job scheduling, we will send each worker its subset of training data before training. During training, on each worker, next split will be loaded to memory when the current split is being processed to hide data-loading cost. Since all splits contain the same amount of data, for example τ mini-batches, it is

Table 1. *Performance (in %) comparison and training speedups of DBLSTMs trained by CSC-BPTT with SGD, MA and BMUF approach on "SWB task".*

Trainng Method	Partition Config.	WER (%)		Training Speedup
		Eval2000	RT03S	
MA	8×104	15.4	22.9	7.7
	16×52	16.0	23.4	15.3
BMUF -CBM	8×104	14.7	22.7	7.7
	16×52	15.0	22.7	15.3
BMUF -NBM	8×104	14.9	22.3	7.7
	16×52	14.8	22.4	15.3
Single-GPU SGD Baseline		14.8	22.9	1.0

Table 2. *Performance (in %) comparison and training speedups of DBLSTMs trained by epoch-wise BPTT with SGD, MA and BMUF approach on "SWB task".*

Training Method	Partition Config.	WER (%)		Training Speedup
		Eval2000	RT03S	
MA	8×104	15.6	23.5	7.9
	16×52	16.2	24.0	15.8
BMUF -CBM	8×104	14.7	23.1	7.9
	16×52	14.8	23.4	15.8
BMUF -NBM	8×104	14.5	22.8	7.9
	16×52	14.3	23.0	15.8
Single-GPU SGD Baseline		14.8	22.9	1.0

equivalent to conducting BMUF after all workers have updated their local models for τ times respectively. In practice, we only need to partition training set into N subsets instead of $N \times M$ splits.

2.3. Discussion

In mini-batch based SGD with momentum trick, momentum can be treated as a way to delay parameter update. For example, $\mathbf{B}_i$, which is the i-th mini-batch in $\mathbf{D}$, contributes to not only the i-th update, but also all the following updates. Assume the final parameter of mini-batch SGD optimized model is $\mathbf{W}_s$ and $\mathbf{\Delta}_s = \mathbf{W}_s - \mathbf{W}(0)$, where $\mathbf{W}(0)$ is initial model, then the contribution of the i-th mini-batch to $\mathbf{\Delta}_s$ is

$$\boldsymbol{\delta}_s^{(i)} = \gamma_s \mathbf{g}_s^{(i)} (1 + \epsilon_s + \epsilon_s^2 + \cdots) \approx \frac{\gamma_s}{1 - \epsilon_s} \mathbf{g}_s^{(i)} \ , \tag{7}$$

where γ_s, ϵ_s, and $\mathbf{g}_s^{(i)}$ are the learning rate, momentum and gradient of the i-th mini-batch, respectively.

In incremental block training, we assume $\mathbf{B}_i$ is the l-th mini-batch of the k-th split in block $\mathbf{D}_j$ and each split contains τ mini-batches. τ is always set to be a relatively small value to avoid divergence of local models. When MA is used, $\mathbf{B}_i$ only contributes to local and global model update at current split and block, and will have no direct influence in successive training, so we can express its contribution to $\mathbf{\Delta}_m$ as follows:

$$\begin{aligned} \boldsymbol{\delta}_m^{(i)} &= \frac{1}{N} \gamma_m \mathbf{g}_m^{(i)} (1 + \epsilon_m + \epsilon_m^2 + \cdots + \epsilon_m^{\tau - l}) \\ &= \frac{1}{N} \cdot \frac{\gamma_m (1 - \epsilon_m^{\tau - l + 1})}{1 - \epsilon_m} \mathbf{g}_m^{(i)} \ . \end{aligned} \tag{8}$$

Compared with Equation (7), we should set γ_m about N times of γ_s to ensure enough per mini-batch contribution. As we know, an important role of momentum in SGD is to attenuate influence of noisy component in gradients by history update information. For per split optimization, lack of update information from previous blocks will weaken the attenuation effect while larger γ_m enlarges influences of noise. Consequently, model-update resulting from a single block is pretty noisy and the performance of MA becomes poorer with more parallel workers.

Assume constant BLR ζ and BM η are used in BMUF. Since η builds links between successive blocks, $\mathbf{B}_i$'s contribution to $\mathbf{\Delta}_b$ is

$$\begin{aligned} \boldsymbol{\delta}_b^{(i)} &= \frac{\gamma_b (1 - \epsilon_b^{\tau - l + 1})}{N(1 - \epsilon_b)} \mathbf{g}_b^{(i)} \zeta (1 + \eta + \eta^2 + \cdots) \\ &\approx \frac{\zeta}{N(1 - \eta)} \cdot \frac{\gamma_b (1 - \epsilon_b^{\tau - l + 1})}{1 - \epsilon_b} \mathbf{g}_b^{(i)} \ . \end{aligned} \tag{9}$$

Akin to the behavior of a mini-batch in momentum SGD training, current block contributes to all the following global model updates, and noisy components of model-update resulting from current block will be attenuated while informative ones will be enlarged. We use the following formula

$$\frac{\zeta}{N(1 - \eta)} = C \tag{10}$$

to set empirically the values of η and ζ, where C is a constant slightly larger than 1. Therefore γ_b can be set at the same range of γ_s to ensure enough per mini-batch contribution. The above analysis is applicable to both CBM and NBM versions of BMUF.

Another possible solution to mitigate the degradation problem in MA is to warm-up per split optimization by introducing history update information to SGD. However, its gain is limited in comparison with BMUF, therefore we will not report its result here.

3. EXPERIMENTS AND RESULTS

3.1. Experimental Setup

We choose two LVCSR benchmark tasks to conduct evaluations. The first one is Switchboard-I conversational telephone speech transcription task [40], which contains about 309 hours of training speech and is referred to as "SWB task". The second one consists of Switchboard-I corpus and Fisher English corpus (part 1 and part 2) [41], which contains about 1,860-hour training speech data and is referred to as "SWB+Fisher task". For both tasks, we use the 2000 NIST Hub5 evaluation (Eval2000, about 2 hours of speech) and Spring 2003 NIST rich transcription set (RT03S, about 6.3 hours of speech) as testing sets, and word error rate (WER) as performance metric. A 52-dimensional PLP feature vector with up to 3rd order time derivatives is extracted for each frame. The LVCSR systems take a hybrid neural network and HMM architecture (e.g., [1]).

For "SWB task", we train DBLSTM as acoustic model, which has 5 hidden layers, each containing 512 memory blocks (256 for forward and 256 for backward states), and 9,304 HMM tied-states as output classes, resulting to about 11 million free parameters. Both epoch-wise BPTT and context-sensitive-chunk (CSC) BPTT [11] are used to train DBLSTM. In CSC-BPTT training, each utterance is partitioned into CSCs of 64 frames long with 21 past and 21 future frames appended as context, which is denoted as "21-64+21", while a 32-frame overlap is used in decoding. For "SWB+Fisher task", we train DNN as acoustic model, which has 11 consecutive frames of feature vectors as input, 7 hidden layers with 2,048 ReLUs per layer, and 18,002 HMM tied-states as output classes, resulting to about

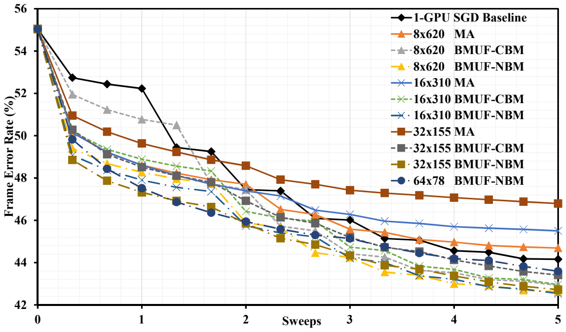

Fig. 1. *Learning curves of FER on validation set with different methods and data partition configurations for DNN training.*

63 million free parameters. During training, L2 constraint is used for regularization. For both DBLSTM and DNN training, a frame-level CE criterion is used as objective function. The frame-level ground truth of training data is obtained by the forced alignment of the training sentences using the corresponding ML-trained GMM-HMM systems. Learning rate scheduling is the same as in [11]. For both tasks, we randomly sample 30 hours of speech as validation set. In DBLSTM training, validation set is evaluated every sweep of data, while in DNN training, it is evaluated every 600 hours of data. Moreover, learning rates are carefully tuned for all training configurations and the one leading to the best validation set performance is chosen to decode testing sets. In order to make fair comparison, all methods start from the same initial model and process the training set for the same number of sweeps. For DBLSTM, initial model is obtained by 1-sweep SGD with respective algorithms and the training set is processed for 6 sweeps, while for DNN, initial model is obtained by 1-sweep SGD of 309 hours of data and the training set is processed for 5 sweeps. According to Equation (10), the BM η is set as 0.9, 0.94, 0.97 and 0.986 in 8-, 16-, 32-, 64-GPU experiments respectively and BLR ζ is always set as 1.0.

3.2. Experimental Results

3.2.1. SWB task

The partition configurations are "8×104" and "16×52" (about 22.5 minutes of speech per split). The number of GPUs equals to the split number per block. Tables 1 and 2 compare WERs on testing sets and training speedups with CSC and epoch-wise BPTT training per worker, respectively. Both MA and BMUF achieve linear speedup in terms of the throughput of processing training data. As for model performance, consistent with our analysis, with more GPUs, the performance gap between MA and SGD becomes larger, while BMUF-trained models outperform that of SGD-trained ones in most cases. For our BMUF approach, NBM performs better than CBM.

3.2.2. SWB+Fisher task

In DNN training, data set is partitioned at frame level and the partition configurations are "8×620", "16×310", "32×155" and "64×78" (about 22.5 minutes per split). Again, the number of GPUs equals to the split number per block. Fig. 1 shows the learning curves of different methods and data partitions in terms of frame error rate (FER) on validation set. Significant performance degradations are observed for MA, while BMUF approach performs similarly with different configurations, yet outperforms SGD. For our BMUF approach, NBM learns faster yet converges to better solutions than CBM. It is noted that NBM experiments with 8-32 GPUs converge to almost the same FER. In terms of testing set performance, Table 3 shows that in comparison with single-GPU SGD training, MA incurs WER degradations, while BMUF approaches achieve about 5.0% and 5.3% relative WER reductions on Eval2000 and RT03S, respectively. Again, NBM performs better than CBM. We also compare the elapsed time per sweep of data in Table 4. Obviously, a linear speedup is also achieved on this task.

Table 3. *Performance (in %) comparison and training speedups of DNNs trained by single-GPU SGD, MA and BMUF approach on "SWB+Fisher task".*

Training Method	Partition Config.	WER (%)		Training Speedup
		Eval2000	RT03S	
MA	8×620	14.2	18.8	7.3
	16×310	14.8	19.3	14.5
	32×155	15.5	19.9	28.7
BMUF -CBM	8×620	13.4	18.0	7.3
	16×310	13.4	18.1	14.4
	32×155	13.5	18.2	28.4
BMUF -NBM	8×620	13.3	17.8	7.3
	16×310	13.4	17.9	14.4
	32×155	13.4	17.9	28.4
	64×78	13.6	18.1	56.2
Single-GPU SGD Baseline		14.0	18.8	1.0

Table 4. *Elapsed time (in minutes) per sweep of 1,860-hour training data in DNN training with different optimization methods.*

Training Method	Partition Config.	Elapsed Time (minutes)			
		optimize	aggregate	validate	SUM
MA	8×620	320.1	16.5	2.8	339.4
	16×310	159.9	10.3	1.4	171.6
	32×155	81.0	4.7	0.7	86.4
BMUF -CBM	8×620	320.1	17.2	2.8	340.1
	16×310	159.5	11.3	1.4	172.2
	32×155	81.6	5.0	0.7	87.3
BMUF -NBM	8×620	319.8	17.4	2.8	340.0
	16×310	159.6	11.9	1.4	172.9
	32×155	81.5	5.2	0.7	87.4
	64×78	40.3	3.5	0.4	44.2
Single-GPU SGD Baseline		2460.6	N/A	22.5	2483.1

4. CONCLUSION AND DISCUSSION

From the above results, we conclude that the proposed BMUF approach can indeed scale out deep learning on a GPU cluster with almost linear speedup and improved or no-degradation of recognition accuracy compared with mini-batch SGD on single GPU. In addition to the verified cases for DBLSTM and DNN training on LVCSR tasks, we have also verified its effectiveness up to 16 GPUs for CTC-training of DBLSTM on a handwriting OCR task using about one million training text line images. Our ongoing and future work include 1) Scale out to more GPUs; 2) Evaluate our approach to CNN and other types of discriminative sequence training for D(B)LSTM and DNN; 3) Develop even better parallel training approach.

5. REFERENCES

[1] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp.30-42, 2012.

[2] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-depedent deep neural networks," *Proc. INTERSPEECH-2011*, pp.437-440.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 16, pp.82-97, 2012.

[4] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*, *Springer*, 2015.

[5] A. Graves, A. R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," *Proc. ICASSP-2013*, pp.6645-6649.

[6] A. Graves, N. Jaitly, and A. R. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," *Proc. ASRU-2013*, pp.273-278.

[7] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," *Proc. ICML-2014*, pp.1764-1772.

[8] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," *Proc. INTERSPEECH-2014*, pp.338-342.

[9] H. Sak, O. Vinyals, G. Heigold, A. Senior, E. McDermott, R. Monga, and M. Mao, "Sequence discriminative distributed training of long short-term memory recurrent neural networks," *Proc. INTERSPEECH-2014*, pp.1209-1213.

[10] H. Sak, A. Senior, K. Rao, and F. Beaufays, "Fast and accurate recurrent neural network acoustic models for speech recognition," *Proc. INTERSPEECH-2015*, pp.1468-1472.

[11] K. Chen, Z.-J. Yan, and Q. Huo, "Training deep bidirectional LSTM acoustic model for LVCSR by a context-sensitive-chunk BPTT approach," *Proc. INTERSPEECH-2015*, pp.3600-3604.

[12] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Proc. NIPS-2012*, pp.1097-1105.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on ImageNet classification," *Proc. ICCV-2015*.

[14] B. Moysset, T. Bluche, K. Maxime, M. F. Benzeghiba, R. Messina, J. Louradour, and C. Kermorvant, "The A2iA multi-lingual text recognition system at the second Maurdor evaluation," *Proc. ICFHR-2014*, pp.297-302.

[15] T. Bluche, H. Ney, J. Louradour, and C. Kermorvant, "Framewise and CTC training of neural networks for handwriting recognition," *Proc. ICDAR-2015*, pp.81-85.

[16] K. Chen, Z.-J. Yan, and Q. Huo, "A context-sensitive-chunk BPTT approach to training deep LSTM/BLSTM recurrent neural networks for offline handwriting recognition," *Proc. ICDAR-2015*, pp.411-415.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," in D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp.318-362, MIT Press, 1986.

[18] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp,1-17, 1964.

[19] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. A. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," *Proc. NIPS-2012*, pp.1232-1240.

[20] A. Coates, N. Huval, T. Wang, D. Wu, B. Catanzaro, and A. Y. Ng, "Deep learning with COTS HPC systems," *Proc. ICML-2013*, pp.1337-1345.

[21] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, "Improving deep neural network acoustic models using generalized maxout networks," *Proc. ICASSP-2014*, pp.215-219.

[22] T. N. Sainath, I. Chung, B. Ramabhadran, M. Picheny, J. Gunnels, B. Kingsbury, G. Saon, V. Austel, and U. Chaudhari, "Parallel deep neural network training for LVCSR tasks using Blue Gene/Q," *Proc. INTERSPEECH-2014*, pp.1048-1052.

[23] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-Bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," *Proc. INTERSPEECH-2014*, pp.1058-1062.

[24] F. Niu, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Proc. NIPS-2011*, pp.693-701.

[25] G. Heigold, V. Vanhoucke, A. Senior, P. Nguyen, M. Ranzato,M. Devin, and J. Dean, "Multilingual acoustic models using distributed deep neural networks," *Proc. ICASSP-2013*, pp.8619-8623.

[26] G. Heigold, E. McDermott, V. Vanhoucke, A. Senior, and M. Bacchiani, "Asynchronous stochastic optimization for sequence training of deep neural networks," *ICASSP-2014*, pp.5624-5628.

[27] S. Zhang, C. Zhang, Z. You, R. Zheng, and B. Xu, "Asynchronous stochastic gradient descent for DNN training," *Proc. ICASSP-2013*, pp.6660-6663.

[28] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," *Proc. North American Chapter of the Association for Computational Linguistics (NAACL)*, 2010, pp.456-464.

[29] M. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized Stochastic Gradient Descent," *Proc. NIPS-2010*, pp.2595-2603

[30] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of DNNs with natural gradient and parameter averaging," *Proc. ICLR-2015*.

[31] A. L. Maas, A. Y. Hannun, C. T. Lengerich, P. Qi, D. Jurafsky, and A. Y. Ng, "Increasing deep neural network acoustic model size for large vocabulary continuous speech recognition," *arXiv:1406.7806*, 2014.

[32] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," *Proc. INTERSPEECH-2015*, pp.1488-1492.

[33] S. Zhang, H. Jiang, S. Wei, and L. Dai, "Rectified linear neural networks with tied-scalar regularization for LVCSR," *Proc. INTERSPEECH-2015*, pp.2635-2639.

[34] S. Boyd, N. Parikh, E. Chu, B, Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp.1-122, 2010.

[35] Q. Huo, Z.-J. Yan, and K. Chen, "Deep learning using alternating direction method of multipliers," *USA Patent Application, 2014*.

[36] S. Zhang, A. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," *Proc NIPS-2015*.

[37] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *Proc. ICML-2013*, pp. 1139-1147.

[38] Y. Nesterov, " A method of solving a convex programming problem with convergence rate O(1/sqr(k))," Soviet Mathematics Doklady, vol. 27, pp. 372-376, 1983.

[39] Z.-J. Yan, T. Gao, and Q. Huo, "Designing an MPI-based parallel and distributed machine learning platform on large-scale HPC clusters," *2012 International Workshop on Statistical Machine Learning for Speech Processing*, Kyoto, Japan, March 31, 2012 (http://www.ism.ac.jp/IWSML2012/).

[40] J. J. Godfrey, E. C. Holliman, and J. McDaniel, "Switchboard: telephone speech corpus for research and development," *Proc. ICASSP-1992*, pp.I-517-520.

[41] C. Cieri, D. Miller, and K. Walker, "The Fisher corpus: A resource for the next generation of speech-to-text," *Proc. ICLRE-2004*, pp.69-71.