INTERSPEECH 2017
August 20–24, 2017, Stockholm, Sweden

Empirical Evaluation of Parallel Training Algorithms on Acoustic Modeling

Wenpeng Li[1], *Binbin Zhang*[1], *Lei Xie*[1]*, *Dong Yu*[2]

[1]School of Computer Science, Northwestern Polytechnical University, Xi'an, China
[2]Tencent AI Lab, Seattle, USA

{wpli, bbzhang, lxie}@nwpu-aslp.org, dongyu@ieee.org

Abstract

Deep learning models (DLMs) are state-of-the-art techniques in speech recognition. However, training good DLMs can be time consuming especially for production-size models and corpora. Although several parallel training algorithms have been proposed to improve training efficiency, there is no clear guidance on which one to choose for the task in hand due to lack of systematic and fair comparison among them. In this paper we aim at filling this gap by comparing four popular parallel training algorithms in speech recognition, namely asynchronous stochastic gradient descent (ASGD), blockwise model-update filtering (BMUF), bulk synchronous parallel (BSP) and elastic averaging stochastic gradient descent (EASGD), on 1000-hour LibriSpeech corpora using feed-forward deep neural networks (DNNs) and convolutional, long short-term memory, DNNs (CLDNNs). Based on our experiments, we recommend using BMUF as the top choice to train acoustic models since it is most stable, scales well with number of GPUs, can achieve reproducible results, and in many cases even outperforms single-GPU SGD. ASGD can be used as a substitute in some cases.

Index Terms: speech recognition, parallel algorithm, ASGD, BMUF, BSP, EASGD

1. Introduction

Since 2010, the year in which deep neural networks (DNNs) were successfully applied to the large vocabulary continuous speech recognition (LCVSR) tasks [1, 2, 3] and led to significant recognition accuracy improvement over the then state of the art, various deep learning models, such as convolutional neural networks (CNNs) [4, 5, 6, 7, 8, 9], long short-term memory (LSTM) recurrent neural networks (RNNs) [10, 11, 12, 13, 14, 15, 16, 17] and their variants [18, 19, 20, 21, 22, 23], have been developed to further improve the performance of automatic speech recognition (ASR) systems. Albeit achieving the state-of-the-art performance, these deep learning models are time consuming to train well, especially when trained on single-GPU. Trade-offs often need to be made between the scale of model size and training corpora (and thus recognition accuracy) and the training time because even with today's massively parallel GPU it usually takes days or weeks to train large models to desired accuracy on a single GPU.

Many parallel training algorithms have been proposed to speed up training. These algorithms can be categorized into two classes: model parallelism (e.g., [24, 25]), which exploits and splits the structure of neural networks to distribute computation across GPUs, and data parallelism (e.g., [24, 26, 27]), which splits and distributes data across GPUs to achieve speedup. Model parallelism focuses on computing more parameters at the same time. It allows and is more suitable for training models that are too big to fit in the memory on a single device. On the other hand, data parallelism concentrates on processing more training samples at the same time and is thus best used when there are enormous training samples. In speech recognition, data parallelism is more important since ASR models usually fit well on a single GPU while the training set is often large.

The core problem data parallelism algorithms try to solve is the difficulty in achieving parallelization of mini-batch based stochastic gradient descent (SGD) algorithm [28], which is the most popular technique to train deep learning models (DLMs). Several successful techniques, such as asynchronous stochastic gradient descent (ASGD) [29, 30, 31], blockwise model-update filtering (BMUF) [32], bulk synchronous parallel (BSP) [33, 34, 35, 36], 1-bit SGD [26] and elastic averaging stochastic gradient descent (EASGD) [37], have been proposed recently. Unfortunately, these techniques solve the problem with different assumptions and strategies, have been evaluated only on vastly different data sets and tasks, and there is no theoretical guarantee on their behavior when used to train DLMs. This causes the difficulty in selecting the right parallel algorithm for training models on industrial-size corpora.

In this paper, we evaluate and systematically compare four parallel training algorithms, namely BSP, ASGD, BMUF and EASGD with regard to training speed, convergence behavior, final model's performance, reproducibility, and robustness across models, number of GPUs, and learning control parameters. For all we know, this is the first time these algorithms are compared relatively thoroughly on ASR tasks. It is also the first time EASGD is evaluated for acoustic model training. All the four algorithms were implemented in Kaldi toolkit [38] using message passing interface (MPI) for parameter exchange across GPUs. Using the same communication protocol guarantees that the comparison is fair and reliable. To evaluate these algorithms, we train DNNs and CLDNNs [23] (an architecture that stacks CNNs, LSTMs and DNNs) on 1000hr LibriSpeech [39] corpus.

The rest of the paper is organized as follows. In Section 2, we introduce BSP, ASGD, BMUF and EASGD, and discuss relationships between them. In Section 3 we describe series of experimental setups and report related results. We conclude the paper in Section 4.

2. Parallel training algorithms

2.1. BSP

The bulk synchronous parallel (BSP) [33] algorithm is often referred to as model averaging. In this model, data are distributed across multiple workers. Each worker updates its local model replica independently using its own portion of data with SGD. Periodically the local models are averaged and the generated global model is synchronized across workers. We denote w_t^i as the i-th worker's local model at time t. The global model $\tilde{w}_t$ is

* Corresponding author

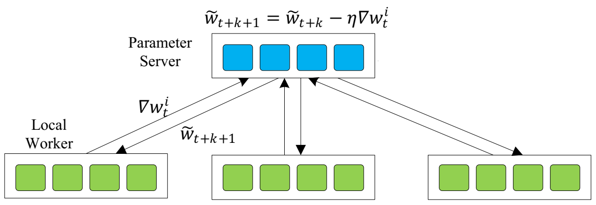

Figure 1: *ASGD architecture. Arrows indicate communication between the parameter server and the workers.*

computed as

$$\tilde{w}_t = \bar{w}_t = \frac{1}{N}\sum_{i=1}^{N} w_t^i, \tag{1}$$

where N is the number of local workers and $\bar{w}_t$ is the average model of the local models. This algorithm is easy to implement and can achieve linear speedup when communication cost can be ignored (e.g., with large synchronous time) at the cost of recognition accuracy degradation, esp. when the number of workers becomes large.

2.2. ASGD

The asynchronous stochastic gradient descent (ASGD) algorithm is the distributed version of SGD. It is proved [40] that ASGD converges for convex problems. As shown in Figure 1, ASGD uses a parameter sever and several local workers. Each worker independently and asynchronously pulls the latest global model $\tilde{w}_t$ from the parameter server, computes the gradient ∇w_t^i with a new minibatch, and sends it to the parameter server. The parameter server always keeps the current model. When it receives the gradient ∇w_t^i from worker i it generates the new model

$$\tilde{w}_{t+k+1} = \tilde{w}_{t+k} - \eta \nabla w_t^i \tag{2}$$

where η is the learning rate.

Before worker i sends gradient ∇w_t^i back to parameter server, some other workers may have already added their local gradients to the model and updated the model k times to become $\tilde{w}_{t+k}$. Therefore ASGD essentially adds a "delayed" gradient ∇w_t^i computed based on the model $\tilde{w}_t$ to the model $\tilde{w}_{t+k}$ [41]. This may be the reason that ASGD can be unstable: sometimes the model can converge to the same accuracy as that trained with SGD but with more iterations, and sometimes it can never achieve the same performance as SGD, esp. when there are many workers.

2.3. BMUF

The blockwise model-update filtering (BMUF) algorithm [32] can be considered as an improved model averaging technique in which the global model update is implemented as a filter.

In BMUF, the full training set D is partitioned into M non-overlapping blocks and each block is further partitioned into N non-overlapping splits, where N is the number of workers. Each worker updates its local model with its portion of data. The N optimized local models are then averaged using Eq. (1). Unlike BSP, which treats the average model $\bar{w}_t$ as the global model, BMUF generates the global model $\tilde{w}_t$ as

$$\tilde{w}_t = \tilde{w}_{t-1} + \Delta_t, \tag{3}$$

where

$$\Delta_t = \zeta \Delta_{t-1} + \eta G_t, 0 \le \zeta < 1, \eta > 0, \tag{4}$$

is the global-model update,

$$G_t = \bar{w}_t - \tilde{w}_{t-1} \tag{5}$$

is the model-update resulted from a block, ζ is called block momentum (BM) and η is called block learning rate (BLR). We use the formula

$$\frac{\eta}{N(1-\zeta)} = C \tag{6}$$

to set the values of ζ and η empirically, where C is a constant slightly large than 1 and N is the number of workers. Usually, the value of η and C both are set to 1.0 and the value of ζ is calculated based on Eq. (6). We implemented CBM-BMUF [32] in this work.

2.4. EASGD

In elastic averaging stochastic gradient descent (EASGD) [37], the loss function is defined as

$$\min_{w_t^1,\dots,w_t^N,\tilde{w}_t} \sum_{i=1}^{N} f(\mathrm{D}|w_t^i) + \frac{\lambda}{2}||w_t^i - \tilde{w}_t||^2 \tag{7}$$

where D is the training set, $f(.)$ is the loss function for local sequential training, λ is a hyper-parameter for the quadratic penalty term, w_t^i represents model for the i-th worker, and $\tilde{w}_t$ represents the global model.

From Eq. (7) we observe that EASGD minimizes the loss summed over all workers, as well as the quadratic difference between the global model and local models. $\frac{\lambda}{2}||w_t^i - \tilde{w}_t||^2$ is a quadratic regularization term, which forces local workers to stay close to the global model.

By taking the derivative of w_t^i and $\tilde{w}_t$ in Eq. (7), we get the update rules for w_t^i and $\tilde{w}_t$ in synchronous EASGD as

$$\begin{aligned} w_{t+1}^i &= w_t^i - \eta \nabla w_t^i - \eta\lambda(w_t^i - \tilde{w}_t) \\ \tilde{w}_{t+1} &= \tilde{w}_t - \eta\lambda \sum_{i=1}^{N}(\tilde{w}_t - w_t^i) \end{aligned} \tag{8}$$

where ∇w_t^i is the stochastic gradient of $f(.)$ with respect to w_t^i.

In asynchronous EASGD, ∇w_t^i is only used in local updating, and the update rules for local and global models become

$$\begin{aligned} w_{t+1}^i &= w_t^i - \alpha(w_t^i - \tilde{w}_t) \\ \tilde{w}_{t+1} &= \tilde{w}_t - \alpha(\tilde{w}_t - w_t^i) \end{aligned} \tag{9}$$

where $\alpha = \eta\lambda$, which controls the update step for the variable. Small α allows for more exploration as it allows w^i to fluctuate further from $\tilde{w}$ while large α makes local model perform more exploitation. We only implemented asynchronous EASGD in this work.

2.5. Relationships between algorithms

These four algorithms, although are different, have relations.

First, ASGD and EASGD are asynchronous algorithms based on the client/server framework, in which the global model is stored on and updated by a parameter server, and each worker computes gradients and updates its local model independently. Workers only exchange parameters with the server and do not communicate with each other. BSP and BMUF, on the other hand, are synchronous algorithms that do not use a server. All workers exchange parameters synchronously with each other.

Second, in ASGD the global model is updated based on the local gradients computed by and sent from workers. In BSP, EASGD and BMUF, however, the global model is a weighted sum of local models instead of gradients.

Third, EASGD and BMUF both introduce extra hyper-parameters whose values may affect the training behavior, while ASGD and BSP have no extra hyper-parameter and thus require less tuning in practice.

Forth, we argue that BMUF actually minimizes the difference

$$\min_{\tilde{w}_t} F(\tilde{w}_t) = \frac{1}{2}\sum_{i=1}^{N} ||w_t^i - \tilde{w}_t||^2 \tag{10}$$

between the global and local models. By taking the derivative of $\tilde{w}$, we get

$$\nabla\tilde{w}_t = \sum_{i=1}^{N}(\tilde{w}_t - w_t^i). \tag{11}$$

Let $\nabla\tilde{w}_t = 0$. By directly solving $\tilde{w}_t$, we get

$$\tilde{w}_t = \frac{1}{N}\sum_{i=1}^{N} w_t^i. \tag{12}$$

This is the same as BSP in Eq. (1). If we optimize $\tilde{w}_t$ using SGD, then

$$\tilde{w}_t = \tilde{w}_{t-1} - \eta\sum_{i=1}^{N}(\tilde{w}_{t-1} - w_t^i) \tag{13}$$

which is the same as Eq. (8) in EASGD.

Further, if we optimize $\tilde{w}_t$ using momentum SGD, then

$$\begin{aligned}\tilde{w}_t &= \tilde{w}_{t-1} - \eta\sum_{i=1}^{N}(\tilde{w}_{t-1} - w_t^i) - \zeta\nabla\tilde{w}_{t-1} \\ &= \tilde{w}_{t-1} + \eta' G_t + \zeta'\Delta_{t-1} \\ &= \tilde{w}_{t-1} + \Delta_t\end{aligned} \tag{14}$$

where $\eta' = N\eta, \zeta' = N\zeta$. This is exactly the BMUF update rule in Eq. (3).

Therefore we conclude that the global model updates of BSP, EASGD and BMUF are derived from the same objective function with different optimization strategies.

3. Experiments

3.1. Experimental setup

In this work, all the models are trained on the 1000hr LibriSpeech [39] dataset. The 40-dim FBANK features computed on a 25ms window shifted by 10ms are used. The lexicon and language model (LM) are provided by the dataset. Specifically, the results reported here are all achieved with a full 3-gram LM. We used test-clean and test-other sets for evaluation.

To evaluate the parallel training algorithms, we trained two types of DLMs: DNNs and CLDNNs [23]. The input to DNNs is the 40-dim FBANK feature with first and second order time derivatives and 11 frame context. The input to CLDNNs is the same as that to DNNs but without the 2nd order time derivatives. The DNN has 6 hidden layers, each containing 1024 neurons. With 5723 HMM tied-states as output classes, it has about 13.5 million parameters. The CLDNN consists of 1 CNN layer (128 feature maps), 2 DNN layers (1024 neurons) and 2 LSTM layers (1024 memory blocks and 512 projections). With the same output classes as that in DNNs, CLDNNs have about 13.8 million parameters. Both models use the ReLU activation function.

In order to ensure the fairness of the comparison and the credibility of the experimental results, we take the following measures: First, all experiments in this work were carried out on the same computing node with 8 GTX1080 GPUs. Second, the four parallel training algorithms were implemented in the KALDI toolkit. The parameter exchange among GPUs is based on OpenMPI. Third, in all parallel training we used the same initial model which was obtained by one-epoch minibatch-SGD on a single-GPU. Finally, we used the identical learning rate schedule and the same initial learning rate. The learning rate keeps fixed as long as the cross entropy loss on a cross-validation (CV) set decreases by at least 1%. Then, the learning rate is halved each epoch until the optimization terminates when the cross entropy loss on the CV set decreases by less than 0.1%.

Table 1: *WER and training speedup of DNNs trained by single-GPU minibatch-SGD, ASGD, BMUF, BSP, and EASGD. The synchronization period is 5 minibatches for each algorithm.*

Parallel Algorithm	GPU Number	Training Speedup	WER(%)	
			test-clean	test-other
ASGD	4	2.74X	5.91	15.55
	8	4.72X	6.09	16.20
BMUF	4	2.68X	**5.70**	**15.01**
	8	4.56X	**5.99**	**15.66**
BSP	4	2.68X	6.01	16.03
	8	4.56X	6.21	16.55
EASGD	4	2.80X	6.04	16.02
	8	5.00X	6.22	16.64
Minibatch -SGD	1	1.0X	5.83	15.44

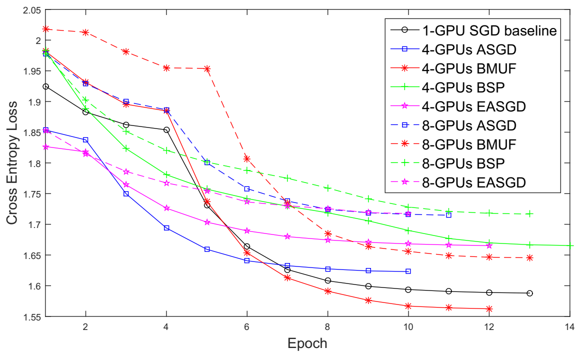

Figure 2: *Learning curves of CE loss on CV set with different algorithms and GPU numbers for DNN training.*

3.2. Experimental results

3.2.1. DNN results

In DNN training, the minibatch size was set to 4096. Table 1 compares training speedups and word error rate (WER) on testing sets with four parallel training algorithms on 4 GPUs and 8 GPUs. When using same synchronization period, EASGD achieved the best training speedup, followed by ASGD, BMUF and BSP. However, BMUF achieved the best WER in both 4 and 8 GPU cases, and even outperformed the single-GPU SGD. As Figure 2 shows, the convergence trend of BMUF is similar to minibatch-SGD with single-GPU in the CV set. To further verify our conclusions, we chose the most appropriate synchronization period for each algorithm based on the literature. The results in Table 2 show that BMUF still performed the best.

3.2.2. CLDNN results

In CLDNN training, we computed gradients on 100 subsequences from different utterances in the same time. The truncated BPTT with truncation step of 20 was used to train the models. The appropriate synchronization period was chosen for each algorithm based on the literature. Specifically, the synchronization periods for BSP, ASGD, BMUF, and EASGD are 5, 5, 80, and 64 minbatches, respectively. Table 3 shows that BMUF achieved the best WER and training speedup with 4 GPUs and ASGD achieved the best WER with 8 GPUs.

3.2.3. Synchronization period

The choice of synchronization period τ affects the behavior of BMUF and ASGD. As Table 4 shows, the WER of ASGD gradually increased and the training even diverged when increasing τ. This is because ASGD suffers from the problem of de-

Table 2: *WER and training speedup of DNNs trained by ASGD, BMUF, BSP and EASGD. The most appropriate synchronization period is chosen for each algorithm, respectively.*

Parallel Algorithm	Sync Period	Training Speedup	WER(%)	
			test-clean	test-other
ASGD	1	2.22X	5.85	15.54
BMUF	80	2.93X	**5.74**	**14.87**
BSP	5	2.68X	6.01	16.03
EASGD	64	2.99X	6.06	15.97

Table 3: *WER and training speedup of CLDNNs trained by ASGD, BMUF, BSP and EASGD. The most appropriate synchronization period is chosen for each algorithm, respectively.*

Parallel Algorithm	GPU Number	Training Speedup	WER(%)	
			test-clean	test-other
ASGD	4	3.42X	5.37	14.08
	8	6.11X	**5.48**	**14.29**
BMUF	4	3.84X	**5.26**	**13.80**
	8	6.88X	5.63	14.65
BSP	4	3.50X	5.43	14.08
	8	6.03X	5.64	14.74
EASGD	4	3.53X	5.44	14.31
	8	7.45X	5.55	14.54
Minibatch -SGD	1	1.0X	5.26	13.76

layed gradient update and larger synchronization period results in greater latency. In contrast, we observed no performance degradation on BMUF when varying the synchronization period. As for training speedup, when synchronization period τ is small, the parameter exchange among multi-GPUs is quite frequent and the communication overhead is the main bottleneck of training speed. So as τ increases (from 5 to 20 minibatches in Table 4), the communication overhead decreases and training speedup increases. When the value of τ continues to increase, the communication overhead is reduced so that the computation speed becomes the main bottleneck. Therefore the training speedup almost keeps unchanged as the synchronization period increases (from 20 to 80 minibatches in Table 4).

3.2.4. Minibatch size

Table 5 compares three different minibatch sizes in BMUF. With single-GPU training, the training speed is lower when smaller minibatch size is used. This is because with small minibatch size the GPU power is not fully utilized and the model is updated more frequently.

The training speedup s of multi-GPU training is calculated as

$$s = \frac{t_s}{f(t_s, N) + t_c} \tag{15}$$

where t_s is the computation time through one epoch of dataset on single-GPU, N is the number of GPUs, t_c is the communication overhead, and

$$f(t_s, N) = \alpha \frac{t_s}{N} \tag{16}$$

is a decreasing function over N. When the minibatch can fill the GPU α is 1, otherwise α is greater than 1. According to Eqs. (15) and (16), we get

$$s = \frac{1}{\frac{\alpha}{N} + \frac{t_c}{t_s}} \tag{17}$$

This means the training speedup s of multi-GPU parallel training depends on $\frac{t_c}{t_s}$. When synchronization period τ is small, the communication overhead t_c is large due to frequent parameter exchange among GPUs. Although t_c and t_s decrease

Table 4: *WER and training speedup of DNNs trained by ASGD and BMUF on 4 GPUs with various synchronization periods.*

Parallel Algorithm	Sync Period	Training Speedup	WER(%)	
			test-clean	test-other
ASGD	5	2.74X	5.91	15.55
	20	2.96X	5.97	15.78
	80	divergence		
BMUF	5	2.68X	5.70	15.01
	20	2.90X	5.73	14.86
	80	2.93X	5.74	14.87
Minibatch -SGD	1	1.0X	5.83	15.44

Table 5: *WER and training speedup of DNNs trained by BMUF on 4 GPUs with various minibatch sizes.*

Minibatch Size	Sync Period	Training Speedup	WER(%)	
			test-clean	test-other
256	Single-GPU	1.0X	5.80	15.13
	5	1.85X	5.86	15.31
	20	2.54X	5.86	15.26
	80	3.21X	5.87	15.13
1024	Single-GPU	1.0X	5.72	14.80
	5	2.14X	5.71	15.02
	20	2.76X	5.74	15.02
	80	2.98X	5.69	14.90
4096	Single-GPU	1.0X	5.83	15.44
	5	2.68X	5.70	15.01
	20	2.90X	5.73	14.86
	80	2.93X	5.74	14.87

with the increase of minibatch size, t_c decreases more quickly. Therefore the larger minibatch size leads to the greater training speedup. When τ is large, however, t_c is small. t_s decreases more quickly with the increase of minibatch size and causes decreasing in training speedup. However, from the perspective of absolute training speed, it benefits from the growth of minibatch size. Eqs. (17) also explains why the speedup of CLDNN (Table 3) is much larger than DNN (Table 1). In the same GPU number and synchronization period, t_c of DNN and CLDNN is similar, but t_s of CLDNN is larger than DNN .

4. Conclusions

We implemented four parallel training algorithms, discussed the relationship among them, and evaluated them on speech recognition tasks. The experimental results show that BMUF and ASGD consistently outperform BSP and EASGD. BMUF, in particular, achieved the best performance without frequent synchronization, and even outperformed the single-GPU SGD in some cases. We conjecture that the momentum used in BMUF global model update makes the global model not only related to each local model but also the previous global model. ASGD also achieved pretty good performance and the lager training speedup with the same synchronization period. Profit fully from the asynchronous properties, ASGD is insensitive to the difference of computing capacity of workers, but sensitive to the synchronization period and suffers from the poor reproducibility.

5. Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant No. 61571363).

6. 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 Transactions on Audio Speech & Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[2] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks." in *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, and T. N. Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *ICASSP*, 2012, pp. 4277–4280.

[5] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition." in *INTERSPEECH*, 2013, pp. 3366–3370.

[6] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin, and B. Ramabhadran, "Improvements to deep convolutional neural networks for LVCSR," in *ASRU, 2013 IEEE Workshop on*, 2013, pp. 315–320.

[7] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *ICASSP*, 2013, pp. 8614–8618.

[8] O. Abdel-Hamid, A. R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on Audio Speech & Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[9] A.-r. Mohamed, "Deep neural network acoustic models for ASR," Ph.D. dissertation, University of Toronto, 2014.

[10] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[11] ——, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." in *INTERSPEECH*, 2014, pp. 338–342.

[12] Y. Miao, J. Li, Y. Wang, S. X. Zhang, and Y. Gong, "Simplifying long short-term memory acoustic models for fast training and decoding," in *ICASSP*, 2016, pp. 2284–2288.

[13] H. Sak, A. Senior, K. Rao, and F. Beaufays, "Fast and accurate recurrent neural network acoustic models for speech recognition," *arXiv preprint arXiv:1507.06947*, 2015.

[14] H. Sak, A. Senior, K. Rao, O. Irsoy, A. Graves, F. Beaufays, and J. Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural networks," in *ICASSP*, 2015, pp. 4280–4284.

[15] A. Senior, H. Sak, and I. Shafran, "Context dependent phone models for LSTM RNN acoustic modelling," in *ICASSP*, 2015, pp. 4585–4589.

[16] K. Chen and Q. Huo, "Training deep bidirectional LSTM acoustic model for LVCSR by a context-sensitive-chunk BPTT approach," *IEEE/ACM Transactions on Audio, Speech & Language Processing*, vol. 24, no. 7, pp. 1185–1193, 2016.

[17] Y. Zhang, G. Chen, D. Yu, K. Yao, S. Khudanpur, and J. Glass, "Highway long short-term memory RNNs for distant speech recognition," in *ICASSP*, 2016, pp. 5755–5759.

[18] D. Yu, W. Xiong, J. Droppo, A. Stolcke, G. Ye, J. Li, and G. Zweig, "Deep convolutional neural networks with layer-wise context expansion and attention," in *INTERSPEECH*, 2016, pp. 17–21.

[19] S. Sun, B. Zhang, L. Xie, and Y. Zhang, "An unsupervised deep domain adaptation approach for robust speech recognition," *Neurocomputing*, 2017.

[20] T. Sercu and V. Goel, "Advances in very deep convolutional neural networks for LVCSR," in *INTERSPEECH*, 2016, pp. 3429–3433.

[21] S. Zhang, C. Liu, H. Jiang, S. Wei, L. Dai, and Y. Hu, "Feedforward sequential memory networks: A new structure to learn long-term dependency," *arXiv preprint arXiv:1512.08301*, 2015.

[22] S. Zhang, H. Jiang, S. Xiong, S. Wei, and L. R. Dai, "Compact feedforward sequential memory networks for large vocabulary continuous speech recognition," in *INTERSPEECH*, 2016, pp. 3389–3393.

[23] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *ICASSP*, 2015, pp. 4580–4584.

[24] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *NIPS*, 2012, pp. 1223–1231.

[25] A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, and B. Catanzaro, "Deep learning with COTS HPC systems," in *ICML*, 2013.

[26] 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," in *INTERSPEECH*, 2014, pp. 1058–1062.

[27] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *NIPS*, 2011, pp. 873–881.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*. MIT Press, 1986.

[29] G. Heigold, E. Mcdermott, V. Vanhoucke, and A. Senior, "Asynchronous stochastic optimization for sequence training of deep neural networks," in *ICASSP*, 2014, pp. 5587–5591.

[30] S. Zhang, C. Zhang, Z. You, and R. Zheng, "Asynchronous stochastic gradient descent for DNN training," in *ICASSP*, 2013, pp. 6660–6663.

[31] T. Paine, H. Jin, J. Yang, Z. Lin, and T. Huang, "GPU asynchronous stochastic gradient descent to speed up neural network training," *arXiv preprint arXiv:1312.6186*, 2013.

[32] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *ICASSP*, 2016, pp. 5880–5884.

[33] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[34] H. Ma, F. Mao, and G. W. Taylor, "Theano-mpi: a theano-based distributed training framework," *arXiv preprint arXiv:1605.08325*, 2016.

[35] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of DNNs with natural gradient and parameter averaging," *arXiv preprint arXiv:1410.7455*, 2014.

[36] H. Su and H. Chen, "Experiments on parallel training of deep neural network using model averaging," *arXiv preprint arXiv:1507.01239*, 2015.

[37] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *NIPS*, 2015, pp. 685–693.

[38] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The Kaldi speech recognition toolkit," in *ASRU, 2011 IEEE workshop on*, no. EPFL-CONF-192584, 2011.

[39] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an ASR corpus based on public domain audio books," in *ICASSP*, 2015, pp. 5206–5210.

[40] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 7, pp. 2121–2159, 2011.

[41] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation for distributed deep learning," *arXiv preprint arXiv:1609.08326*, 2016.