1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs

Frank Seide[1], *Hao Fu*[1,2], *Jasha Droppo*[3], *Gang Li*[1], *and Dong Yu*[3]

[1] Microsoft Research Asia, 5 Danling Street, Haidian District, Beijing 100080, P.R.C.
[2] Institute of Microelectronics, Tsinghua University, 10084 Beijing, P.R.C
[3] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{fseide,jdroppo,ganl,dongyu}@microsoft.com, fuhao9202@hotmail.com

Abstract

We show empirically that in SGD training of deep neural networks, one can, at no or nearly no loss of accuracy, quantize the gradients aggressively—to but *one bit* per value—if the quantization error is carried forward across minibatches (error feedback). This size reduction makes it feasible to parallelize SGD through data-parallelism with fast processors like recent GPUs.

We implement data-parallel deterministically distributed SGD by combining this finding with AdaGrad, automatic minibatch-size selection, double buffering, and model parallelism. Unexpectedly, quantization *benefits* AdaGrad, giving a small accuracy gain.

For a typical Switchboard DNN with 46M parameters, we reach computation speeds of 27k frames per second (kfps) when using 2880 samples per minibatch, and 51kfps with 16k, on a server with 8 K20X GPUs. This corresponds to speed-ups over a single GPU of 3.6 and 6.3, respectively. 7 training passes over 309h of data complete in under 7h. A 160M-parameter model training processes 3300h of data in under 16h on 20 dual-GPU servers—a 10 times speed-up—albeit at a small accuracy loss.

1. Introduction and Related Work

At present, the best context-dependent deep-neural-network HMMs, or CD-DNN-HMMs [1, 2], are trained primarily with error back-propagation, or BP. BP is a form of stochastic gradient descent, or SGD. For production-size models and corpora, this is time consuming and can take many days or weeks, even on the currently fastest hardware, graphics processing units (GPUs). While attempts at parallelizing SGD training across multiple compute nodes were successful for sparsely connected networks like those used for image processing, success has been moderate for speech DNNs which are fully connected.

For example, Google's DistBelief system successfully utilizes 16,000 cores for the ImageNet task [3] through asynchronous SGD, an implementation of Hogwild [4]; while for a speech model with 42M parameters, a 1,600-core DistBelief [5] is only marginally faster than a single recent GPU; and [6] achieved a 28-fold speed-up with 64 GPUs for their 1.9B-parameter vision network, while [7] reports a 3.2-times speed-up using 4 GPUs for speech.

This paper focuses on parallelization in a data-parallel fashion. In data parallelism, each minibatch is split over multiple compute nodes. Each node computes a sub-gradient on its sub-minibatch. These sub-gradients, of the same dimension as the full model, must be summed over all nodes and redistributed.

Applied directly to typical training configurations, this process is infeasible due to the high bandwidth that it takes to exchange sub-minibatch gradients across nodes. Avenues for improving efficiency for data parallelism are to increase the minibatch size and to reduce how much data gets exchanged [8].

We focus on the latter and propose to reduce bandwidth by *aggressively quantizing* the sub-gradients—to but *one bit per value*. We show that this does not or almost not reduce word accuracies—but only if the *quantization error is carried forward across minibatches*, i.e. the error in quantizing the gradient in one minibatch is added (fed back) to the gradient of the next minibatch. This is a common technique in other areas, such as sigma-delta modulation for DACs [9], or image rasterization. It is a key difference to the well-known R-prop method [27].

Some prior work on speeding up model training considered changes of model structure and training approach, e.g. [10, 11] where the network was factored into a hierarchy; low-rank approximations [12, 13]; second-order methods ("Hessian-Free") [14, 15]; model averaging [16]; or ADMM which cleverly tweaks the objective function for better parallelizability [17, 18]. The last three typically require more data passes, but make up for it through good parallelization properties.

In the paper at hand, we aim at unchanged convergence behavior. Also, unlike Hogwild/ASGD [4, 5], we desire deterministic behavior. In this category, an alternative to data parallelism is model parallelism, where models are distributed over nodes [5, 8]. One can also parallelize over layers [19]: Each GPU processes one or more consecutive layers, where data flows up and down through the layers between GPUs, and, as a consequence, gradients only become available at a delay of one or more minibatches (depending on the layer). This achieved a 3.3-times speed-up on 4 GPUs, but it does not scale beyond the number of layers, and load balancing is problematic. That work showed, however, that delayed updates can work, and motivated the double-buffering technique we apply in this paper.

We will next describe data-parallel DNN training. Then, Section 3 will introduce the 1-bit quantization approach, and Section 4 the data-parallel SGD system we implemented based on this. Finally, Section 5 will give experimental results for quantization, interaction with AdaGrad, impact of double buffering, and combination with model parallelism.

2. Data-Parallel Deterministically Distributed SGD Training

A deep neural network (DNN) is a conventional multi-layer perceptron (MLP [20]) with many layers, where training is commonly initialized by a pretraining algorithm [21, 22, 23]. A CD-DNN-HMM models the posterior probability $P(s|o)$ of a tied triphone state, or senone s [24, 1], given an observation vector o. For details, please see, for example, [23].

The best DNNs to this date are often trained using the common error back-propagation (BP) technique [25], which is a

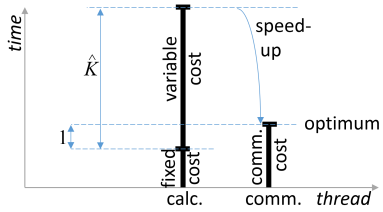

Figure 1: *Illustration of optimal number of compute nodes $\hat{K}$. Parallelizing over K nodes reduces variable cost K-fold. K is optimal when computation and communication cost are equal.*

form of stochastic gradient descent. Even when other techniques are used, such as the Hessian-free method [26, 15, 12], BP still constitutes a significant portion of the training time.

In its typical minibatch form, BP can be written as:

$$\lambda(t+N) = \lambda(t) + \epsilon(t) \cdot G(t) \quad (1)$$

$$G(t) = \sum_{\tau=t}^{t+N-1} \frac{\partial \mathcal{F}_\lambda(o(\tau))}{\partial \lambda}\bigg|_{\lambda=\lambda(t)} \quad (2)$$

where $\lambda(t)$ denotes the model at "current" sample index t which increases in steps of the minibatch size N (e.g. 1024), and $\mathcal{F}_\lambda$ is the partial gradient of the objective function for sample vector $o(\tau)$. The learning rate $\epsilon(t)$ is variable as training progresses.

2.1. Data-Parallel Distributed SGD

Eq. (2) can be parallelized by splitting the sum over compute nodes: Each node processes part of each minibatch—*data parallelism*—and the sub-minibatch gradients are then summed up over all nodes. Generally, an optimal number of nodes[1], $\hat{K}$, is one where computation and data exchange happen *concurrently with perfect overlap*; that is, simultaneously saturating the communcation channel and the processing resources:[2]

$$T_{\text{calc}}(\hat{K}) = T_{\text{comm}}(\hat{K}). \quad (3)$$

T_{calc} and T_{comm} are the time *per minibatch* for concurrent per-node calculation and inter-node communication, respectively [8]. If we break T_{calc} further into *variable* (parallelizable) and *fixed computation cost*, then the optimal number of nodes $\hat{K}$ for data parallelism is given, as illustrated by Fig. 1, as:

$$\hat{K} = \frac{N/2 \cdot T^{\text{frm}}_{\text{calc}} + C \cdot T^{\text{post}}_{\text{calc}}}{\frac{1}{Z} \cdot T^{\text{float}}_{\text{comm}} - T^{\text{upd}}_{\text{calc}}} \quad (4)$$

with the *variable* (parallelizable) costs $T^{\text{frm}}_{\text{calc}}$ (cost to process one frame of data[3], dominated by three large matrix products per layer) and $T^{\text{post}}_{\text{calc}}$ (component-wise gradient post-processing steps of which there are C, e.g. 3 for momentum + AdaGrad accumulation and application). $T^{\text{float}}_{\text{comm}}$ is the communication cost for exchanging sub-gradients as plain "float" values[4], and Z a compression factor due to using less bits per gradient value. In particular, $Z = 32$ for quantization to 1 bit. Lastly, $T^{\text{upd}}_{\text{calc}}$ is the cost of adding the gradient to the model, a component-wise operation that is not parallelized—it is *fixed* w.r.t. K.

Roughly, for a model with M parameters, $T^{\text{frm}}_{\text{calc}} \sim \frac{M}{\text{FLOPS}}$, $T^{\text{post}}_{\text{calc}}$ and $T^{\text{upd}}_{\text{calc}} \sim \frac{M}{r}$ with RAM bandwidth r, and $T^{\text{float}}_{\text{comm}} \sim \frac{M}{b}$ with peer-to-peer bandwidth b. Importantly, unlike $T^{\text{frm}}_{\text{calc}}$, $T^{\text{post}}_{\text{calc}}$ and $T^{\text{upd}}_{\text{calc}}$ are memory-bound, in particular on GPUs.

2.2. Double Buffering with Half Batches

The ominous /2 in Eq. (4) stems from double buffering. To achieve concurrent computation, we break each minibatch in half and exchange sub-gradients of one half-minibatch while computing the sub-gradients of the next half-minibatch, using a model that is outdated by $N/2$ samples (*delayed update* [19, 8]). We hope by keeping the overall delay in the order of a minibatch size, convergence will not change fundamentally [8].

2.3. Potential Faster-Than-Fixed-Cost Communication

Eq. (4) breaks down when communication cost falls below the fixed cost. Then we can no longer saturate the communication channel; the speed-up is limited by the fixed cost.[5] This happened to our 1-bit SGD with earlier, less optimized code. In this case, double buffering with half-minibatches no longer makes sense, as it masks communication cost at the expense of an additional fixed cost, which is now higher.

2.4. Relation to Hogwild/ASGD

All of the above also applies to the Hogwild method, also known as "asynchronous SGD" [4, 5, 7]. Hogwild differs in that it uses an unsynchronized gradient exchange (via a parameter server). It is another form of delayed update where the delay varies non-deterministically across model parameters. This is beneficial for inhomogeneous server farms or shared networks, but it does not improve parallelizability in a fundamental way.

3. 1-Bit SGD with Error Feedback

The core of this paper is our approach to reducing the bandwidth requirement for the data exchanges in data parallelism: We (1) design the system to exchange *gradients* (as opposed to model parameters) and (2) *quantize* those gradients during data exchange—aggressively so.

The inevitable quantization errors should, however, not just be left to the SGD process to be caught and corrected; we find that it does not work well and can lead to divergence.

Instead, inspired by Sigma-Delta modulation [9], when quantizing a gradient parameter $G_{ij\ell}(t)$, we keep the quantization error $\Delta_{ij\ell}(t)$ and *add it into the respective next minibatch gradient* before quantization:

$$G^{\text{quant}}_{ij\ell}(t) = \mathcal{Q}(G_{ij\ell}(t) + \Delta_{ij\ell}(t-N))$$

$$\Delta_{ij\ell}(t) = G_{ij\ell}(t) - \mathcal{Q}^{-1}(G^{\text{quant}}_{ij\ell}(t))$$

where $\mathcal{Q}(\cdot)$ is the quantization function and the $G^{\text{quant}}_{ij\ell}$ are packed integers representing the quantizated values. This *error-feedback* ensures that all gradients are eventually added up into the model with (in the limit) full accuracy; just split across multiple minibatches—another form of delayed update. We find that as long as error feedback is used, we can quantize all the way to 1 bit at no or nearly no loss of accuracy.

[1] A node can be defined as a GPU, a CPU core, or a multi-core server.

[2] If the communication channel is not saturated, then the system could be improved by parallelizing more. If the processing resources are not saturated, it could be improved by parallelizing (communicating) less or by processing more data.

[3] Here we ignore the fact that both computation and communication become less efficient if we reduce the data size, due to caching and/or overhead. The results section will show actual time measurements.

[4] Note that communication cost is independent of K, cf. Section 3.1.

[5] Noone escapes Amdahl's law.

For our 1-bit implementation, we find that using a constant quantization threshold of 0 is a good (and cheap) choice, whereas the reconstruction values used by the unquantizer $\mathcal{Q}^{-1}(\cdot)$ are tied within each weight-matrix *column* (j, ℓ). The two values per column are recomputed as to minimize the square quantization error and transmitted in each data exchange.

3.1. Aggregating the Gradients

The algorithm we use for aggregating the sub-gradients over compute nodes [8] is of $\mathcal{O}(1)$ w.r.t. the number of nodes: Each compute node is responsible for aggregating a $1/K$-th subset of model parameters, which it will receive in quantized form from all peer nodes (K concurrent transfers of $M \cdot \frac{K-1}{K}$ values). These are then summed up in *unquantized* form, post-processed (AdaGrad, momentum), and redistributed to all peers, again *quantized*. Thus, each minibatch gradient undergoes quantization *twice*. The first quantization is applied to sub-gradients which are summed up, reducing the quantization error through averaging. The second quantization happens after AdaGrad, where gradient values are in more homogeneous numeric range.

4. System Description

We implemented data-parallel deterministically distributed SGD by combining 1-bit quantization and double buffering with automatic minibatch-size selection, AdaGrad, and model parallelism. We want to briefly describe these other aspects.

Eq. 4 reveals the main avenues to increase parallelizablity: (a) growing N—*maximizing minibatch size*; (b) increasing Z—*data compression*; and (c) reducing fixed cost $T^{\text{upd}}_{\text{calc}}$. We addressed (b) by quantization in section 3.

For (a), we find that at any given point in the training, the minibatch size N has an upper limit above which convergence slows notably or fails [19, 8]. Thus, every 24h of data, we process the next ≈45 minutes at different minibatch sizes and pick the largest N that does not hurt convergence, based on training-set frame accuracy. We find that more mature models allow for larger N. So do smaller learning rates, so we also use a gradually decaying learning-rate profile that was determined automatically using frame accuracy on a cross-validation set on an earlier configuration. Lastly, we use AdaGrad—a technique to normalize gradients by their standard deviation over time or recent samples [30, 31]. AdaGrad leads to faster convergence and allows to increase the minibatch size earlier.

Our system can apply AdaGrad at three different places: Locally on each node before quantization (which may benefit quantization at the risk of introducing inconsistencies across nodes); during data exchange (risking interference from quantization) after aggregation; and after momentum smoothing (saving memory and fixed cost while reducing the effect of AdaGrad due to peaks being smoothed out). We find AdaGrad responds best to quantized, unsmooted gradients.

To address (c), the fixed cost, and to benefit from dual-GPU servers, we combine our system with model parallelism. Model parallelism distributes model parameters over multiple GPUs and can perfectly parallelize component-wise operations. It can also reduce the variable cost, albeit with suboptimal efficiency.

5. Experimental Results

We evaluate our 1-bit data-parallel deterministically distributed SGD training primarily on a Switchboard speech-to-text system. The CD-DNN-HMM in this system is trained on the 309-hour SWBD-I training set [28]. The model has 7 hidden layers of dimension 2048, and an output dimension of 9304, a total of $M = 46$M model parameters. The test set is Hub-5'00 (1831 utterances). We will also show a few results on different configurations as noted. Our hardware consists of a server equipped with 8 NVidia Tesla K20Xm GPU cards, and a server farm of 24 dual-K20Xm servers connected through Infiniband.

Table 1: *Sub-batch computation time and $T^{\text{frm}}_{\text{calc}}$ (variable computation cost per frame) for different sub-batch sizes.*

sub-batch size	256	512	1024	2048	4096	8192
sub-b. time [ms]	59	89	143	260	490	955
$T^{\text{frm}}_{\text{calc}}$ [μs]	156	137	122	118	115	114

5.1. Cost Measurements

First, we want to provide measurements for our three cost factors. Eq. (4) had assumed that $T^{\text{frm}}_{\text{calc}}$ does not depend on the number of nodes, but Table 1 shows that it indeed does vary to some degree with the sub-batch size in which computations are performed. These times were measured on a single K20X using momentum and AdaGrad but no data parallelism. For this configuration, the cost that does not depend on the sub-batch size, that is, the gradient post-processing (momentum, two AdaGrad steps) and fixed cost (model update), was measured as $C \cdot T^{\text{post}}_{\text{calc}} + T^{\text{upd}}_{\text{calc}} = 18.2$ ms.

On our server farm, $Z^{-1} \cdot T^{\text{float}}_{\text{comm}}$ over Infiniband/MPI fluctuates in the range 3 to 10 ms (with $Z = 32$), a little behind our expectations for an 8 GB/s Infiniband link. With quantization, our fixed cost $T^{\text{upd}}_{\text{calc}}$ is in the range of 9 ms. With half-batch double buffering, the fixed cost per minibatch is $2 \cdot T^{\text{upd}}_{\text{calc}}$, while without it, communication and computation get serialized such that $T^{\text{float}}_{\text{comm}}$ has to be added into the fixed cost. Two-way model parallelism cuts the fixed cost in half.

5.2. Effect of 1-bit Quantization

Table 2 shows the effect of 1-bit quantization on our main configuration as well as two alternate setups, one using rectified linear units (ReLUs) instead of sigmoids, and one using singular-value decomposed weight matrices [13] and a low-latency front-end (SVD-LL). Shown are word error rates (WERs) for Hub-5'00 and training-set frame accuracies. These setups use a fixed two-step learning rate schedule per [2] and no AdaGrad. Quantized setups are data-parallel with $K = 4$ nodes, except for the first 24h of data, which was processed without parallelism or quantization (cold start). 1-bit quantization works well across all setups, at minor but consistent impact on training-set frame accuracy. WER actually improves a little for the main setup, which is probably noise. In all of these, error feedback is essential. Without it, training quickly diverges.

The table also shows that double buffering has minor impact on accuracy—it undoes the small gain on the main setup, while it does not change the WER of the ReLU setup.

Table 2: *Effect of 1-bit quantization and double buffering. Shown are Hub-5'00 WER and, in parentheses, training-set frame accuracy.*

	WER / training frame acc [%] on		
	main setup	alternate setups	
configuration		ReLU	SVD-LL
baseline (not parallel)	16.5 (55.6)	16.5 (62.8)	17.6 (61.5)
+ 1-bit quant/data para.	16.3 (55.5)	16.6 (61.7)	-
+ double buffering	16.5 (55.3)	16.6 (62.4)	17.7 (61.3)

Table 3: *Effect of AdaGrad at several points and double buffering. Shown are WER on Hub-5'00 and training-set frame accuracy (in parentheses), and end-to-end BP training times.*

AdaGrad applied to...	WER [%]	GPUs	time
momentum-smoothed gradient	16.5 (57.4)	1	41h
raw gradient (not parallelized)	16.2 (58.2)	1	35h
partial gradients (parallel, 4 nodes)	16.1 (57.4)	4×2	-
aggregate gradient (4 nodes)	15.8 (59.1)	4×2	8.1h
+ MB size tuning 3 x less often	15.9 (59.2)	4×2	7.3h
+ double buffering (DB)	15.8 (59.4)	4×2	7.3h
vs. no DB for MB-size selection	15.9 (59.1)	4×2	6.3h

5.3. When to do AdaGrad?

Table 3 analyzes where to apply AdaGrad in the process. First, we can see that applying AdaGrad to the raw gradients before momentum, rather than the momentum-smoothed gradient, leads to higher training frame accuracy and 0.3 points better WER. We believe that this is because momentum smoothing reduces the standard deviation and thus the effect of AdaGrad.

Row "partial gradients" adds data parallelism over $K = 4$ compute nodes, combined with 2-GPU model parallelism in each compute node (more on that in Section 5.5). AdaGrad is applied locally before quantization. While it leads to a small WER gain, the training frame accuracy drops a little.

Shifting AdaGrad to after data aggregation, i.e. applying it to quantized gradients (row "aggregate gradient"), presents us with a pleasant surprise: frame accuracy jumps 1.7 points, and WER drops 0.3 points. We have no good explanation for this, other than quantization distributes outliers over multiple minibatches, so that they impact the standard-deviation estimate less. We take away that AdaGrad is best applied here.

This configuration reduces end-to-end training time from 35h to 8.1h (where the first 24h of data did not use data parallelism/quantization and consumed 22 min alone).

5.4. Impact of MB-Size Selection and Double Buffering

The initial change from 41 to 35h is due to automatic minibatch-size selection, which selected larger N. MB-size selection itself has a cost. Doing it only every 72h of data instead of 24h reduces the time further to 7.3h, losing 0.1 point WER.

The next row ("+ double buffering") of Table 3 shows that double buffering (DB) does not give further speed-up; however, we find that smaller MB sizes are selected. Disabling DB only for MB-size selection reverts this, and now, DB reduces the runtime to 6.3h. This represents a total 5.5-times speed-up from using $4 \times 2 = 8$ GPUs instead of one. As we hoped, half-batch DB is both efficient and does not seem to change convergence.

Table 4 shows the training speeds with and without half-batch DB over minibatch sizes N for 4×2 GPUs. DB up to 21% faster ($N = 32$k).

5.5. Combination with Model Parallelism

Table 5 shows results for two-way model parallelism (MP) for our two systems, the compute server "farm" with Infiniband connection, and the single "8-GPU" server. Without data parallelism, a second GPU (1×2) is 55% efficient. Comparing the same number of GPUs, MP only helps in one configuration, the communication-bound minibatch size 2880 with 16 GPUs. In all other settings, MP is less efficient due to caching. The reason we used MP throughout this paper is that using earlier, less optimized code versions, we had measured a different balance.

Table 4: *Training speed in 1000 frames per second (kfps) over minibatch size, without and with double buffering, 4 × 2 GPUs.*

N:	1k	2k	2.8k	4k	8k	16k	32k	64k	92k
no DB	-	23	28	32	36	38	39	44	45
DB	15	23	28	34	39	45	47	49	50

Table 5: *Combining data and model parallelism.*

data× model parallelism	training speed [kfps] for $N =$ 2880		16384	
	farm	8-GPU	farm	8-GPU
1×1	7.5	7.4	8.2	8.0
1×2	11.7	11.7	12.7	12.6
4×2	26.9	25.0	42.3	40.9
8×1	26.9	26.5	51.6	50.6
8×2	34.8	-	72.5	-
16×1	30.2	-	77.9	-

5.6. Training a Production-Scale Model

Table 6 shows results for a production-scale model with 160M parameters trained on 3300h of multi-bandwidth data including Switchboard/Fisher and 1300h of wideband lectures. We test on Hub-5'00 and RT03S of Switchboard, internal tele-conference recordings, and three IWSLT sets (dev10, dev12, tst10) [29]. The two data passes use GMM and DNN alignments (generated with a slightly outdated DNN for time reasons), respectively. DB is not used. For this large model, model parallelism is about 90% efficient. The '1×1' setups use fixed $N = 1$k, but '1×1 realign' started from an earlier first pass model that used MB-size control and has on av. 0.1 worse WER.

The '10×2' setup uses conservative MB-size control and achieves a 6 to 7 times speed-up using 20 GPUs. '20×2' does more aggressive MB-size control every 72h, and achieves over 10-fold speed-up, but at more notable accuracy loss.

6. Conclusion

We have shown that 1-bit quantization allows to significantly reduce data-exchange bandwidth for data-parallel SGD at no or nearly no loss of accuracy, making data-parallel distribution of SGD feasible even with modern fast hardware (GPUs). For this to work, quantization-error feedback is essential. Quantization interacts well with AdaGrad. A 7 data-pass 309h-Switchboard BP training of a 46M-parameter model completes in under 7 hours. A production-scale model of 160M parameters completes one pass through 3300h of data in under 24 hours, and under 16 if a small accuracy loss is acceptable.

In [8], we had concluded that dramatic speed-ups from parallelizing standard SGD are not to be expected for speech-scale DNNs. The proposed 1-bit quantization improves the situation somewhat, but we still believe that the true breakthrough will have to come from a more fundamental change of the training algorithm that allows for greater parallelizability by its nature.

7. Acknowledgements

We'd like to thank John Langford for pointing us to the $\mathcal{O}(1)$ all-reduce algorithm in Section 3.1.

Table 6: *Training a 160M-parameter model over 3300h. Shown is WER in % over 5 test sets, training time for two data passes.*

data × model parallelism	Hub-5 '00	RT03S FSH	RT03S SWB	IWSLT all	tele-conf.	time (3300h)
1×1	14.5	15.1	21.2	15.0	19.4	157h
+ realign	13.2	14.1	19.8	14.1	18.5	155h
10×2	14.2	14.8	20.8	14.9	19.1	24.0h
+ realign	13.2	14.1	19.8	14.2	18.6	21.5h
20×2	14.3	14.9	20.8	15.1	19.2	15.6h
+ realign	13.1	14.4	20.1	14.5	18.7	14.0h

8. References

[1] D. Yu, L. Deng, and G. Dahl, "Roles of Pretraining and Fine-Tuning in Context-Dependent DNN-HMMs for Real-World Speech Recognition," NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Dec. 2010.

[2] F. Seide, G. Li, and D. Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks," Interspeech, 2011.

[3] Q.-V. Le, M.-A. Ranzato, R. Monga, M. Devin, K. Chen, G.-S. Corrado, J. Dean, and A.-Y. Ng, "Building High-Level Features Using Large Scale Unsupervised Learning," ICML, 2012.

[4] F. Niu, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," arXiv preprint arXiv:1106.5730, 2011.

[5] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M.-A. Ranzato, A. Senior, P. Tucker, K. Yang, A. Y. Ng, "Large Scale Distributed Deep Networks," NIPS, 2012.

[6] A. Coates, B. Huval, T. Wang, D.-J. Wu, and A.-Y. Ng, "Deep Learning with COTS HPC Systems," ICML, 2013.

[7] S. Zhang, C. Zhang, Z. You, R. Zheng, and B. Xu, "Asynchronous Stochastic Gradient Descent for DNN Training," ICASSP, 2013.

[8] F. Seide, H. Fu, J. Droppo, G. Li, D. Yu, "On Parallelizability of Stochastic Gradient Descent for Speech DNNs," ICASSP 2014.

[9] "Delta-Sigma Modulation," Wikipedia, `http://en.wikipedia.org/wiki/Delta-sigma_modulation`.

[10] H. Franco *et al.*, "Context-Dependent Connectionist Probabilty Estimatation in a Hybrid Hidden Markov Model–Neural Net Speech Recognition System," Computer Speech and Language, vol. 8, pp. 211–222, 1994.

[11] P. Zhou, C. Liu, Q. Liu, L. Dai, and H. Jiang, "A Cluster-Based Multiple Deep Neural Networks Method for Large Vocabulary Continuous Speech Recognition," ICASSP, 2013.

[12] T.-N. Sainath, B. Kingsbury, H. Soltau, and B. Ramabhadran, "Optimization Techniques to Improve Training Speed of Deep Neural Networks for Large Speech Tasks," IEEE Trans. on Audio, Speech, and Language Processing, Vol. 21, No. 11, Nov. 2013.

[13] J. Xue, J. Li, and Y. Gong, "Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition," Interspeech 2013.

[14] S. Wiesler, J. Li, and J. Xue, "Investigations on Hessian-Free Optimization for Cross-Entropy Training of Deep Neural Networks," Interspeech, 2013.

[15] B. Kingsbury, T. Sainath, and H. Soltau, "Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Optimization," Interspeech, 2012.

[16] X. Zhang, J. Trmal, D. Povey, S. Khudanpur, "Improving Deep Neural Network Acoustic Models Using Generalized Maxout Networks," ICASSP 2014.

[17] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," in Foundations and Trends in Machine Learning, Vol. 3, No. 1 (2010) 1–122.

[18] Q. Huo, Z. Yan, K. Chen, "Deep Learning Using Alternating Direction Method of Multipliers," U.S. Patent Application, filed on 4/8/2014.

[19] X. Chen, A. Eversole, G. Li, D. Yu, and F. Seide, "Pipelined Back-Propagation for Context-Dependent Deep Neural Networks," Interspeech, 2012.

[20] F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms", Spartan Books, Wash. DC, 1961.

[21] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large Vocabulary Speech Recognition," IEEE Trans. Speech and Audio Proc., Special Issue on Deep Learning for Speech and Language Processing, 2011.

[22] G. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets", Neural Computation, vol. 18, pp. 1527–1554, 2006.

[23] F. Seide, G. Li, X. Chen, and D. Yu, "Feature Engineering in Context-Dependent Deep Neural Networks for Conversational Speech Transcription," Proc. ASRU, Waikoloa Village, 2011.

[24] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," ICASSP, 2009.

[25] D. Rumelhart, G. Hinton, and R. Williams, "Learning Representations By Back-Propagating Errors," Nature, vol. 323, Oct. 1986.

[26] J. Martens, "Deep learning via Hessian-free optimization," ICML, 2010.

[27] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the Rprop algorithm," International Conference on Neural Networks, 1993.

[28] J. Godfrey and E. Holliman, "Switchboard-1 Release 2," Linguistic Data Consortium, Philadelphia, 1997.

[29] "Evaluation Campaign," IWSLT 2013, `http://www.iwslt2013.org/59.php`.

[30] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," `http://www.cs.berkeley.edu/~jduchi/projects/DuchiHaSi10.pdf`, 2010.

[31] A. Senior, G. Heigold, M.-A. Ranzato, K. Yang, "An Empirical Study of Learning Rates in Deep Neural Networks for Speech Recognition," ICASSP, 2013.