# Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training

**Yujun Lin** [*]
Tsinghua University
yujunlin@stanford.edu

**Song Han**
Stanford University
Google Brain
songhan@stanford.edu

**Huizi Mao**
Stanford University
huizi@stanford.edu

**Yu Wang**
Tsinghua University
yu-wang@mail.tsinghua.edu.cn

**William J. Dally**
Stanford University
NVIDIA
dally@stanford.edu

## Abstract

Large-scale distributed training requires significant communication bandwidth for gradient exchange. The intensive gradient communication limits the scalability of multi-node training, and requires expensive high-bandwidth network infrastructure. The situation gets even worse with distributed training on mobile devices (federated learning), which suffers from higher latency, lower throughput, intermittent poor connections. In this paper, we propose Deep Gradient Compression that can reduce the communication bandwidth by two orders of magnitude. We proposed four techniques that also preserves the accuracy: momentum correction, local gradient clipping, momentum factor masking, and warm-up training. We extensively applied Deep Gradient Compression to many types of machine learning tasks including image classification, speech recognition, and language modeling with multiple datasets including Cifar10, ImageNet, Penn Treebank, and Librispeech Corpus. On these scenarios, Deep Gradient Compression achieved a compression ratio from 270$\times$ to 600$\times$ without losing accuracy, cutting the gradient size of ResNet-50 from 97MB to 0.35MB, DeepSpeech from 488MB to 0.74MB. Deep gradient compression enables large-scale distributed training on inexpensive commodity 1Gbps Ethernet and facilitates distributed training on mobile.

## 1 Introduction

Large-scale distributed training improves the productivity of training deeper and larger models [1–4]. Synchronous stochastic gradient descent (SGD) is widely used for distributed training. By increasing the number of training nodes and taking advantage of data parallelism, the total computation time of the forward-backward passes on the same size training data can be dramatically reduced. However, gradient exchange is costly and dwarfs the savings of computation time [5, 6], especially for recurrent neural networks (RNN) where the computation-to-communication ratio is low. Therefore, the network bandwidth becomes a significant bottleneck for scaling up distributed training. This bandwidth problem gets even worse when distributed training is performed on mobile devices (such as federated learning [7, 8]). Training on mobile devices is appealing due to better privacy and

[*]Work done while at Stanford CVA lab.

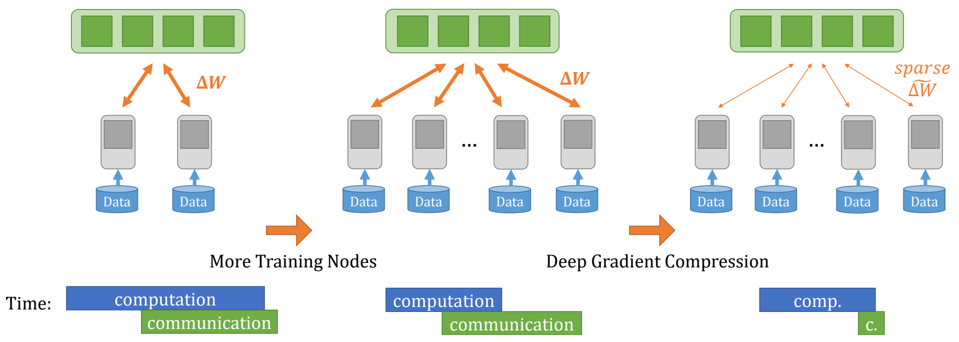


Figure 1: Deep Gradient Compression reduces the communication bandwidth, improves the scalability and speed up for distributed training.

better personalization [9], but a critical problem is that those mobile devices suffer from even lower bandwidth and intermittent Internet connection.

There have been many approaches to overcome the communication bottleneck in distributed training. Asynchronous SGD accelerates the training by removing gradient synchronization and updating parameters immediately once a node has completed back-propagation [5, 10, 11]. Quantizing the gradients to low-precision values are also extensively studied [6, 12, 13]. Aji and Heafield [14] proposed a gradient sparsification method to effectively saves 99% of gradients exchange while incurring 0.3% loss of BLEU score on a machine translation task.

*Deep Gradient Compression* (DGC) solves the communication bandwidth problem by compressing the gradients which effectively reduces the communication bandwidth. DGC has several benefits. First, DGC can save communication time, achieve better speedup and scalability. Second, DGC reduces the infrastructure cost not having to buy expensive switches (e.g. Infiniband). Third, DGC helps federated learning on mobile devices.

We provide several techniques to ensure that DGC incurs no loss of accuracy. We propose *momentum correction* and *local gradient clipping* on top of the gradient sparsification to maintain model performance. Furthermore, we propose *momentum factor masking* and *warmup training* to overcome the staleness problem caused by reduced communication.

We empirically verified Deep Gradient Compression on a wide range of tasks, models, and datasets: CNN for image classification (with Cifar10 and ImageNet), RNN for language modeling (with Penn Treebank) and speech recognition (with Librispeech Corpus). We showed that up to 600× gradient size reduction can be achieved on all these models without any accuracy degradation, which is an order of magnitude higher than previous work.

## 2 Deep Gradient Compression

### 2.1 Gradient Sparsification

We reduce the communication bandwidth by sending only the important gradients (sparse update). We use magnitude as a simple heuristics for importance: only gradients larger than a threshold are transmitted. To avoid loosing information, we accumulate the rest of the gradients locally. Eventually, these gradients become large enough to be transmitted. Thus, we send the large gradients immediately but eventually send all of the gradients over time. The method is shown in Algorithm 1.

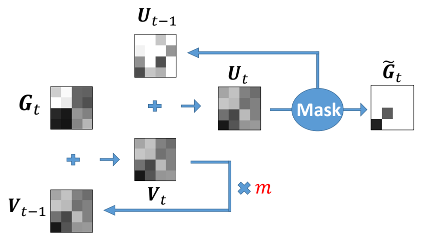
The insight is that local gradient accumulation is equivalent to increasing the batch size over time. Let $F(w)$ be the loss function which we want to optimize, and Synchronous Distributed SGD performs the following update with $N$ training nodes in total:

$$F(w) = \frac{1}{|\chi|}\sum_{x\in\chi} f(x,w), \qquad w_{t+1} = w_t - \eta\frac{1}{Nb}\sum_{k=0}^{N}\sum_{x\in\mathcal{B}_{k,t}} \nabla f(x,w_t) \tag{1}$$

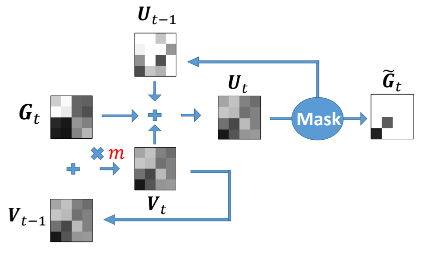
where $\chi$ is the training dataset, $w$ are the weights of a network, $f(x,w)$ is the loss computed from samples $x \in \chi$, $\eta$ is the learning rate, $N$ is the number of training nodes, and $\mathcal{B}_{k,t}$ for $0 \le k < N$ is a sequence of $N$ minibatches sampled from $\chi$ at iteration $t$, each of size $b$.

**Algorithm 1** Gradient Sparsification on node $k$

```
Input: dataset $\chi$
Input: minibatch size $b$ per node
Input: the number of nodes $N$
Input: optimization function $SGD$
Input: init parameters $w = \{w[0], w[1], \cdots, w[M]\}$
 1: $G^k \leftarrow 0$
 2: for $t = 0, 1, \cdots$ do
 3:     $G^k_t \leftarrow G^k_{t-1}$
 4:     for $i = 1, \cdots, b$ do
 5:         Sample data $x$ from $\chi$
 6:         $G^k_t \leftarrow G^k_t + \frac{1}{Nb} \nabla f(x; w_t)$
 7:     end for
 8:     for $j = 0, \cdots, M$ do
 9:         Select threshold: $thr \leftarrow s\%$ of $|G^k_t[j]|$
10:         $Mask \leftarrow |G^k_t[j]| > thr$
11:         $\widetilde{G}^k_t[j] \leftarrow G^k_t[j] \odot Mask$
12:         $G^k_t[j] \leftarrow G^k_t[j] \odot \neg Mask$
13:     end for
14:     All-reduce $G^k_t$ : $G_t \leftarrow \sum_{k=1}^{N} sparse(\widetilde{G}^k_t)$
15:     $w_{t+1} \leftarrow SGD(w_t, G_t)$
16: end for
```



(a) Vanilla momentum correction



(b) Nesterov momentum correction

Figure 2: Gradient Sparsification with momentum correction

Consider the weight value $w^{(i)}$ of $i$-th position in $w$. After $T$ iterations, we have

$$w^{(i)}_{t+T} = w^{(i)}_t - \eta T \cdot \frac{1}{NbT} \sum_{k=0}^{N} \left( \sum_{\tau=0}^{T-1} \sum_{x \in \mathcal{B}_{k,t+\tau}} \nabla^{(i)} f(x, w_{t+\tau}) \right) \tag{2}$$

Equation 2 shows that local gradient accumulation can be considered as increasing the batch size from $Nb$ to $NbT$ (the second summation over $\tau$), where $T$ is the length of the *update interval* between two iterations at which the gradient of $w^{(i)}$ is sent. Learning rate scaling [15] is commonly used technique to deal with large minibatch. This is automatically satisfied in Equation 2 where the $T$ in the learning rate $\eta T$ and batch size $NbT$ are canceled out.

## 2.2 Improving the Local Gradient Accumulation

Without care, sparse update will greatly harm convergence when sparsity is extremely high (99.9%). For example, Algorithm 1 incurred more than 1% loss of accuracy on Cifar10 dataset. We find momentum correction and local gradient clipping can mitigate this problem.

**Momentum Correction** Momentum SGD is widely used in place of vanilla SGD. Distributed training with vanilla momentum SGD on $N$ training nodes follows [16],

$$u_t = m u_{t-1} + \sum_{k=1}^{N} (\nabla_{k,t}), \quad w_{t+1} = w_t - \eta u_t \tag{3}$$

where $m$ is the momentum, $N$ is the number of training nodes, and $\nabla_{k,t} = \frac{1}{Nb} \sum_{x \in \mathcal{B}_{k,t}} \nabla f(x, w_t)$. However, Algorithm 1 doesn't directly apply to SGD with the momentum term, since it ignores the discounting factor between the sparse update interval $T$. When the gradient sparsity is high, the update interval $T$ dramatically increases, and thus the significant momentum effect will harm the model performance. To avoid this error, we need momentum correction on top of Algorithm 1 to make sure the sparse update is equivalent to the dense update as in Equation (3).

If we regard the velocity $u_t$ in Equation (3) as "gradient", the second term of Equation (3) can be considered as the vanilla SGD for the "gradient" $u_t$. The local gradient accumulation is proved to be effective for the vanilla SGD in Section 2.1. Therefore, we can locally accumulate the velocity $u_t$ instead of the real gradient $\nabla_{k,t}$ to migrate Algorithm 1 to approach Equation (3):

$$u_{k,t} = m u_{k,t-1} + \nabla_{k,t}, \quad v_{k,t} = v_{k,t-1} + u_{k,t}, \quad w_{t+1} = w_t - \eta \sum_{k=1}^{N} sparse(v_{k,t}) \tag{4}$$

where the first two terms are the corrected local gradient accumulation, and the accumulation result $v_{k,t}$ is used for the subsequent sparsification and communication. We refer to this migration as the *momentum correction*. Beyond the vanilla momentum SGD, we also look into Nesterov in Appendix B.1. The momentum correction for Nesterov is similar to momentum SGD, as shown in Figure 2(a) and 2(b). The Algorithm 1 with momentum correction is provided in Appendix B.

**Local Gradient Clipping** Gradient clipping is widely adopted to avoid the exploding gradient problem [17]. The method proposed by Pascanu et al. [18] rescales the gradients whenever the sum of their L2-norms exceeds a threshold. This step is conventionally executed *after* gradient aggregation from all nodes. Because we accumulate gradients over iterations on each node independently, we perform gradient clipping locally *before* adding the current gradient $G_t$ to previous accumulation ($G_{t-1}$ in Algorithm 1 and $V_{t-1}$, $U_{t-1}$ in Figure 2(a), 2(b)). We scale the threshold by $N^{-1/2}$, the current node's fraction of the global threshold if all $N$ nodes had identical gradient distributions.

### 2.3 Overcoming the Staleness Effect

Because we delay the update of small gradients, when these updates do occur, they are outdated or *stale*. Staleness can slow down convergence and degrade model performance. We mitigate staleness with momentum factor masking and warm-up training.

**Momentum Factor Masking** Mitliagkas et al. [19] discussed the staleness caused by asynchrony and attributed it to a term described as *implicit momentum*. Inspired by their work, we introduce *momentum factor masking*, to alleviate staleness. Instead of searching for a new momentum coefficient as suggested in Mitliagkas et al. [19], we simply apply the same mask to both the gradients $G_t$ and the momentum factor $V_t$ in Figure 2(a) and 2(b):

$$V_t[j] \leftarrow V_t[j] \odot \neg Mask$$

This mask stops the momentum for delayed gradients, preventing the stale momentum from carrying the weights in the wrong direction.

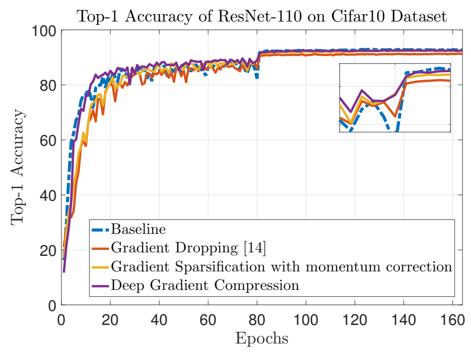
**Warm-up Training** In the early stages of training, the network is changing rapidly, and gradients are more diverse and aggressive. Sparsifying gradients limits the range of variation of the model, and thus prolongs the period of drastic gradients. Meanwhile, the remaining aggressive gradients from the early stage are accumulated before being chosen for the next update, and therefore they may outweigh the latest gradients and misguide the optimization direction. The *warm-up training* method introduced in large minibatch training [15] is helpful. During the warm-up period, we use a less aggressive learning rate, to slow down the changing speed of the neural network at the start of training, and also less aggressive gradient sparsity, to reduce the number of extreme gradients being delayed. Instead of linearly ramping up the learning rate during the first several epochs, we exponentially increase the gradient sparsity from a relatively small value to the final value.

## 3 Experiments

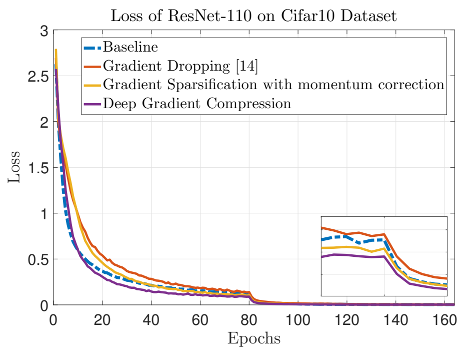
We validate our approach on three types of machine learning tasks: image classification with ResNet and AlexNet [20, 21] on Cifar10 [22] and ImageNet [23], language modeling with 2-layer LSTM [24, 25] on Penn Treebank dataset [26, 27], and speech recognition [28] with 5-layer LSTM on AN4 [29] and 7-layer GRU on Librispeech 1000h corpus [30].

We first examined deep gradient compression on image classification task. Figure 3(a) and 3(b) are the Top-1 accuracy and training loss of ResNet-110 on Cifar10 with 4 nodes. The sparsity of the gradient is 99.9% (only 0.1% is non-zero). The learning curve of Gradient Dropping [14] (red) is worse than the baseline due to gradient staleness. With momentum correction (yellow), the learning curve converges slightly faster, and accuracy is much closer to the baseline. Adding momentum factor masking and warm-up training techniques (blue), gradient staleness is eliminated and the learning curve closely follows the baseline.

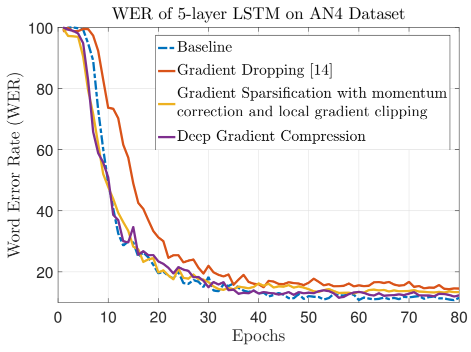
Table 1 shows the results of AlexNet and ResNet-50 training on ImageNet with 4 nodes. We compared the gradient compression ratio with Terngrad [6] on AlexNet (ResNet is not studied in [6]). Deep



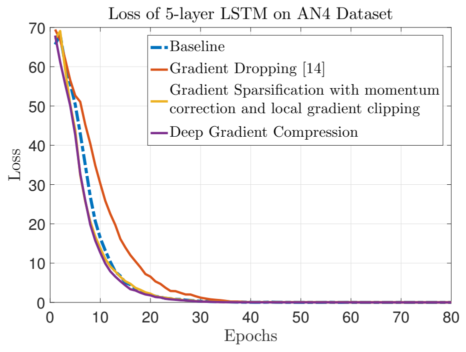
(a) Top-1 accuracy of ResNet-110 on Cifar10



(b) Training loss of ResNet-110 on Cifar10



(c) WER of 5-layer LSTM on AN4



(d) Training loss of 5-layer LSTM on AN4

Figure 3: Learning curves when gradient sparsity is 99.9%.

Table 1: Comparison of gradient compression ratio on ImageNet Dataset

| Model | Training Method | Top-1 Accuracy | Top-5 Accuracy | Gradient Size | Compression Ratio |
|---|---|---|---|---|---|
| AlexNet | Baseline | 58.17% | 80.19% | 232.56 MB | 1 × |
| | TernGrad [6] | 57.28% (-0.89%) | 80.23% (+0.04%) | 29.18 MB[2] | 8 × |
| | Deep Gradient Compression | **58.20% (+0.03%)** | **80.20% (+0.01%)** | **0.39 MB**[3] | **597** × |
| ResNet-50 | Baseline | 75.96 | 92.91% | 97.49 MB | 1 × |
| | Deep Gradient Compression | **76.15 (+0.19%)** | **92.97% (+0.06%)** | **0.35 MB** | **277** × |

Gradient Compression gives 75× better compression than [6] with no loss of accuracy. For ResNet-50, the compression ratio is slightly lower (277× vs. 597×) with a slight increase in accuracy.

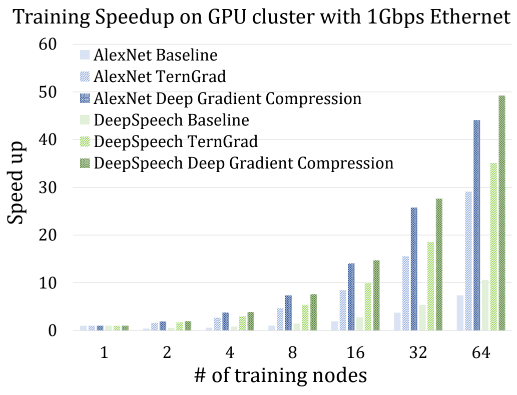
For language modeling, Table 2 shows the perplexity of a 2-layer LSTM trained on the PTB dataset using 4 nodes when 99.9% of the gradient exchange is removed. Deep Gradient Compression compresses the gradient by 462 × with a slight reduction in perplexity. For speech recognition, Figure 3(c) and 3(d) show the word error rate (WER) and training loss curve of 5-layer LSTM on AN4 Dataset with 4 nodes when the gradient sparsity is 99.9%. The learning curves show the same improvement acquired from techniques in Deep Gradient Compression as for the image network. Table 2 shows word error rate (WER) performance on LibriSpeech test dataset, where *test-clean* contains clean speech and *test-other* noisy speech. The model trained with Deep Gradient Compression gains better recognition ability on both clean and noisy speech, even when gradients size is compressed by 608×.

[3]The gradient of last fully-connected layer of Alexnet is 32-bit float. [6]

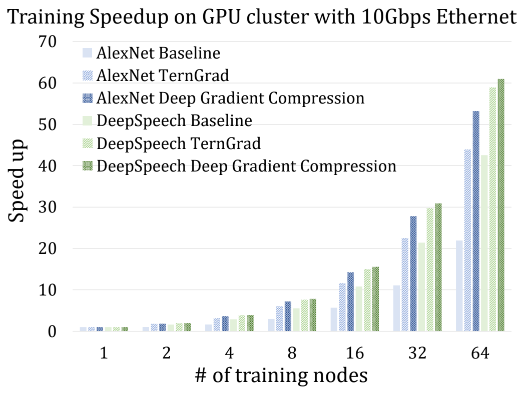
[3]We only transmit 16-bit index distances and 32-bit values of non-zeros in flattened gradients.

Table 2: Training results of language modeling and speech recognition with 4 nodes

| Task | Language Modeling on PTB | | | Speech Recognition on LibriSpeech corpus | | | |
|---|---|---|---|---|---|---|---|
| Training Method | Perplexity | Gradient Size | Compression Ratio | Word Error Rate (WER) test-clean | Word Error Rate (WER) test-other | Gradient Size | Compression Ratio |
| Baseline | 72.30 | 194.68 MB | 1 × | 9.45% | 27.07% | 488.08 MB | 1 × |
| Deep Gradient Compression | **72.24 (-0.06)** | **0.42 MB** | **462 ×** | **9.06% (-0.39%)** | **27.04% (-0.03%)** | **0.74 MB** | **608 ×** |



(a)



(b)

Figure 4: The total time consumption speedup compared to one training node when the number of nodes increases. Each node has 4 NVIDIA Titan XP GPUs and one PCI switch.

## 4 System Implementation and Performance Analysis

One critical step to implement Deep Gradient Compression is gradient top-$k$ selection. Given the target sparsity ratio of 99.9%, picking the top 0.1% largest over millions of weights can be slow (the complexity is $O(n\log(n \cdot s))$, where $s$ is the expected sparsity of gradients). We use sampling to reduce top-$k$ selection time. We sample 0.1% to 1% of the gradients and sort this sample to estimate the threshold for the entire population. If the number of gradients exceeding the threshold is far more than expected, a precise threshold is calculated from already-selected gradients. Hierarchically calculating the threshold significantly reduces sorting time.

We use the performance model proposed in Wen et al. [6] to perform the scalability analysis, combining the lightweight profiling on single training node with the analytical communication modeling. With the all-reduce communication model [31, 32], the density of sparse data doubles at every aggregation step in the worst case. However, even considering this effect, Deep Gradient Compression still significantly reduces the network communication time, as implied in Figure 4.

Figure 4 shows the speedup of multi-node training compared with single-node training. Conventional training achieves much worse speedup with 1Gbps (Figure 4(a)) than 10Gbps Ethernet (Figure 4(b)). Nonetheless, Deep Gradient Compression enables the training with 1Gbps Ethernet to be competitive with conventional training with 10Gbps Ethernet. For instance, when training AlexNet with 64 nodes, conventional training only achieves about 30× speedup with 10Gbps Ethernet [33], while with DGC, more than 40× speedup is achieved with only 1Gbps Ethernet. From the comparison of Figure 4(a) and 4(b), Deep Gradient Compression benefits even more when the communication-to-computation ratio of the model is higher and the network bandwidth is lower.

## 5 Conclusion

Deep Gradient Compression compresses the gradient by 270-600× for a wide range of CNNs, RNNs, and LSTMs. To achieve this compression without slowing convergence, DGC employs momentum correction, local gradient clipping, momentum factor masking, and warm-up training. Our implementation uses hierarchical threshold selection to speed up gradient sparsification. Deep Gradient Compression improves the scalability of distributed training with inexpensive, commodity networking infrastructure.

## References

[1] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pages 571–582, 2014.

[2] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015.

[3] Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael I Jordan. Sparknet: Training deep networks in spark. *arXiv preprint arXiv:1511.06051*, 2015.

[4] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.

[5] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[6] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*, 2017.

[7] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[8] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[9] Google Research Blog. Federated learning: Collaborative machine learning without centralized training data. URL `https://research.googleblog.com/2017/04/federated-learning-collaborative.html`.

[10] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[11] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[12] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[13] Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. *arXiv preprint arXiv:1610.02132*, 2016.

[14] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[15] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[16] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1): 145–151, 1999.

[17] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[18] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[19] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*, pages 997–1004. IEEE, 2016.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] S. Gross and M. Wilber. Training and investigating residual nets. `https://github.com/facebook/fb.resnet.torch`, 2016.

[22] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[24] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[25] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

[26] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330, 1993.

[27] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[28] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[29] Alejandro Acero. Acoustical and environmental robustness in automatic speech recognition. In *Proc. of ICASSP*, 1990.

[30] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE, 2015.

[31] Rolf Rabenseifner. Optimization of collective reduction operations. In *International Conference on Computational Science*, pages 1–9. Springer, 2004.

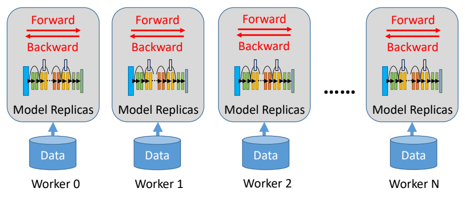
[32] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, Eli Upfal, and Derrick Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on parallel and distributed systems*, 8(11):1143–1156, 1997.

[33] Image classification with mxnet. `https://github.com/apache/incubator-mxnet/tree/master/example/image-classification`.

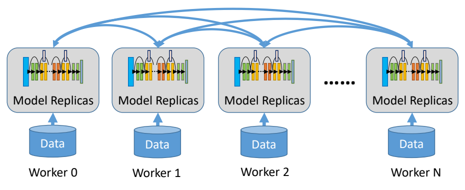
[34] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

## A Synchronous Distributed Stochastic Gradient Descent

In practice, each training node performs forward inference and backpropagation on different batches sampled from training dataset with the same network model. The gradients calculated by all nodes are summed up and broadcast to every node to optimize their models. By this synchronization step, models on different nodes are always the same during the training. The aggregation step can be achieved in two ways. One is using the parameter as the intermediary which store the parameters among several servers[10]. The nodes push the gradients to the servers while servers are waiting for gradients from all nodes. Once gradients are all sent, the servers update the parameters and then nodes pull the latest parameters from the servers. The other is done by an *all-reduce* operation [15] among all nodes as shown in Algorithm 2 and Figure 5. In this paper, we adopt the latter approach by default.



(a) Each node independently calculates gradients



(b) All-reduce operation of gradient aggregation

Figure 5: Distributed Synchronous SGD

**Algorithm 2** Distributed Synchronous SGD on node $k$

```
Input: Dataset $\chi$
Input: minibatch size $b$ per node
Input: the number of nodes $N$
Input: Optimization Function $SGD$
Input: Init parameters $w = \{w[0], \cdots, w[M]\}$
1: for $t = 0, 1, \cdots$ do
2:     $G_t^k \leftarrow 0$
3:     for $i = 1, \cdots, B$ do
4:         Sample data $x$ from $\chi$
5:         $G_t^k \leftarrow G_t^k + \frac{1}{Nb}\nabla f(x; w_t)$
6:     end for
7:     All-reduce $G_t^k : G_t \leftarrow \sum_{k=1}^{N} G_t^k$
8:     $w_{t+1} \leftarrow SGD(w_t, G_t)$
9: end for
```

## B Gradient sparsification with momentum correction

**Algorithm 3** Gradient sparsification with momentum correction on node $k$

```
Input: dataset $\chi$
Input: minibatch size $b$ per node
Input: momentum $m$
Input: the number of nodes $N$
Input: optimization function *momentum_SGD*
Input: initial parameters $w = \{w[0], \cdots, w[M]\}$
 1: $U^k \leftarrow 0, V^k \leftarrow 0$
 2: for $t = 0, 1, \cdots$ do
 3:     $G_t^k \leftarrow 0$
 4:     for $i = 1, \cdots, b$ do
 5:         Sample data $x$ from $\chi$
 6:         $G_t^k \leftarrow G_t^k + \frac{1}{Nb}\nabla f(x; \theta_t)$
 7:     end for
 8:     $V_t^k \leftarrow V_{t-1}^k + G_t^k$
 9:     $U_t^k \leftarrow U_{t-1}^k + V_t^k$
10:     $V_t^k \leftarrow m \cdot V_t^k$
11:     for $j = 0, \cdots, M$ do
12:         $thr \leftarrow s\%$ of $|U_t^k[j]|$
13:         $Mask \leftarrow |U_t^k[j]| > thr$
14:         $\widetilde{G}_t^k[j] \leftarrow U_t^k[j] \odot Mask$
15:         $U_t^k[j] \leftarrow U_t^k[j] \odot \neg Mask$
16:     end for
17:     *All-reduce*: $G_t \leftarrow \sum_{k=1}^N sparse(\widetilde{G}_t^k)$
18:     $\theta_{t+1} \leftarrow$ *momentum_SGD* $(\theta_t, G_t)$
19: end for
```

**Algorithm 4** Gradient sparsification with Nesterov momentum correction on node $k$

```
Input: dataset $\chi$
Input: minibatch size $b$ per node
Input: momentum $m$
Input: the number of nodes $N$
Input: optimization function *Nesterov_momentum_SGD*
Input: initial parameters $w = \{w[0], \cdots, w[M]\}$
 1: $U^k \leftarrow 0, V^k \leftarrow 0$
 2: for $t = 0, 1, \cdots$ do
 3:     $G^k \leftarrow 0$
 4:     for $i = 1, \cdots, b$ do
 5:         Sample data $x$ from $\chi$
 6:         $G_t^k \leftarrow G_t^k + \frac{1}{Nb}\nabla f(x; \theta_t)$
 7:     end for
 8:     $V_t^k \leftarrow m \cdot (V_{t-1}^k + G_t^k)$
 9:     $U_t^k \leftarrow U_{t-1}^k + V_t^k + G_t^k$
10:     for $j = 0, \cdots, M$ do
11:         $thr \leftarrow s\%$ of $|U_t^k[j]|$
12:         $Mask \leftarrow |U_t^k[j]| > thr$
13:         $\widetilde{G}_t^k[j] \leftarrow U_t^k[j] \odot Mask$
14:         $U_t^k[j] \leftarrow U_t^k[j] \odot \neg Mask$
15:     end for
16:     *All-reduce*: $G_t \leftarrow \sum_{k=1}^N sparse(\widetilde{G}_t^k)$
17:     $\theta_{t+1} \leftarrow$ *Nesterov_momentum_SGD* $(\theta_t, G_t)$
18: end for
```

### B.1 Nestrov Momentum SGD

The conventional update rule for Nesterov momentum SGD [34] follows,

$$u_{t+1} = mu_t + \sum_{k=1}^{N} \nabla_{k,t}, \quad w_{t+1} = w_t - \eta \left( m \cdot u_{t+1} + \sum_{k=1}^{N} \nabla_{k,t} \right) \tag{5}$$

where $m$ is the momentum, $N$ is the number of training nodes, and $\nabla_{k,t} = \frac{1}{Nb} \sum_{x \in \mathcal{B}_{k,t}} \nabla f(x, w_t)$.

Before momentum correction, the sparse update follows,

$$v_{k,t+1} = v_{k,t} + \nabla_{k,t}, \quad u_{t+1} = mu_t + \sum_{k=1}^{N} sparse\left(v_{k,t+1}\right), \quad w_{t+1} = w_t - \eta u_{t+1} \tag{6}$$

where the first term is the local gradient accumulation. Once the accumulation result $v_{k,t}$ is larger than a threshold, it will pass hard thresholding in the $sparse\,()$ function, and gets sent over the network in the second term.

After momentum correction sharing the same methodology with Equation (4), it becomes,

$$u_{k,t+1} = mu_{k,t} + \nabla_{k,t}, \quad v_{k,t+1} = v_{k,t} + \left(m \cdot u_{k,t+1} + \nabla_{k,t}\right), \quad w_{t+1} = w_t - \eta \sum_{k=1}^{N} sparse\left(v_{k,t+1}\right) \tag{7}$$