WeightGrad: Geo-Distributed Data Analysis Using Quantization for Faster Convergence and Better Accuracy

Syeda Nahida Akter and Muhammad Abdullah Adnan
Bangladesh University of Engineering and Technology (BUET)
Dhaka, Bangladesh
1505027.sna@ugrad.cse.buet.ac.bd,abdullah.adnan@gmail.com

ABSTRACT

High network communication cost for synchronizing weights and gradients in geo-distributed data analysis consumes the benefits of advancement in computation and optimization techniques. Many quantization methods for weight, gradient or both have been proposed in recent years where weight-quantized model suffers from error related to weight dimension and gradient-quantized method suffers from slow convergence rate by a factor related to the gradient quantization resolution and gradient dimension. All these methods have been proved to be infeasible in terms of distributed training across multiple data centers all over the world. Moreover recent studies show that communicating over WANs can significantly degrade DNN model performance by upto 53.7× because of unstable and limited WAN bandwidth. Our goal in this work is to design a geo-distributed Deep-Learning system that (1) ensures efficient and faster communication over LAN and WAN and (2) maintain accuracy and convergence for complex DNNs with billions of parameters. In this paper, we introduce WeightGrad which acknowledges the limitations of quantization and provides loss-aware weight-quantized networks with quantized gradients for local convergence and for global convergence it dynamically eliminates insignificant communication between data centers while still guaranteeing the correctness of DNN models. Our experiments on our developed prototypes of WeightGrad running across 3 Amazon EC2 global regions and on a cluster that emulates EC2 WAN bandwidth show that WeightGrad provides 1.06% gain in top-1 accuracy, 5.36× speedup over baseline and 1.4×-2.26× over the four state-of-the-art distributed ML systems.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Distributed computing methodologies**; *Distributed algorithms.*

KEYWORDS

geo-distributed datasets; deep neural networks; quantization; data analytics

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403097

ACM Reference Format:
Syeda Nahida Akter and Muhammad Abdullah Adnan. 2020. WeightGrad: Geo-Distributed Data Analysis Using Quantization for Faster Convergence and Better Accuracy. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403097

1 INTRODUCTION

Large scale cloud organizations are establishing data centers and "edge" clusters around the world to provide their users low latency access to their services. For instance, Google [3] and Microsoft [5] have tens of data centers, with the former also operating 1500 edges worldwide. The services deployed on these geo-distributed sites continuously produce large volumes of data. To analyze these huge amount of data with billions of features Deep neural networks (DNN) have shown the most success. Deep neural networks (DNN) are powerful category of machine learning algorithms implemented by combining layers of neural networks along the depth and width of smaller architectures. It has shown major improvement in several application domain including text and image classification, object recognition and detection, automatic speech recognition, natural language processing, video classification and topic modeling. These complex DNN models can analyze massive amounts of data using thousands of neurons and billions of parameter values. To analyze this massive amount of data, common approach is to centralize all data into one data center and perform training using complex DNN models which needs powerful machine with large amount of memory and top-processing power and takes huge amount of time. Clearly, centralization is not a feasible solution which motivates the need to distribute the DNN system across multiple data centers.

For distributed setup, the main challenges are (i) efficiently utilize limited WAN b/w, (ii) faster convergence without loss of accuracy. To attenuate the scalability challenges that arise from huge amount of data used for training complex deep neural network models with millions of weights and biases, data parallelism can be a plausible solution. Synchronous SGD [7, 12, 19] for distributed training assures the convergence of the model and also the accuracy, though total communication to exchange parameters can be dependent on the slowest worker. Asynchronous SGD [40] alleviates the problem but creates inconsistency in structure, thus can lose the convergence of the model. Many corrections and modifications to both Synchronous SGD and Asynchronous SGD have been proposed to address their drawbacks. To overcome the communication bottleneck and accelerate convergence in distributed training of dense deep learning models, sparse and quantized deep learning models have become one of the most researched areas in machine learning and deep learning world [14, 21, 32].

In this work, we propose WeightGrad, a geo-distributed Deep Learning system that has been designed to efficiently operate over a collection of data centers. We use the widely used Parameter Server architecture [6, 8] for communication over both LAN and WAN bandwidth. According to the architecture, inside a data center, there are several Local Parameter Servers (LPS) and each LPS has several Worker Machines (WM) assigned to it. The key idea of WeightGrad is to maintain the consistency, accuracy and faster convergence within a data center (LAN) and propagate the aggregated update in timely manner to other data centers when it crosses a dynamic threshold value. To ensure accuracy and faster convergence within LAN, we design quantized network introducing both gradient and weight quantization for communication. To globally synchronize our system, we develop a Gradient Synchronizer inside each Local Parameter Server (LPS) which maintains significant update passing among the data centers. We propose a Two Level Structure (TLS) to achieve this consistency, first level structure consists of LPS and WMs and second level structure consists of Global Server (GS). WeightGrad maintains tight synchronization inside a LAN where communication cost is the least and loose synchronization globally to eliminate performance degradation due to scarce bandwidth. We deploy WeightGrad across 3 AWS EC2 regions and also on local cluster that emulates the WAN bandwidth across the different Amazon EC2 regions. Our experiment including intensive comparison with four state-of-the-art systems deployed on WANs validates that WeightGrad: (1) gets top-1 accuracy gain of 1.06% over baseline, 1.26% over Gaia [24], 1.96% over TernGrad [38] and 1.08% over DGC [30], and (2) shows upto 5.36× speedup over baseline, 2.16×-2.26× over Gaia [24] and 1.42×-1.67× over TernGrad [38] running the same DNN model. Our major contributions include:

- We propose a geo-distributed Deep learning system that uses quantized network to communicate over LAN and WAN and make efficient use of the scarce WAN bandwidth by considering the significant update thus minimizing the communication over WAN. To the best of our knowledge, this is the first work on geo-distributed Deep learning system that uses both weight and gradient quantization for communication across the data centers.
- We design a Gradient Synchronizer in the LPS inside each data center to communicate outside the data center and to eliminate the inconsistency among them within a fixed time interval. Gradient Synchronizer reduces usage of WAN bandwidth by sending the significant quantized gradients within a fixed time period and to preserve the full-precision accuracy and convergence, it applies local gradient clipping and momentum correction before sending it to the worker machines and GS.
- We present a Two Level Structure (TLS) to synchronize communication among the data centers over WANs. The first level maintains the communication intensive individual data centers running under LAN on specified shard of data and the second level handles the scarce WAN bandwidth in the most efficient way by passing only the significant gradient updates and monitoring the convergence and synchronization using Global Server.
- We implement our system on both CPU and GPU-based DNN models. We experiment with Cifar-10 and ImageNet dataset and compare the system outcome with four established state-of-the-art distributed systems which shows significant performance gain.

2 RELATED WORKS

To analyze the nature of the enormous amount of data growing at each moment all around the world, distributed arrangement is the only way to maintain feasibility, scalability and cost. As the intrinsic nature of machine learning algorithms to be iterative, the output should be distributed among the workers after every iteration and with the increased number of worker nodes, more and more messages need to be exchanged to update the internal state of the workers which results in additional convergence time. Numerous works have been proposed to overcome the communication bottleneck and for faster convergence in distributed training. Asynchronous SGD [29, 40] accelerates the training by removing gradient synchronization and updating parameters immediately once a node has completed back-propagation. But it introduces inconsistent state among the workers which can engender loss in accuracy. On the other hand, Gradient sparcification [17, 30, 37] and quantization [10, 20, 23, 34] try to reduce communication cost by focusing on the size of information sent in each epoch. Apart from the idea of model parallelism and data parallelism, synchronous SGD [12] and asynchronous SGD, weight quantization and gradient quantization, this paper is the first work that applies the idea of merging both weight and gradient quantization with structured distributed Deep Learning system. Prior to this, the works and ideas that motivated our work are as follows:

Distributed ML Systems. To address the communication bottleneck on WAN and LAN, Gaia [24] proposed a structured model that decouples the communication within a data center from the communication between data centers. Gaia is built on the frame of Parameter Server architecture. According to the architecture, each data center has some worker machines and parameter servers and each worker holds a model replica and a shard of data in the data center. Worker machines process their shard of data and send parameter to their local parameter server. Parameter Server aggregates the parameters and send the value to the worker machines. For convergence of the distributed training, Gaia proposed a synchronization model, Approximation Synchronous Parallel (ASP) to ensure that all significant updates are synchronized in time. ASP is based on the key finding that vast majority of the updates to the global ML model parameters from each ML worker machine are insignificant. A parameter server shares only the significant updates with other data centers and ASP ensures that these updates can be seen by all data centers in a timely fashion. Gaia provides 1.8–53.5× speedup running across 11 Amazon EC2 global regions and on a cluster that emulates EC2 WAN bandwidth, and is within 0.94–1.40× of the speed of running the same ML algorithm on machines on a local area network (LAN).

Gradient Sparsification. Threshold quantization was developed by Strom [37] to only send gradient larger than a predefined constant and thus reduce the volume of data sent. But maintaining the best possible threshold needs large amounts of experiments and largely varies from model to model, dataset to dataset. A fixed threshold can also lead to a situation where error feedback builds up, leading to large values of data being transmitted. Dryden et al. [17] addressed this problem by introducing their own adaptive algorithm which uses a fixed proportion π and sends both positive and negative gradient updates maintaining that proportion. Aji and

Heafield [9] conducted further improvement on Dryden et al. [17] which achieved up to 49% speed up on MNIST and 22% on NMT without damaging the final accuracy. They used a single threshold based on absolute value, instead of dropping the positive and negative gradients separately. DGC [30] tried to solve bandwidth problem by compressing gradients. To compensate the loss, they further introduced *momentum correction* and *local gradient clipping* on top of the gradient sparsification and *momentum factor masking* and *warmup training* to overcome staleness problem.

Parameters and Gradient Quantization. Seide et al. [34] proposed 1-bit quantization which allows to significantly reduce data-exchange bandwidth for data-parallel SGD and achieved 10× speedup for Switchboard speech-to-text system. As the gradient quantization is conducted by columns, a floating point scalar per column is required. So, it cannot yield speed benefit on CNN. Moreover, the "cold start" method of this paper requires first 24h of data processed without parallelism or quantization for floating point gradients to converge to a good initial point for the following 1-bit SGD. This method can significantly reduce accuracy if correct conditions are not maintained. S. Gupta [20] successfully trained deep networks on MNIST and CIFAR-10 datasets using only 16-bit wide fixed-point number representation. Admitting the problem of compressing gradient i.e, they don't always converge, Dan Alistarh et al. [10] proposed QSGD with convergence guarantees. For ResNet-152 network, they achieved 1.8× speedup than full precision variant on ImageNet dataset to full accuracy. Terngrad [38] further applied this idea to quantize gradients to ternary levels. They introduced layer-wise ternarizing and gradient clipping to reduce accuracy loss while it assured faster convergence. They achieved 0.92% top-1 accuracy improvement for AlexNet while 1.73% top-1 accuracy loss was observed in QSGD with 4 levels. The closest to our work, i.e, both weight and gradient quantization have also been adapted for reducing communication cost and faster convergence [23]. This method showed that gradient clipping before quantization gives faster convergence though it sacrificed upto 1.22% loss in top-1 accuracy for ImageNet dataset. In contrast to our work, this method (a) does not differentiate the communication on LAN from the communication on WAN and (b) does not provide any synchronization model across multiple data centers. Passing quantized weights and gradients using WAN bandwidth in each epoch will wipe off the additional speed gained due to quantization. XNOR-Net [33] proposed two efficient approximations to standard Convolutional Neural Networks: Binary-Weight-Networks and XNOR-Networks. In Binary-Weight-Networks, the filters are approximated with binary values resulting in 32× memory savings. This results in 58× faster convolutional operations (in terms of number of the high precision operations) and 32× memory savings on CNN models. DoReFa-Net [41] trained CNN with low bitwidth weights and activations using low bitwidth parameter gradients. They experimented with AlexNet that has 1-bit weights and 2-bit activations and trained the model with 6-bit gradients to get 46.1% top-1 accuracy on ImageNet validation set.

3 OUR APPROACH: WEIGHTGRAD

In this section, we introduce WeightGrad, a consistent and efficient architecture for distributed deep learning and communication among the data centers across all over the world. We address the limitations of previous works for distributed system in Section 3.1 and then propose our model architecture that successfully trains state-of-the-art deep learning models in a distributed manner and communicates efficiently among the data centers without any accuracy loss. WeightGrad acknowledges the fact that for distributed setup averaging weights or gradients received from all the worker nodes creates distinction between centralized and distributed training. Moreover, quantizing gradients and weights for faster communication between worker and parameter server also causes loss in precision thus some loss in accuracy [10, 34]. Deploying quantized model in WAN can further deteriorate the situation by adding more inconsistency and loss in accuracy due to limited WAN bandwidth. This ultimately results in taking more time to converge and more epochs to reach to the standard accuracy. On the basis of this findings, we design our model which focuses on this missing accuracy while ensuring faster convergence.

Precision	Model	Parameters	Total Size
32bits/4bytes	ResNet50	23M	92MB
	ResNet152	60M	240MB
	AlexNet	64M	256MB
	VGG-Net	138M	552MB

Table 1: Parameters of DNN models

3.1 Key Challenges

There are two key challenges we generally face in designing a structure for training DNN models over data centers across the WAN.

Efficient and faster communication over LAN and WAN. For distributed setup, scarcity of WAN bandwidth can significantly slowdown the convergence of the Deep neural networks. The state-of-the-art DNN models hold billions of parameters and exchanging them among the data centers in each epoch is a major challenge that can dismiss the benefit of distributed setup. To overcome this challenge, Gaia [24] proposed significance threshold, that helps to reduce unnecessary communication over WAN, but significance threshold value itself is a hyper-parameter which changes dynamically with each epoch, so for initial epochs, there will be same number of communication over LAN and WAN which can cause significant slowdown in convergence and thus major fall in accuracy in dearth of WAN bandwidth. Moreover, it exchanges parameters (weights and biases) with full-precision. For complex DNN models hundreds to thousands mega bytes of data needs to be exchanged for each epoch. Table 1 shows size and number of parameters of some state-of-the-art CNN models [22, 28, 35]. Therefore sending full-precision parameters can disparage the effect of significance threshold for distributed training. Communication pattern among the data centers considerably influences communication and convergence time. Reducing communication using threshold value creates inconsistency among the data centers. Choosing asynchronous structure for communication over WAN can deteriorate the situation resulting in divergence of the model [11, 31]. Hence we need a model which not only reduces communication time and bandwidth efficiently, but also provides a proper architecture to ensure convergence.

Maintaining accuracy and convergence. From the previous challenge it is evident that full-precision parameter passing can considerably affect convergence time for DNN models. To alleviate this problem, idea of quantization to the parameters and

gradients have been introduced [10, 14, 20, 21, 34]. Though quantization has perceived significant speedup than the full precision models, quantizing gradients slows convergence by a factor related to the gradient quantization resolution and dimension which contributes to fall in accuracy (Section 4.3). Various approaches e.g, gradient clipping [38], momentum correction and momentum factor masking [30] have been applied to improve accuracy, but they still fail to guarantee convergence for distributed training across data centers where communication in each epoch can cause significant slowdown in convergence due to scarce b/w. Moreover for distributed data architecture, data characteristics can vary so strikingly that gradients generated from a single worker node can hold an important feature to the model. Therefore, we need to design a structure that utilizes the speedup achieved from the quantization and guarantees same accuracy as the full precision model.

3.2 WeightGrad System Overview

We propose a new architecture, WeightGrad that acknowledges two major problems discussed above and gives feasible and effective solution to train complex and large deep learning models across the data centers over both LAN and WAN. WeightGrad takes the idea of the state-of-the-art parameter server architecture Gaia [24] which has proven its effectiveness over other architectures on a wide variety of machine learning algorithms [15, 16]. As discussed in Section 2, Gaia mainly adapts the popular parameter server architecture, but modifies the communication condition among the data centers which subsequently reduces the communication overhead over WANs by eliminating insignificant and thus unnecessary communication across different data centers. We observe this advantage of Gaia and also acknowledge their slow convergence rate while training large and complex deep neural net models due to exchanging full-precision parameters (32-bit) in each epoch which can be of size upto thousands of megabytes. WeightGrad introduces quantized network architecture to solve this problem of passing large amount of data within a data center, a Gradient Synchronizer inside each Local Parameter Server (LPS) to aggregate the insignificant update and propagate the significant update within fixed time interval and a Two Level Structure (TLS) to handle the synchronization across the data centers. To mitigate the loss of accuracy due to quantized parameters and gradients, LPS and worker machines also maintain layer-wise gradient clipping before quantization [38] and momentum correction [30] (Section 4.3).

Figure 1 shows system overview of WeightGrad. Each data center has some worker machines and parameter servers. Each worker machine contains model replica and shard of input data from data center to achieve data parallelism. Worker machines in a data center communicates with each other through the parameter servers over LAN. We assign inference and gradient descent calculation to the Worker machines and parameter-update function to the LPS. We apply both weight and gradient quantization approach to reduce the size of data exchanged for faster communication over data centers. LPS accumulates quantized gradients from the worker machines and updates the model parameters serially. It simulates centralized training as we consider gradients from each worker machine for updating parameters of the DNN model instead of taking average of the gradients. It preserves similar accuracy as centralized training but ensures faster convergence in distributed

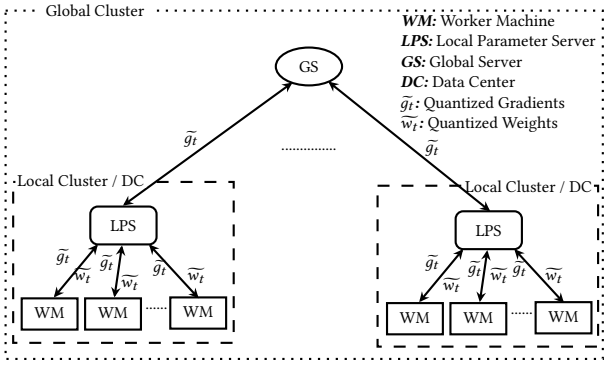


Figure 1: WeightGrad Tree Structure

training setup. Section 3.3 describes the details of each LPS. To synchronize the communication over WAN, WeightGrad maintains TLS where the first level contains individual data center and the second level contains a Global Server to propagate the significant update across all data centers in a timely manner (Section 3.4).

3.3 LPS with Gradient Synchronizer

Inside each LPS, we design a Gradient Synchronizer to synchronize the model parameters inside and outside the data center. According to Gaia [24] vast majority of updates of global ML model parameters from each worker machine are insignificant, as their experiment shows that 95% of the updates produce less than 1% change to the parameter value and DGC [30] shows 99.9% of the gradient exchange in distributed SGD are redundant. Using the idea of discarding redundant update passing, Gradient Synchronizer maintains a dynamic threshold (Appendix A.2) for distributing the gradient updates. LPS accumulates the quantized gradients from the worker machines and the Gradient Synchronizer checks its significance by comparing it with the threshold. For initial epochs gradients tend to have higher values as the loss is higher. To match with this decreasing property of the loss and gradient value, the threshold value also gets updated with each epoch and momentum. Gradient Synchronizer also receives average gradient update from Global Server (GS) within a predefined time interval T and updates the model parameters accordingly. During scarce bandwidth, Gradient Synchronizer may not receive any update from the GS, in such case LPS stops sending parameter updates to the WMs until it receives any update from GS. It helps all the data centers to reach in consistent state within a fixed time interval.

3.4 Two Level Structure for Synchronization

To eliminate inconsistency among the data centers WeightGrad proposes a Two Level Structure that distributes the significant update among the parameter servers in different data centers. Each individual data center with strong LAN network forms the first level of the structure. Each data center in this level handles computation and communication intensive parameter update process and it maintains tight consistency, convergence and accuracy on its own shard of data. The second level of the structure consists of a Global Server (GS). First level communicates with the second level through the Gradient Synchronizer. GS receives all the significant gradients from the LPSs and sends the average to all the LPSs. GS maintains

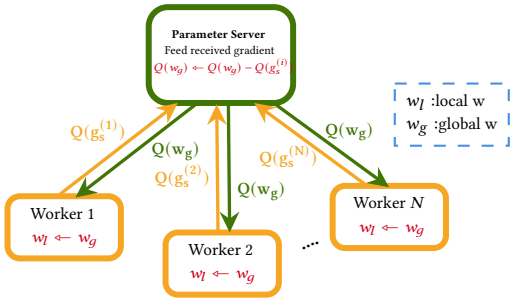


Figure 2: WeightGrad: Local Cluster

a time interval against each LPS (T_{local}) while sending the average quantized gradients. It also maintains a global time interval (T_{global}) within which model replicas of all data centers achieves global consistency. When the GS can not send update to a data center due to limited b/w within a T_{local}, it stops sending updates to other LPSs until the temporarily unreachable LPS catches up. Thus GS ensures global consistency and convergence.

4 ALGORITHM FORMULATION

In this section, we formulate the algorithm for distributed deep learning ensuring accuracy and convergence. First we present the state-of-the-art distributed training process using quantization (Section 4.1) and then we update the basic algorithm and discuss the motivation and intuition behind each algorithm (Section 4.2). We also justify how these algorithms overcome the drawbacks of the basic algorithm. Each algorithm has two parts, (i) Parameter Server and (ii) WM. For WeightGrad, we also need a Global Server to distribute the significant updates among the data centers.

4.1 Basic Parameter Server Architecture with Gradient Quantization

In basic quantized gradients based Parameter Server systems, worker machines produce gradients and pass the quantized gradients to the parameter server. Parameter server averages the quantized gradients and sends the value to each worker. So, the input and output of the worker machine algorithm follows:

- **Input:** Total number of epochs, S. Shard of data assigned to worker node i, z^i.
- **Output:** Quantized gradients $\widetilde{g}_s^{(i)}$ for epoch s from worker i.

Data is partitioned across different data shards and stored in each worker machine, z^i and the deep learning model is replicated at each worker node. Here $W^{(i)}, B^{(i)}$ are random initial weights and biases for a worker node i. Then each worker node runs the forward propagation and calculates the gradients using its own shard of dataset ${z_s}^{(i)}$ and model parameters ${W_s}^{(i)}, {B_s}^{(i)}$. Then the generated gradients are quantized, $\widetilde{g}_s^{(i)}$ and sent to the parameter server. The input and output for Parameter Server algorithm follows:

- **Input:** Total number of worker nodes, N. Quantized gradients $\widetilde{g}_s^{(i)}$ for epoch s received from worker node i. Total number of epochs, S.
- **Output:** Average of all the quantized gradients $\overline{g}_s$ received from N worker nodes for epoch s.

Algorithm 1 Traditional Worker-Parameter Server

Worker: $i = 1, ..., N$

Input: $S, z^{(i)}$ ▷ S: Number of epochs

$W^{(i)}, B^{(i)} \leftarrow$ initial parameter values (weights and biases)
for $s \in S$ **do**
  $\widetilde{g}_s^{(i)}$ = ComputeQuantizedGradient(${z_s}^{(i)}, {W_s}^{(i)}, {B_s}^{(i)}$)
  sendToPS($\widetilde{g}_s^{(i)}$) ▷ PS: Parameter Server
  $\overline{g_s}^{(i)} \leftarrow$ PullAverageGradientFromPS()
  ${W_{s+1}}^{(i)}, {B_{s+1}}^{(i)} \leftarrow$ UpdateParameters($\eta, \overline{g_s}^{(i)}$)
end

Server:

Input: N, S

for $s \in S$ **do**
  for $i \in N$ **do**
    $\widetilde{g}_s^{(i)}$ = PullGradientFromWorker()
  $\overline{g}_s$ = CalculateAverage($\widetilde{g}_s$)
  sendToWorkers($\overline{g}_s$)
end

Parameter servers receive the quantized gradients $\widetilde{g}_s^{(i)}$ from each worker node i in epoch s and send the average $\overline{g}_s$ to them.The worker nodes update the parameter values of their model using the averaged quantized gradients. This is how a single epoch gets executed in both worker and parameter server.

4.2 Weight-Gradient Quantization Model

A. Issues with the Basic Algorithm

We first examine why Algorithm 1 is different from the centralized training and why it can lose accuracy and convergence.

- **First**, quantization causes a loss of precision as it converts 32-bit floating point numbers into 1,2 or 3 bits of discretized values. This conversion greatly reduces the size of the data needed to pass in each epoch, but at the price of precision loss. Quantization slightly deviates gradients from the full-precision gradients in each epoch $\delta_s = g_s - \widetilde{g}_s$ and this difference increases with the number of epochs $\Delta = \sum_{s=0}^{S} \delta_s$. Moreover, the LPS averages the quantized gradient which creates another approximation to the precision and the difference becomes more acute which results in loss in accuracy.
- **Second** and most importantly, this algorithm does not consider scarce and heterogeneous WAN b/w for distributed training across different data-centers all over the world. All previous systems experimented quantization within a single data centers under stable b/w which gave significant speedup and small loss in accuracy. But communicating over WANs in each epoch can significantly degrade ML system performance (around 53.7× than training within LAN) [24].

B. Our Proposed Algorithm

To account for the limitations of Algorithm 1, we design our own system algorithm that maintains the best training method within a single datacenter and communicates and synchronizes the parameter values across multiple data centers in an efficient way. WeightGrad algorithm has three parts: (a) Worker Machine (WM) (b) Local Parameter Server (LPS) (c) Global Server (GS).

(a) Worker Machines: WMs calculate gradients from their own shard of data, quantize the gradients and send them to the LPS. The input and the output of each worker machine are follows:

Algorithm 2 WeightGrad (Worker)

Worker: $i = 1, ..., N$

Input: $S, z^{(i)}$ ▷ S: Number of epochs

$W^{(i)}, B^{(i)} \leftarrow$ initial parameter values received from LPS
for $s \in S$ **do**
  $\widetilde{g_s}^{(i)}$ = ComputeTernarizedGradient($z_s{}^{(i)}, W_s{}^{(i)}, B_s{}^{(i)}$)
  sendToLPS($\widetilde{g_s}^{(i)}$) ▷ LPS: Local Parameter Server
  $\widetilde{W_{s+1}}^{(i)}, \widetilde{B_{s+1}}^{(i)} \leftarrow$ pullGlobalValuesFromLPS()
end

- **Input:** Total number of epochs, S. Shard of data assigned to worker node i, $z^{(i)}$.
- **Output:** Quantized gradients $\widetilde{g}_s^{(i)}$ for epoch s from worker i.

This WM works exactly like the basic WM except the fact that in WeightGrad, workers receive quantized parameters instead of quantized gradients and it uses TernGrad [38] for quantization.

(b) Local Parameter Server: LPS receives quantized gradients from the worker machines, updates the parameters and sends the quantized parameters to each worker node. The input and the output of LS are follows:

- **Input:** Total number of workers, N, fixed interval, T.
- **Output:** Quantized weights and biases $\widetilde{W}_g, \widetilde{B}_g$.

We experiment with two gradient accumulation techniques. Algorithm 3 gathers all the gradients from the worker machines and use the average gradients to update the parameter values. To scale WeightGrad for distributed training across multiple data centers, we apply TLS to preserve accuracy and ensure convergence. Each LPS inside a data center maintains a Gradient Synchronizer which monitors the significance threshold value of gradients, G_{thresh} (Appendix A.2). When the significance value of the average gradients ($|\frac{Aggregated\ Gradient\ Update, g_u}{Current\ Average\ Gradients, \overline{g}_s}|$) becomes more than the threshold significance value (Appendix A.2), LPS sends the gradient update to the GS. Here to synchronize the communication process, Gradient Synchronizer maintains a fixed interval T, within which it receives aggregated gradient values from the GS. If an LPS does not get update from the GS within T, it stops sending updates to the WMs until gradient updates are received from the GS.

(c) Global Server: GS mainly follows the work graph of basic local server. It collects significant gradients from the datacenters and sends the average gradient values to the LPS of each datacenter. It also maintains a time interval T_{local} against each LPS and sends gradient update to each LPS within T_{local}. In case of any slow LPS, GS maintains a T_{global} within which all including the slowest LPS will receive the aggregated quantized gradients.

C. Tuning Aggregation Method of Gradients

In Algorithm 3, LPS collects all the gradients from the worker nodes and update the parameter values using the average of the received gradients. We tune this aggregation process in Algorithm 4 to update the parameter values whenever LPS receives gradients from a worker node. For each update, it checks whether the aggregated gradients are significant enough ($\frac{g_u}{g_{max}} \geq G_{thresh}$) and sends the significant update to the GS. LPS receives the average gradients from the GS and if the average gradient value is greater than the current maximum gradient value, then the parameter values get updated by the difference between them ($g_{avg} - g_{max}$).

Algorithm 3 WeightGrad (Local and Global Server)

Local Server:

Input: N, T

SendInitialParameters(W_g, B_g) ▷ W_g, B_g : global weights and biases
for $i \in N$ **do**
  $\widetilde{g_s}^{(i)}$ = PullGradientFromWorker()
end
$\overline{g}_s$ = CalculateAverage($\widetilde{g_s}$)
g_u = AggregateGradients($\overline{g}_s$)
if $\frac{g_u}{\overline{g}_s} \geq G_{thresh}$ **then**
  SendGradientToGS(g_u)
$\overline{g}_s \leftarrow$ PullAverageGradientFromGSWithinT()
$\widetilde{W_g}, \widetilde{B_g} \leftarrow$UpdateParameters($\eta, \overline{g}_s$)
SendToWorkers($\widetilde{W_g}, \widetilde{B_g}$)

Global Server:

Input: N, T_{local}, T_{global}

for $i \in N$ **do**
  $\widetilde{g_s}^{(i)}$ = PullGradientFromLPS()
end
$\overline{g}_s$ = CalculateAverage($\widetilde{g_s}$)
SendToAllLPSWithinT_{local}($\overline{g}_s$)

Algorithm 4 WeightGrad (Local Server)

Local Server:

Input: N, T

SendInitialParameters(W_g, B_g) ▷ W_g, B_g : global weights and biases
for $i \in N$ **do**
  $\widetilde{g_s}^{(i)}$ = PullGradientFromWorker()
  $\widetilde{W_g}, \widetilde{B_g} \leftarrow$UpdateParameters($\eta, \widetilde{g_s}^{(i)}$)
  g_u = AggregateGradients($\overline{g}_s$)
end
g_{max} = MaxGradient($\widetilde{g_s}$)
if $\frac{g_u}{g_{max}} \geq G_{thresh}$ **then**
  SendGradientToGS(g_u)
$g_{avg} \leftarrow$ PullAverageGradientFromGSWithinT()
if $g_{avg} > g_{max}$ **then**
  $\widetilde{W_g}, \widetilde{B_g} \leftarrow$UpdateParameters($\eta, g_{avg} - g_{max}$)
SendToWorkers($\widetilde{W_g}, \widetilde{B_g}$)

4.3 Quantized Network Accuracy

In most state-of-the-art parameter-server architecture, the Parameter Server (PS) aggregates the gradients from the worker nodes and sends the average of the gradients to them. For reducing the expensive communication cost in distributed setup, quantization and sparsification are two major approaches. In weight quantization, the average regret (Appendix A) converges to an error related to the weight quantization resolution Δ_w and dimension d regardless of the full-precision or quantized gradients (Theorem A.1 and A.2). The greater the value of Δ_w or d, the larger is the error. Similarly, quantized gradients reduces the convergence rate by a factor of gradient quantization resolution Δ_g and dimension d. The convergence rate falls with increasing Δ_g or d. For very complex and deep neural net models, d is significantly higher. And to overcome the communication bottleneck in the distributed training, quantization

with fewer bits is applied which results in larger Δ_w (weight quantization) or Δ_g (gradient quantization). To mitigate this problem, we consider both weight and gradient quantization for distributed training. Figure 2 shows the structure of WeightGrad inside a data center. We quantize weights and gradients to ternary levels {-1, 0, 1} proposed in TernGrad [38]. We use a layer-wise quantization scheme to reduce the performance degradation. A smaller learning rate is maintained to preserve the good performance achieved from quantization [39]. Again to reduce the fall of convergence rate due to gradient quantization, we observe that gradient clipping with momentum correction and layer-wise ternarizing gives the best convergence rate among the state-of-the-art systems (Section 5.3). Applying both weight and gradient quantization with the modifications mentioned above outperforms the baseline along with some other state-of-the-art distributed training systems (Section 5.3). Moreover, the other state-of-the-art structures do not consider gradients from a single worker, rather they grow with the average of the gradients which is the major distinction between centralized and distributed training setup. For this reason, centralized training loss graph is much smoother than distributed training. Considering this distinction, we design our own local parameter server (within a data center) which considers all the gradient update serially to update the model parameters and sends the final updated parameters to the WMs for next epoch (Appendix B).

5 EXPERIMENTS

5.1 Experiment Settings

We first validate the convergence of WeightGrad under various training schemes on relatively small datasets. For this, we use Cifar-10 [27] dataset which consists of 50,000 training images and 10,000 validation images in 10 classes. Then we check convergence by scaling our system to different EC2 regions and observe the fall of loss and final accuracy by training different deep learning models on ImageNet dataset which contains over 1 million training images and 50,000 validation images in 1000 classes. All the experiments are performed by Keras [13] and Tensorflow [6]. We maintain the exponential moving average of parameters by employing an exponential decay of 0.9999 and batchnorm moving average decay of 0.9997 [12]. For fair comparison among the systems using either floating or ternary gradients, all the training hyper-parameters are kept same throughout the whole training process. We use initial learning rate of 0.1 and learning rate decay factor of 0.1. While polynomial decay is applied to decay the LR, the power of 0.5 is used to decay LR from the base LR to zero [38].

5.2 Experiments Platforms

- **Amazon-EC2**: We deploy WeightGrad to total 10 machines spread across 2 EC2 regions (N. California and N. Virginia) [1], each region having 4 worker machines (WM) and 1 local parameter server (LPS). We deploy global server (GS) in another EC2 region (Ohio). In each EC2 region, we use instances of type g3s.xlarge [2] for computation intensive backpropagation phase of deep neural nets. Each instance of type g3s.xlarge has 4 vCPU with 30.5 GiB RAM, NVIDIA Tesla M60 GPU, 1 accelerator with network performance 10 Gbps, running 64-bit Ubuntu Server 16.04 LTS (HVM).

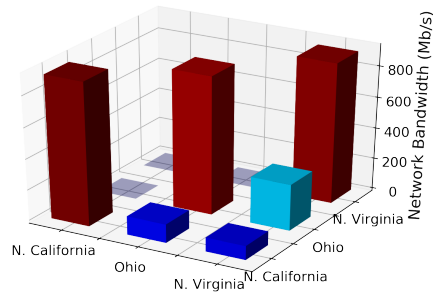


Figure 3: Measured network bandwidth between Amazon EC2 sites

- **Local Cluster**: For our initial experiment in local cluster, we use two Titan RTX with 16 GiB RAM, 4 vCPU for parameter servers and for workers 4 machines each with 8 GiB RAM and 1 CPU, running 64-bit Ubuntu 16.04 LTS.

Figure 3 shows the network b/w between each of two pairs of 3 EC2 regions. We use *iperf3* [4] to measure the b/w and transferred data. We take total of 10 b/w values between each two regions and calculate the average to plot the graph. As we can observe from the graph that the WAN b/w between data centers is 3.06× smaller than the LAN b/w within a data center in the best case (Ohio ↔ N. Virginia) and upto 10.1× smaller in the worst case (N. Virginia ↔ N. California). Thus the systems designed for single data center can perform terribly while deploying them over WAN b/w.

5.3 Results and Analysis

5.3.1 Experiment on Cifar-10 dataset. We investigate the convergence of WeightGrad by training AlexNet on Cifar-10 dataset. We first deploy WeightGrad (Algorithm 2, 3, 4) on two EC2 regions and from both region we take two instances as worker node and one instance as LPS. Then we repeat the training using the same model and same dataset on three and five workers in each region. Figure 4a shows the sharp fall in loss regardless of the number of worker nodes. Figure 4b depicts the same training result using time and loss, here 5 workers setup shows faster fall in loss as large number of workers offers more parallelism.

Next, we compare WeightGrad with baseline, Gaia and TernGrad by CifarNet [26] and VGGNet-A [36] trained on Cifar-10 dataset. We take 4 workers and 1 LPS from each region (N. California and N. Virginia) and a GS from Ohio. Figure 5a and 5b shows convergence of WeightGrad within initial epochs where other systems are struggling to get the efficient training loss. Table 2 summarizes the results of CifarNet and VGGNet-A training on Cifar-10 dataset for Baseline, Gaia, TernGrad and WeightGrad with total of 4 worker nodes and 8 worker nodes, where all trainings terminate after the same steps. While training CifarNet model, WeightGrad achieves 0.03% accuracy gain in spite of using a quantizated network (Section 4.3).

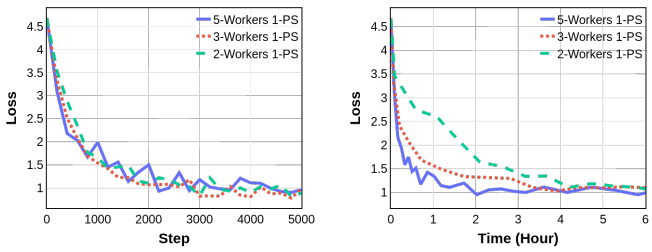
5.3.2 Experiment on ImageNet dataset. We also evaluate WeightGrad by AlexNet trained on ImageNet dataset to observe the convergence and accuracy for large dataset. Applying quantized gradients and weights to large-scale DNNs is more challenging than the floating weights and gradients based systems. TernGrad faces some accuracy loss when simply replacing the floating gradients

Model	SGD	Base LR	Total mini-batch size	Steps	Gradients	Workers	Accuracy
CifarNet	GD	0.1	128	50k	Baseline	4	84.56%
					Gaia	4	83.48%(-1.08%)
					TernGrad	4	82.41%(-2.15%)
					WeightGrad	**4**	**84.56%(-0.00%)**
	GD	0.1	512	50k	Baseline	8	83.19%
					Gaia	8	83.04%(-0.13%)
					TernGrad	8	81.40%(-1.79%)
					WeightGrad	**8**	**83.21%(+0.03%)**
VGG-Net	GD	0.1	512	50k	Baseline	8	88.14%
					Gaia	8	87.19%(-0.95%)
					TernGrad	8	86.3%(-1.84%)
					WeightGrad	**8**	**88.13%(-0.01%)**

Table 2: Result of WeightGrad on CIFAR-10 dataset

Model	Steps	Training Method	Top-1 Accuracy	Top-5 Accuracy
AlexNet	185k	Baseline	58.17%	80.19%
		Gaia	58.02%(-0.15%)	80.20%(+0.01%)
		TernGrad	57.32%(-0.85%)	80.18%(-0.01%)
		Deep Gradient Compression [30]	58.20%(+0.03%)	80.20%(+0.01%)
		WeightGrad	**59.28%(+1.06%)**	**80.25%(+0.06)**

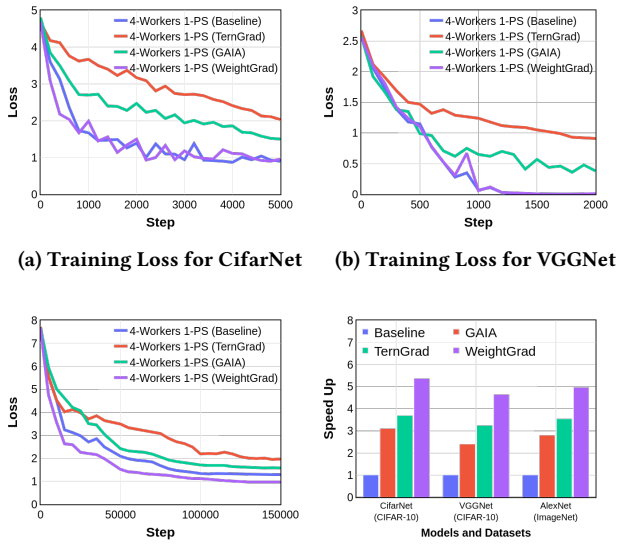
Table 3: Comparison of training methods on ImageNet data



(a) Loss Vs Step **(b) Loss Vs Time (hour)**

Figure 4: Convergence of WeightGrad for CIFAR10 dataset

with ternary gradients while keeping other hyper-parameters unchanged. The approximate and inconsistent values of parameters are generated in each epoch by the worker machines because of using average gradients received from the parameter server and the average gradients further deviate from the exact values of gradients because of quantization. Moreover, TernGrad needs to distribute the same scaler among all the worker machines so that the number of gradient levels after summing up the gradients in the parameter server can be minimized. This sharing creates additional overhead of tranferring $2N$ floating scalers. WeightGrad solves the accuracy problem using the quantized gradients serially instead of averaging and applying momentum correction, gradient clipping, smaller learning rate discussed in Section 4.3. Figure 5c shows that WeightGrad closely follows full-precision baseline system. Table 3 summarizes the result of AlexNet training on ImageNet with same number of steps. WeightGrad converges to the baseline accuracy level and top-1 error of training with WeightGrad decreases faster than the baseline with same training loss, which results in 1.06% gain in accuracy where only Deep Gradient Compression shows 0.03% gain and others show loss in accuracy.



(a) Training Loss for CifarNet **(b) Training Loss for VGGNet**

(c) Training Loss for ImageNet **(d) Training Speed Comparison**

Figure 5: (a) Training loss for CifarNet model on CIFAR-10 dataset (b) Training loss for VGGNet model on CIFAR-10 dataset (c) Training loss for AlexNet on ImageNet dataset (d) Training speedup comparison from convergence time with the WAN b/w among Ohio, N. California and N. Virginia

5.3.3 Convergence Speed Comparison. Figure 5d shows the convergence speed gain of WeightGrad compared to other systems. WeightGrad can have upto 5.36× speedup than baseline in CifarNet using CIFAR-10 dataset, for parameter intensive deep VGGNet model, WeightGrad shows 4.64× and for ImageNet 4.96× speedup over baseline. WeightGrad gains 2.16×-2.26× speedup over Gaia and 1.4×-1.67× over TernGrad training different DNN models.

6 CONCLUSION

In this paper, we present WeightGrad, a loss-aware weight-quantized deep learning system with quantized gradient that efficiently runs complex deep neural networks with billions of parameters on globally generated data over WAN. WeightGrad maintains a tight synchronization over LAN to simulate each data center as a centralized system and a loose synchronization over WAN to reduce communication cost. Empirical experiments comply with our claim and demonstrate that WeightGrad can achieve significant speedup in training complex DNN models maintaining the similar and even better accuracy compared to the full-precision baseline and other state-of-the-art systems.

ACKNOWLEDGMENTS

This research is partially supported by AWS Cloud Credits for Research.

REFERENCES

[1] [n.d.]. AWS. http://aws.amazon.com/about-aws/global-infrastructure/.
[2] [n.d.]. AWS EC2 Instance. https://aws.amazon.com/ec2/instance-types/.
[3] [n.d.]. Google Datacenter Locations. http://www.google.com/about/datacenters/inside/locations/.
[4] [n.d.]. isperf3. https://software.es.net/iperf/.
[5] [n.d.]. Microsoft Datacenters. http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx.
[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi
[8] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J Smola. 2012. Scalable inference in latent variable models. In *Proceedings of the fifth ACM international conference on Web search and data mining*. 123–132.
[9] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse Communication for Distributed Gradient Descent. arXiv:1704.05021 [cs.CL]
[10] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. arXiv:1610.02132 [cs.LG]
[11] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. 2011. Parallel Coordinate Descent for L1-Regularized Loss Minimization. arXiv:1105.5379 [cs.LG]
[12] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting Distributed Synchronous SGD. arXiv:1604.00981 [cs.LG]
[13] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.
[14] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. arXiv:1602.02830 [cs.LG]
[15] Henggang Cui, Alexey Tumanov, Jinliang Wei, Lianghong Xu, Wei Dai, Jesse Haber-kucharsky, Qirong Ho, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. [n.d.]. Exploiting iterative-ness for parallel ML computations.
[16] Henggang Cui, Hao Zhang, Gregory Ganger, Phillip Gibbons, and Eric Xing. 2016. GeePS: scalable deep learning on distributed GPUs with a GPU-specialized parameter server. 1–16. https://doi.org/10.1145/2901318.2901323
[17] Nikoli Dryden, Sam Ade Jacobs, Tim Moon, and Brian Van Essen. 2016. Communication Quantization for Data-parallel Training of Deep Neural Networks. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments* (Salt Lake City, Utah) *(MLHPC '16)*. IEEE Press, Piscataway, NJ, USA, 1–8. https://doi.org/10.1109/MLHPC.2016.4
[18] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, Jul (2011), 2121–2159.
[19] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677 [cs.CV]
[20] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. arXiv:1502.02551 [cs.LG]
[21] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149 [cs.CV]
[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]
[23] Lu Hou, Ruiliang Zhang, and James T. Kwok. 2019. Analysis of Quantized Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ryM_IoAqYX
[24] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 629–647. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh
[25] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]
[26] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[27] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n.d.]. CIFAR-10 (Canadian Institute for Advanced Research). ([n. d.]). http://www.cs.toronto.edu/~kriz/cifar.html
[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
[29] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. 2017. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952* (2017).
[30] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. 2017. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. arXiv:1712.01887 [cs.CV]
[31] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. 2011. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. arXiv:1106.5730 [math.OC]
[32] Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. 2016. Faster CNNs with Direct Sparse Convolutions and Guided Pruning. arXiv:1608.01409 [cs.CV]
[33] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*.
[34] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs. In *Interspeech 2014* (interspeech 2014 ed.). https://www.microsoft.com/en-us/research/publication/1-bit-stochastic-gradient-descent-and-application-to-data-parallel-distributed-training-of-speech-dnns/
[35] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]
[36] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[37] Nikko Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *INTERSPEECH*.
[38] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. arXiv:1705.07878 [cs.LG]
[39] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and Inference with Integers in Deep Neural Networks. arXiv:1802.04680 [cs.LG]
[40] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. 2016. Asynchronous Stochastic Gradient Descent with Delay Compensation. arXiv:1609.08326 [cs.LG]
[41] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *CoRR* abs/1606.06160 (2016). http://arxiv.org/abs/1606.06160
[42] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*. 928–936.

A APPENDIX A

A.1 Convergence Analysis and Theorems

We use the online learning framework proposed in [42] to analyze the deep learning optimizers. At each time $t \in T$, the algorithm selects a parameter $\theta_t \in \Theta$ for the model which incurs a loss of $f_t(\theta_t)$. Our goal is to find the best model parameter $\theta_t^* \in \Theta$. So the performance can be evaluated by measuring the distinction between them which is usually known as regret:

$$R(T) = \sum_{t=1}^{T} f_t(\theta_t) - f_t(\theta_t^*) \tag{1}$$

where $\theta_t^* = argmin_{\theta \in \Theta} \sum_{t=1}^{T} f_t(\theta)$ and average regret $R(T)/T$. So, the convergence of model depends on the average regret value. As the model will converge if $f_t(\theta_t)$ converges to $f_t(\theta_t^*)$ with $t \to \infty$, i.e, $R(T)/T \to 0$. For online learning framework [25] proved that, $\frac{R(T)}{T} = O(\frac{d}{\sqrt{T}})$, i.e, $lim_{T\to\infty} \frac{R(T)}{T} = 0$. Gaia [24] proved convergence for distributed SGD under ASP (Approximate Synchronous Parallel). In contrast to WeightGrad, we need to prove this under quantized weights and quantized gradients scenario instead of full-precision network. For this we use the two theorem stated and proved in [23].

(a) Weight Quantization: Let the full-precision weights from all L layers in the deep network be w. The corresponding quantized weight is denoted $Q_w(w) = \hat{w}$, where $Q_w(\cdot)$ is the weight quantization function.

Before applying quantization to the weight, a gradient descent is obtained from the previous weight [23],

$$\hat{w}_{t+1} = Q_w(w_{t+1}) = Q(w_t - \eta_t Diag(\sqrt{\hat{v}_t})^{-1} \hat{g}_t) \tag{2}$$

where $\hat{v}$ is the moving average and $\hat{g}_t$ is the full-precision gradient. Moving average at timestamp t can be written as a function of the gradients at all previous timesteps [25]:

$$\hat{v}_t = (1-\beta) \sum_{i=1}^{t} \beta^{t-i} \hat{g}_i^2 \tag{3}$$

with $\beta \simeq 1$, used in popular DNN optimizers [25].

For simplicity Equation 2 can be written as:

$$\begin{aligned} \hat{w}_{t+1} &= argmin_{\hat{w}} ||w_{t+1} - \hat{w}||^2_{Diag(\sqrt{\hat{v}_t})} \\ s.t., \hat{w} &= \alpha b, \alpha > 0, b \in (S_w)^d \end{aligned} \tag{4}$$

Here $S_w = \{-C_l, \ldots, -C_1, C_0, C_1, \ldots, C_l\}$ and $l = 2^{k-1} - 1$, for k-bit quantization with weight quantization resolution $\Delta_w = C_{l+1} - C_l$. For TernGrad [38], $S_w = \{-1, 0, 1\}$ and $\Delta_w = 1$.

(b) Gradient Quantization: For gradient quantization, we observe the quantization of TernGrad [38]. Quantized gradient $\tilde{g}_t$ can be formulated as:

$$\tilde{g}_t = Ternarize(g_t) = s_t \cdot sign(g_t) \odot b_t \tag{5}$$

where, $s_t = ||g_t||_\infty \triangleq \max(abs(g_t))$, $\odot$ is the Hadamard product and each element of b_t independently follows the Bernoulli distribution,

$$\begin{cases} P(b_{tk} = 1 | g_t) = |g_{tk}|/s_t \\ P(b_{tk} = 0 | g_t) = 1 - |g_{tk}|/s_t \end{cases} \tag{6}$$

here, $b_{tk}, g_{tk} = k-$th element of b_t, g_t, respectively. b_t can be expressed as another way, $b_t \in (S_g)^d$, where $S_g = \{-P_l, ..., -P_1, P_0, P_1, ..., P_l\}$ and $l = 2^{k-1} - 1$, for k-bit quantization with gradient quantization resolution $\Delta_g = P_{l+1} - P_l$.

(c) Average Regret for Weight-Gradient Quantized System: To formulate the error related to quantized weights and quantized gradients, we make some assumptions on loss function f_t commonly used in convex online learning [18] [23]:

- f_t is convex (the convexity assumption does not hold for nonconvex deep nets)
- f_t is twice differentiable with Lipschitz-continuous gradient
- f_t has bounded gradient i.e, $||\nabla f_t(w)|| \le G$ and $||\nabla f_t(w)||_\infty \le G_\infty$ for all $w \in S$ where S is convex set holding the model weights.

Though the convex f_t assumption does not hold for DNN, it facilitates the analysis of DNN models and helps to explain the empirical behavior. We assume that $||w_i - w_j|| \le D$ and $||w_i - w_j||_\infty \le D_\infty$, for $w_i, w_j \in S$.

Now for full precision gradient and quantized weights the average regret converges to Theorem A.1, which is:

THEOREM A.1. *[23] For quantized weight with full-precision gradients and $\eta_t = \eta/\sqrt{t}$,*

$$R(T) \le \frac{D_\infty^2 \sqrt{dT}}{2\eta} \sqrt{\sum_{t=1}^{T} (1-\beta)\beta^{T-t} ||\hat{g}_t||^2} + \frac{\eta G_\infty \sqrt{d}}{\sqrt{1-\beta}} \sqrt{\sum_{t=1}^{T} ||\hat{g}_t||^2} + \sqrt{L} D \sum_{t=1}^{T} \sqrt{||w_t - \hat{w}_t||^2_{H'_t}} \tag{7}$$

$$\frac{R(T)}{T} \le O\left(\frac{d}{\sqrt{T}}\right) + LD\sqrt{D^2 + \frac{d\alpha^2\Delta_w^2}{4}} \tag{8}$$

From [25] $\frac{R(T)}{T} = O(\frac{1}{\sqrt{T}})$, for online learning framework proposed in [42]. From Theorem A.1, the average regret converges at the same rate but with an error of $LD\sqrt{D^2 + \frac{d\alpha^2\Delta_w^2}{4}}$, which is related to the weight quantization resolution Δ_w and dimension d. Detailed proof is provided by [23].

For both quantized weights and gradients the error can be formulated from the Theorem A.2.

THEOREM A.2. *[23] For quantized weights with quantized gradients and $\eta_t = \eta/\sqrt{T}$,*

$$\begin{aligned} E(R(T)) \le \frac{D_\infty^2 \sqrt{dT}}{2\eta} \sqrt{\sum_{t=1}^{T} (1-\beta)\beta^{T-t} E(||\hat{g}_t||^2)} + \\ \frac{\eta G_\infty \sqrt{d}}{\sqrt{1-\beta}} \sqrt{\sum_{t=1}^{T} E\left(||\hat{g}_t||^2\right)} \\ + \sqrt{L} D \sum_{t=1}^{T} E\left(\sqrt{||w_t - \hat{w}_t||^2_{H'_t}}\right) \end{aligned} \tag{9}$$

$$E\left(\frac{R(T)}{T}\right) \le O\left(\sqrt{\frac{1+\sqrt{2d-1}}{2}} \Delta_g + 1.\frac{d}{\sqrt{T}}\right) + LD\sqrt{D^2 + \frac{d\alpha^2\Delta_w^2}{4}} \tag{10}$$

From the comparison between Theorem 8 and 10, quantized gradients slows convergence by a factor of $\sqrt{\frac{1+\sqrt{2d-1}}{2}\Delta_g + 1}$ which is dependent upon dimension d and gradient quantization resolution Δ_g. So, for complex deep neural net models with huge numbers of parameters result in large value of d and to reduce communication cost we need fewer bits for these complex DNN, which results in large Δ_g.

To eliminate the slow convergence due to quantized gradient, [23] provides Theorem A.3, which is based on the idea of gradient clipping before quantization, which is:

THEOREM A.3. *[23] Assume that $\hat{g}_t$ follows $\mathcal{N}\left(0, \sigma^2 I\right)$. For loss-aware weight quantization with quantized clipped gradients and $\eta_t = \eta/\sqrt{T}$,*

$$E(R(T)) \leq \frac{D_\infty^2 \sqrt{dT}}{2\eta} \sqrt{\sum_{t=1}^{T} (1-\beta)\beta^{T-t} E(||\hat{g}_t||^2)} + \frac{\eta G_\infty \sqrt{d}}{\sqrt{1-\beta}} \sqrt{\sum_{t=1}^{T} E\left(||\hat{g}_t||^2\right)} + \sqrt{L} D \sum_{t=1}^{T} E\left(\sqrt{||w_t - \hat{w}_t||^2_{H'_t}}\right) + D \sum_{t=1}^{T} E\left(\sqrt{||Clip(\hat{g}_t) - \hat{g}_t||^2}\right) \tag{11}$$

$$E\left(\frac{R(T)}{T}\right) \leq O\left(\sqrt{\left(\frac{2}{\pi}\right)^{\frac{1}{2}} c\Delta_g + 1.} \frac{d}{\sqrt{T}}\right) + LD\sqrt{D^2 + \frac{d\alpha^2\Delta_w^2}{4}} + \sqrt{d} D\sigma \left(\frac{2}{\pi}\right)^{\frac{1}{4}} \sqrt{F(c)} \tag{12}$$

Here we can see that gradient clipping before quantization slows convergence by a factor $\sqrt{\left(\frac{2}{\pi}\right)^{\frac{1}{2}} c\Delta_g + 1}$ but convergence rate does not depend on the dimension d rather it depends on clipping factor c and gradient quantization resolution Δ_g, hence a large Δ_g can be used which can significantly reduce communication cost. So, in WeightGrad we use this technique with momentum correction. DGC [30] obtained upto 0.19% accuracy gain only by adapting momentum correction on compressed gradients.

A.2 Dynamic Threshold, G_{thresh}

To obtain better threshold for distribution of significant update across the data centers, LPS maintains threshold G_{thresh}, initially set as $G_{thresh} = \frac{||g_t||_2}{||g_t||_\infty}$. WeightGrad updates threshold value as:

$$G_{thresh}^{s+1} = \delta_\eta \cdot \Delta_v \cdot G_{thresh}^{s} \tag{13}$$

here δ_η is the change in learning rate with epoch s and Δ_v is the change in momentum correction for $v_{s+1} = v_s + \nabla_s$, $\nabla_s = \frac{1}{Nb} \sum_{x \in WM_{iX}} \nabla f(x, w_s)$; WM_{iX} dataset for WM_i where total N workers and $f(x, w)$ is the loss computed from samples $x \in WM_{iX}$.

B APPENDIX B

B.1 Centralized training in guise of Distributed setup

In the centralized training process, gradients generated from forward passing are fed directly in backpropagation. Gradients from each mini-batch are considered as-they-are for determining the weights and biases for next epoch. It helps the model to choose the best weights and biases and thus ensures convergence and best accuracy. But in distributed training, we don't have this flexibility where each Local Parameter Server collects gradient and distributes the average gradient among the workers. Here, we make the trade-off between training efficiency and model accuracy by considering approximate gradients (average gradients) that produces approximate weights and biases for next epoch. It creates a gap between centralized (standard/true) training and distributed training (approximate) and this gap increases with each epoch. This gap is the major reason that results in divergence of the model in distributed training which never happens in the case of centralized training. In this paper, we claim that our system simulates the centralized training under the guise of distributed training. In WeightGrad, as a parameter server can control parameter-update phase, it can use gradients directly without averaging or approximating and produce the exact weights and biases that could have been produced in centralized training. This distinction in our system gives major boost up in accuracy compared to other state-of-the-art systems and ensures convergence. Partitioning the deep learning model between worker machines and the parameter servers offers all the benefits from distributed training e.g, scalability, resource constraints, faster training for bigger data set etc and also preserves the notion of centralized training by ensuring convergence and the best accuracy where other state-of-the-art systems make trade-off between training efficiency and accuracy.

B.2 Challenges of incorporating Parameter update in a single parameter server

In WeightGrad, we execute the inference (forward propagation) and gradient descent parts of neural network in the worker machine and parameter-update part in parameter server. But in neural network, the learning phase (backpropagation) is slower than the inference phase (forward propagation)[1] as we have to consider additional gradient descent calculation in learning phase which often has to be repeated many times. For state-of-the-art systems, they execute inference and backpropagation parallelly in worker machines and do the aggregation part in parameter server. But in WeightGrad, we are distributing two tasks between the worker machines and parameter servers. Deploying the parameter-update phase in a single parameter server inside a local data center can create a little slowdown in speed and convergence if there are many worker machines assigned to a single LPS. To address this problem we have assigned at most 5 worker machines to a parameter server which reduces the computational complexity in the parameter server. It also provides accuracy gain which the former systems were trading off against speed.

[1] https://kasperfred.com/series/introduction-to-neural-networks/computational-complexity-of-neural-networks