eSGD: Commutation Efficient Distributed Deep Learning on the Edge

Zeyi Tao
College of William and Mary

Qun Li
College of William and Mary

Abstract

Training machine learning model on IoT device is a natural trend due to the growing computation power and the great ability to collect various data of modern IoT device.In this work, we consider an edge based distributed deep learning framework in which many edge devices collaborate to train a model while using an edge server as the parameter server. However, the high network communication cost of synchronizing gradients and parameters between edge devices and cloud is a bottleneck. We propose a new method called edge Stochastic Gradient Descent (eSGD) for scaling up edge training of convolutional neural networks. eSGD is a family of sparse schemes with both convergence and practical performance guarantees. eSGD includes two mechanisms to improve the first order gradient based optimization of stochastic objective functions in edge scenario. First, eSGD determines which gradient coordinates are important and only transmits important gradient coordinates to cloud for synchronizing. This important update can aggressively reduce the communication cost. Second, momentum residual accumulation is designed for tracking out-of-date residual gradient coordinates to avoid low convergence rate caused by sparse updates. Our experiments show that we reach 91.2%, 86.7%, 81.5% accuracy on MNIST data set with gradient drop ratio 50%, 75%, 87.5% respectively.

1 Introduction

The Internet-of-things (IoT) has been widely used in many areas. Billions of edge devices have already been deployed for monitoring natural environment, predicting weather, smart housing, cities, personalized fitness and healthcare, etc [14]. Edge devices successfully prove their great ability to collect realtime, local, sensor-based data from a variety of environments. We also notice that these edge devices can easily access wealth of data suitable for machine learning model [4], which in turn can greatly improve user's QoE. In recent years, not surprisingly, training on a personal mobile device is appealing due to the concerns about better privacy and better personalization [6], but a critical problem is current edge machine learning is so far limited to cloud-based training. Edge devices such as mobile phones, environment sensors, HD cameras and other portable devices fetch the real time environment data and push this data to cloud server. The cloud server collects data from all edge nodes, and trains the large scale machine learning (ML) model according to this data, then sends the training result back to edge devices. The cloud-based training therefore suffers from low throughput, high latency, and expensive mobile data plan [11]. To mitigate the above issues, we prefer the training process can be performed on local devices. That is, the real time data can be used to train in local ML model without pushing it to cloud server and waiting for responses from cloud.

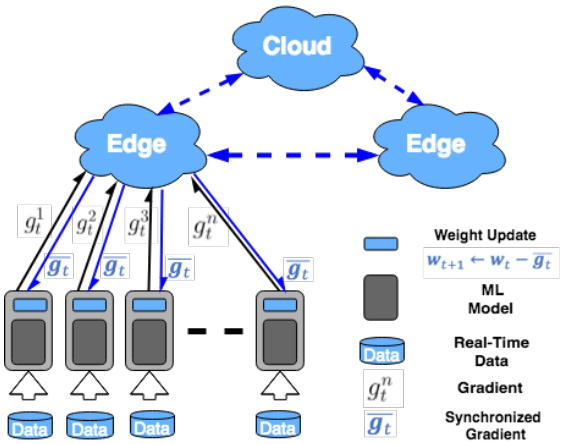


Figure 1: Example of Edge Training

Inspired by distributed training system [1, 8, 4, 5, 13] and recent work DIANNE [7], we deploy the ML train-

ing model on edge devices, and make the edge serve function as parameter server so that we can both execute and train the ML model on the edge device. Figure 1 illustrates the edge training framework. Edge devices keep fetching the trainable environment data and run learning model locally, and separately. As a result, users can receive real-time predictions and at same time, data parallelism is adopted to exploit the compute capability empowered by multiple edge devices. To achieve this, we chose stochastic gradient descent (SGD) as optimization method due to its high computation efficiency. As we mentioned above, the copy of learning model in each edge device is trained separately by feeding individual data. And the edge server performs gradient synchronization by collecting all gradients and averaging them to update parameters. The updated parameters will be sent back to each edge node for next training step, which is known as parameter synchronization [19]. However, as the scale of edge training grows up, a large amount of training metadata including gradients and parameters will be pushed to the edge server. This will result in a high communication overhead due to low network bandwidth and highly frequent gradient exchange [15, 20].

Edge Stochastic Gradient Descent (eSGD) solves the communication bandwidth problem by taking advantage of gradient sparsification. In fact, an empirical observation mentioned that *training gradients* are very sparse and most of weights have value close to zero [16]. Here, *training gradients* are series of multidimensional vectors and we denote the element of training gradient as **gradient coordinate**. Due to the limited training samples on edge device, sub-gradients are normally even sparser than the weight distribution. Thus we notice that only a small fraction of the gradient coordinates are required to be updated after each mini-batch. To guarantee zero accuracy loss, eSGD employs two mechanisms: important updating and momentum residual accumulation. To determine which gradient coordinates are selected to be synchronized, we introduce a historical liked method called random weight selection to select most popular gradient coordinates according on its hidden weight. *Hidden weight* is a real positive value. Every time gradient coordinate participates in gradient synchronization, the hidden weight value associate with this gradient coordinate is increased. Therefore, large hidden weight value indicates that the coordinate participates synchronization more frequently and is more likely to be selected than other gradient coordinates in the next round, that is, importance update. Meanwhile momentum residual accumulation is applied for tracking and accumulating unsynchronized coordinates value for delay updates. We empirically verified eSGD performance as desired on MNIST data set. We have achieve 81.5%, 86.7%, 91.2% accuracy with dropping ratio 87.5%, 75% 50% respectively.

2 Related Work

In recent year, most of communication efficient approaches are done by distributed training scheme. Each worker keeps a copy of training model and for each iteration, all gradients from workers are synchronized and averaged at parameter server and then sent back to update workers. To accelerate training speed, asynchronous SGD [10] was proposed by updating gradient immediately when the training worker finished their batch training. To reduce the communication cost between work and parameter server, two type approaches include vector quantization and gradient sparsification are well studied.

Vector Quantization Quantizing the gradient vector to low precision value or approximation value are commonly used. Seide et al. [9] proposed 1-bit SGD to reduce gradients transfer data size and track the quantization error by adding it into the respective next mini-batch gradient before quantization which achieved 10x speedup in traditional speech DNNs. Similarly, Zhou et al. [18] created DoReFa-Net which uses 1-bit weights and 2-bit gradients to exchange information in network. Another approximation based approach is Wen et al. [19] developed TernGrad which uses 3-level $\{-1,0,1\}$gradients. Alistarh et al. [17] proposed QSGD which gives a general method of quantized gradient and it balances the trade-off between accuracy and gradient precision.

Gradient Sparsification This type of work follows the very simple intuition that not full gradient will be update to server, instead only some important gradient's value(coordinate-based) will participate averaging at parameter server. How to choose important gradient value to update is major task of gradient sparsification. Strom et al. [16] proposed threshold gradient sparsification that is only gradient elements whose absolute values exceed a predefined threshold can be updated. However, the threshold is hard to choose in practice due to the variation of its value. Then Dryden et al. [12] chose a fixed proportion of gradient value to avoid the threshold drawback. Gradient Dropping, introduced by Aji & Heafield [2] found gradient updates are positively skewed as most updates are near zero. Gradient Dropping saves 99% of gradient exchange and 11% speed improvement but it has bad BLEU score. Chen et al. [21] proposed AdaComp which can adaptively adjusts compression ratios in different mini-batches, epochs, network layers and bins. AdaComp gained compression ratio around 200x for fully-connected layers and 40x for convolutional layers with negligible degradation of top-1 accuracy on ImageNet dataset.

3 eSGD Technique

3.1 Observations

The first empirical, observation is that sub-gradients are very sparse. In general, the weights of a fully connected DNN are sparsely distributed with most weight values close to zero [16]. Therefore, it is very obvious that the sub-gradients of these sparse weights are also sparse. In addition, considering the capacity on the edge devices, small training data set can greatly impact on the sparsity of weights and its sub-gradients. This leads to the core idea of our method, that is, only a small fraction of the parameters is necessarily to be updated after each training batch. Another observation made by

Algorithm 1 Edge Stochastic Gradient Descent

1: **Initialization** Hidden Weight $H_0 \leftarrow \mathbf{0}$, Parameter x_0, Fixed proportion $k\%$

EdgeNode:

2: Update parameter $x_t \leftarrow x_{t-1} - \bar{g}_{t-1}$
3: $threshold \leftarrow \sum_d g_{i,t-1}/d$
4: **if** $t > 1$ **then**
5: **if** $l(x_{t-1}) > l(x_t)$ **then**
6: $g_{i,t}^{update} \leftarrow g_{i,t-1}^{update}$
7: Record the index i of updating gradient coordinates
8: Update Hidden Weight $H_t(i)$ at given coordinates i
9: **else**
10: $g_t^{update} \leftarrow$ Weighted Random Selection $k\%$ coordinates from g_{t-1}
11: **end if**
12: Accumulate Residual Gradient $g_t^{residual} = Residual(g_{t-1}^{residual})$
13: **for all** $g_{i,t}^{residual} > threshold$ **do**
14: Replace $g_{j,t}^{update}$ with $g_{i,t}^{residual}$ according to least significant
15: **end for**
16: **else**
17: $g^{update} \leftarrow$ randomly select with fixed proportion $k\%$ from g_1
18: **end if**
19: Push g^{update} to Cloud/Fog

Cloud:

20: Average gradients $\bar{g}_t = \frac{1}{N}\sum_n g_t^{update,n}$
21: Send $\bar{g}_t$ back to edge node

[11] is that many techniques for optimizing SGD can be presented as delaying parameter updates [16]. Generally, the distributed training performs the following synchronous SGD update with N training nodes:

$$x_t = x_{t-1} - \gamma \frac{1}{Nb} \sum_i^N \sum_{z \in B} \nabla L(x,z) \tag{1}$$

where x is the weights of network, γ is the learning rate, N is the number of training nodes, B is a sequence of samples from universal dataset $\mathscr{X}$ with size b and a loss objective function $L(x,z)$ is used to measure the performance of current system with parameter x and input z. We rephrase the above update equation (1) to coordinate-based form. We define $x_{i,t}$ is the value of weight in t^{th} iteration at coordinate i, after T iterations, we have:

$$x_{i,t} = x_{i,t-1} - \gamma T \cdot \frac{1}{NbT} \sum_i^N \sum_{z \in B} \sum_{t=0}^{T-1} \nabla L^i(x,z) \tag{2}$$

The equation above shows that local gradient accumulation can be regarded as changing the min-batch size from Nb to NbT. In this case, updates of sub-gradients for individual samples are delayed until the end of the sparse update interval.

3.2 Importance Updating

As we mentioned in section 2, neither threshold nor fixed proportion approaches are suitable for edge training. The main drawback behind gradients dropping approaches is how we define the importance of a single coordinate of the entire gradient so that important gradient coordinates will be selected for synchronizing and less important gradient coordinates are going to be ignored. In reality, drop-like approaches are only working on the large scale distributed training with plenty of computational resources. All workers keep a copy of training data and run training task with more than $10^k (4 \leq k \leq 8)$ epochs. However, in edge training, edge nodes have limited storage space and battery supply. Simply dropping the small value gradient coordinates can degrade training accuracy and incur occasional divergence [3].

Therefore, the main challenge of our work is how to select a group of important gradient coordinates so that training parameters can be converged fast without accuracy loss. To address this, we introduce our novel technique in Algorithm 1 line 5-8 named random hidden weight selection. We define $l(x)$ is the loss function which we want to minimize. Let $g_{i,t}^{update}$ denote the i^{th} coordinate of gradient g to be updated at time step t. $l(x_{t-1})$ and $l(x_t)$ are the loss value at time steps $t-1$ and $t-2$ with parameter x_{t-1} and x_t respectively. And there is a hidden weight value of each gradient coordinate which stores in H_t.

Intuitively, we want the loss function to be converged at each step, which means the loss value is getting close to the local optimal value. Following this basic trend,

in ideal case, the loss value on time step t should be smaller than its value on $t-1$. However, SDG method does not always act in this way. The loss value is erratic fluctuation. So, in our design, we keep tracking the loss values at two consecutive times $t-1$ and t. If $l(x_{t-1}) > l(x_t)$(line 5), it indicates that we have better result at time step t with current gradient g_t, we should record all index of participated gradient coordinates and label them as important gradient and sign a positive value to its hidden weight $H_t(i)$. For next iteration $t+1$, we use the same gradient coordinates at time t because we assume these gradient coordinates can be beneficial to our loss function.

Once $l(x_{t-1}) > l(x_t)$ does not hold, it indicates that at time t the loss function value is undesirable. Then we stop updating gradient coordinates by using same indexes gradient in previous step, instead, we randomly select the indexes among the all coordinates according to their hidden weight values $H_t(i)$. Why? Because a coordinate with large hidden weight value is more likely to be selected for next round and it also indicates that this coordinate is labeled as important many times during the training process.

Algorithm 2 Residual

1: **Initialization** Decay rate $\beta = 0.9$
2: **Input:** gradient $g_t$previous
3: **for all** coordinate i in g_t not update $g_{i,t}$ **do**
4: $\quad g_{i,t}^{residual} = \beta \cdot g_{i,t-1}^{residual} + (1-\beta) \cdot g_{i,t}$
5: **end for**
6: **Return:** $g_t^{residual}$

3.3 Momentum Residual Accumulation

In addition, we should also take care of the gradient coordinates value below the threshold. Because ignoring these small values will greatly harm convergence. To choose the threshold, we set a dynamic threshold for each iteration according to current gradient average value (Algorithm 1, line 3). We do not set a fix threshold due to the variation of gradient coordinate value and we also do not use fixed proportion approach to avoid the case that some small gradient values could never be used or updated.

We keep tracking the residual gradient value, in algorithm 1 line 12. On edge node, each iteration, small gradient values are accumulated in what we call the gradient residual accumulation. Since residual gradient coordinates have very different dynamic range, therefore, inspired by momentum SGD, and deep compression [11], we want to use momentum correction technique to overcome this issue. Momentum SGD is widely used in place of vanilla SGD. However, in our scenario, we do not apply the momentum to whole gradient but only use it on residual gradient (Algorithm 2). Since we delay the update of small gradients, when updates do occur, they are outdated and have weak influence for next iteration. Therefore, we set a discount factor β to correct residual gradient accumulation. This technique ensures that parameters with small but biased sub-gradient values are eventually updated whenever they reach the current threshold.

When residual gradients reach the threshold, we need to synchronize them to cloud server. Algorithm 1, line 13-15 shows how do we replace the updating gradient coordinates with qualified residual gradient coordinates. Remember, each gradient coordinate has a hidden weight value associated with, at this point, we sort the gradient coordinates in descending order via their hidden weight. To replace the updated gradient coordinate, we drop the least significant (small hidden weight value) coordinates and fill in with the qualified residual gradient coordinates.

4 Experiment

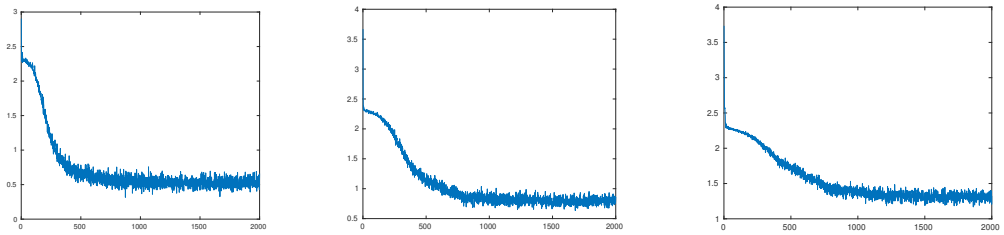
4.1 Experiment Settings

We validate our approach on MNIST data set. The experiments are performed by MATLAB 2018R. We first investigate the convergence of eSDG under various training schemers. We maintain the hyper-parameter for residual gradients to be exponential decay β as 0.9 and learning rate $\gamma = \frac{0.1}{2 \cdot e}$ where e is number of epochs. We fix the drop ratio as 50% and no drop for standard SGD. The accuracy is evaluated by compare the final training parameter result with MNIST test set. For fair comparison, in each experiment set, we training the model by using standard SGD, threshold SGD, and eSGD. Then we focus on experiment with different drop ratio. We perform each experiment set 3 times and pick the best result.

4.2 Result and Analysis

Figure 2 is the loss convergence with different drop ratio. It is clearly to see full gradient reaches the fast convergence rate, and if only 50% gradient coordinates are persevered, convergence rate can be degraded and for drop rate with 75% the result gets worse. However, standard SGD receive zero feedback from loss value, therefore, standard SGD converge with high frequency fluctuate but eSGD obviously is more smooth.

The first 3 rows in Table 1 shows that the accuracy result of 3 different training algorithms. Using same min-batch size, standard SGD optimization can reach the



a. no drop b. drop ratio=50 c.drop ratio=75

Figure 2: Convergence with different drop ratio

highest accuracy for 99.99% in the only 10 epochs. Compared with previous work by using fixed threshold value to drop the gradient coordinates, eSGD shows much better result with accuracy 86.01%, since it is not considerable for threshold-like method to choose a reasonable value. On the row 4-6, we exam training with eSGD but different min-batch size, this is useful because we assume training in different edge devices but has different capacity for training sample. We notice that eSGD is sensitive when mini-batch size changed. It is due to small mini-batch result in the gradient much more sparse and as the consequence, momentum residual accumulation process takes more time to reach the upload condition which can greatly impact the result of our experiment.

SGD	batch size	iterations	accuracy
std SGD	128	200000	99.77
eSGD	128	200000	86.01
thresholdSGD	128	200000	78.23
eSGD	32	200000	82.22
eSGD	64	150000	82.89
eSGD	128	100000	80.43

Table 1: Compare with stdSGD and threshold SGD

Drop Ratio	batch size	iterations	accuracy
25%	128	200000	95.31
50%	128	200000	91.22
87.5%	128	200000	88.46
75%	32	200000	83.85
75%	64	150000	83.76
75%	128	100000	81.13

Table 2: Drop ratio

In Table 2, we explore different drop ratio on MNIST CNN model. We see when the drop ratio is low such as 25% and 50%, we still can have high accuracy with 95.31% and 91.22% respectively. Compared to the threshold like method, when we drop more than 80% gradient, we still have desirable result.

5 Conclusions

Edge Stochastic Gradient Descent (eSGD) can shrink the gradient size by 90% of a CNN training model without slowing down the convergence rate. eSGD employs important updating method with a core technique of random weighted selection. To avoid the exploding gradient problem, eSGD applies momentum residual accumulation. Edge Stochastic Gradient Descent reduces the required communication bandwidth and improves the scalability of edge training .

6 Acknowledgments

The authors would like to thank all the reviewers for their constructive suggestions.

References

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, and Jeffrey Dean. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. In: *arXiv preprint:1603.04467* (2016).

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

[3] Xinghao Pan Jianmin Chen Rajat Monga Samy Bengio and Rafal Jozefowicz. “Revisiting distributed synchronous sgd”. In: *arXiv preprint:1702.05800* (2017).

[4] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems”. In: *arXiv preprint:1512.01274* (2015).

[5] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. "Project adam: Buildingan efficient and scalable deep learning training system". In: *USENIX Symposium on Operating Systems Design and Implementation* 14 (2014), pp. 571–582.

[6] *Collaborative machine learning without centralized training data*. URL: https://research.googleblog.com/2017/04/federated-learning-collaborative.html (visited on 05/02/2018).

[7] Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Steven Bohez, Sam Leroux, and Pieter Simoens. "DIANNE: Distributed Artificial Neural Networks for the Internet of Things". In: (2015), pp. 19–24. URL: http://doi.acm.org/10.1145/2836127.2836130.

[8] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. "Large scale distributed deep networks". In: *Advances in Neural Information Processing Systems* (2012), pp. 1223–1231.

[9] 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: *INTERSPEECH* (2014).

[10] Martin Zinkevich Markus Weimer Lihong Li and Alex J Smola. "Parallelized stochastic gradient descent". In: *In Advances in neural information processing systems* (2010), pp. 2595–2603.

[11] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training". In: *arXiv:1712.01887* (2018).

[12] Nikoli Dryden Sam Ade Jacobs Tim Moon and Brian Van Essen. "Communication quantization for data parallel training of deep neural networks". In: *In Proceedings of the Workshop on Machine Learning in High Performance Computing Environments* (2016).

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

[14] *Resource efficient ML for Edge and Endpoint IoT Devices*. URL: https://www.microsoft.com/en-us/research/project/resource-efficient-ml-for-the-edge-and-endpoint-iot-devices/ (visited on 05/02/2018).

[15] MuLi David G Andersen JunWooPark Alexander J Smola AmrAhmed VanjaJosifovski James-Long Eugene J Shekita and Bor-Yiing Su. "Scaling distributed machine learning with the parameter server". In: *OSDI* (2014), pp. 583–598.

[16] Nikko Strom. "Scalable Distributed DNN Training Using Commodity GPU Cloud Computing". In: *INTERSPEECH* (2015).

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

[18] Shuchang Zhou Yuxin Wu Zekun Ni Xinyu Zhou He Wen and Yuheng Zou. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160* (2016).

[19] 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: *arXiv print:1705.07878* (2017).

[20] Benjamin Recht Christopher Re Stephen Wright and Feng Niu Hogwild. "A lock free approach to parallelizing stochastic gradient descent". In: *In Advances in Neural Information Processing Systems* (2011), pp. 693–701.

[21] Chia-Yu Chen Jungwook Choi Daniel Brand Ankur Agrawal Wei Zhang and Kailash Gopalakrishnan. "Adacomp: Adaptive residual gradient compression for data-parallel distributed training". In: *arXiv preprint arXiv:1712.02679* (2017).