HPC2020 Project Report: Performance Comparison of Distributed Machine Learning with Different Communication Overheads

Hai Cheng
NUID: 001449866

April 17, 2020

1 Introduction

In the era of big data, machine learning (ML) especially deep learning (DL) is the most hot research field since it achieves near human level performance in computer vision applications (like image classification, object detection) and natural language processing application (like English to/from Chinese translation, speech to text generation). In (most) ML/DL, a large number of labeled data are required to feed the algorithm model. Due to the parallel processing over the dataset, it is intuitively to bring up with parallel training (like using Graphical Processing Card, GPU) or distributed training (like carry training work over different devices distributly). To make use of computation capacity of a high performance computer cluster, distributed training on multiple computers is a trend.

1.1 Communication in Distributed Systems

As we know, two key steps in distributed system are allocating workload to each slave node and gathering the intermediate results to a master node. Among the two steps, reliable and high speed communication links between master node and slave nodes are important for the distributed system. What's more, the communication links can be the bottleneck of the whole system since the system generally need all results to continue the next processing step.

A hot research topic in the intersection of Internet of Things(IoT) and Artificial Intelligence (AI) is the Edge Machine Learning. In edge machine learning, the training work are done in power-constraint devices, like mobile phone and other IoT devices. A natural extension from edge machine learning is distributed edge machine learning, or say distributed edge training. There are two challenges to be solved in distributed edge training:

- Since most edge devices communicate through wireless channel(like 4G/5G, WIFI, and so on), the spectrum resource would be more and more rare. Therefore, how to make the communication between those edge devices be more efficient is a big challenge.
- Since most edge devices is power-constraint and computation-constraint, the common neural network and its framework(like Tensorflow, PyTorch, and MXNet) cannot be used on edge devices directly. How to do the neural network training in a low-computation complexity and low-power cost manner need many researches.

1.2 Specific Problem to be Investigated

In this project, we consider the performance comparison of different distributed training schemes with various communication overheads. We try to investigate two major distributed training schemes: Distributed Stochastic Gradient Descent [1](called D-SGD in the later) and Federated Learning [4] (called FL in the later). Since the two scheme need different communication overhead, they are great benchmarks to show the performance comparison of different distributed training schemes.

Specifically, we assume that each user node(called user in the later) has its own dataset and is connected to a common center node(called cloud in the later) via a communication link. Each user trains a local model(or do optimization like Stochastic Gradient Descent) according to its local dataset and the cloud aggregates all local model (or local gradient vector) to update the global model. For different schemes, the communication rounds and the data size each round transmitted are different. Also, the performance of distributed training networks could be different. The main step of D-SGD and FL are given as follows.

1.2.1 D-SGD

The core steps of D-SGD are given by:

1. Initialise the weights for the global model.
2. Download the current global model.
3. Run a single step of forward prediction and backward error propagation for a given batch.
4. Compress the gradients (optional).
5. Upload the gradients to the cloud.
6. Decompress the incoming gradients(optional) and compute their average
7. Perform a single step of SGD to update the global model.
8. Go to step 2.

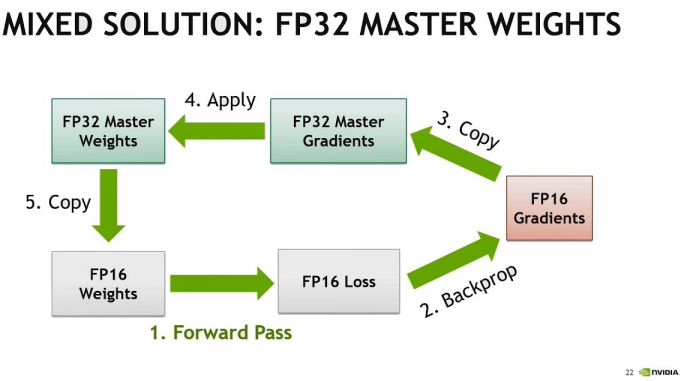

Figure 1: FP16 Neural Network Training.

1.2.2 FL

The core steps of FL are given by:

1. Initialise the weights for the global model.
2. Download the current global model at each user.
3. Run SGD iterations based on the local data.
4. Compress the updated model weights (optional).
5. Upload the update to the cloud.
6. Decompress the incoming models weights (optional).
7. Compute their average as the updated global model.
8. Go to step 2.

1.2.3 Mixed Precision Neural Network

Apart from the two schemes, we will also train low-precision neural network, which could be used in many kinds of edge devices, to compare the performance. What does that mean? In common neural network running on powerful server with GPUs, the model parameters (or say neural network weights) are represented in 32 bits floating numbers (FP32). This FP32 representation cost a lot of memory to storage the whole model and also cost a lot of communication resources. A natural way is to reduce the parameter representation from 32 bits (FP32) to 16 bits (FP16) or even 8 bits (INT8). Many literatures [3, 5, 6] demonstrate the low-precision parameters would not reduce the network performance too much.

A figure to depict the mixed precision training is given in Fig.1.

1.3 What experiment to operate and What results to expect

1.3.1 Experiments to operate

As mentioned above, we will compare D-SGD and FL two distributed training schemes. What's more, the classical SGD (the non-distributed or centralized SGD, called C-SGD in the later) will serve as baseline scheme in the final performance comparison. Thus, the performance of 3 training schemes would be compared. For each scheme, FP32 neural network and FP16 neural network would be both implemented and the corresponding performance would be compared. Note that the performance indicates the validation accuracy of the trained neural network.

1.3.2 Prospective results

For different schemes, we should see apparent performance and communication trade-off. That is to say, a large communication overhead corresponds to a better performance while a small communication overhead corresponds to a worse performance. Also, different schemes may have different trade-off.

For FP32 and FP16 neural network implementations, we should see a little performance loss in FP16 neural network when compared to FP16 neural network.

2 Pytorch-based CIFAR10 Classification ResNet Training

In the project, we consider CIFAR10 image classification with Pytorch based Residual Network (ResNet) implementation. For distributed training, Pytorch support Gloo, MPI, NCLL 3 backends for communication in distributed system. We use NCCL since it is the best for GPU/CUDA. Details about CIFAR10, PyTorch and NCLL are given as follows.

- The CIFAR-10 is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. For each class, there are 6,000 images where 5000 images are in the training set and 1000 images are in the testing set. Thus, CIFAR10 is splitted as a training set with 50000 images and testing set with 10000 images.

- PyTorch is an open source machine learning library based on the Torch library. It is primarily developed by Facebook's AI Research lab (FAIR) and now is one of the most widely used deep learning framework.

- NCCL: The NVIDIA Collective Communications Library (NCCL) is a library of multi-GPU collective communication primitives that are topology-aware and can be easily integrated into many machine learning framework. It is similar to the famous MPI library .

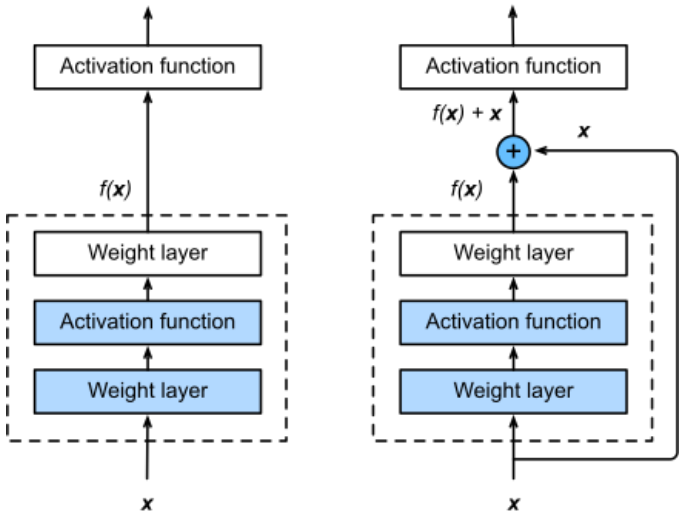

Figure 2: The difference between a regular block (left) and a residual block (right)

2.1 Residual Neural Network

Residual Neural Network(ResNet) is proposed by Kaiming He in [2]. Recently, ResNet has shown its ability in image classification. Thus, we use ResNet to do the CIFAR10 classification job.

ResNet is a sequence of basic building block. A building block is a sequence of layers. For a normal block, it can be represented as

$$\mathrm{y} = f(\mathrm{x}).$$

For a residual block, it can be represnted as

$$\mathrm{y} = f(\mathrm{x}) + \mathrm{x}.$$

We see that the extra term x is the residual. This residual term is the base of the fact: a deeper neural network could not be worse than a shallow neural network. Since the investigation of neural network architectura is far from the topic of this project, we present the architecture of ResNet building block as well as the architecture of ResNet18(18 building blocks). Note that ResNet18 is used in our training.

The basic residual block is given in Fig.2, Fig. 3, and Fig. 4.

2.2 Nodes in Discovery Cluster

To train the ResNet18, we use the Discovery cluster GPU partition with Nvidia V100-sxm2 GPU. In GPU partition, we can apply N nodes serving as N user where there is local dataset and would do training job locally. The classification accuracy will be measured in cloud.

In Discovery cluster, gpu partition and multigpu partition can offer Nvidia GPU for neural network training. Since i doesn't get the approval for the accesss to multigpu partition, i

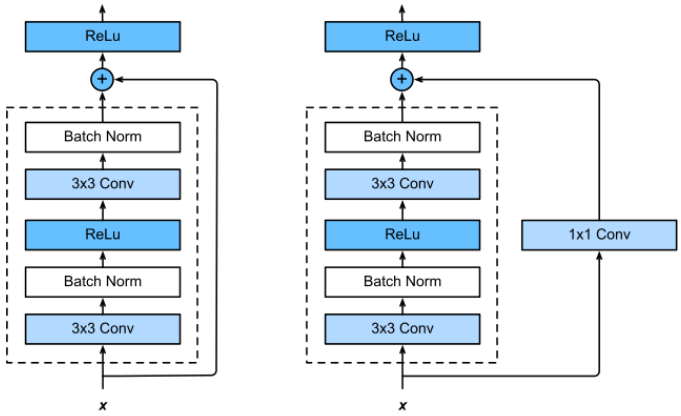

Figure 3: Left: regular ResNet block; Right: ResNet block with 1x1 convolution

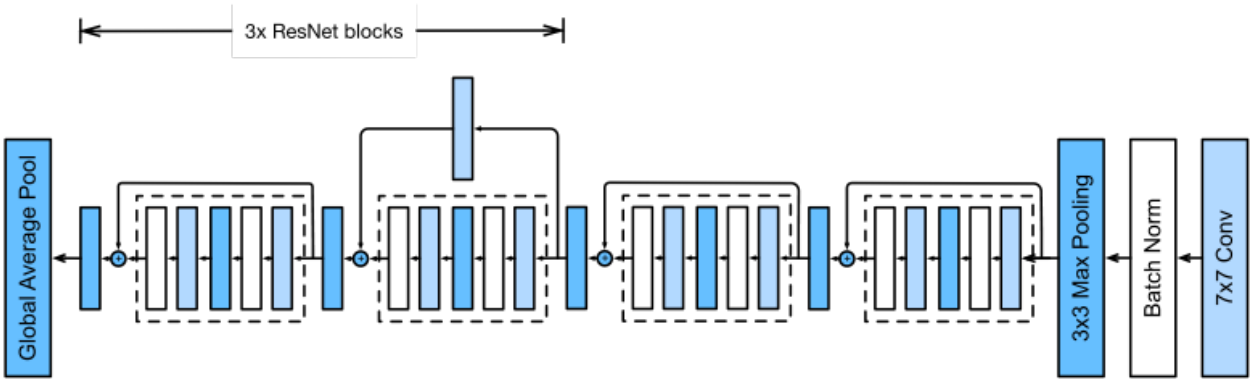

Figure 4: ResNet 18

apply one node(with one GPU) in gpu partition each time. We can have multiple gpu nodes access in the same time. The slurm command to apply one gpu nodes in interactive mode is: "*srun –pty –export=ALL –partition=gpu –gres=gpu:v100-sxm2:1 –tasks-per-node 1 –nodes 1 –mem=10Gb /bin/bash*".

3 Experiment Results

3.1 Performance Comparison

We first describe the super parameters used in ResNet18 training. The following setting are applied to all training jobs:

1. Dataset pre-processing: we use "*transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))*" to normalize both training dataset and testing dataset.

2. Loss function: Cross Entropy loss function is used.

3. Optimizaer: SGD optimized is used with constant learning rate 0.1, momentum $= 0.9$, weightdecay $= 1e-4$.

4. Epoches: 100 epoches, no extra terminating condition.

5. Number of users: 4 GPU nodes are used to serve as 4 users. The cloud is served by first node(the node with rank $= 0$).

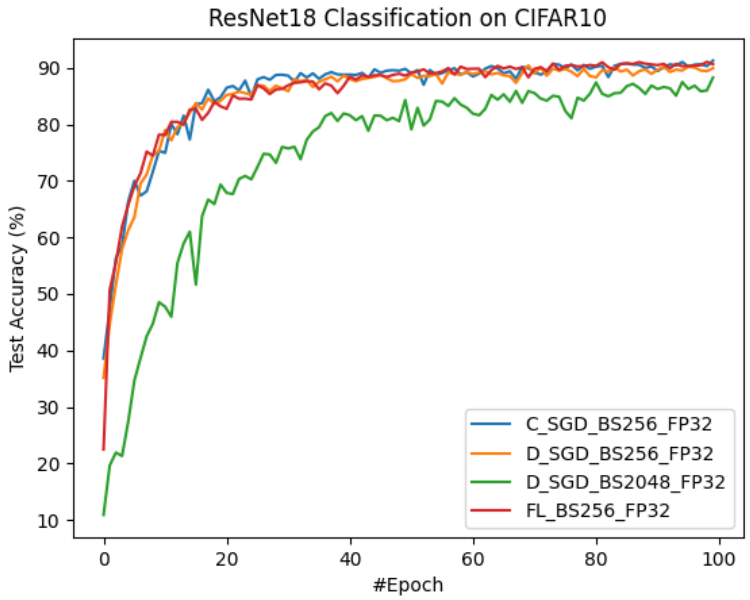

Figure 5: Validation accuracy of ResNet18 on CIFAR10 with FP32 precision.

The CIFAR10 classification validation accuracy are given in Fig. 5, 6. The result of Fig. 5 are from FP32 precision neural network. The four schemes are

- C_SGD_BS256_FP32: C-SGD with batch size 256.
- D_SGD_BS256_FP32: D-SGD with batch size 256. The (local) batch size in each node is $256/4 = 64$. Each node send their gradient to cloud after one (local) batch.
- D_SGD_BS2048_FP32: D-SGD with batch size 2048. The batch size in each node is $2048/4 = 512$. Each node send their gradient to cloud after one (local) batch.
- FL_BS256_FP32: FL with local batch size 256. Each node send their local model to cloud after one local epoch ($50000/4 = 12500$).

The result of Fig. 6 are from FP16 mixed precision neural network. The four scheme are the same as the schemes in Fig. 5, except the precision.

From the two figures, we see that for batch size 256, the performance of C-SGD, D-SGD, FL are almost the same. Even with precision loss, the performance loss is negligible.

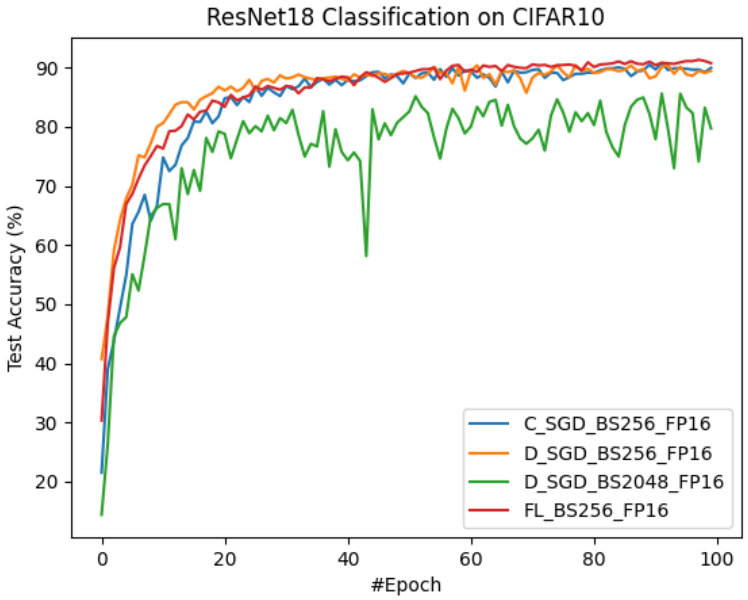

Figure 6: Validation accuracy of ResNet18 on CIFAR10 with FP16 mixed precision.

3.2 Communication Overhead

In ResNet18, there are $11,173,962$ parameters. That indicates

$$11,173,962 \times 4B \approx 44MB$$

model storage size with FP32 precision or

$$11,173,962 \times 2B \approx 22MB$$

model storage size with FP16 precision. Note that the number of gradients and the precision of gradients are the same as model parameters. The total communication data size are calculated by

$$\#\text{Round} \times \text{sizeperround}.$$

Therefore, for the above four schemes in FP32 precision, the total communication data size per epoch are

- C_SGD_BS256_FP32: $0 \times 44MB = 0MB$.
- D_SGD_BS256_FP32: $(50000/256) \times 44MB \approx 8.6GB$.
- D_SGD_BS2048_FP32: $(50000/2048) \times 44MB \approx 1.1GB$.
- FL_BS256_FP32: $1 \times 44MB \approx 44MB$.

We see that different training scheme have very different communication overhead. Among the above cheme, FL is the best one in terms of communication overhead. In addition, by utilizing FP16 precision, we can achieve half communication overhead with almost no performance loss.

4 Conclusion

In this project, we consider the performance of various distributed training scheme. Specifically, we implement ResNet18 image classification over CIFAR10 by Pytorch framework. Three training scheme are used: classical SGD(serve as performance baseline), distributed SGD, and federated learning. Nvidia V100 smx2 GPU in Northeastern Discovery cluster are applied to train the ResNet18. The experiment results compares the performance of the training schemes and show the influence of communication overhead to performance.

5 Acknowledgement

I want to thanks Prof. David Kaeli for his careful instruction to my project proposal and his answers to our question in Piazza.

References

[1] Mohammad Mohammadi Amiri and Deniz Gunduz. Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air. *IEEE Transactions on Signal Processing*, 2020.

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

[3] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

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

[5] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[6] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.