# Decentralized Distributed Machine Learning through Adapted Data Parallelism

Amruth Nare

University of Maryland

Robert H. Smith School of Business

**Problem Statement**

Machine Learning is quickly becoming one of the biggest fields in the world of computer science. However not everybody has access to powerful CPUs to run their complex machine learning algorithms on. The advent of distributed computing proves to be a possible solution that would democratize machine learning, allowing everyone to run algorithms regardless of their computer's specifications. Though the field of distributed computing is quickly growing, there is yet to be a model that accurately and efficiently runs a machine learning algorithm on multiple nodes collaboratively.

**Background**

Currently in order to address the necessity of powerful CPUs for machine learning algorithms, researchers have proposed various distributed computing solutions. In order to consider a protocol or system an adequate solution to distributed machine learning, I designated a few requirements to be satisfied.

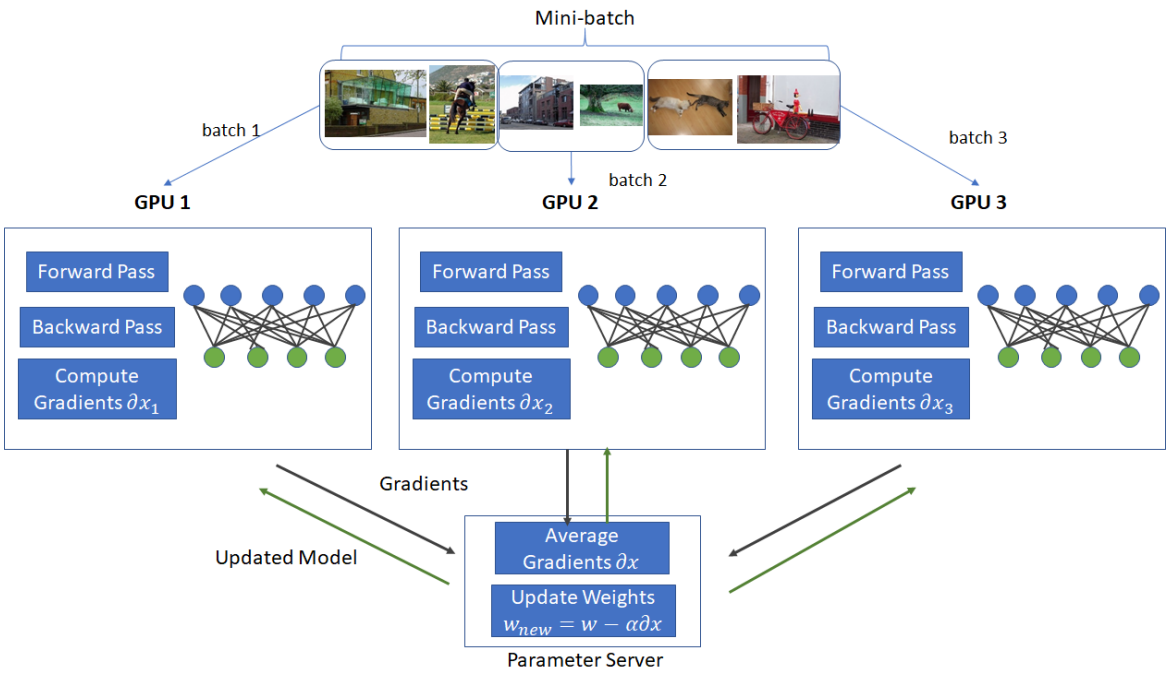
*Requirements*

1) Fault Tolerance
2) Data Privacy
3) High Accuracy
4) Efficiency

Fault tolerance is the ability of the algorithm to work regardless of missing nodes. Research on distributed machine learning proves to have much significance in the future as distributed computation democratizes machine learning. This would allow anyone to be able to create their own machine learning algorithms.

*Data Parallelism & Parameter Servers*

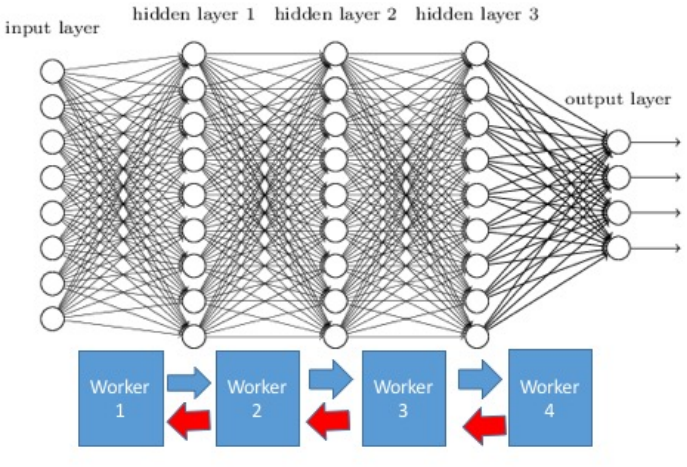
Initially proposed by researchers at Carnegie Mellon University, data parallelism works by splitting the training data for the machine learning model in mini-batches to perform distributed machine learning(Li, 2014). These batches are then distributed among the nodes(external computers), each individual node will calculate the accompanying gradients for their dataset. Gradients are the values in a neural network that optimize the accuracy and minimize loss. These respective gradients are then passed onto an overruling parameter server, which calculates the weighted average of the gradients and passes the updated model back to the worker nodes (Illustrated Below).




With the concept of parameter servers there is the advantage of improved accuracy for the machine learning model as each node is performing the learning individually. However, this specific design fails to pass the requirement of fault tolerance as if one of the worker nodes wasn't present the parameter server would not be able to conduct the gradient calculations. Another inherent problem with parameter servers is that it introduces additional latency as each worker has to pass the data to the parameter server and wait for its response. Also parameter servers are inherently centralized so it is not reasonable on a larger scale involving more worker nodes.

*Model Parallelism*

In model parallelism rather than splitting up the data into batches, the machine learning model itself is split into parts and distributed among different systems. So rather than each system running the model synchronously, model parallelism runs serially as each part of the machine learning model depends on the prior results. With model parallelism a neural network's layers would be split across various different nodes and the last node would have the model's parameters.




Contrary to its name, model parallelism doesn't truly run in parallel but rather in a serial pattern. In a study by researchers at Taiwan University, they constructed a deep neural network that implemented model parallelism(Chen, 2019). However the researchers noticed that the neural network had poor efficiency due to its back propagation feature. Back propagation is the process of propagating the calculated cost of the neural network back through its prior layers to recalculate gradients. Each time the model used back propagation its efficiency sunk, as the layers were spread across various GPUs(nodes) which increased latency. The researchers also noted that model parallelism is not fault tolerant as each GPU is required to run a specific layer so if one was missing the machine learning model would not work. Since model parallelism is inherently inefficient and not fault tolerant it is not a viable method of distributed machine learning.

*Inter-Batch Pipelining*

In order to combat the problems that arise from back propagation, the concept of inter-batch pipelining was developed by researchers from Stanford University(Narayanan, 2019). Inter-batch pipelining is the process of scheduling forward and backward passes of different mini-batches concurrently through different workers. This way no worker is sitting idle after completing the training for their specific minibatch. However this model developed by the researchers was meant for data parallelism and hasn't been developed for model parallelism so it still isn't an option.

**Materials**

The research project required a variety of materials that are essential to the development environment as well as the testing of the program. In order to perform the programming and host a python script, a linux machine from Amazon Web Services(AWS) was implemented through their workspaces product. The language the program was written in was Python 3.7 which was used in conjunction with TensorFlow 2 for the machine learning functionality. Along with TensorFlow the research project used the python package Horovod by uber which is the library responsible for the distributed data parallelism. In order to connect various nodes together OpenMPI an open source library was used to transfer data between nodes. However any implementation of MPI (Message Passage Interface) is acceptable for the program. For the various nodes used in the project itself, DLAMI EC2 instances from AWS were implemented. DLAMI instances are deep learning amazon machine images which are servers which a machine learning environment already set up for development. For the actual machine learning models that were tested on the network, the MNIST database from TensorFlow was used.

**Methods**

To conduct decentralized distributed machine learning various algorithms and methods were used to connect nodes and run the script.

*Setting up MPICluter*

Prior to the start of any development, the nodes had to be set up to run the machine learning models on. All the nodes(servers) that were used throughout the research project were DLAMI EC2 instances from AWS. These instances were initialized through SSH and their private IP's were documented. In order to improve efficiency all instances also downloaded a local version of the MNIST dataset to take the stress of the algorithm itself.

*Horovod Program & Convolutional Neural Network*

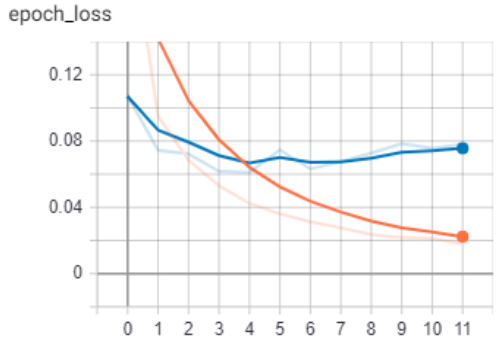
The program to conduct the actual machine learning used the Horovod library to conduct the adapted data parallelism. The initial machine learning model used was a convolutional neural network(CNN) taken from TensorFlow's github repository. Along with raw code required for the ML model, Horovod also requires a hosts file for multi-node GPU training. The private IP's from the EC2 instances were entered into the Horovod hosts files. Some aspects of the CNN were changed in order to standardize results against a single node running a CNN. The CNN's hyperparameters had to be changed as the learning rate of the neural network was not proportional to the amount of data it was processing. For the research project 2 worker nodes were used to conduct the actual machine learning processing so the data would be split into two equal mini batches. To accommodate for the smaller amount of data, the learning rate of the worker nodes was proportionally halved as the learning rate is dependent on the confidence in

each adjustment to the gradients which would be reduced due to less data. This proportionality is outlined in this research paper by Facebook(Goyal, 2018).

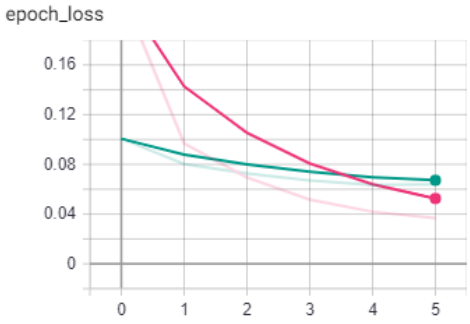
*Adapted Data Parallelism*

The program itself utilized the concept of data parallelism in order to conduct the actual distributed machine learning. Data parallelism is the process of dividing the training data into mini batches and having each worker node run their machine learning model on the smaller set of data. After these models are done running a parameter server will aggregate the gradients and establish a fully trained machine learning model by taking a weighted sum of the gradients. However in this research project, a different form of data parallelism was used that decentralizes the process so that all nodes will have a fully trained model. This adapted data parallelism leverages OpenMPI technology and the Baidu Ring All-Reduce algorithm(Gibiansky, 2018). The OpenMPI library was used to communicate between nodes directly rather than through an overruling parameter server. The ring all-reduce algorithm was implemented by having each node transfer their gradients [node-1] in a ring pattern and performa  reduction operation. By the end each node will have a reduced copy of all the gradients. Then a simple weighted mean is taken by every node in the ring which results in the gradients to the trained machine learning model.

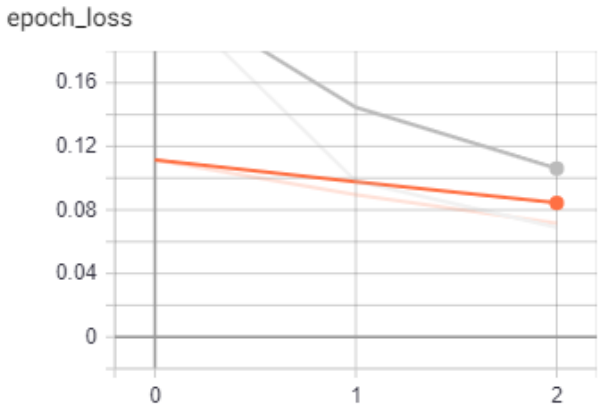
**Results**



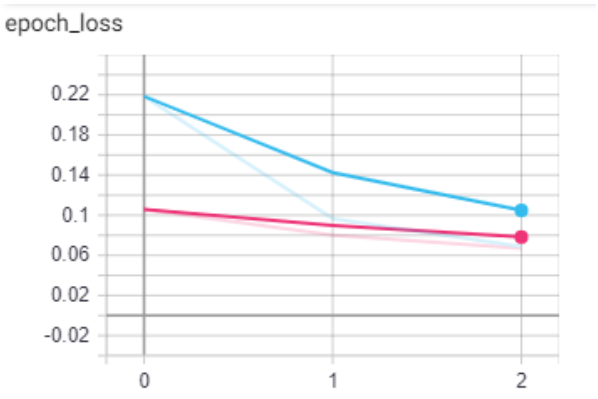

Local CNN trained with 12 epochs




2 Processes on GPU




CNN Trained on 4 Nodes




CNN Trained on 3 Nodes with .003 LR

| Theoretical Cores | Time for 12 epochs |
|---|---|
| 2 | 482 |
| 2 | 473 |
| 6 | 125 |
| 6 | 131 |
| 8 | 67 |
| 8 | 62 |

**Discussion**

Throughout the testing of the distributed learning protocol, I used the MNIST database provided by the Keras framework. The MNIST database contains images of the numbers 1-9 so the neural network can be trained to identify handwritten digits. For the majority of the testing a 4 layered deep neural network that implemented the HVD-Adam optimizer was used. The testing was done for 3 different iterations of the protocol: 2 local processes on GPU, 4 Multi-Node, and 3 Multi-Node. For each of the different tests I adjusted the learning rate based on the amount of processes as more processes results in smaller batches of test data.

Each of the tests has their loss functions graphed over the training epochs and were used to verify whether there was any difference in accuracy between single and multi-node neural network training. While the neural network run on the 4 node network had the lowest loss, it wasn't significant enough to warrant a difference from the local training. However there was a

clear difference in time to compute when using the multi-node network. Since the multiple node network has multiple GPU's that are each processing mini-batches of data they essentially had more cores. The data reflected this claim because as the number of theoretical cores for processing increased the time to complete training decreased. The improvement in efficiency was approximately 677% between the local node and the four node process however there was only a 400% increase in the computing power. This can be explained by the Horovod technologies framework that uses OpenMPI to interact between nodes. Since OpenMPI only distributed minibatches of the dataset each respective node is performing less computation than the local network so it will be more efficient.

**Citations**


Gibiansky, A. (2018). Andrew Gibiansky :: Math → [Code]. Retrieved November 07, 2020, from https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., . . . He, K. (2018, April 30). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. Retrieved November 07, 2020, from https://arxiv.org/abs/1706.02677

Weng, J., Weng, J., Zhang, J., Li, M., Zhang, Y., & Luo, W. (2019). DeepChain: Auditable and Privacy-Preserving Deep Learning with Blockchain-based Incentive. *IEEE Transactions on Dependable and Secure Computing,* 1-1. doi:10.1109/tdsc.2019.2952332

Chen, Xuhui & Ji, Jinlong & Luo, Changqing & Liao, Weixian & Li, Pan. (2018). -1178-1187. 10.1109/BigData.2018.8622598.

Garcia, E. (2019, August 10). Visual intuition on ring-Allreduce for distributed Deep Learning. Retrieved June 16, 2020, from https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da

Sergeev, A. (2019, April 08). Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow. Retrieved June 16, 2020, from https://eng.uber.com/horovod/

Sundaram, Narayanan, et al. "Streaming Similarity Search over One Billion Tweets Using Parallel Locality-Sensitive Hashing." Proceedings of the VLDB Endowment, vol. 6, no. 14, 2013, pp. 1930–1941., doi:10.14778/2556549.2556574.

Sgantzos, K., & Grigg, I.. (n.d.). Artificial Intelligence Implementations on the, fi11080170.

McConaghy, T., & al, E. (n.d.). "How Blockchains Could Transform Artificial Intelligence." Dataconomy, 21 Dec. Retrieved May 3, 2020,

Ibanez, T. (n.d.). “LearningChain.” Devpost, devpost.com/software/learningchain. Retrieved May 3, 2020,

Baldominos, Alejandro, and Yago Saez. “Coin.AI: A Proof-of-Useful-Work Scheme for Blockchain-Based Distributed Deep Learning.” Entropy, vol. 21, no. 8, 2019, p. 723., doi:10.3390/e21080723.

Ibanez, Thomas. “LearningChain.” Devpost, devpost.com/software/learningchain.

McConaghy, Trent, et al. “How Blockchains Could Transform Artificial Intelligence.” Dataconomy, 21 Dec. 2016, dataconomy.com/2016/12/blockchains-for-artificial-intelligence/.

Sgantzos, Konstantinos, and Ian Grigg. “Artificial Intelligence Implementations on the Blockchain. Use Cases and Future Applications.” Future Internet, vol. 11, no. 8, 2019, p. 170., doi:10.3390/fi11080170.

Engdahl, S. (2008). Blogs. Retrieved June 16, 2020, from https://aws.amazon.com/blogs/machine-learning/launching-tensorflow-distributed-training-easily-with-horovod-or-parameter-servers-in-amazon-sagemaker/

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N., Ganger, G., . . . University of Calgary. (2019, October 01). PipeDream: Generalized pipeline parallelism for DNN training. Retrieved June 16, 2020, from https://dl.acm.org/doi/10.1145/3341301.3359646

Li, M., Andersen, D., Smola, A., Google, A., Ahmed, A., Google, V., . . . University of Michigan,University of Washington. (2014, October 01). Scaling distributed machine learning with the parameter server. Retrieved June 15, 2020, from https://dl.acm.org/doi/10.5555/2685048.2685095

Chen, Chi-Chung, Yang, Chia-Lin, Cheng, & Hsiang-Yun. (2019, October 28). Efficient and Robust Parallel DNN Training through Model Parallelism on Multi-GPU Platform. Retrieved June 15, 2020, from https://arxiv.org/abs/1809.02839