Layer-wise Asynchronous Training of Neural Network with Synthetic Gradient on Distributed System

Xupeng Tong
Carnegie Mellon University
Computational Biology Department
xtong@andrew.cmu.edu

Hao Wang
Carnegie Mellon University
Language Technologies Institute
haow2@andrew.cmu.edu

ABSTRACT

In this proposal, we describe a potential design a distributed training of convolutional neural network by synthetic gradient.

Keywords

Synthetic gradient, asynchronous, neural network

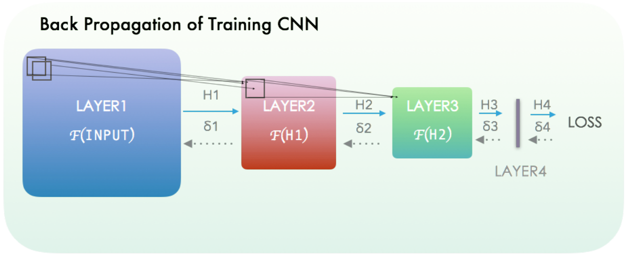
1. INTRODUCTION

Parallelization is always an intriguing idea for researchers working with exceptional large datasets or complex models in machine learning community. Recently, the emergence of affordable cloud services and multi-core machines has made it feasible more than ever.

Paralleling the training of neural network is always a challenge as each layer takes input from its previous layer, and has to wait for the gradient passed from its next layer. An intuitive way of doing parallelization in such structure is to use Asynchronous Stochastic Gradient Descent, which is extensively discussed by the paper published by Google at 2012 [1], that made a compromise on the mathematical effectiveness and the scalability of distributed system, by paralleling the model update asynchronously with independent workers managed by a parameter server. This technique has been extensively used and significant improvement has been made on both the parameter server [2,3] and the algorithm itself [4] later on and becomes a industrial standard for large scale training of neural networks.

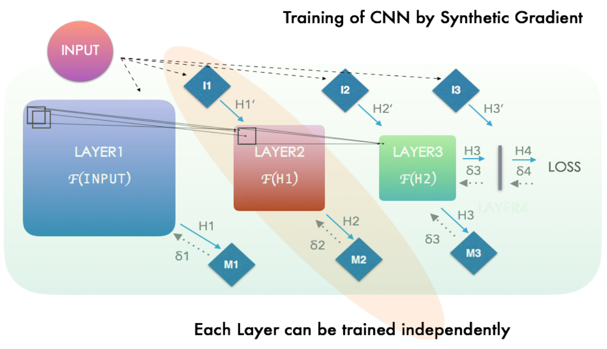
Though the exceptional popularity that Asynchronous Stochastic Gradient Descent has gained in the recent years, question exists as it parallels data over the model itself, where at here, is composed of end to end layers. Are we able to parallel the training of each layers instead of each data block? It is made possible by the paper published this year by Google Deepmind [5].

Normally the training of neural network requires back propagation to update parameters from each layer of the neural network, through which the gradient calculated accumulatively, percolate down from the output layer to the input layer and, sequentially update the weights and biases in each layer. Repetitive calculation of the gradient is greatly reduced through this is way because the loss function is the composite of many sub-functions. This method, though established as the standard training pipeline, suffers from the fact that the the parameter update in each layer is dependent on its adjacent layers. The proposed method by [5] uses two types



of additional small neural networks, that simulate the input and gradient update whenever a layer makes a request.

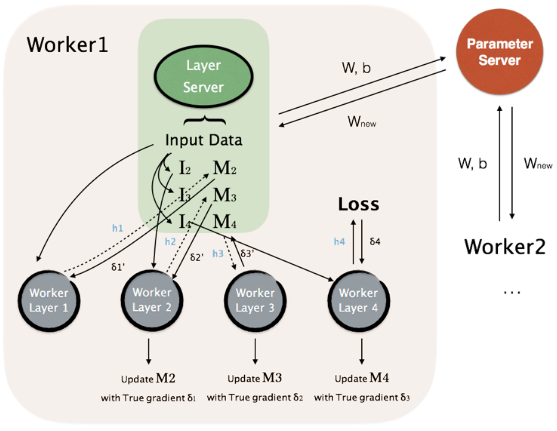
The advantage of this lock-free version of training of neural network could be applied in tasks that involves multiple small neural networks that communicate with each other. However, this is not the focus on our project. In this project, we tend to discover and implement a more robust model that utilizes this elegant idea of layer wise training. The complete decoupled model [5] has achieved proven effectiveness on simple dataset like MNIST, and now we want to extend its application in larger dataset with slightly more complicated models by changing training model dynamically on distributed system.



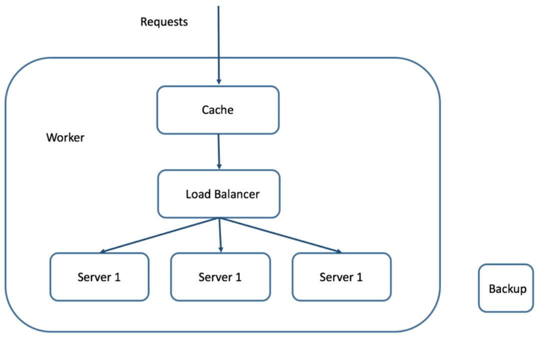
By this time, we haven't seen any distributed version of this model and we tend to discover the effectiveness of this model by doing extensive experiments from naive to complex models with our knowledge in machine learning and distributed system. Trivial but important works should be done focusing on decreasing the variance of input and gradient update simulated by auxiliary neural networks. Also, how to scale the data by improving the data storage and cache system is also things that awaits us to tackle with.

2. IDEA

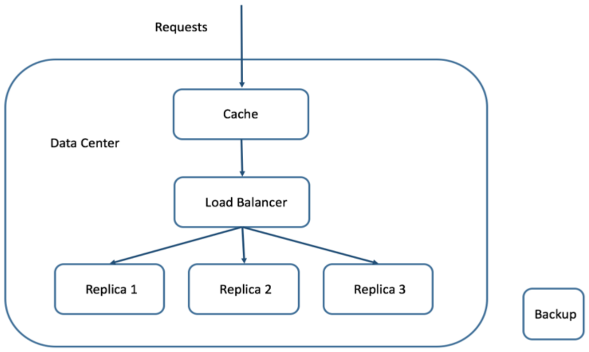
We tend to train the synthetic gradient model in a distributed way by using parameter server. The big picture of our model might look like the picture described below.



3. IMPLEMENTATIONS



As this is a brand new idea for the whole data pipeline, there are many factors that needed to be taken into consideration. As the amount of data is very huge, the storage of them might still be a problem. We might use a NoSQL database to deal with the storage problem. For the inner implementation of each single worker, we plan on using TensorFlow [6] for gradient calculation and weight update. Further, due to the large amount of data, more machines might be required for each worker. They should follow the master slave schema, and the auto-scaling setup for each machine should also be considered. Besides, for each worker, the unlabeled data of the input is will be repeatedly used across iterations, so a front-end cache might be able to speed it up. The



communication among different servers might need the timeout mechanism as the data might be outdated for too much waiting time, or be left behind due to blocking. To solve this problem, we might need an additional machine or databases like Memcached or Redis to decrease the burden for each worker. From the side of the data center, backup mechanisms is required, which will involve replication and a load balancer as well.

Finally, to reduce the variance problem among layers, the network would not be necessarily trained with independent layers, and we plan to group several layers into one and train that group as one. Moreover, we still manage to change the splitting of the network dynamically thus the variance might be further reduced.

4. DATASET

We tend to test out our framework with datasets varies in their sizes from MNIST to Amazon review dataset [7] to do classification task with either fully connected neural network and convolutional neural network.

5. EXPECTATION

As this is a new algorithm and no distributed version of this has been released, we might not expect the accuracy of this framework to be as high as some state of art models, however, we expect it to achieve a better convergence rate than asynchronous stochastic gradient descent alone, though with potential loss of accuracy due to the instability of the calculation among layers. We also want to discover what modification on the model may help improving the system in either accuracy and convergence time, this might be extremely useful for big data environment as we always want to make a balance between the precision and the efficiency.

6. REFERENCES

[1] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

[2] Ho, Qirong, et al. "More effective distributed ml via a stale synchronous parallel parameter server." *Advances in neural information processing systems*. 2013.

[3] Li, Mu, et al. "Scaling distributed machine learning with the parameter server." *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 2014.

[4] Zhang, Ruiliang, Shuai Zheng, and James T. Kwok. "Fast distributed asynchronous sgd with variance reduction." *arXiv preprint arXiv:1508.01633*(2015).

[5] Jaderberg, Max, et al. "Decoupled neural interfaces using synthetic gradients." *arXiv preprint arXiv:1608.05343* (2016).

[6] Abadi, Martın, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." *arXiv preprint arXiv:1603.04467* (2016).

[7] http://snap.stanford.edu/data/amazon/productGraph