Data Structure and Algorithms
16:332:573

Term Paper: Parallelization of Stochastic Gradient Descent

Xin Yang

April 3, 2018

Electrical and Computer Engineering Department
Rutgers University, Piscataway, NJ 08854

1 Introduction

The rapid development of machine learning has pressed an urgent demand for a faster approach to computation. Working as one of the most widely used optimizers [1], the performance of gradient descent algorithms plays a decisive role. With the boost of the training data magnitude, the traditional gradient descent has become a bottleneck for its serial procedure, thus the idea of effective parallel approaches came up. However, as the core of gradient descent is a serial iterative algorithm, how to distribute the work and communicate asynchronously between clusters are the main challenges. This article will mainly talk about several successful parallel stochastic gradient descent algorithms.

2 Gradient Descent

2.1 Batch Gradient Descent

The gradient descent is defined as a first-order iterative optimization algorithm, which can help finding the minimum of a given function in lots of unconstrained optimization problems [2].

The gradient is the derivative of parameters for the multi-variable function in vector format. A direct geometrical explanation of gradient is it reflects the changing rate of the function and is exactly pointing to the fastest changing direction. In machine learning, it's much easier to find the minimum of a function(usually the cost function) by following the opposite side of the gradient, which is like going down a mountain by taking the most steer path at each step.

In a mathematical way, we assume our target is a basic linear regression:

$$h_\theta(x_1, x_2, ...x_n) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$$

where $\theta_i (i = 0, 1, 2, ...n)$ are the parameters of our model, $x_i (i = 0, 1, 2, ...n)$ are the features. By adding a feature $x_0 = 1$, we can simplify it to be:

$$h_\theta(x_0, x_1, ...x_n) = \sum_{j=0}^{n} \theta_i x_i$$

Assuming our cost function to be square loss, the cost function is:

$$J(\theta_0, \theta_1, ...\theta_n) = \frac{1}{2m} \sum_{j=0}^{m} (h_\theta(x_0^{(j)}, x_1^{(j)}, ...x_n^{(j)}) - y^j)^2$$

The gradient for a certain position is $\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1 ..., \theta_n)$. The distance of each step can be calculated by using step length a multiply the gradient. Thus the iterative formula is $\theta_i = \theta_i - a \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1 ..., \theta_n)$. By substituting the $J(\theta)$, we can get the final iteration of gradient descent for a linear regression problem:

$$\theta_i = \theta_i - a \frac{1}{m} \sum_{j=0}^{m} (h_\theta(x_0^{(j)}, x_1^{(j)}, ...x_n^{(j)}) x_i^{(j)}$$

The above formula shows the current direction of the gradient is decided by all the samples, which means although the participation of all data can speed up the converging, the iteration can be extremely slow when the datasets are huge. This method is the batch gradient descent(BGD).

The main problems for BGD are that under some circumstances, the batch gradient descent would trap into local minimum convergence(saddle point) rather than the global minimum. The performance will decrease when datasets get larger as well.

2.2 Stochastic Gradient Descent

To overcome the problems of BGD, the stochastic gradient descent(SGD) is developed. Stochastic gradient descent will randomly choose a sample j to calculate the gradient, which makes the training process significantly faster, but it sacrifices the accuracy of convergence. The gradient updating method for SGD is:

$$\theta_i = \theta_i - a(h_\theta(x_0^{(j)}, x_1^{(j)}, ...x_n^{(j)}) - y_j)x_i^{(j)}$$

Although the speed of SGD is faster, the problems are obvious: SGD has to tolerate more noise owing to for each step, it might not towards the global optimization, meanwhile the iteration times is more than BGD.

2.3 Mini-batch Gradient Descent

Mini-batch gradient descent is the compromising solution, which will pick a batch of data to iterate, having the convergence speed faster than BGD and accuracy better than SGD, the parallelized stochastic gradient descent are similar to the concept mini-batch gradient descent in some ways because they both partition the datasets and train only one part of them each time.

2.4 Challenges

As mentioned earlier, the increasing magnitude of datasets can slow down the performance of batch gradient descent, as well as the stochastic gradient descent although it's faster than BGD by definition.

The process of stochastic gradient descent is highly serial, i.e., the update of next step depends on the current result, which makes the parallelization difficult. Also, one main problem for many parallelization problems is how to lower the communication cost between distributed nodes. By specific is how to asynchronously exchanging the data between the master nodes and worker nodes in the cluster.

3 Parallel Stochastic Gradient Descent

The growth of the data size makes the gradient descent under the pressure of big data. Distributed systems have significant superiority over serial computation systems. Also with the development of parallel computing, we can easily achieve impressive computing capabilities utilizing multi-core CPU and GPU as well as distributed computing frameworks such as Apache Hadoop, Apache Spark, and MPI with multi-core machines or multi-machine clusters. If we can find an effective way to transplant the gradient descent into parallel, the improvement would work for lots of areas from scientific research to business engineering.

3.1 Parallelized Stochastic Gradient Descent for MapReduce

MapReduce is a concept inspired by functional languages. The main idea is to map a huge task to small ones and distribute to clusters, then compute separately on each worker node, finally aggregate the results from clusters and process to be the result. One successful application is Apache Hadoop developed by Yahoo, inspired by the paper of Google [3].

One way of parallelizing the stochastic gradient descent is developed by Yahoo! Labs [4]. The main idea is to uniformly separate the workload D, and send each part D_i to the i_{th} machine. Before the jobs start, all learning rates η will be set to the same.

For each worker node in the cluster, it will shuffle the given dataset to be $\{\hat{x_j}, \hat{y_j}\}_{j=1}^{T}$, and initialize the parameters as a vector w_0, and run T steps of stochastic gradient descent on each node. Assuming that the cost function is $l(x_i, y_i, w)$ for a single training set. The iterative update function for step j is:

$$w_j^{(i)} = w_{j-1}^{(i)} - \eta \frac{\partial l(x_i, y_i, w)}{\partial w}\bigg|_{(\hat{x_j}, \hat{y_j}, w_{j-1}^{(i)})}$$

After the computing of each node, it will come to the aggregate stage, where will send each machine the mean weight vector $v = \frac{1}{c}\sum_{i=1}^{c} w_T^{(i)}$. If the aggregated result is qualified for convergence, the algorithm will end, or it will recalculate the weights based on the current results.

This algorithm is the combination of a simple average method [5] and the asymptotic analysis. This strategy only requires one MapReduce pass, which cut down the cost of communication because it only happens when computation finished. Meanwhile, this algorithm will converge anyway regardless of the number of samples. The only factors that would affect the performance are the expected error and regularization parameter since the regularization will go down if the dataset is big, and the convergence is not guaranteed.

However, this parallelization is based on data parallelization without model parallelization, i.e., only the training data are distributed and computed separately, the parameters are still stored on all worker nodes. So this algorithm only fits for models with fewer parameters with massive training sets.

3.2 Downpour Stochastic Gradient Descent

Despite the data parallelized approach proposed by Yahoo, Google raised another algorithm which supports both data parallelization and model parallelization. This algorithm is named as Downpour Stochastic Gradient Descent(Downpour SGD), which is a component of the DistBelief framework, the predecessor of TensorFlow [6].

The traditional parallelizations are based on the MapReduce, which mainly focus on computing small datasets on a single node, or GraphLab, which can't make full advantage of structured graphs. Downpour SGD can train large-scale deep neural network models with the achieved balance between model parallelization and data parallelization.

The DistBelief framework allows users to partition the neural network models, and assign partitions to different workers. In this approach, the improvement will be restricted by the connections between neurons, e.g., partially connected neural networks are considered to have better communication performance than fully connected neural networks.

With the help of graph partitioning, DistBelief parallelized the computation on another dimension: multiple copies of DistBelief can run at the same time, and the synchronization of parameters is on a centralized sharded parameter server.

The original training set will be partitioned, each node in the cluster will take one part of the training set and request a copy of the model w from the server, once finished computing, the worker will send the result Δw to the server, and the server will do the update of gradient in the following way:

$$w' = w - \eta \Delta w$$

It's notable that the centralized server can be made up of multiple machines and each server will equally store one independent part of parameters. In this way, both workers and master servers are parallelized asynchronously. Then there comes the problem of communication costs. In Downpour SGD, two extra parameters n_{fetch} and n_{push} are imported to control how often should the workers connect to the server. The n_{fetch} defines after how many steps should the worker request for the latest parameters, and the n_{push} defines after how many batches should the worker send the gradient to the parameter server.

Comparing with the parallelized SGD proposed by Yahoo, the Downpour SGD is much more robust as it keeps the final parameters on the centralized sharded parameter server, and all workers will have a copy of the full training model, this can avoid risks of node failures.

Problems for the Downpour SGD are brought by the asynchronism. The copies owned by worker nodes may not be the latest because other workers might be pushing their gradients during computation. Also since the parameter servers are independent, the order and rounds of update are not fixed, so the convergence efficiency could be slower. But with the improvement of speed, this approach is still faster than the parallelized stochastic gradient descent based on MapReduce or GraphLab [6].

4 Applications of Parallelized Stochastic Gradient Descent

Currently, the gradient descent is commonly implemented in machine learning areas, especially in the deep neural networks. From image classification to voice recognition, even the autonomous driving cars and the famous Google AlphaGo are utilizing it. The properties of deep neural networks determine that a good neural network must be trained by large amounts of data [7]. Thanks to the data mining, we can now access the data easily, but the performance of a single machine can be incapable in face of serial large-scale data processing. A more efficient way is to distribute a single big job to a cluster to scale down the workload of a single machine. This is why we need to parallelize the stochastic gradient descent.

One successful example of exploiting parallelized stochastic gradient descent is the Google TensorFlow, which is developed based on the DistBelief framework. TensorFlow can make full advantage of the properties of graphics processing units to achieve high-performance parallel computing and can speed up to hundred times faster than running relatively serially using CPU [8].

For leader companies like Google or Amazon, their products also need to train large-scale deep neural networks, their approaches are launching clusters consisting of thousands of servers. The efficiency of parallelized stochastic gradient descent no doubt has a priceless impact on these companies.

In scientific areas like medication or biological engineering, the explorations of nature now depend on the simulation on computers, e.g., simulating the folding of proteins to find the secrets hidden in our genes, which also requires the participation of neural networks. The improvement of parallelized gradient descent will speed up the evolution of human civilization in a macroscopic view.

5 Conclusion

There are now several successful strategies can successfully turn the serial iterative stochastic gradient descent algorithm into a parallelized way and been tested can benefit from the increasing numbers of machines [6] [4]. But problems still exist as the ideal parallelization must be achieved by perfectly leverage the data parallelization and the model parallelization. The invention of Downpour SDG proposed by Google is already an excellent discovery, which has combined the merits of lots of previous works. For future work, the main point is to solve the problems came with asynchronism, and meanwhile make sure the convergence will not be slowed down. The problems of communication delays are also worth taking care of in large-scale clusters.

References

[1] Y. LeCun, L. Bottou, G. Orr, and K. Muller, “Efficient backprop,” *Neural Networks: Tricks of the trade*, 1998.

[2] Wikipedia, “Gradient descent,” 2018, https://en.wikipedia.org/wiki/Gradient_descent.

[3] Jeffrey Dean, Sanjay Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *OSDI*, 2004.

[4] Martin A. Zinkevich, Markus Weimer, Alex Smola, Lihong Li, “Parallelized stochastic gradient descent,” *Advances in Neural Information Processing Systems year = 2011.*

[5] G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker, “Efficient large-scale dis- tributed training of conditional maximum entropy models,” *IEEE Transactions on Neural Networks.*

[6] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc’Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng, “Large scale distributed deep networks,” *NIPS*, 2012.

[7] Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A.Y. Ng., “On optimization methods for deep learning,” *ICML*, 2011.

[8] R. Raina, A. Madhavan, and A. Y. Ng, “Large-scale deep unsupervised learning using graphics processors,” *ICML*, 2009.