GAMES AND AI

Incentivizing Workers in a Federated Learning Model By Stackelberg Game Formulation

Kaan Güney KEKLİKÇİ - kaanguney@sabanciuniv.edu
Computer Science & Economics / FENS & FASS

Yavuz GÜLEŞEN -
ygulesen@sabanciuniv.edu
Computer Science / FENS

Yiğit SEVİM - yigitsevim@sabanciuniv.edu
Industrial Engineering / FENS

Oğuzhan TAVŞAN - oguzhantavsan@sabanciuniv.edu
Computer Science / FENS

Yunus SARIKAYA
Electrical & Electronics Engineering

Özgür ERÇETİN
Electrical & Electronics Engineering

Abstract

In this paper, we centralize a federated learning approach and evaluate on a two-stage model where we measure response times from the devices in the model and perform weight optimization for the model in each iteration. The starting node(owner), optimizes the model in such a way that workers in the model utilize their allocated CPU power in the most efficient way and at optimum delay(response) time. Furthermore, the model forces the workers to work at optimum efficiency per second by introducing a target error rate. In the end, our goal is to optimize the trade-off between the most efficient CPU-cycle usage and training delay.

1 Introduction

Recently, systems that are based on a deep neural network prediction model have started to draw significant amount of attention. Replacing the primitive versions of artificial networks, these models are consisted of hundreds of layers. The determination behind the idea is to increase the number of parameters in the model such that in each iteration(run) performed in the model; the number of parameters updated is very high. Therefore, in the end, the model has a higher success rate at predicting the output. However, using models with extended depths are costly; even with the best hardware accelerators; training may require unreasonable amount of time before even starting to test/validate the model on smaller samples. Additionally; since there are thousands of parameters in the model, memory requirement is also too high. However, there is a widely used solution for memory and time requirements which is allocating parallel computational power. Using this technique, input data is divided into subsets and is fed into the same network system but in different computers with advanced hardware accelerators. In other words, the network allocates the data using the same model but waits for the response times of all devices running on the model separately, which brings us to the maintenance of the network.

The model works in sequential manner, every input vector is passed through every layer of the network one by one. Then, the network calculates the sum of squared errors. From now on, after the first iteration, the network's determination is to minimize that error rate. In order to satisfy, for that goal in each iteration, the model sends back its parameters to the owner for the owner to optimize them. To perform this operation, the model uses a fully synchronous stochastic gradient descent (SGD)[1] as the optimizer. SDG moves towards the target rate in almost random directions and tries to obtain a minimum target error. Even though SGD optimization delivers promising results for federated learning, since we are dependent in each separate device to get a response in order to update the parameters, the device with the highest delay determines the performance of the neural network. This is, in fact, called the straggler effect.[2]

There are many models with optimistic assumptions about all devices in the model contributing to the model unconditionally. However, in real world applications, this assumption does not hold. Assuming all devices in the model contribute their allocations to the model as needed as possible is wrong due to high resource expenditures, making these models impractical. In our approach, we introduced an incentive mechanism[3] where a node (device) sends its parameters to the model if and only if a reasonable result above the threshold is obtained. Since this project's ambition is to optimize CPU allocation for each device by parameter update in the network's structure, the incentive level we set while experimenting with delay measurement is a target accuracy level which the model tries to reach.

The experimentation was done by inducing that incentive mechanism to the model at each gradient update step, aiming for the local minima for delay at different target errors under the budget constraint of the owner and CPU power available. In order to perform optimally, we needed to use our resources as much as we can such as performing maximum number of gradient updates at each minibatch with minimum amount of delay possible at that target error level. For these conditions to satisfy, we applied a game theoretical approach to the artificial neural network by injecting Stackelberg game modeling to our model. First, we define the lower sub-level game where we calculate for maximum CPU power that may be applicable to the model from a utility function. Then, we move on to the upper sub-level game where the objective is to minimize the cost function of the owner with respect to minimum cost per CPU power. The game is going to be considered Pareto efficient when the owner cost function and all weights in the model are optimized thanks to optimum number of mini-batch updates for the optimum number of K workers.

2 FEDERATED MODEL

The model starts with the model owner forward passing the inputs to the network. At each iteration, inputs, which are essentially data vectors, pass all the hidden layers and reach to the end of the network where a loss function value is calculated. Then for every i^{th} worker, weight optimization is done for all possible weights in the neural network by back propagation. Each weight is represented by w_i such that each w_i represents a worker's CPU optimization in a sense. At each gradient update, this method is repeatedly done. However, remark that the optimal solution may not be the case where gradient update is applied to all weights running in the network. There may be some unnecessary workers using unnecessary amount of resources and causing unnecessary amount of delay. To sum up, optimization relies on gradient updates but is never fully done accurately under non-optimal minibatch size.

[1] **Sarıkaya and Erçetin, 2019**
[2] **Sarıkaya and Erçetin, 2019**

[3] **Sarıkaya and Erçetin, 2019**

Let T_i, t be the time elapsed for the worker i to update the gradient in iteration t. Here,

we consider plain *fully* synchronous SGD such that the model owner waits for all the workers to push their gradients. Thus, iteration t is completed in maxi Ti,t time, i.e., when all workers send their gradient updates. Each worker i allocates a CPU power, i.e., P_i^t to complete minibatch in iteration t. Let the total number of CPU cycles required to complete the computation of gradient of minibatch in iteration t be c_i^t .We assume that c_i^t is random and independently distributed across P_i mini-batches and workers. Hence, Ti,t has a mean $P_{i\,/\,c_{i\,(sample)}}$, where $c_{i\,(sample)}$ is the mean of c_i^t which may follow any arbitrary distribution.**4** Remark that, the model owner is the node to announce a target delay under a target error rate. Furthermore, the incentive (yardstick) is defined to be the difference between the $T_{desired}$ and T_{actual}. This delay difference is calculated at each stochastic step and the workers are rewarded with respect to a function such that:

$$q_i^t = Q_0 - \alpha\, d_i^t,$$

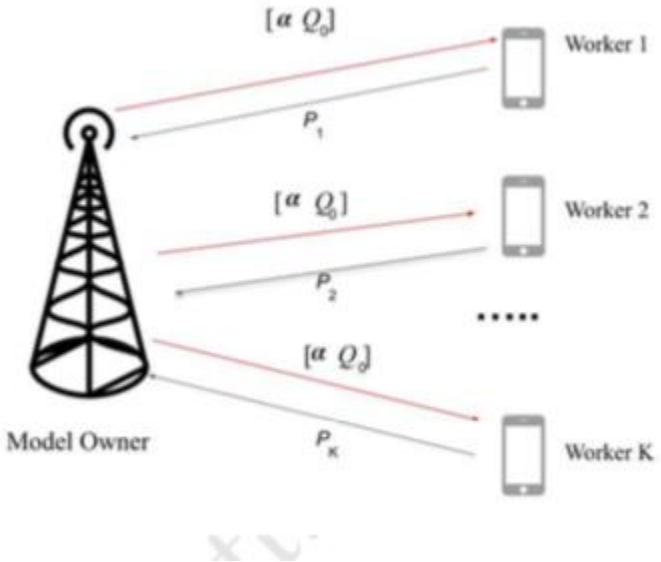

Figure 1. Federate Model with SGD weight optimization

[4] **Sarıkaya and Erçetin, 2019**

where α and Q_0 are design parameters. A node improves on its contribution to the model by α if the worker exceeds the yardstick. From that point, it's also clear that delay is dependent on time, hence iteration. Therefore, the final form of the equation is now:

$$Q_i = Q_0 - \alpha\, d_i$$

Up until now, we have evaluated on the yardstick we proposed for the workers in the model. Next, worker optimization was evaluated and tested for the model. A utility function was introduced for the workers in the network to maximize their CPU allocations in such a way that the utility function returns the max. utility for the worker with respect to the max CPU-cycle usage at a local max for P_i. The utility function introduced was:[5]

$$U_i(P_i) = q_i P_i - Kc_i(P_i)2.$$

With max. P_i derivation, this results in $P_{max} = q_{i\,/\,(2KCi).}$

Now, a worker has calculated for the optimum CPU-cycle allocation for his/her work. At last, the worker's determination to achieve this goal is completed. In every iteration, the worker will send back the parameters of his work back to the model owner and the model owner is going to focus on his/her budget optimization by taking the derivative of his cost function with respect to minimal cost per CPU-cycle. The function is provided as such:

$$\Delta(qi,Pi) = VE\ maxTi,t + \sum qiPi^{6}$$

Optimization part of the project is yet to be evaluated as a whole and there have not been any tests in that field. Below are some experimental runs for the neural network for different yardstick approaches and delay measurements:

```
Hiddenlayer1: 5

Hiddenlayer2: 5

Stopping Criteria: 0.2

Average CPU Delay: 0.5042739143500018

Average iteration count: 160
```

Figure 2. (Ex.1) Sequential Neural Network Run

[5] **Sarıkaya and Erçetin, 2019**

[6] **Sarıkaya and Erçetin, 2019**

```
Hiddenlayer1: 5

Hiddenlayer2: 10

Stopping Criteria: 0.2

Average CPU Delay: 0.5460267177300006

Average iteration count: 316
```

Figure 3. (Ex.2) Sequential Neural Network Run Analysis

3 CONCLUSION

With increasing memory requirements to mine training data, deep neural schemes have become the new trendline for training approaches such as federated learning. In this paper, we focused on a yardstick model where we used stochastic gradient descent to perform for parameter optimization (i.e weight & CPU-cycle allocation).

In our tests, we determined different batch sizes and optimized the weights in this approach. We calculated for an approximate iteration level of 1000-1500 to qualify for a reward under a specific yardstick. Although experimentations were done in MNIST, we are planning to use a larger dataset to get better performance metrics and analyze the model better in the future. To be more specific, during experimentation, two hidden layers were used since we were restricted to limited computational power and worked on a relatively small dataset. The pushback updates that we collected from SGD were well optimized and qualified our project for 90% in some extreme cases. Naturally, the model needs significantly more parameter adjustment, however, results were quite promising. Finally, only the process times were measured with Colaboratory GPU for the project in the tests and our goal towards the future includes timing by CPU.

References

- Sarıkaya, Y., & Erçetin, Ö. (2019, August 6). Motivating Workers in Federated Learning: A Stackelberg Game Perspective.