CS 5287: Final Report

Jingtao Cheng
Vanderbilt University
jingtao.cheng@vanderbilt.edu

Shizhen Li
Vanderbilt University
shizhen.li@vanderbilt.edu

Aniruddha Gokhale
Vanderbilt University
a.gokhale@vanderbilt.edu

Abstract

Federated learning is a learning strategy to collaborate machine learning to train a centralized model with decentralized training data. The mechanism for this learning algorithm requires each client (device) to compute an update for the shared prediction model based on their local data and then send this update to the central server on each epoch. The collected update will be aggregated to update the global model to keep it up to date. Some common edge devices that can perform federated learning are mobile phones, IoT devices, and other devices that support efficient communications. In this report, we will first introduce the notion of edge computing, distributed computing, and explain the state of art solution of federated learning and its challenges.

Keywords: federated learning, machine learning, distributed learning, cloud computing, Edge computing, machine intelligence.

Introduction

Our motivation for this topic is simple, Jingtao has been working in the Machine Learning/Deep Learning field for quite some time, and Shizhen is interested in Cloud Computing and Security in general. Thus picking Federated Learning as our topic will benefit both of us. The keyword involved such as Machine Learning, Data Privacy, Edge Computing, Decentralized Machine Learning. The most common use cases involve self-driving cars, smart manufacturing, digital health.

Background

This section will mainly focus on the evolution of cloud computing from 2000 to 2017. The first part of the evolution is from 2000 to 2014. And with the continuously evolving, the notion of Edge computing started to emerge. So in the rest of this section, we will include an overview of Edge Computing and why this technology is becoming the center of today's technology.

From 2000 to 2014, the most predominant way of using Cloud services is to process data. A massive amount of data got pushed into the Cloud center, and it either got stored in the Cloud Storage or went into the processing cycle right away. We can either consume the result or use certain services to visualize the result. For example, our programming assignment for CS5287 uses this type of pipeline. We choose to use Chameleon for the cloud platform, a free cloud service provided for research purposes. In addition, we use Apache Kafka to stream data into the Cloud and deploy multiple brokers in the Cloud to read the data. Finally, after we finish processing the data, we have deployed Apache Couchdb in the Cloud to visualize our result, accessed using an URL, as shown in Figure 1.

The combination of Kafka and Cloud is like bread and butter. But it also has complications, something we are taking for granted in this pipeline: the Internet. In our case, the amount of data we are producing at once is at most 10GB. It is seasonable to move that 10 GB of data into the Cloud through the Internet. But what if the data is large enough which we cannot simply transfer it through the Internet? For example, an airplane flying internationally generates 5TB of data per day. A smart

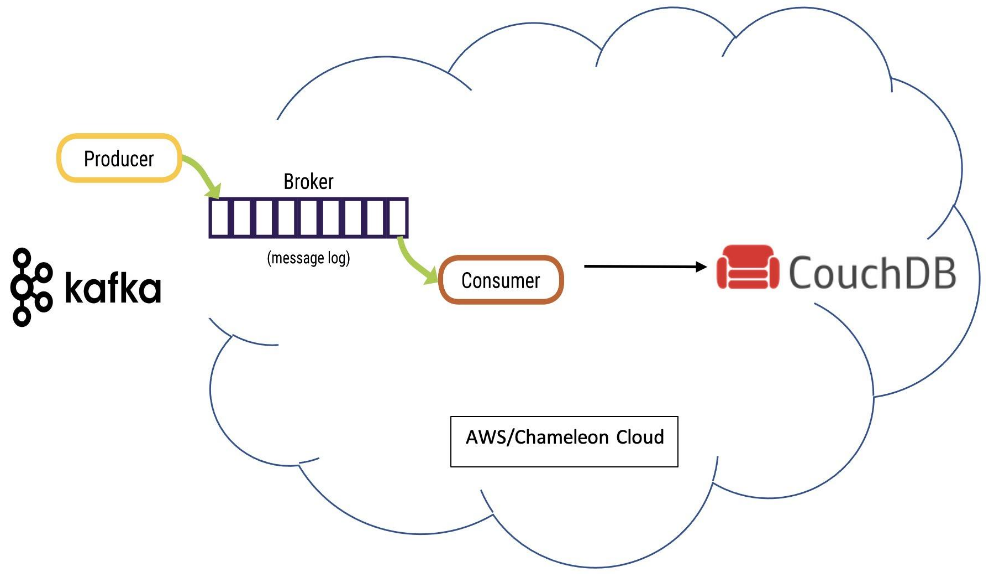


Figure 1. Cloud architecture for course assignments

the hospital produces 3TB of data. Each self-driving car generates 4TB of data every day, and a connected factory could produce 3PB of data every day. It is impractical to transfer all of those data into the Cloud through the Internet. No matter how fast your Internet is. Therefore, using Cloud to stream and process the data on this scale seems cumbersome. As a result, a new variant of Cloud Computing started to emerge.

Around 2014, Docker and Kubernetes became popular. As a result, more devices and systems are running in Containerized environment. The lightweight nature of Containerized applications is being deployed into billions of devices. Those devices can be either offline or online, but since everything requires access to the Internet. As a result, most of the devices are connected to the Internet. Therefore, all of those edge devices extended Cloud Computing into a new form of distributed computing, edge Computing.

Method

1. Data preprocessing

The training data we will use is the most common Boston Housing dataset. It is easy to fetch from Kaggle.com. We only trim a small section of the original data for experiments to eliminate the computational resource costs, since we will evoke five edge workers to asynchronously computing the gradient descent function and send the update to the central server. Also, we use the pickle to serialize our data to further optimize the performance of processing computational intensive tasks. Although pickle format is not a good solution for network security and encryption, our project is just for academic use and our data is an open-source public dataset, so there will be no privacy issues.

For data preprocessing, we will go check the data and give those null objects a numeric zero value or simply remove those no-meaning data. Secondly, we will convert the NumPy array into the pickle data format for further use. After that, we will get cleaned data and will divide those data evenly into five parts which is the number of our worker nodes. In the end, we will send those separated data to each corresponding edge node by Python socket server.

2. Training method

Although there are various machine learning models, here in our project, we are trying to gain an in-depth understanding and analysis of federated learning. So, we will perform a simple and explainable neural network model, a two fully connected linear layer with ReLU activation function at the intermediate layer and sigmoid function at the last layer. And use the Stochastic gradient descent iterative method for optimizing training loss. The architecture for the neural network model is shown below in Figure 2.

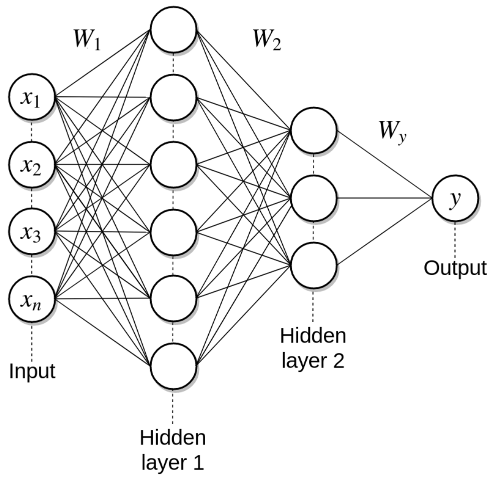


Figure 2. Architecture for neural network model

The basic idea of our project is to simulate the process of federated learning to solve a traditional prediction task. Each simulated edge worker will be given the same nn model and they will train the model based on their assigned local data. In each epoch, the computed new feature weights will be sent back to the central server and the server will do an aggregation for all updates to upgrade the current model. The architecture for the nn model is shown in Figure 3.

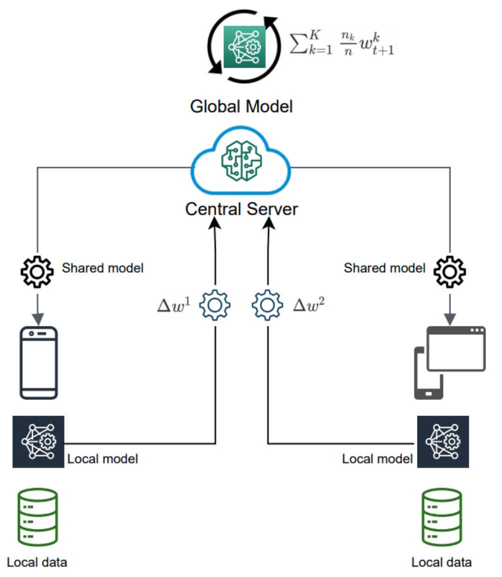


Figure 3. Simplified Architecture for Federated Learning

As we can see in figure 3, first, the central server will share the current best model stored in the central server's database. This model will be sent to each edge device as the same version. Then, the edge device will perform training tasks based on their local historical datasets to calculate new feature weights. Those weights will be collected by the central server and be aggregated and doing an average for updating the current model by the following mathematical formula. This is the formula for parameter averaging:

$$\min_{\mathbf{w}} \left\{ F(\mathbf{w}) \triangleq \sum_{k=1}^{N} p_k F_k(\mathbf{w}) \right\},$$

where N is the number of devices, and pk is the weight of the k-th device such that pk ≥ 0 and ∑ k=1to N for pk = 1. Suppose the k-th device holds the nk training data: xk1, xk2, · · · , xknk . The local objective Fk(·) is defined by

$$F_k(\mathbf{w}) \triangleq \frac{1}{n_k} \sum_{j=1}^{n_k} \ell(\mathbf{w}; x_{k,j}),$$

where ℓ(;) is a user-specified loss function.

From this strategy, the central server will keep its model up to date and maintain the best performance on real-world problem solving. In the meanwhile, the edge will keep their local model up to date and keep feeding new weights back to the server. This state of art learning method can not only protect the user's privacy data, because it will never spread local data out of local network, but also eliminate the data transition costs.

Experiments Result

The training epoch and it's loss plot is shown below in Figure 4 and 5.

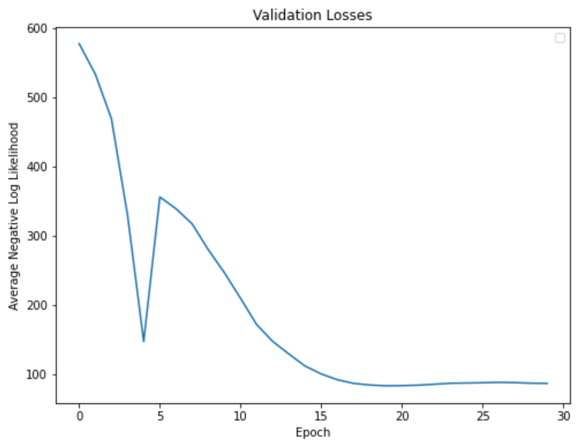
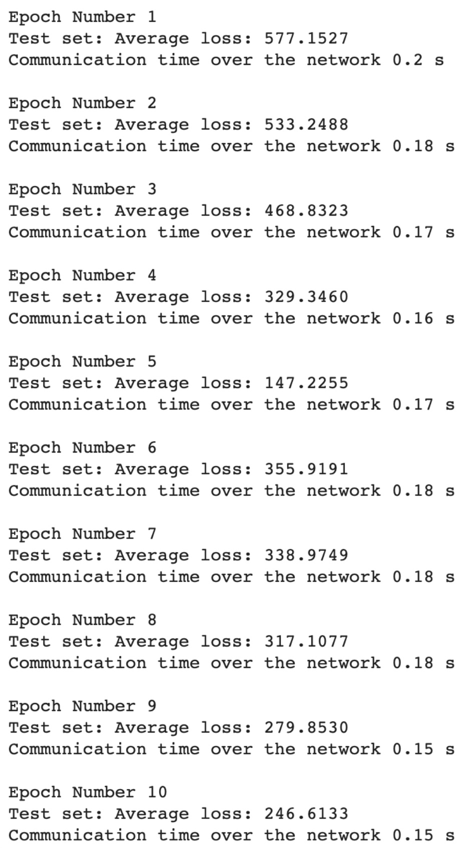


Figure 4. Validation loss plot of Fedavg



```
Epoch Number  1
Test set: Average loss: 577.1527
Communication time over the network 0.2 s

Epoch Number  2
Test set: Average loss: 533.2488
Communication time over the network 0.18 s

Epoch Number  3
Test set: Average loss: 468.8323
Communication time over the network 0.17 s

Epoch Number  4
Test set: Average loss: 329.3460
Communication time over the network 0.16 s

Epoch Number  5
Test set: Average loss: 147.2255
Communication time over the network 0.17 s

Epoch Number  6
Test set: Average loss: 355.9191
Communication time over the network 0.18 s

Epoch Number  7
Test set: Average loss: 338.9749
Communication time over the network 0.18 s

Epoch Number  8
Test set: Average loss: 317.1077
Communication time over the network 0.18 s

Epoch Number  9
Test set: Average loss: 279.8530
Communication time over the network 0.15 s

Epoch Number  10
Test set: Average loss: 246.6133
Communication time over the network 0.15 s
```

Figure 5. Training loss descent of first 10 epochs

Challenge and Limitation

Above all, privacy is still one of the most challenging problems in federated learning. For example, most of the distributed learning builds up the model at the edge; thus, it requires the user's consent to extract features from those data. However, even if users give their full consent, data leakage could still happen when updating the parameters. Therefore, the inevitable trade-off between privacy and accuracy is pushing researchers to develop privacy policies for federated learning.

And through our experiments, we also find a big challenge for the federated learning is the diversity of the local data. The Fedavg aggregation method is not a good solution for a complicated and multi-task federated learning problem. It is possible for IoT devices to contain a lot of un-meaningful data and highly biased data. How to pre-clear and select the key feature to update our model is still a big problem.

Moreover, federated learning needs a very high speed network bandwidth to ensure it’s frequent communication between edge and server as well as sufficient local computing power and memory.

Another point we can learn from our experiments is the challenge of temporal heterogeneity. It is possible that each local dataset's distribution may change with time, so how to overcome this inevitable change is a good topic that we need to overcome in the future.

Reference

[1] Federated Learning with Non-IID Data, Yue Zhao et al, arXiv: 1806.00582v1, 2 Jun 2018

[2] Communication-Efficient Learning of Deep Networks from Decentralized Data, H. Brendan McMahan et al, arXiv:1602.05629v3 [cs.LG] 28 Feb 2017

[3] OpenMined. (n.d.). OpenMined/PySyft. GitHub. https://github.com/OpenMined/PySyft.

[4] Ko, Y. (2021, January 5). Federated Learning Aggregate Method (1). Medium. https://medium.com/disassembly/federated-learning-aggregate-method-1-c2f96bc03f59.

[5] T. Yu et al., "Learning Context-Aware Policies from Multiple Smart Homes via Federated Multi-Task Learning," 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), 2020, pp. 104-115, doi: 10.1109/IoTDI49375.2020.00017.

[6] Bhattacharjee, A., Chhokra, A. D., Sun, H., Shekhar, S., Gokhale, A., Karsai, G., & Dubey, A. (2020). Deep-Edge: An Efficient Framework for Deep Learning Model Update on Heterogeneous Edge. *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*. Published. https://doi.org/10.1109/icfec50348.2020.00016

[7] McMahan, H.B., Moore, E., Ramage, D., & Arcas, B.A. (2016). Federated Learning of Deep Networks using Model Averaging. ArXiv, abs/1602.05629.