De-Centralized Training of Deep Neural Networks using Federated Learning

Meet N. Gandhi

George Mason University, mgandhi3@gmu.edu

Juhi D. Kamdar

George Mason University, jkamdar@gmu.edu

Simran Manturgimath

George Mason University, smanturg@gmu.edu

There has been a sudden increase in the use of Deep Learning from tasks ranging from image classification to generating complex models like GPT 3. One of the most common things with the Machine Learning or Deep Learning pipeline is that the entire process is done on the same server. The training, testing, and validation data along with the pre-processing and training of the model are on the same machine. In this project, we are going to talk about how decentralized training of deep learning models takes place and what are the advantages associated with this approach.

Additional Keywords and Phrases: Federated Learning, Tensor Flow Federated, Deep Learning, Distributed Training.

ACM Reference Format:

Meet Gandhi, Juhi Kamdar, Simran Manturgimath. 2022. The Title of the Paper: De-Centralized Training of Deep Neural Networks using Federated Learning.

1 INTRODUCTION

With the increase in the availability of data and processing power, deep learning has already surpassed traditional machine learning and statistical methods. There are several deep learning models which outperform humans in tasks like image classification, text generation, text summarization, and many more. Most of these models are trained on data that is publicly available. Datasets like MNIST, ImageNet, and NLP models from English text corpus are widely available, thus getting access to the data is not an issue here. But in the case of more sensitive data like tumor detection, we might face several challenges which are not faced by us in traditional datasets like MNIST. The first challenge is to gather millions of images of tumors, this kind of data is personal and its usage is legally complicated due to certain policies like HIPAA. In case if we want to use this data to train our model in order to identify tumors, we might need to purchase the data. Here, we can see a clear difference between working with public data and private data.

Due to the private nature of data, using a traditional machine learning pipeline for gathering data into one server and applying machine learning or deep learning methods to it might not be useful. To overcome such issues federated learning was introduced, in this approach, the data is not stored in a concentrated way on one server rather there is a system of server and client [4]. The data and model are distributed among clients, clients process with their individual model training for a few epochs and once the training is completed the updated weights are sent to the server. The server then merges the updated weights from all the clients and finally, the updated weights are passed down to the clients. Not only privacy, but federated learning can be also useful in applications like Improving latency, using models to work offline, better battery life, etc [1].

One of the common examples where federated learning is used, are voice assistants like Siri, Alexa, and even Google's keyboard, Gboard. Let's consider the example of Alexa to understand this concept. There would be thousands of EcoDots (Device that uses Alexa) deployed, all of which are connected to the internet. These clients are trained on an individual voice by commands like "Hey Alexa, set an alarm" or "Hey Alexa, what's today's date?" (depicted in part A of Figure 1). All these individually trained clients (depicted in part B of Figure 1) send their model weights to the server. The server performs several operations and aggregates the learned weights. The updated aggregated weights are then sent to the clients thus improving the overall accuracy in terms of identifying words/ accents more accurately (depicted in part C of Figure 1).[8].

In this project we are working with MNIST dataset and testing the training process with clients ranging from 2 to 10. We evaluate the testing and training accuracy of the model along with the processing time taken in the entire process.

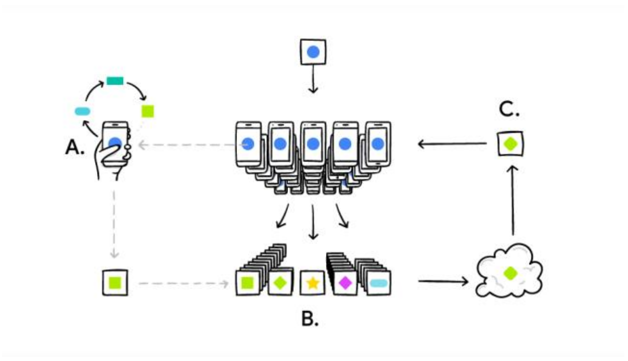

Figure 1: Overview of Federated Learning System [Public domain], via Google AI Blog [8].

2 Method

There are several ways the federated learning environment can be replicated, one of the ways for it to be done is to use Docker for generating servers and clients, and transfer data between them. To make this task easier there are several federated learning frameworks like PySyft by PyTorch[6], Flower[5], Federated by Tensorflow[7], and many more. For this project, we are using the Tensorflow framework "Tensorflow-Federated".

The TensorFlow-Federated framework provides us with several modules and methods which can be directly used to create a federated learning environment. Federated learning requires a federated data set, i.e., a collection of data from multiple users. Federated data is typically non-i.i.d., which poses a unique set of challenges. Tensorflow-Federated provides us with a few inbuilt non-IID data sets like StackOverflow, Federated EMNIST, Shakespeare CIFAR-100, and CelebA. For the sake of this project, we are using the Federated EMNIST dataset. Using this dataset will enable us to divide the training data among all the clients, typically they have different distributions of data depending on usage patterns. Some clients might have more data depending on the usage and the storage availability while some clients might have less data on them.

Once the dataset is decided there are several methods provided by TensorFlow-federated which are as follows:

Functionality	TensorFlow Federated Methods
Load Data	client_data, _ = tff.simulation.datasets.emnist.load_data()
Create Single Client	first_client_id = client_data.client_ids[0]
Assigning subset of data to	first_client_dataset = client_data.create_tf_dataset_for_client(first_client_id)
Iterative Process	for i in range(6):
	client_dataset = emnist_train.create_tf_dataset_for_client (emnist_train.client_ids[i])

Table 1: Tensorflow Federated pre-defined command for Federated Learning

For the experimentation, we have selected up to 10 clients, with a batch size of 20 and 25 epochs. For the following distribution of clients, we can simply use the commands provided in the above table.

Once the dataset is chosen and clients are generated we can move forward with the model part. For generating the model we are using the TensorFlow Keras package. Here in this project, we are not focusing on making more complex and deep models rather we are focusing on implementing the training process in a distributed manner. Thus we make a simple model with two dense layers, one with 512 neurons and the second with 256 neurons, then we add a drop-out layer with 50% probability and finally 10 neurons at the end with a softmax activation function. The structure of the model can be seen below.

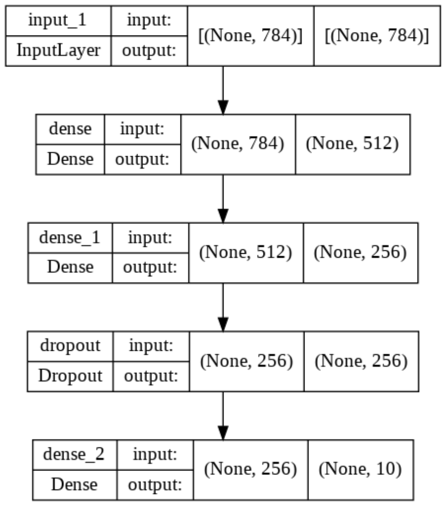

Figure 2: CNN model used for experimentation.

Finally, we set the client and server optimization function as SGD, once the learning process starts. Each client is given a chunk of data to train the model on, for each epoch, the clients train their model with the provided data and sends that update to the server. The major work done by the server here is to aggregate the updates obtained from all the clients and sending the aggregated weight back to clients.

3 EXPERIMENTATIONS / RESULT

Here we perform and analyze our experiments, there are plenty of hyper-parameters that we can change like the deep learning model, learning rate, and adding drop out and normalization layers but rather than focusing on deep learning models we are focusing here on the number of clients and number of epoch each client is been trained for, by keeping the deep learning model constant.

3.1 Analyze model performance with different numbers of epochs keeping the number of clients constant.

Keeping the computational resources in mind we are testing with a number of clients equal to 5 and 10 respectively and monitoring our model with different numbers of epochs. As the number of clients is 5 and 10 the dataset will be divided into 5 and 10 parts respectively and distributed among the clients. Ideally increasing the number of epochs should gradually increase the accuracy of the model. As shown in the table below we can see the accuracy for 5 clients increases from 46.35% to 94.16% and for 10 clients is 76.16% at the 5th epoch and it keeps on increasing up to 97.52% in the 25th epoch. Both the training and testing accuracy is as expected, hence proving that the model has been trained well.

Number of Clients	Epochs	5	10	15	20	25	Average time/ epoch
5	Training Accuracy	46.35	71.29	83.46	91.43	94.16	4.09 seconds
	Testing Accuracy	70.17	71.92	75.43	78.94	84.21	
10	Training Accuracy	76.16	89.15	93.24	97.17	97.52	6.77 seconds
	Testing Accuracy	66.37	74.13	78.44	81.89	82.75	

Table 2: Training and Testing accuracy vs Number of Epochs

We are also taking the execution time into consideration as the number of devices increases the load at the server side increases, as it needs to manage more devices simultaneously and thus training a higher number of clients takes more computation time as compared to training with a lesser number of clients. Thus a clear distinction in computation time is seen for 5 and 10 clients.

3.2 Analyze model performance by keeping the number of epochs constant and changing the number of clients.

In this experimental part, we are comparing the model result with increasing the number of clients. We are testing with 2, 4, 6, 8, and 10 clients, each of the clients is trained for 25 epochs. Now if the number of clients is 1 then it is similar to the normal machine learning training approach because having only 1 client means all the data and training is done on just one client. On increasing the number of clients the server needs to manage all the transactions and updates, for that it takes the average of all the updates. Now while each client is sending its update to the server a small random noise is introduced to the updating weights such that even if the update has been intercepted by a 3rd party they will not be able to reverse engineer the model or the data. Thus as the number of clients increases this random noise also increases and thus when the weights are aggregated on the server, we can see a slight decline in model performance due to that added randomness in the weights.

Number of Clients	2	4	6	8	10
Training Accuracy for 25 epochs	99.20	93.11	92.43	91.81	91.35
Testing Accuracy for 25 epochs	99.99	77.77	84.05	81.52	81.03
Average time/ epoch (Seconds)	1.62	2.90	4.34	5.57	6.97

Table 3: Training and Testing accuracy vs Number of Clients

Finally, we again take processing time into consideration. As explained in the previous experiment, computation time increases, if we increase the number of clients, as the time duration of each iteration also increases. Here we can clearly see that the average time for each epoch increased from 1.62 seconds for 2 clients to 6.97 seconds for 10 clients.

4 CONCLUSION

Deep Learning has been used in numerous tasks and even out-performed humans in some of the tasks. But the data hungry nature of such models is one of the major limitations which restricts us to utilize these models for more sensitive data. Federated Learning helps us to solve such issues and takes us one step closer in terms of using private data without actually sharing it, but also taking advantage of such data to generate models which can help professionals like surgeons to identify diseases like tumors or cancer. In this project, we took the example of the MNIST dataset, and with the help of the TensorFlow Federated framework, we showcase how can we train a deep learning model in a distributed manner and how its performance and computation time might vary according to the number of clients we have in the system. With just 25 epochs and 5 clients, we were able to obtain a testing accuracy of 84%, and a much deeper Deep Neural Network architecture can be later added to increase the accuracy.

REFERENCES

[1] Jakub Konečný and H. Brendan McMahan and Felix X. Yu and Peter Richtarik and Ananda Theertha Suresh and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. NIPS Workshop on Private Multi-Party Machine Learning. https://doi.org/10.48550/arxiv.1610.05492

[2] Kai Hu, Yaogen Li, Min Xia, Jiasheng Wu, Meixia Lu, Shuai Zhang, Liguo Weng, "Federated Learning: A Distributed Shared Machine Learning Method", Complexity, vol. 2021, Article ID 8261663, 20 pages, 2021. https://doi.org/10.1155/2021/8261663

[3] Qinbin Li and Zeyi Wen and Zhaomin Wu and Sixu Hu and Naibo Wang and Yuan Li and Xu Liu and Bingsheng He, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection", IEEE Transactions on Knowledge and Data Engineering, https://doi.org/10.1109%2Ftkde.2021.3124599

[4] McMahan, H. Brendan and Moore, Eider and Ramage, Daniel and Hampson, Seth and Arcas, Blaise Agüera y, "Communication-Efficient Learning of Deep Networks from Decentralized Data", https://doi.org/10.48550/arxiv.1602.05629

[5] https://flower.dev/docs/

[6] https://github.com/OpenMined/PySyft

[7] https://www.tensorflow.org/federated

[8] Brendan McMahan and Daniel Ramage, Research Scientists Google AI Blog, "Federated Learning: Collaborative Machine Learning without Centralized Training Data" https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

Final Project Video Link: https://drive.google.com/file/d/1-y8LTM3Rddm9N_Su8dz2h84S71BCRU5u/view