UiT The Arctic University of Norway

Faculty of Science and Technology Department of Computer Science

Investigating and developing efficient federated learning for air pollution monitoring

Jørgen Reinnes

INF-3981 Master's Thesis in Computer Science - June 2022

“Simplicity is prerequisite for reliability.”
–Edsger Dijkstra

“Beware of bugs in the above code;
I have only proved it correct, not tried it.”
–Donald Knuth

Abstract

Location-based data may be considered highly private; as such, handling location-based data requires that it cannot be used to track a user. In a network of multiple edge devices that each collect data, training a machine learning model would typically involve transmitting the data securely to a central server which requires strict privacy rules.

Federated learning solves the privacy problem by not requiring data to be shared; instead, training of a machine learning model is performed on the device that gathered the data itself. Using federated learning with the Federated Stochastic Gradient Descent (FedSGD) algorithm, a similar training performance is expected as training a machine learning model on a single server with data transmitted to it. Overall less bandwidth may be used for communication between edge devices and the server. However, a higher computational cost is seen due to having to perform model training on the edge device, which lowers the potential data points that can be processed each day given the lower computational performance of an edge device versus a high power server. Whilst only a single edge device may train the model at a time, a different federated learning algorithm may be used on the server to enable multiple to train simultaneously.

Contents

List of Figures

1 Introduction

Data directly linked to a person's whereabouts is considered private information and is thus under strict data handling guidelines to avoid misuse. Using a device mounted on something like a bike. The device will be gathering positional data in addition to the sensory data. Since the positional data can be used to track the user, it is considered private and must thus be handled accordingly such that it does not get stolen or misused.

Handling private information can be solved by storing it securely, but if it never leaves the device as it is, then the chance for accidental misuse is drastically lowered. If private data must be handled, then security requirements are placed on how the data is transmitted, stored, and processed. All of which increase the cost and complexity. Instead of only sending anonymous data from the device, more focus can be put on what to use the data on rather than how to safely keep it.

Algorithms running on remote monitoring devices will be limited by battery and processing power. Given a constraint where raw, private data cannot leave the device. Data must be fully or partially processed on the device to become anonymous. This imposes restrictions based on the device, such as battery and processing power. Which, in turn, puts restrictions on the algorithm. An algorithm that can only run on one device in a network of devices at a time will have scaling limits.

Federated learning is a relatively new computer science field which means new

changes may come, or issues to be solved may appear. The basic setup of a machine learning-based system is changed with federated learning. Rather than train the model on a server, it is trained on edge devices themselves. Additional computing resources are required, but the privacy of edge device users is preserved as no raw data leaves the device. Given the current implementation, only a single edge device can train the model at a time, but there are federated learning algorithms that allow for multiple to train simultaneously. However, training performance may suffer.

1.1 Related Work

The federated learning FedAvg algorithm is created by the authors of paper [8] as a method for communication efficient training on decentralized data. The paper investigates how federated learning can be used to train a deep neural network on decentralized data with multiple clients simultaneously. By copying the model to each client, who then trains using local data, an averaging algorithm is used to combine all clients' results into a single model.

2

Technical Background

Introduced here are some technical terms and technologies to understand this project.

2.1 HAPADS: Highly Accurate and Autonomous Programmable Platform for Providing Air Pollution Data Services to Drivers and the Public

The HAPADS project aims to create a low-cost air monitoring platform for monitoring air quality such that end-users can decrease exposure to air pollution. Air pollution is a common problem in densely populated areas, but it is heavily dependent on environmental factors such as weather and motor traffic. As such, HAPADS has been developed with small cost and size requirements such that it can be widely deployed, offering data such that end-users may avoid certain times of the day or locations when air pollution is high. [4]

The platform is low-cost and small in size, using custom sensors. By being custom-designed with custom sensors and software, costs are lower, and it can perform better for the target task by being specialized. However, the sensors have a lower accuracy compared to more expensive options. In addition, they

have a geographical dependency where they must be calibrated to a local area. [4]

2.2 Machine Learning

2.2.1 What is Machine Learning

Rather than manually code how an algorithm behaves with different inputs, machine learning is a way to train a model to create that algorithm. An algorithm that a person has coded will behave in the way it was programmed, and if the algorithm is exposed to something unexpected, it is likely not to get the expected results. Machine learning would be likewise, except for not requiring a person to update the algorithm. The algorithm can train on this new occurrence to thus produce the expected result[5].

2.2.2 Neural Network

Neural network is a subset of machine learning that simulates the behavior of the brain's neurons to compute a set of results based on a set of inputs. A neural network is composed of three main parts. The input layer with one or more input nodes, one or more hidden layers, and an output layer with one or more output nodes. Each node connects to nodes in the next layer through a connection which is a weight factor that is applied to the value of the node. The node's value is an input value for the input nodes or the sum of the connections for other nodes. A node is activated if its value is greater than a threshold set for it [6].

Training a neural network involves using input with expected output to adjust connection weights and thresholds to fit the expected value better using statistical methods. A neural network can be run using input data with known outputs such as labeled images and then adjust its weights and thresholds to fit the training data better. To calculate the error mean squared error (MSE) may be calculated from the Cost Function(2.1). The error is lowered using a function like a gradient descent which modifies weights and thresholds as seen on 2.1 [6].

The output of a neural network labeling images is not a specific answer but rather a percentage of how confident the input is of a specific label. A neural network that detects what digit an image is displaying may have ten output nodes—each with a value between 0 and 1 interpreted as between 0% and 100%. Given an image of 2, a network may output 0.2 or 20% from the output

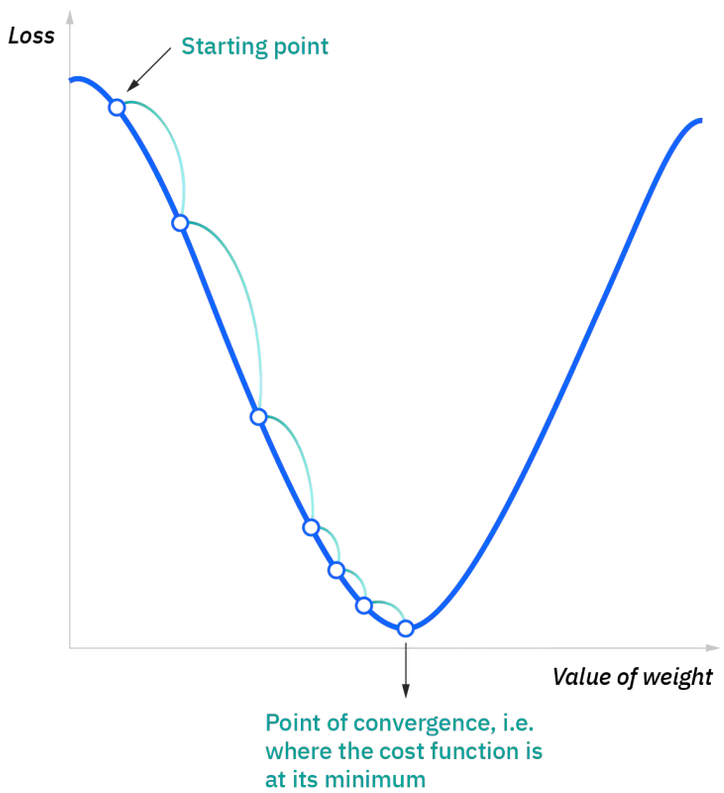

Figure 2.1: Error is high to begin with, but will converge to a lower error as more training sessions are performed [6]

representing the digit 1, 0.95 or 95% from the output representing the digit 2, and values from the other outputs. Given this, the neural network is 20% confident that it is an image of the digit 1 and 95% confident it is the digit 2 [6].

$$CostFunction = MSE = \frac{1}{2m}\sum_{i=}^{m}(\hat{y} - y)^2 \tag{2.1}$$

2.3 Federated Learning

In a typical setting, machine learning is performed on a single computer only; however, Federated Learning (FL) expands upon it to allow for a single machine learning model to be trained using multiple computers. The data stays on each

device in a federated learning setting, preserving privacy. Instead, the machine learning model is transferred to each device such that the model is only trained with locally acquired data. Multiple federated learning algorithms exist, which mainly offer the main benefit of data privacy, but differ in their performance metrics, such as being able to train on multiple devices simultaneously or not [7].

By distributing the machine learning model, as opposed to training data, the total amount of data transmissions may be lower. Given a federated learning algorithm that transfers a machine learning model to a device for training on local data. The amount of data transmitted depends on how often the model is transferred between devices—as such, increasing the size of the training session may lower the total amount of data transmission by requiring fewer training sessions.

2.3.1 Federated Stochastic Gradient Descent Algorithm

Federated Stochastic Gradient Descent (FedSGD) is a federated learning algorithm that offers the training efficiency of a centralized machine learning model with decentralized data. FedSGD is, in practical terms, a federated learning algorithm, where the machine learning model is transferred between devices to train on local data. Given that it may only train on a single device at a time, it has a peak for how much data it can be trained on in a given period. However, it offers training efficiency equal to training a centralized machine learning model, which means less training data is required for the model to converge.

2.3.2 Federated Averaging Algorithm

An improvement to FedSGD was proposed in [8] known as FederatedAveraging (FedAvg) which allows for simultaneous training on multiple devices but lowers training efficiency. FedAvg allows for a model to be copied onto multiple devices during a training session. The devices may be chosen at random, and a device can be aborted. The resulting models are fetched and then averaged from the devices that finish getting the updated model. Additional training rounds are required as the final average may cause the model to diverge, given that different devices may train on different kinds of data.

2.3.3 Split Learning

Rather than running the whole machine learning model on a single device, split learning splits the model such that one part may run on the edge device, and the other part may run on a server. The machine learning model is split into multiple pieces at layers in a split learning setting. Such that the input layer and a few hidden layers are the first part. While the rest of the hidden layers and the output layer are on the other part. Training such a model involves running the first part with the input data, then the second part with the output of the first part. Data from the first part may be considered obfuscated [3].

2.4 TensorFlow

TensorFlow is a programming interface used to express and implement machine learning algorithms to be run on heterogeneous computers. By abstracting away much of the complexity involved with machine learning, a more straightforward setup for training and production can be achieved. TensorFlow exists for multiple programming languages, notably Python and JavaScript, and can thus be run on most modern computers with none or minimal modifications to the program code [1] [2].

3

Method

To explore the field of federated learning, this paper will be conducting investigative research into if federated learning can be beneficial to a distributed network of low-cost sensory devices. Given the test devices, a focus will be on end device power usage and bandwidth usage.

Given the usage of both new algorithms and hardware, results found in this paper may change as better iterations of the algorithms come, or more efficient hardware is created. Given that federated learning is a relatively new field with open research topics[7], algorithms used may be improved through future iterations, which may improve the training efficiency.

The research in this paper is mainly done through practical experiments and thus follows the action research strategy. Machine learning models will be implemented using Python with TensorFlow and are performed on a single computer.

4

Design

This chapter will describe the design of a federated machine learning system for data coming from mobile devices.

4.1 Chapter Outline

- **Section 4.2** explains the reason for using federated learning.
- **Section 4.3** illustrates the initial solution using only federated learning.
- **Section 4.4** adds onto Section 4.3 on how split learning can be utilized for an alternative algorithm.

4.2 Introduction

Collecting air quality data from large areas using mobile edge devices poses several difficulties regarding system requirements and privacy. With a device intended to be deployed on anything from a bike to vehicles, certain system constraints need to be worked around. There will be a battery constraint such that the device must run efficiently when performing work or when transmitting the data. Secondly, data that is collected must be handled in such a way that

privacy is preserved.

This chapter describes a proposed system using federated learning for collecting and processing sensory data to build a machine learning model to calibrate the edge device sensors. The system will consist of edge devices deployed on a vehicle such as a bike, where it will collect sensor data. The data may be collected at specific locations using the low power communication method Bluetooth.

Data is typically processed in the cloud for centralized learning, not on the edge device. To ensure users' privacy using the edge devices, raw data cannot leave the device. Two different methods for training the machine learning model will be compared. Split learning as described in [3], but without backpropagation, and FedSGD. With federated learning, the training runs entirely on the edge device, while with split learning, only the first couple of layers are run for the rest to be processed on a server.

4.3 Federated Learning

Using federated learning, machine learning model training is performed on the edge devices rather than collecting the data. A centralized server, accessible by all edge devices, holds the machine learning model. When an edge device has data to process and is within a location for data collection, the machine learning model is transmitted to it. The edge device trains the model using locally acquired data, then transmits back the updated model to the server.

4.3.1 FedSGD

Federated Stochastic Gradient Descent (FedSGD) is a method of federated learning which works as a distributed centralized training with one device being able to train the model at a time. In FedSGD, a server may hold the machine learning model. Clients may request the model to perform local training on it. One client at a time may receive the model to perform training on it using local data. When complete the updated model is returned to the server.

4.4 Split Learning

As outlined in [3] split learning has a lower computation usage that directly correlates to the size of the part of the machine learning model the edge device must compute. Instead of running the entire model on a single device, split learning is a method in which the model is split between devices. Contrary to the method in the paper, this implementation will not perform backpropagation. Instead, an edge device receives a part of the current model at the start of each batch, then executes it with the sensory data. The output data is then transmitted to the server where the rest of the model is run, and backpropagation is performed. This is repeated for all batches—one device at a time.

5

Implementation

Using TensorFlow, a simplified implementation of the system was created. The simulation simulates the computational work the client would do each training session using real-world test data and gathers information on the computational time of the client.

5.1 Languages and libraries

Python 3 version 3.8.8 is chosen for having proven machine learning libraries and the lower implementation complexity.

Major libraries used:

- **Keras**[1] - Provides method to create a machine learning model. Version 2.9.0
- **TensorFlow**[2] - Provides a high level interface for performing machine learning operations. Version 2.9.1

1. https://keras.io/api/
2. https://www.tensorflow.org/api_docs/python/tf

5.2 Dataset

The dataset used is raw data gathered from a weather station (ARMAG) and a prototype of the HAPADS edge device HAPADS located at the weather station during two time periods. The HAPADS research group provides the data. Temperature, humidity, pressure, PM1, PM10, and NO2 were gathered every hour from 11.01.2022 to 18.01.2022 and up to every minute between 01.02.2022 to 15.02.2022.

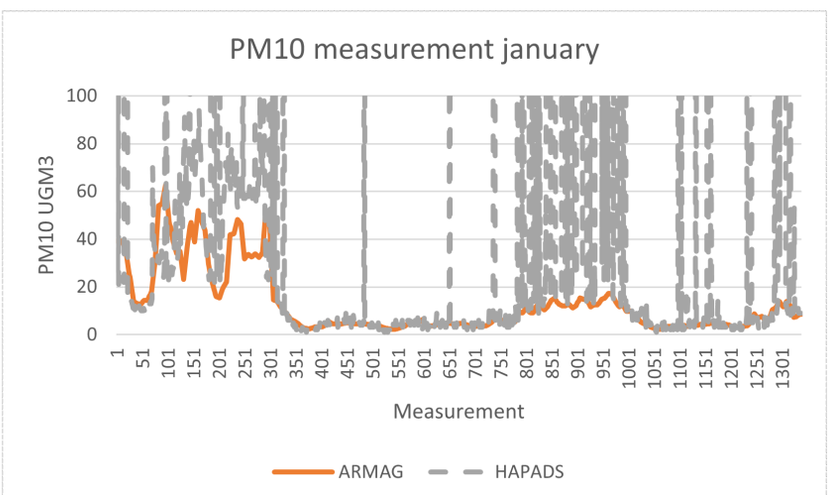

Figure 5.1: PM10 data from January plotted over time, Y-axis has been limited to 100 to fit the ARMAG results, but HAPADS has spikes to 30 thousand which are clipped

Because of anomalies with the HAPADS platform, the PM10 sensor data includes data spikes significantly higher than the surrounding data points, as such values above 70 were omitted. The ARMAG data has a stable data output from near 0 to one peak at 60. The HAPADS data closely follow the trend of the ARMAG data but has some spikes up to 30 thousand. Since these spikes would significantly degrade the model, data above 70 were omitted, or 200 out of 1300 data points, as seen in figure 5.2.

5.3 Model

A machine learning model for calibrating the edge device sensors has been created using Keras. It consists of an input layer with three inputs, one or more hidden dense layers with three nodes each, and one output. The model takes the temperature and humidity from ARMAG and PM10 from the edge device and outputs a value for PM10, which is compared with the PM10 from the

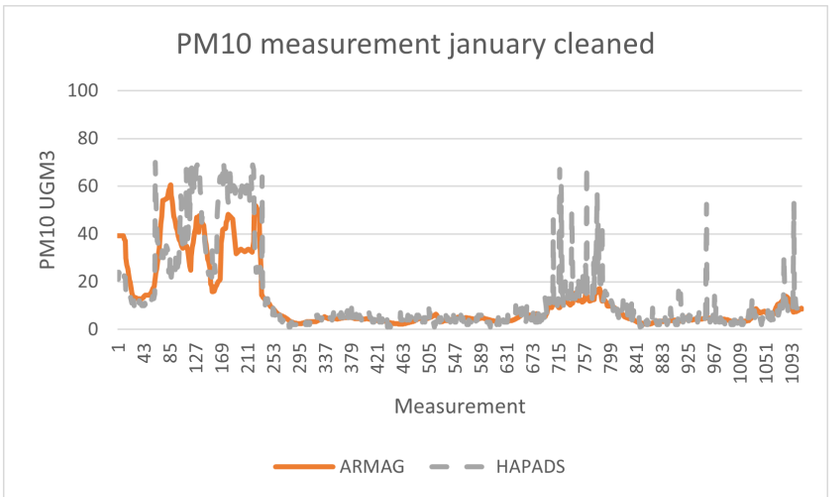

Figure 5.2: PM10 data from January plotted over time, and Y-axis has been limited to 100 to fit the ARMAG results, data points with spikes over 70 have been omitted

weather station.

5.4 Server

The server manages the machine learning model and distributes the model to contact clients. As described in algorithm 1 below, a server runs in a loop waiting for clients to request the machine learning model. One client at a time may be sent the machine learning model to train on it as outlined in algorithm 2 below. When the client finishes, the server receives the updated model.

5.4.1 Client

The same devices that collect the data also perform the training of the machine learning model. There may be one or more devices in a network that collects data from sensors. Data is stored locally on the device until it can connect with the server to receive the current machine learning model. It trains the model with a batch size of 128 upon receiving it before transmitting the updated model.

Algorithm 1 Federated Learning Server

```
for round = 1, 2, ... do
    Receive client request
    if Model is locked then
        Put client on wait
    else
        Lock model
        Send model
    end if
    Wait for client to train model as per algorithm 2
    Receive updated model from client
    Set received model as current model
end for
```

Algorithm 2 Client Training

```
while true do
    while No data OR not near a server do
        Gather local data T
    end while
    Request model from server
    Wait until server sends current model
    Set local model as received model
    for Batch D in T do
        Train local model on D
        Update model
    end for
    Send updated model back to server
end while
```

5.4.2 Split Training variation

In the split learning alternative, the logic for training and data transmissions is altered on both the client and server-side. The client receives only the first hidden layer of the model and transmits the output from running the data through the layers to the server. The server then completes the training using the client output and updates the weights.

6 Evaluation

This paper aims to evaluate methods to efficiently create machine learning models from training on data generated from multiple edge devices, locally on the devices themselves. A high-level implementation is created using Python, demonstrating federated learning as a method for training locally on the devices that created the data. The method is compared with split learning based on the client computation time and the total amount of data transferred between client and server.

6.1 Experimental Setup

Experiments were performed on the same computer using the GPU for training:

Desktop Computer	
CPU	AMD Ryzen 5900x, 12-Core, 24-Thread, 3.7/4.8GHz
GPU	NVIDIA RTX 3090
RAM	32GB DDR4@3600MHz
OS	Windows 10 Professional Build 19043

6.2 Benchmarks

Running in a virtual environment, time is measured for TensorFlow to perform all the computations on the edge device. With the data from February sorted by time. Training is done on each data-point multiple times to simulate the computation the edge device would perform. The benchmark will run for 50 epochs with a batch size of 128, at which point total time spent is recorded, and the total amount of bandwidth is calculated. Execution time is measured for the training duration. Split learning and federated learning are tested independently, with split learning only performing model prediction with the partial model. Each method is tested with 1, 2, 4, 8, and 16 hidden three-node layers in the model to artificially increase the computing complexity and size of the model.

Assuming 4-byte float types, the input to the model consists of three 4-byte values, and a single 4-byte value is output. Likewise, each weight is 4 bytes. This gives a total size for the model of (24 + 36 * hidden layers) bytes that is transferred both before and after each training session consisting of 2 batches during federated learning. On the other hand, the first layer is received as 36 * split hidden layers bytes of data during split learning. While the output, which totals to 12 bytes, is transferred to the server. Multiple batches may be run per training session.

Results

Increasing the number of hidden layers increases the computational time on the client-side for federated learning throughout the benchmark. Split learning is constant at 2.8s no matter the number of hidden layers. Likewise, split learning is constant in data transfer amount, but for federated learning, the model data transferred is larger the more hidden layers there are, and thus more data is transferred.

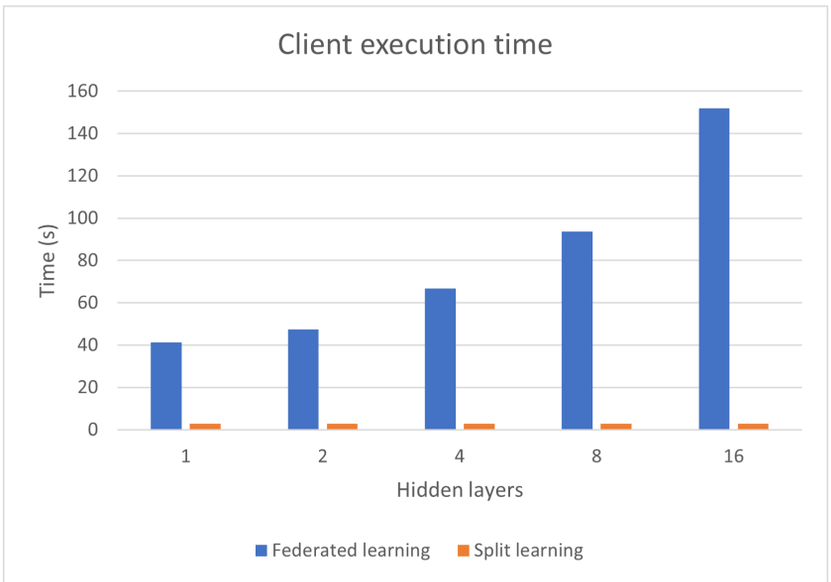

Figure 6.1: Raw execution time to perform each methods training on the client for 50 epochs

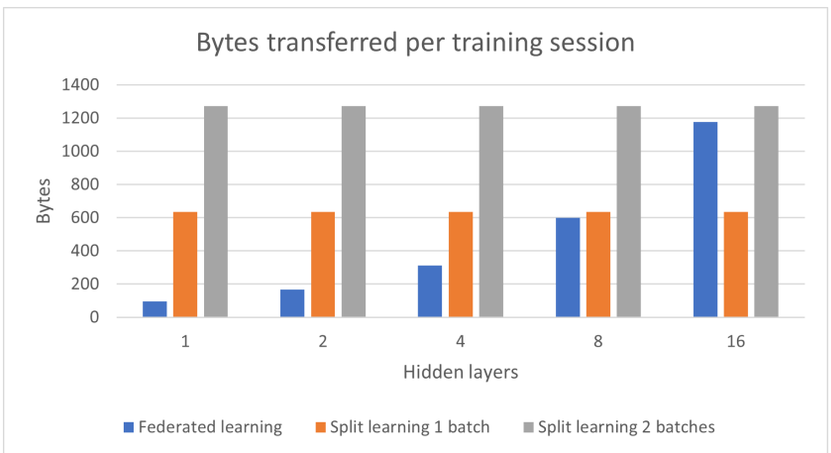

Figure 6.2: Data transferred between server and client for each method. Because federated learning allows for multiple batches to be processed each session, it does not affect data transfer size

7

Discussion

Using Federated learning data from edge devices can be used to train a single model without private data leaving each device. During each training session, only the model itself needs to be transmitted between server and client at the start and end. Depending on the model's size and data trained on, less data transfer occurs than sending the raw data itself.

7.1 Client Time Usage

Federated learning and split learning differ heavily in the client execution time, with federated learning requiring much more execution time. Federated learning has an initial execution time of 41 seconds with one hidden layer but goes up to 152 seconds with 16 hidden layers. Given that split learning always perform the same amount of computation client-side, it computes all epochs in 2.8 seconds.

There would be some computational time spent to perform communication between client and server in the real world. However, given the small amount of data transmitted, this would be insignificant compared to performing the client-side training. Given that only the training computation is measured but not the expected computation for communication, there is some but not significant computational time that is not measured.

7.2 Data Transfer Size

Data transferred differs drastically based on each method, with a low of 96 bytes and a peak of 1272 bytes transferred between client and server each training session. For federated learning, where the model is transferred between each training session, the data transfer size increases with model size from 96 bytes with one hidden layer to 1176 bytes with 16. On the other hand, with split learning, the data transfer size is static at 1272 bytes, as the split learning method only transfers up to the first hidden layer of the model.

7.3 Test Data

Given that the test data is from a single location, it is not necessarily a perfect example of a real-world example that would gather data from multiple locations. In a real-world example where multiple devices gather data, it is expected that the devices could gather radically different data based on the locations they visit—for instance, traveling on a motor vehicle road versus using the bike trail through a forest.

Secondly, while there were still anomalies in the cleaned-up test data, it would not affect the results, given that model accuracy was not the focus. Some of the test data had to be omitted due to anomalies where the experimental PM10 sensor recorded some significantly larger values at random. Some unexplained spikes in the HAPADS test data were omitted as the focus was not on model accuracy. As seen in figure 5.1 above, the HAPADS platform follows ARMAG closely and generally has a value a little higher than ARMAG except for the few spikes.

7.4 Results

Federated learning drastically increases the client computation time with model size compared to split learning. Even with just one hidden layer in the model, which means split learning lacks the output layer versus federated learning, there is a significant difference of 41s with federated learning versus 2.8s with split learning in client computation time. This difference is thus mainly attributed to the time it takes to perform the backpropagation for updating the weights after training an epoch, as split learning does not perform backpropagation on the client.

Since an edge device is power-constrained and thus has lower computational

performance than a server, lower throughput is expected. In a network of edge devices, each edge device has a lower computational performance as it is limited by the battery instead of a server that can use powerful hardware. Thus training on the edge device takes longer than done on the server. However, federated learning can use a lower-cost server since the server does not need to perform much computation. As such federated learning with the chosen algorithm FEDSGD has an expected lower throughput versus split learning as both are limited to one client training at a time.

8 Conclusion

8.1 Concluding Remarks

This paper presents a federated learning experiment to create a machine learning model based on data from multiple edge devices without private data leaving the devices. Training occurs on the edge devices themselves rather than on a centralized server with the federated learning model. Thus instead of transmitting raw data from edge device to server, the model itself is transmitted. Performing 50 epochs of training took 152s with the biggest model, but only 2.8s with split learning. However, whilst federated learning is static in the amount of data transmitted between client and server, no matter the number of data points trained on, split learning would see an increase.

Based on the experimental results, federated learning can be used to train a model based on private data from edge devices. It cannot, however, in the implementation using Federated Stochastic Gradient Descent used in this paper scale to accommodate an arbitrary amount of edge devices and is thus limited by the performance of each edge device. A different algorithm may be optimal to be able to handle a larger number of edge devices.

8.2 Future Work

Given the current implementation, there are multiple possible future iterations to federated learning that can be explored, which could solve the main issue of scaling.

Since FedSGD does not scale due to the limit of only being able to run a single training session at a time, a better algorithm such as FedAvg, which allows for multiple clients to train simultaneously, should be explored. However, given the limited battery of edge devices, it may be beneficial to attempt a combination method that uses both FedSGD when required throughput is low and another like FedAvg when required throughput is high.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org/`, 2015. Software available from tensorflow.org.

[2] Google. Neural networks. `https://www.tensorflow.org/about`. Last accessed 10-05-2022.

[3] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. `https://arxiv.org/abs/1810.06060`, 2018.

[4] HAPADS. Hapads. `https://hapads.eu/`. Last accessed 03-05-2022.

[5] IBM. Machine learning. `https://www.ibm.com/cloud/learn/machine-learning`. Last accessed 1-06-2022.

[6] IBM. Neural networks. `https://www.ibm.com/cloud/learn/neural-networks`. Last accessed 30-05-2022.

[7] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[8] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–

1282. PMLR, 2017.