Imperial College London

Department of Electrical and Electronic Engineering

M.Eng. Electronic and Information Engineering

Final Year Project - Interim Report

Hierarchical Federated Learning with Client Mobility

Author:
Alessandro Serena
alessandro.serena16@imperial.ac.uk
CID: 01188591

Supervisor:
Dr Deniz Gündüz
Imperial College London
EEE Department
Information Processing and Communications Lab

Date: January 13, 2020

Contents

1 Project Specification

Many modern applications rely on models generated using Machine Learning techniques on large-scale datasets. However, in cases when data is generated by edge devices (e.g. mobile phones, IoT smart sensors, wearable devices, etc.), it is often not possible or inefficient to harvest this information and send it to a central server to perform a centralized fitting of the model due to bandwidth constraints and privacy concerns.
In Federated Learning (§2.2) the model is updated in a distributed fashion. Edge devices update the global model on their local datasets, and then send the updates to a central server that aggregates them generating a new global model.

Considering that large-scale wireless implementations of this process can involve millions of devices, it is trivial to see that having only one aggregation server constitutes a huge bottleneck due to wireless communication latency and processing time.
To solve this issue, Hierarchical Federated Learning (HFL) has been proposed (§2.3). In this setting, an intermediate layer of small cell base stations (SBSs) is introduced. The clients communicate with one of the local base stations, which then periodically send their local models to the macro base station (MBS) to generate a global model.

A common shortfall of [1] and [6] is the fact that both papers present HFL assuming that the clients are fixed and always communicate with the same SBS, and that the clients are equally distributed in the clusters. These assumptions clearly do not hold in reality, where FL is meant to be performed by edge devices such as smartphones or wearables
The aim of this project is hence to investigate how wireless HFL systems perform when clients are allowed to move and to communicate with different SBSs, and how the existing HFL algorithms can be modified to take advantage of client mobility.

Understanding the behaviour of a HFL system when clients are allowed to move is not only important from a communications or distributed systems point of view (e.g. how devices can connect to a new SBS when they change geographical zone, how SBSs should handle the connection/disconnection of a user, etc.), but it is also crucial with regards to the learning task. In facts, as presented in [1], SBSs collaborate together with the MBS in the inter-cluster model averaging because otherwise they would not have any knowledge learnt from datasets of clients that do not belong to their own pool of devices.
Intuitively, if clients are allowed to move and hence to carry their local datasets to different SBSs, the advantage of performing inter-cluster model averaging decreases. If all devices were moving and ended up in all the clusters, every SBS would have received updates based on all the data held by the clients and therefore would end up with the same model parameters without having to share its model with the MBS.
One intuition is that it might be possible to reduce the frequency of the inter-cluster model averaging (thus saving bandwidth) as devices move more on average. This will be tested during the experimental phase.

The experimental data for the evaluation of the algorithms will be obtained through simulating an HFL system in which clients can move in between each round to adjacent cells (and hence communicate with a different SBS) depending on some probability distribution specific for each client. The main idea is to have different classes of edge devices with different mobility factors, mimicking real devices. For example:

3%	10%	3%
10%	48%	10%
3%	10%	3%

Table 1: Table of probabilities of a device to move to an adjacent cluster or to remain in the same.

- High mobility devices$\longrightarrow$ smartphones, wearables, self-driving cars.
- Low mobility devices $\longrightarrow$ IoT devices that can be moved, laptops, tablets.
- Fixed devices $\longrightarrow$ fixed IoT sensors, security cameras.

From a mathematical perspective this translates to having a 2-dimensional distribution in a 3x3 matrix of the probability of the point to remain in the current cluster (the central element of the matrix) or move to an adjacent cluster (the external elements of the matrix). See Table 1 for an example.

The aim of the simulator will be to evaluate the algorithms on two benchmark tasks:

- object recognition (CIFAR-10) with a ResNet18, as in [1];
- next-word prediction with a LSTM, as in [7].

Both IID and non-IID data distributions among clients will be tested. *Large mini-batch SGD*(§2.1.A), *local SGD* or *post-local SGD*(§2.1.B) will be used by the clients to perform their local updates.

The comparison of the algorithm behaviour will be based on two parameters:

- number of **communication rounds** needed to converge;
- **latency** analysis of upload and download total times given a wireless communication channel.

2 Background

2.1 Machine Learning

The objective of Machine Learning, in the setting of a *supervised learning* task, is to find the parameters of a model that satisfies the an finite-sum objective function of the form

$$\min_{w\in\mathbb{R}^d} f(w) \quad \text{where} \quad f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w) \tag{1}$$

in which $f_i(w) = l(x_i, y_i; w)$ represents the error of the prediction on input data (x_i, y_i) of the model with parameters w.

The model parameters (also called *weights*) are learnt from a training set by performing iteratively a two-step process called *gradient descent*.
In the first step, *forward propagation*, each training sample is input into the model and the output is computed by using the current model weights. The model output is then compared with the ground truth label of the training datapoint. The distance between these two values represents the loss (i.e. error) of the current set of weights[1].
The second step, *backpropagation*, consists in calculating $\nabla l(x, y, w_t)$, the gradient of the

[1]There are various ways of computing the loss, depending on the task that we want the model to perform. Generally, *euclidean distance (L2-norm)* or *mean squared error (MSE)* are used to express the loss in regression problems, *categorical cross-entropy* for classification.

loss on the datapoint x with ground truth label y with respect to the current set of parameters w_t, and applying the following update to the weights

$$w_{t+1} = w_t - \eta_t \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \nabla l(x, y, w_t) \qquad (2)$$

where $\mathcal{D}$ is the training set.
Since our aim is to reduce the loss, the formula above shows that, at each backpropagation step, the current weights are modified in the opposite direction of the average gradient information and the amount of variation is regulated by the *learning rate* η.[2]

Minibatch SGD. One of the most common variations of gradient descent optimization used in machine learning is *minibatch stochastic gradient descent* (minibatch SGD, also known simply as SGD). In minibatch SGD, the gradient descent step described in equation **XXX** is computed on a small batch of datapoints randomly selected from the training set (rather than on the entire dataset). To reuse datapoints in multiple minibatches, minibatch SGD goes over the entire training set more than once; each complete pass over the training set is referred as one *epoch*
This method reduces significantly the computational cost of a backpropagation step because less derivatives have to be computed, and improves generalization accuracy.
It is necessary to note that the performance of this optimization technique is controlled by the newly introduced hyperparameters minibatch size and number of epochs.

2.1.A Large Minibatch SGD

Researchers in [3] have studied how to increase efficiency of minibatch SGD for distributed synchronized learning applications. When the learning task is split among multiple learners, using small batch sizes increases the number of batches per epoch and therefore the number of times that the learners have to communicate with each other. The results in [3] prove that it is possible to increase the batch size up to 8192 and that this reduces drastically the training time required. In their work, they present some hyperparameter tuning techniques that are necessary to achieve said results:

- **Learning rate scaling:** when the minibatch size is multiplied by k, multiply the learning rate by k.
- **Learning rate warmup:** following the expression in **XXX** we can compare the SGD weight vector update step for a small batch size (n) and a larger one (kn). In the first case, after k iterations the resulting weights are given by

  $$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_|} \nabla l(x, w_{t+j}) \qquad (3)$$

  While for a single SGD step with batch size kn we have

  $$w_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_|} \nabla l(x, w_t) \qquad (4)$$

  By setting the learning rate $\hat{\eta} = k\eta$ (following the scaling technique outlined in the point above) we see that the expressions **XXX** and **XXX** yield similar parameter vectors, and therefore a similar learning curve, only if we could assume that $\nabla l(x, w_{t+j}) \approx \nabla l(x, w_t)$. This assumption generally holds in most cases but when the batch size is scaled to very large values and at the beginning

[2] In the expression provided η has a subscript t because the learning rate can change over iterations due to other hyperparameters such as *learning rate decay*.

of training when the network is changing rapidly.
To solve the latter problem, it is suggested to use a warmup strategy to adapt the learning rate in the first epochs of training. Constant warmup, used in [4] can be performed by setting a low constant learning rate for the first epochs[3] and then setting it to $k\eta$. This method, however, yields a similar training curve only for low ranges of batch size scaling factors k.
For large k, it is advised to use a gradual warmup, where the learning rate is iteratively incremented from a low value to the desired $\hat{\eta} = k\eta$ during the initial epochs.
After warmup the learning process goes back to the original learning rate schedule.

2.1.B Local SGD

Since using large minibatches in SGD can cause convergence and generalization problems, researchers in [5] have demonstrated that participants in a distributed learning system can achieve better accuracy and reduce communication rounds by performing locally several parameter updates with small batch sizes sequentially, before sharing their update.

They also discuss two variations of local SGD:

- **Post-Local SGD:** Since in the initial phase of training weights change rapidly, gradient updates need to be communicated as soon as they are computed to achieve better convergence. In post-local SGD, the local SGD algorithm is only started after a few rounds of small minibatch SGD. This lets us take advantage of warmup strategies, as those described in §2.1.A.

- **Hierarchical Local SGD:** in a hierarchically organized distributed learning system local SGD is applied to different layers of the architecture using different numbers of minibatch SGD iterations.

2.2 Federated Learning

Modern applications make use of models trained on large-scale amount of data. However, when this data is generated by edge devices (e.g. text that is typed on a smartphone keyboard, heartbeat information recorded by a wearable device), it is often impossible or inefficient to send it to a central server to be used in the model fitting, due to privacy concerns and bandwidth constraints.
For this reason, Google researchers have developed *Federated Learning* (FL) [7], an algorithm to perform privacy-preserving distributed Machine Learning on the edge.
Federated optimization (i.e. optimization in federated learning) presents several key properties that make it differ from distributed optimization:

- The training data is generated by a particular user acting on a certain device and hence the data of each device would be **non-IID**.

- Some users may use thier device more with respect to others, making the data generated **unbalanced**.

- Since FL is thought to be used on all kinds of edge devices, it is expected that the the number of devices taking part in a FL system to be much larger than the number of data samples per user. The system can be said to be **massively distributed**.

[3] In [3] the warmup phase encompasses the first 5 epochs of training.

- Edge devices have **limited communication**, as they can go offline and they use crowded or expensive connections.

In FL the learning task is handled by a loose federation of clients, coordinated by a central base station. The algorithm that controls FL is called *Federated Averaging* (FedAvg), and is described below and defined in algorithm **XXX**.
The initial parameter vector is generated by the server. Then the algorithm proceeds in rounds all composed of the same steps:

1. A subset of the clients is chosen at random[4] to participate in the current learning round.
2. The latest parameter vector is distributed by the server to the selected clients.
3. Each client updates the model parameters by performing minibatch SGD on their local dataset.
4. The new weights are sent by each client to the server.
5. The server aggregates all the received updates by taking a weighted average

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^{k} \tag{5}$$

Since the training data is never shared by the clients, it is intuitive to understand how this process preserves privacy.

Algorithm 1: `FederatedAveraging`
The K clients are indexed by k; each client uses minibatch size B, number of epochs E, and learning rate η.

```
— INSERT FEDAVG ALGORITHM —
```

2.2.A Client Selection

In [7] the clients are selected randomly to participate in each federated averaging step. This causes the time necessary for each round to be strongly dependent to the random selection. To increase the efficiency of this process, researchers in [8] have added a `Client Selection` step at the beginning of each round before the server sends out the global model.

1. A fraction of clients is selected to participate in the `Client Selection` step.
2. Clients reply to the server with information about the time needed for the update and for the upload of the model.
3. The server chooses the clients to keep by constraining the maximum length of a round to a fixed value, while trying to maximize the number of clients chosen.

2.2.B Secure Aggregation

Secure aggregation[2] is a cryptographic technique that allows the clients to submit their updates encrypted to an *aggregation server*, which is able to aggregate them without decrypting them. The combined update is then sent to the central server by the aggregation server
This allows for increased security because theoretically it is possible to derive the training data that generated an update if the update vector can be seen unencrypted.

[4]A more efficient way to select clients is presented in [8] and discussed in §2.2.A

2.3 Hierarchical Federated Learning

Federated learning was envisioned to be used on large scale networks of devices. When these clients are in the range of millions, having a cloud-based system with only one aggregation server to which all clients send their updates represents a performance bottleneck. In *Hierarchical Federated Learning* [1, 6], we introduce an intermediate layer of small cell base stations (SBSs). Clients are grouped geographically around the closest SBS, to which they send their updates[5]. Periodically SBSs communicate their current model to a macro cell base station (MBS), which averages them and redistributes a new global model.
This process, together with gradient sparsification[9] reduces considerably the communication latency since it allows for carrier reuse in different clusters, without sacrificing model accuracy.

The algorithm described in [1] is presented below.

```
Algorithm 2: Hierarchical Federated Averaging

— INSERT HFL ALGORITHM —
```

3 Implementation Plan

TODO

4 Evaluation Plan

TODO

5 Ethical, Legal, and Safety Plan

TODO

[5]the learning process between clients within a cluster and their SBS works as described in §2.2

References

[1] Mehdi Salehi Heydar Abad et al. *Hierarchical Federated Learning Across Heterogeneous Cellular Networks*. 2019. arXiv: `1909.02362 [cs.LG]`.

[2] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: `10.1145/3133956.3133982`. URL: `https://doi.org/10.1145/3133956.3133982`.

[3] Priya Goyal et al. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2017. arXiv: `1706.02677 [cs.CV]`.

[4] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[5] Tao Lin et al. *Don't Use Large Mini-Batches, Use Local SGD*. 2018. arXiv: `1808.07217 [cs.LG]`.

[6] Lumin Liu et al. "Edge-Assisted Hierarchical Federated Learning with Non-IID Data". In: *CoRR* abs/1905.06641 (2019). arXiv: `1905.06641`. URL: `http://arxiv.org/abs/1905.06641`.

[7] H. Brendan McMahan et al. "Federated Learning of Deep Networks using Model Averaging". In: *CoRR* abs/1602.05629 (2016). arXiv: `1602.05629`. URL: `http://arxiv.org/abs/1602.05629`.

[8] Takayuki Nishio and Ryo Yonetani. "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)* (May 2019). DOI: `10.1109/icc.2019.8761315`. URL: `http://dx.doi.org/10.1109/ICC.2019.8761315`.

[9] Shaohuai Shi et al. "A Distributed Synchronous SGD Algorithm with Global Top-k Sparsification for Low Bandwidth Networks". In: *CoRR* abs/1901.04359 (2019). arXiv: `1901.04359`. URL: `http://arxiv.org/abs/1901.04359`.