The Thesis Committee for Jiahan Liu

certifies that this is the approved version of the following thesis:

Selective Federated Learning for non-IID Training and Test Data

Supervising Committee:

Christine Julien, Supervisor

Jonathan Valvano

Haris Vikalo

Selective Federated Learning for non-IID Training and Test Data

by

Jiahan Liu,

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

The University of Texas at Austin

December 2019

Dedication

To the Zhou Family, Tyler Simmons, Tyler Chen, and my parents, Weize Liu, Mingzhi Liu

Acknowledgments

I would like to thank Christine Julien for her advice and guidance in support my research. Thank you Jonathan Valvano and Haris Vikalo for reading this thesis. The Mobile and Pervasive Computing Lab, Chenguang Liu, Sangsu Lee, Tomasz Kalbarczyk, Jie Hua, Grace Lee, and Yosef Saputra have all provided insightful discussion and reading group contributions.

To my family, the Zhou family, Tyler Chen, and Tyler Simmons who have always been there, I couldn't have done it without you all.

Finally, I would also like to thank Josie Mallery, Yale Patt, and all my teachers for all the kindness and inspiration they have shown me throughout my education.

Jiahan Liu

The University of Texas at Austin

December 2019

Selective Federated Learning for non-IID Training and Test Data

Jiahan Liu, MSE
The University of Texas at Austin, 2019

Supervisor: Christine Julien

Federated learning trains a global model using data distributed across local nodes, and differs from centralized machine learning by moving the computation to the data in order to achieve two main benefits. It improves the privacy of user data and solves the issue of data ownership which potentially unlocks data for training that otherwise would not available. Additionally, as a form of distributed machine learning, federated learning addresses computational, storage, and communication challenges associated with training with a large distributed dataset. While previous research address the effect of non-IID distributed data with a global test set, this research is the first, to our knowledge, to improve the performance of federated learning if the test set was distributed non-IID as well. Afterall, if the data collected at each node was unbalanced then it is reasonable to expect that the test set and inference data to be be unbalanced as well. First, we prove that a converged feder-

ated model may converge to weights which do not provide the optimal local loss for an arbitrary chosen number of training samples on each node. The premise is that if the data on each node is systematically different, then the local loss may provide a better proxy for the node's performance. Then the thesis discusses the experiments that were conducted to examine the effects of model complexity, percentage of unbalanced data, and the current modes of model aggregation on model accuracy. A deployed federated learning library for multiple devices, Jetson Nano, Raspberry Pi, Macbook Pro, and Linux server was built for the purpose of this experiments and provide references for hardware requirements. From the hardware experiments, we observe that while complex models are highly robust to non-IID data, they are not suitable for current edge devices. Finally, we propose selective federated learning algorithm which greatly allows simple models that fit on edge devices to be highly robust to extremely non-IID data.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The current paradigm for machine learning trains model(s) using data stored at a centralized location. The data, however, is often collected on personalized devices, or edge devices. This introduces the challenges of data privacy, ownership, communication, and computation. Federated learning is a form of distributed machine learning first proposed by (Konecny et al., 2015)[4] which seeks to tackle these challenges. Federated learning brings the training computation to the data instead of bringing the data to the computation and averages the model weights to create an aggregate model.

Federated learning works by training model weights locally on each node then aggregating the weights on a centralized server before updating all the local models with the new weights for one round of communication. This continues for multiple rounds of communication until model convergence. This addresses in data ownership because the data never has to leave the device. While a backdoor to federated learning has been proposed by (Bagdasaryan et al.) [1], the paper exposes how the performance of the federated model can be attacked, and not how the

privacy of the model can be compromised. There are currently no known methods to reverse engineering the data from the weights; hence federated learning is a potential paradigm which preserves data privacy. Federated learning is better than centralized learning in terms of communication bandwidth in the cases where the memory size of the weights is less than the memory size of the data. While the memory size of weights is fixed as a function of a model architecture, the amount of data collected by a node devices can vary greatly from application and application. While it's true that models can have millions or billions of parameters, lidar data for the city of Dublin is 0.5 terabytes [2]. Finally, as personalized and edge devices are getting more powerful and numerous, they can be leverage to perform the computations required in federated learning.

An example application where federated learning has been successfully applied is Google's Gboard which uses federated learning to improve query suggestions by training on personal phone data without collecting the data itself in a centralized location [13]. Federated learning, however, still faces both systems and theoretical challenges. The systems challenges are currently bottle-necked by communication efficiency and has been researched by (Konecny et al., 2017)[3]. These communication challenges stem from the fact that the training time is dominated by the time between communication rounds rather than the computation time because of the participation of the devices needs to be synchronized. Synchronized federated learning has produced the best results up to date. The theoretical challenges involve the performance of the federated model. While (Xi et al.) have proven the convergence of FedAvg on non-IID Data [5], (Smith et al.) has shown that federated models may under-perform models trained only local data to each node [7]. In other words, while the convergence of the loss of a global model loss is robust to non-IID Data in federated learning, previous experiments by Smith have shown that federated learning may not be robustness to non-IID data distributions in practice.

1.2 Contributions

To the best of our knowledge, we are the first to conduct experiments for federated learning in which both the training and test sets are distributed non-IID for federated learning. This is a reasonable assumption because the nodes that collect non-IID training data should keep encountering non-iid distributed data so the test set should unbalanced as well. We then prove that it is possible for the global objective to converge to an acceptable loss Q, but for the local loss on a node N to be an unacceptable magnitude times larger, regardless of the number of data points on node N. The loss is empirically chosen to be a proxy for model accuracy, and for non-IID data, the local loss is arguably a better proxy than the global loss for model accuracy for each node. We then conduct and show the results of experiments federated and a new algorithm, selective federated training, to observe the effects of non-IID data on model accuracy and conduct the same experiments for centralized and local learning as a baseline. In these experiments, we vary both percentage of unbalanced data in both the training and test set, and we also vary model complexity. Finally, we propose a selective grouping step to create a selective federated learning algorithm to improve the validation set accuracy of the federated model during training time which in our experiments result in improved test set accuracy. We then deploy selective federated learning as a preliminary feasibility test and find that while complex models are robust to non-IID data for ordinary federated learning, they do not fit on current edge devices. We provide our federated learning library in the Mobile and Pervasive Computing repository for future research

1.3 Technical Problem Statement

The weights of a centralized mode are trained from a set of n training examples $\{(x_i, y_i)|1 \leqslant i \leqslant n\}$ by applying a form of gradient descent to minimize $\frac{1}{n}\sum_{i=1}^{n} f_i(w)$

where $f_i(w) = l(x_i, y_i; w)$ is the loss function on each training example. We denote this optimization the global objective.

Definition 1.3.1. Global Objective

$$\min_{w \in \mathbb{R}^d} F(w) \quad s.t. \quad F(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{1.1}$$

In the federated learning setting, the training data is partitioned over K nodes indexed by k and of size n_k into partitions P_k. Grouping the training data by node, we can rewrite the global objective:

$$F(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad s.t. \quad F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w) \tag{1.2}$$

The rate of convergence of $F(w)$ in a federated model with $FedAvg$ as the aggregation step is proportional to Γ, the degree of non-iid across all the nodes [5].

Definition 1.3.2. Degree of non-iid [5]. Let F^* and F_k^* be the the minimum values of $F(w)$ and $F_k(w)$, respectively. The degree of non-iid, Γ, is defined as:

$$\Gamma = F^* - \sum_{k=1}^{N} p_k F_k^* \tag{1.3}$$

The performance premise behind using federated learning is that a model with weights w_G trained over the global set of data will perform better than a model with weights w_k trained over data from a partition P_k. Empirical results from (Smith et al.) in the team's federated multi-task learning paper [7] provides counterexamples to this premise and shows that for support vector machine models trained on the the google glass, human activity recognition, and vehicle sensor datasets, the global federated model performs worse in terms of accuracy than the local model. In table 1.1 is the average prediction error from the paper:

Model	Human Activity	Google Glass	Vehicle Sensor
Global	2.23 (0.30)	5.34 (0.26)	13.4 (0.26)
Local	1.34 (0.21)	4.92 (0.26)	7.81 (0.13)

Table 1.1: Average prediction error from multi-task federated learning paper

At first glance, these results conflict with the results published by (Konecny et al., 2014) [6]. Upon closer inspect, we see that Konecny's results use a global test set and more complex models than Smith's experiments. Thus we would like to observe the effects of model complexity and unbalanced training as well as test sets in federated learning. Then we propose selective federated learning to improve the performance of federated learning on non-IID data.

Chapter 2

Literature Review

2.1 Federated Learning of Deep Networks using Model Averaging [6]

This paper covers the FederatedAveraging algorithm which uses the weighted average of the model parameters as the aggregation step of federated learning. FederatedAveraging is a concrete way to implement federated learning. It uses the weighted average for the aggregation step which is the key step in federated learning. Below is the algorithm presented by the paper.

Algorithm 1: FederatedAveraging

Server executes:

initialize w_0

for *each round t = 1,2,...* **do**

  S_t = (random set of $max(C \cdot K, 1)$clients

  for *each client* $k \in S_t$ *in parallel* **do**

    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

  end

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

end

ClientUpdate(k, w)**:** *//Executed on client k*

for *each local epoch i from 1 to E* **do**

  batches $\leftarrow$ (data P_k split into batches of size B)

  for *batch b in batches* **do**

    $w \leftarrow w - \eta \nabla l(w; b)$

  end

end

return w to server

Figure 2.1: Federated averaging

This paper shows empirically that FederatedAveraging is robust to non-iid data by showing in experiments which an data was distributed non-IID and then tested on a single test set that spans examples from each node. The dataset that is uses are MNIST and a non-IID modified version of MNIST and the IID and non-IID dataset built from *The Complete Works of William Shakespeare* This experiments in this thesis differs from the ones conducted in Federated Learning of Deep Neural Networks using Model Averaging because it creates test sets which are unbalanced as well and has greater variation in model complexity. It is reasonable to expected the distribution of the test set to be similar to the distribution of the training set.

2.2 Federated Learning: Strategies for Improving Communication Efficiency [3]

This paper examines the engineering challenges associated with deploying a federated learning algorithm to a large number of devices. In practice, the network could be slow and the availability of the clients may be unreliable. The paper proposes two methods, *structured updates* which systematically chooses a subset of devices to aggregate each round, and *sketched updates*, which uses quantization, random rotations, and subsampling of the local dataset before sending it to the server. Both of these methods improve the communication cost of federated training by two orders of magnitude; thereby improving the speed of federated learning.

Our paper focuses on the performance aspect of federated learning and we build a simple deployed model as a preliminary test for feasibility. We did not incorporate *structured updates* and *sketched updates* into our tests but our algorithm could incorporate the two methods as well.

2.3 Federated Multi-Task Learning [7]

In the Federated Multi-Task Learning paper, (Smith et al.) proposed multi-task learning for federated learning on non-iid distributed datasets. In multi-task federated learning, the data from the other nodes are incorporate indirectly as part of the loss function instead of directly by averaging the weights. Her team uses support vector machines to classify three unbalanced datasets, the Google Glass dataset, the Human Activity Recognition dataset, and the Vehicle Sensor dataset. In each dataset, the local model performs better than the global federated model and the Multi-Task learning model performs better than the local model.

This paper is the inspiration for this thesis. It indirectly suggests that the test set should be distributed the same as the training set which the FederatedAv-

eraging did not do. This thesis builds upon Smith's research by exploring another method which train a model to perform better on non-IID test sets. This thesis also explores more model complexity by using multi-layer neural networks with convolutional networks rather than just support vector machines. One key takeaway from the experiments performed in our paper is that exploring performance on different architectures is important as the results of this thesis show that complex models are more robust to non-IID distributed datasets.

2.4 On the Converge of FedAvg on Non-IID Data [5]

This paper was selected from all the proofs of the convergence of federated learning [16, 15, 14, 12, 11, 10, 8, 9] because it was the only work which allows the data to be both non-iid and partial device participation, critical characteristics of the federated setting.

The key concept of this paper shows that the the global loss converges under FederatedAveraging converges to a global minimum with the dominant variable being the number of iterations. Namely, FederatedAveraging converges at rate $O(\frac{1}{T}$ where T is the number of iterations.

This thesis builds upon the proof presented by Li et al. by showing that in the FederatedAveraging algorithm as even as the global loss converges to a minimum, the local loss at each node may not be the minimum.

Chapter 3

Implementation Details

3.1 Requirements

Our experiments contain engineering as well as theoretical requirements.

In the real world, federated learning must be deployable to edge devices of varying computation and storage requirements. In our implementation, we require that the selective federated learning algorithm be deplorably on a CPU-only device as well as devices with GPU. To meet this requirement, we deployed the selective federated learning algorithm onto a Raspberry Pi 4, Nvidia Jetson Nano, and Macbook Pro with a linux server performing the aggregation step of federated learning. We require our weakest device, the Raspberry Pi 4 to finish one round of training within eight hours which correspond to a common use case of cellphones charging over night. We can only train during the time that a personal device is plugged in to prevent from draining the user's battery which would severely impact the usability of any application which implements selective federated learning.

To ensure our experiments are generalizable, the MNIIST dataset is randomly shuffled before any modifications to create an non-IID set by seeding pytorch with the random library from python. We use Kaiming He initialization with ReLu non-

linearity for the weights. At the time of this thesis, ReLu is a popular activation function and Kaiming He is a popular initialization so this is representative of how many machine learning models are configured.

We conduct our experiment with one hundred percent device participation because that was the standard set in the FederatedAveraging paper [6]. This standard is reasonable because methods used to ameliorate non-perfect device participation can also be applied to selective federated learning in the same way that it's applied to FederatedAveraging. The results of this experiment assume ideal device participation.

As with the experiments in FederatedAveraging [6], the majority experiments themselves were conducted on centralized devices with GPUs. To ensure that the results in this thesis are generalize to a real deployment, we conducted a small subset of the experiments on a test bed with a Nvidia Jetson Nano, Raspberry Pi 4, and Macbook Pro as the node devices. The deployed software design used the same code for the federated learning and added networking to service the device to server communication rounds.

3.2 Software Library and Design

All of the code can be found in the Mobile and Pervasive Computing repository on GitHub owned by Christine Julien's research group. In figure 3.1 is the software design diagram.

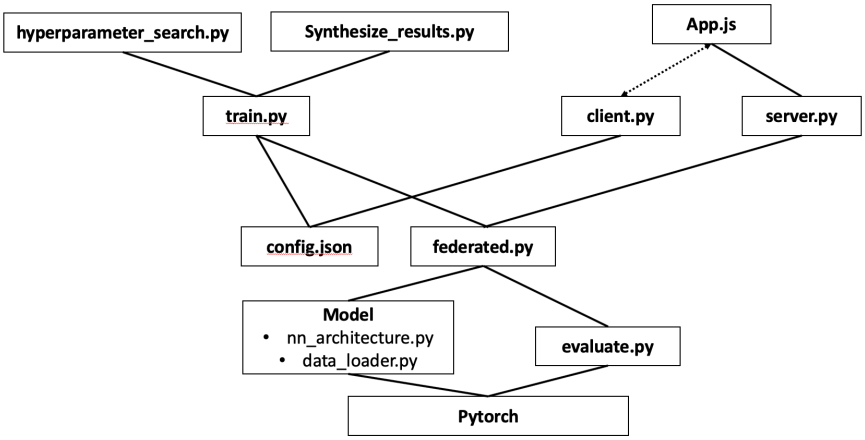

Figure 3.1: Software Design

The software library is built using the Pytorch. Federated.py contains all the code to perform a single round of federated learning in both the experiments as well as the real test bed. It allows creation of a local and global class to represent or be deployed on local nodes and the aggregating server respectively. It allows the data loaders and model architecture to be configurable and passed in as parameters to allow for flexibility in future experiments.

All configurations for the hyperparameters, results file, networking are all refactored out and put inside a json formatted configuration file. The programs are also setup to use both CPU and GPU to do the training. This is configurable through the program args.

The neural network architectures and data loaders used for this experiment are in the model directory. The data loader uses Pytorch's data loaders class to create modified versions of the MNIST dataset to create non-IID versions of the test, validation, and training sets. The code here uses python closures to be reusuable for future experiments.

The bulk of the experiments was conducted with the train.py code which

abstracts out the networking and communication time in federated learning. Instead model parameters are just passed in memory and different objects from the Federated.py classes are created to represent each local node and the aggregation server.

To perform the hyperparameter search, the hyperparameter_search.py program uses random search to train models using different learning rates and batch sizes and each configuration is averaged over N runs where N is configurable. A separate validation set is create by the data loader to perform the hyperparameter optimizations. Random search is used because it is possible that one of the hyperparamaters is more important than the other so we want to be able to sample more fine intervals than grid search. Each training session runs until the validation accuracy not improved for 20 epochs. The hyperparameter search provides the 20 highest validation accuracy and then the researcher must manually and choose the hyperparameter configuration and entered into the configuration file. The parameters themselves are also optimized using ADAM which is a one of the top momentum based optimizers being used at the time of this writing and works well on centralized MNIST.

To generate the results, we must vary both the percentage of unbalanced in the non-IID test, validation, and test set and vary the complexity of the neural network model. The results.py program automatically generates results for the combination of percentage of unbalanced data and complexity of the neural network model for local, centralized, federated, and selective federated learning modes. Because of the limited number of GPUs and GPU memory on devices and the user may pick which results to generate through the program args.

In the deployed version of the experiments each client runs client.py which will perform the local training and send the model weights to a central server. The model parameters themselves are saved inside a file and sent to the central server

using a HTTP request. The clients periodically poll the server until all the federated averaging has been done. The server listens for all HTTP request and saves each file. Once it has recieved all the HTTP request and files, then it calls server.py to perform federated averaging on all the local parameters. Once the federated averaging is finished, then it sends the updated parameters to all the clients. The implemented networking for the federated learning follows is shown in figure 3.2.

In our experiments all the nodes have roughly the same number of data points. The FederatedAveraging paper also conducted their non-IID experiments using the same amount of data points. We also train the dataset once per round, or perform one epoch of training per round of communication. The FederatedAveraging paper was able to get high accuracy results with one epoch of training per round of communication.

Algorithm 2: Deployed Federated Learning

Server Executes

for *each round t = 1,2,...* **do**

waits for each client to upload their model parameters through a HTTP request

saves the model parameter files as each local node uploads their model parameters

if *all the clients have uploaded their model parameters* **then**

take the average of all the weights to create the new weights

wait for all the clients to request the new model parameters

else

end

Client Executes, done in parallel on local nodes

for *each round t = 1,2,...* **do**

for *Total Datasize / Batch Size* **do**

perform ADAM to update the local model parameters

end

send the model parameter to the aggregating server through a HTTP request

poll the aggregating server for when the new parameters are ready

download the new model parameters

end

Figure 3.2: Federated averaging

3.3 Federated Learning

3.3.1 Dataset Divison

To create datasets for federated learning we take disjoint subsets of the MNIST dataset. We choose the MNIST dataset because it is a dataset which very accurate

models have already been found. We would like to perform experiments on datasets in which the centralized version has been solved and models have high accuracy so that we can have an ideal baseline. By using a high accuracy baseline to compare we would be able to more strongly conclude that the error in our predictions is from our experiment rather than the fact that an model with high accuracy is not known yet. In other words, it is not ambiguous whether or not the error can be attributed to non-IID distributions in federated learning.

Our experiments differ from those in the Federated Averaging paper [6] because we also make the test set non-IID. This makes sense because if a node is encountering a biased selection of data during training time, then it should encounter the same distribution data during test and inference time because the partitions between train, test, and inference data should be able to be arbitrary. We also allow the percentage of unbalanced data to be configurable. In our experiments we use three nodes for federated averaging and nine nodes for selective federated averaging. We are interested in the behavior of the federated averaging itself and do not want an insufficient number of data at each node to be a factor. For example, to create a thirty percent unbalanced dataset for each node. We take seventy percent of the data and evenly and random distributed among the nodes. Then we take the data with labels 0-3 and place it in the first node's dataset, 4-6 and place it in the second node's dataset, and 7-9 and place it in the third nodes dataset. For an unbalanced percentage of thirty percent in the case selective federated averaging, we take seventy percent of the data and distribute it evenly and randomly among nine nodes. Then we take the data with labels 0-3 and evenly and randomly place it in the nodes 1-3's datasets, labels 4-6 and place it in nodes 4-6's datasets, and labels 7-9 and place it in nodes 7-9's dataset.

3.3.2 Model Complexity

We conduct our experiments on three models of different complexity. As neural networks get larger they need more data to train. We would like to see the effects of this on federated learning.

Model Complexity 1

Fully Connected Neural Network with one hidden layer composed of thirty neurons. Each neuron uses the ReLu activation function. A output is generated using a log softmax.

Model Complexity 2

A convoluted neural network with two 5x5 convolutional layers. Both convolutional layers have a max pooling layer. The second convolutional layer has a drop out of 0.5. The convolutional layers are followed by two fully connected layers. The first has 320 hidden neurons and the second has 50 hidden neurons. The first hidden layer has a drop out of 0.5. The output is generated using log softmax.

Model Complexity 3

A convoluted neural network with the three 3x3x2 convolutional layers. Each convolutional layer is followed by a batch norm layer, a ReLu layer. The second and third layers have a max pooling layer. The convolutional layers are followed by three fully connect layers with 128, 64, and 10 hidden neurons. Each fully connected layer is followed by a batch norm layer and has a drop out of 0.5. The output is generated using log softmax.

3.3.3 Initialization, Activation Function and Hyper Parameter Search

We used ReLu activation function and Kaiming He initialization modeled by ReLu non-linearity. At the time of the writing, squeaky ReLu has been shown to be decisively better or worse. For networks with drop out, we used a fixed 0.5 drop out. We train for N epochs where for epochs N+1 to N+20 there was no improvement in the validation accuracy. Then we performed random search for the batch size and learning rate. We constrained the batch size to be a power of two because GPU memory is configured in powers of two.

Chapter 4

Hardware and Time Calculations

We also provide calculations on the number of multiply accumulate operations (MACC) that federated learning takes between one round of communication. For federated learning we assume that the personal devices will be charging and able to calculate the new weights between one round of communication over the span of 8 hours. We use the linux time command to get the real, user, and system time for each round of training for the Raspberry Pi 4 which is the slowest of our three devices but still has a NEON instructions for MACCs. The operations are float point 64 but can be floating point 32 as floating point 32 is also support by Pytorch. Here we ignore the back propagation computation in the activation layer since it's dominated by the back propagation of the loss through the weights. When we partition the 50,000 training set into three nodes and then remove 10,000 for a shared validation set we have 13333 images for the training set.

4.1 Number of Multiply Adds

4.1.1 Model 1 MACCs

In model 1, each round of communication 23821N MACC where N is the number of training data on the node. For 13333 images, we would have a little over 300,000,000 MACs. On the Raspberry Pi 4, we timed the average training time per round to complete training (21 rounds). For selective federated training we would need double the amount of time. The calculations for the MACC are shown in table 4.1 and the software timings are shown in table 4.2.

Layer	Activation Shape	Number of Parameters	Number of MACC
Input	784	0	0
FC1	(30,1)	23521	23521
Softmax	(10, 1)	300	300

Table 4.1: Model 1 MACCs

Type	Time (Seconds)
Real	57.90
User	83.42
System	40.56

Table 4.2: Software time for Model 1

4.1.2 Model 2 MACCs

In model 2, each round of communication 21852 MACC where N is the number of training data on the node. For 13333 images, we would have a little over 291,000,000 MACs. On the Raspberry Pi 4, we time the average training time per round to complete training (83 rounds). For selective federated training we would need double the amount of time. The calculations for the MACC are shown in table 4.3 and the software timings are shown in table 4.4.

Layer	Activation Shape	Number of Parameters	Number of MACC
Input	(28,28)	0	0
Conv1	(5,5)	26	250
Conv2	(5,5)	26	5000
FC1	(320, 50)	16001	16001
FC2	(50 , 10)	501	501
Softmax	(10, 1)	100	100

Table 4.3: Model 1 MACCs

Type	Time (Seconds)
Real	169.98
User	503.14
System	133.86

Table 4.4: Software time for Model 1

4.1.3 Model 3 MACCs

In model 3, each round of communication 904188 MACC where N is the number of training data on the node. For 13333 images, we would have a little over 12,000,000,000 MACs. The Raspberry Pi ran out of memory as this model needed 1,003,520,000 bytes of memory. For selective federated training we would need double the amount of time. The calculations for the MACC are shown in table 4.5 and the Raspberry Pi 4 does not support model 3..

Layer	Activation Shape	Number of Parameters	Number of MACC
Input	(28,28)	0	0
Conv1	(3,3)	10	288
Conv2	(3,3)	10	18432
Conv3	(3,3)	10	73728
FC1	(5280, 128)	802816	802816
FC2	(128 , 64)	8193	8193
FC2	(64 , 10)	641	641
Softmax	(10, 1)	100	100

Table 4.5: Model 1 MACCs

Chapter 5

Federated Convergence of Global vs Local Losses

For non-IID distributed dataset, the test set should also be distributed non-IID with the same amount of unbalanced data. With sufficient amount of data on each node, the local loss is a better proxy for local accuracy hence local performance than the global loss. In a federated model in which the global loss converges to a minimum, the local loss may not do the same. We build upon the proof of the convergence of FederatedAveraging by (Li et al.) [5] to show this.

In proof of convergence of $FedAvg$ [5], Li assumes the following assumptions:

Assumption 1. All the subproblems, $F_1, ..., F_N$ are L-smooth.

Assumption 2. All subproblems, $F_1, ..., F_N$ are all μ-strongly convex.

Assumption 3. The variance of stocastical gradients in each device is bounded by σ_k^2.

Assumption 4. That the expected squared norm of stochastic gradients is uniformly bounded by G^2.

Let T be the total number of steps, F^* be the minimum value of F, p_k be the probablity of choosing partition k uniformaly sampled without replacement, E

be the number of local iterations between two rounds of communication, $\kappa = \frac{L}{\mu}$, γ is $max\{8\kappa, E\}$ for learning rate chosen to be $n_t = \frac{2}{\mu(\gamma+t)}$. Li proved that:

$$\mathbf{E}[F(w_T)] - F^* \leqslant \frac{2\kappa}{\gamma + T}\left(\frac{B+C}{\mu} + 2L\|w_0 - w^*\|^2\right)$$

$$where$$

$$B = \sum_{k=1}^{N} p_k^2 \sigma_k^2 + 6L\Gamma + 8(E-1)^2 G^2$$

$$and$$

$$C = \frac{4}{K} E^2 G^2$$

For an arbitrary fixed model and dataset, the only variable in the root convergence rate is T, and as T approaches to infinity we have $E[F(w_T)] - F^* \leqslant 0$. Now, let's use Li's proof on the convergence of $FedAvg$ to prove theorem 1.

Theorem 1. **Worse Case Local Loss for Convergent Federated Model**

Arbitrarily pick any federated learning objective $\min_{w \in \mathbb{R}^d} F(w)$ satisfyingly assumptions 1-4 that uses $FedAvg$ as the aggregation step, any unacceptable loss constant multiplier C, and any number of training data for some node s. We want to show there exist a dataset P distributed into partitions P_k such that as $\mathbf{E}[F(w_T)]$ converges to F^*, $F_s(w_T)$ converges to $C \cdot F_s^*$. Let N be the number of partitions greater than 1.

Proof. Construct dataset P with the following properties. Without loss of generality, let node s to be the last of the nodes, node N. Populate nodes 1 to $N-1$ such that $F_k(w_T)$ converges to F_k^* for $1 \leqslant k \leqslant (N-1)$ or in other words $F(w)$ is a good model for nodes 1 to $N-1$. Choose the degree of non-iid, Γ, and and the probability of choosing from node N, p_N, such that $\frac{\Gamma}{p_N} = (C-1) \cdot F_N^*$. In other words, we can increase the amount of unacceptable loss on node N by increasing the degree of

non-iid or increasing the number of data points on the other nodes to decrease p_N.

To show that the loss on node N converges to $C \cdot F_s^*$, we must be able to find T for any $\epsilon > 0$ that satisfies $|F_N(w_T) - C \cdot F^*| \leqslant \epsilon$. The proof of the convergence of $FedAvg$ by Li et al. shows $\mathbf{E}[F(w_T)] - F^*$ converges at rate $O(\frac{1}{T})$, so we choose T such that $|\mathbf{E}[F(w_T)] - F^*| \leqslant \frac{\epsilon}{2}$ and $|F_k(w_T) - F_k^*| \leqslant \frac{\epsilon \cdot p_N}{2(N-1)}$ for $1 \leqslant k \leqslant (N-1)$.

$$
\begin{aligned}
\mathbf{E}[F(w_T)] - F^* &\leqslant \frac{\epsilon}{2} \\
\mathbf{E}[F(w_T)] &\leqslant F^* + \frac{\epsilon}{2} \\
\mathbf{E}[F(w_T)] &\leqslant \Gamma + \sum_{k=1}^{N} p_k F_k^* + \frac{\epsilon}{2} \\
\sum_{k=1}^{N} p_k F_k(w_T) &\leqslant \Gamma + \sum_{k=1}^{N} p_k F m_k^* + \frac{\epsilon}{2} \\
\sum_{k=1}^{N} p_k F_k(w_T) - \sum_{k=1}^{N} p_k F_k^* &\leqslant \Gamma + \frac{\epsilon}{2} \\
\sum_{k=1}^{N} (p_k (F_k(w_T) - F_k^*)) &\leqslant \Gamma + \frac{\epsilon}{2} \\
\sum_{k=1}^{N-1} (p_k (F_k(w_T) - F_k^*)) + p_N (F_N(w_T) - F_N^*) &\leqslant \Gamma + \frac{\epsilon}{2} \\
\sum_{k=1}^{N-1} \left(p_k \left(\frac{-\epsilon \cdot p_N}{2(N-1)} \right) \right) + p_N (F_N(w_T) - F_N^*) &\leqslant \Gamma + \frac{\epsilon}{2} \\
p_N (F_N(w_T) - F_N^*) &\leqslant \Gamma + \frac{\epsilon}{2} + \sum_{k=1}^{N-1} \left(p_k \left(\frac{\epsilon \cdot p_N}{2(N-1)} \right) \right) \\
F_N(w_T) - F_N^* &\leqslant \frac{\Gamma}{p_N} + \frac{\epsilon}{2} + \sum_{k=1}^{N-1} \left(p_k \left(\frac{\epsilon}{2(N-1)} \right) \right) \\
F_N(w_T) - F_N^* &\leqslant (C-1) F_N^* + \frac{\epsilon}{2} + \sum_{k=1}^{N-1} \left(p_k \left(\frac{\epsilon}{2(N-1)} \right) \right) \\
F_N(w_T) - C \cdot F^* &\leqslant \frac{\epsilon}{2} + \sum_{k=1}^{N-1} \left(p_k \left(\frac{\epsilon}{2(N-1)} \right) \right) \\
F_N(w_T) - C \cdot F^* &\leqslant \epsilon && p_k \leqslant 1
\end{aligned}
$$

The same can be done to show that $F_N(w_T) - C \cdot F^* \geqslant -\epsilon$, hence we have $|F_N(w_T) - C \cdot F^*| \leqslant \epsilon$. In other words the loss at Node N converges to $C \cdot F_s*^*$, a loss that is unacceptable magnitude times larger than the acceptable loss observed at the central server. $\square$

Chapter 6

Selective Federated Averaging

In federated learning, a single model is trained using the data distributed across m nodes to minimize a global loss which acts as a proxy for the accuracy for a global test set. The global test set is, however, not representative of the data which a node may encounter during inference time if the distribution of the dataset is unbalanced. It's reasonable for the data to be unbalanced during inference time if it is unbalanced during training time.

Selective federated averaging aims to partition the set of nodes N into k groups and then create k different models for each of the groups. The algorithm is described in figure 6.2 In selective federated averaging, each of the local node first trains their own local model. In the selective grouping process, the local models use the validation set $V = \{v_1, v_2, v_3...\}$ is used to generate a selection vector by the selection function in figure 6.1 such that $|S| = |V|$ and s_i is a 0 if the local incorrectly predicts the validation set output for input v_i and $s_i = 1$ if the local model correctly predicts the output for input v_i. For every node, a, b, a will be in the same group as b if the percentage of overlap in their selection vector meets a similarity threshold. The similarity threshold is a hyperparameter. Then a separate federated model is trained using data from all the nodes in each group.

Selection Function S(validation set V, model M)

$$s(v_k) = \begin{cases} s_k = 0 & \text{M's prediction for } v_k \text{ is incorrect} \\ s_k = 1 & \text{M's prediction for } v_k \text{ is correct} \end{cases}$$

Figure 6.1: Selection Function

Algorithm 3: Selective Federated Averaging

Train local models for all nodes

for *each node* $i \in |N|$ **do**
    request central server for validation set $V = \{v_1, v_2, v_3...\}$
    form selection vector $S(V, m_i)$
end

initialize set of ungrouped nodes U = N

initialize set of groups G = $\varnothing$

initialized added = False

for *each node* $n_i \in |U|$ **do**
    added = False
    if $G == \varnothing$ **then**
        create group g = $\{n_i\}$
        insert g into G
        continue
    end
    for *each group* $g_j \in G$ **do**
        if $(S_i \cdot S_k)/|V| <$ *similarity threshold for all nodes k in* g_j **then**
            insert node i into g_j
            added = True
        end
    end
    if *False == added* **then**
        create group g = {node i}
        insert g into G
    end
end

Train federated model for all group

Figure 6.2: Selective Federated Averagin

6.1 Time Complexity Analysis

Time complexity in Federated Learning is dominated by communication time [3]. For federated learning model that requires N rounds of communication, selective federated averaging requires 1 additional rounds to communicate the validation set and validation vector.

Chapter 7

Experimental Results

We experimented using modified partitions of MNIST to show the effects of unbalanced training data and model complexity on the test accuracy of an unbalanced test sets. We had three variables which we took combinations of to trained models and tested the test set accuracy. We also did random search to optimize for the hyperparameters. The first variable was the three models architectures of different complexity as discussed in section 3.3.2. The second variable was the percentage of the data which was unbalanced. The last variable was the mode of training, either local, federated, or selective federated. We also provide the centralized accuracy for each model architecture as a benchmark.

For federated learning, we obtained 150 different test accuracy for each combination. Because initialization was random, each accuracy is the average of three different training sessions. The hyperparameters were search 500 combinations of batch size and learning rate for a rough search, then 200 combinations of batch size and learning rate for the fine tune random search. Of the 150 test accuracy results, 50 of them were for each model. Of the 50 test accuracy for each model, each test was trained on a dataset that was distributed with different balance percentages starting from 0 percent balanced (completely non-iid, disjoint labels) to 100 per-

centage balanced with step size of 2. The same was done for selective federated learning. For local learning we generated 60 different test accuracy by using a step size of 5 for the balanced percentages. Below are the results organized by model type.

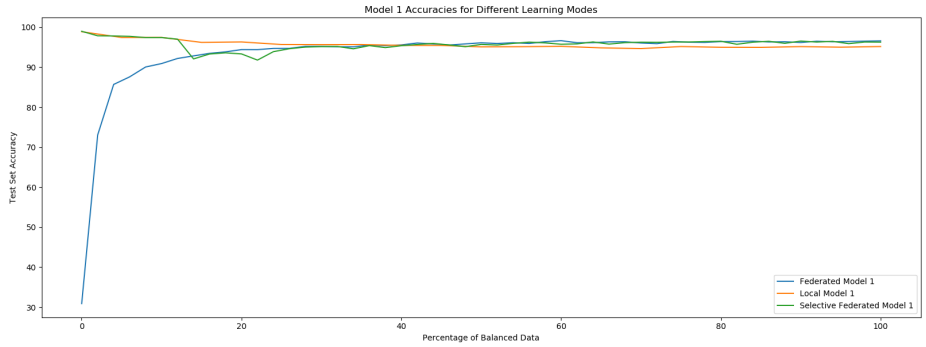

Figure 7.1: Model 1 Accuracy for Different Learning Modes

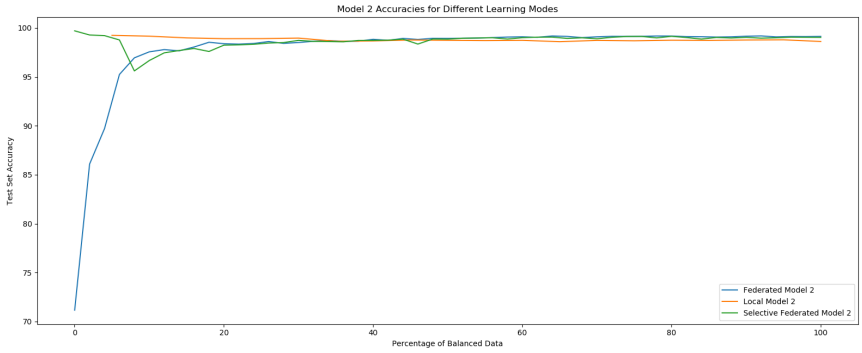

Figure 7.2: Model 2 Accuracy for Different Learning Modes

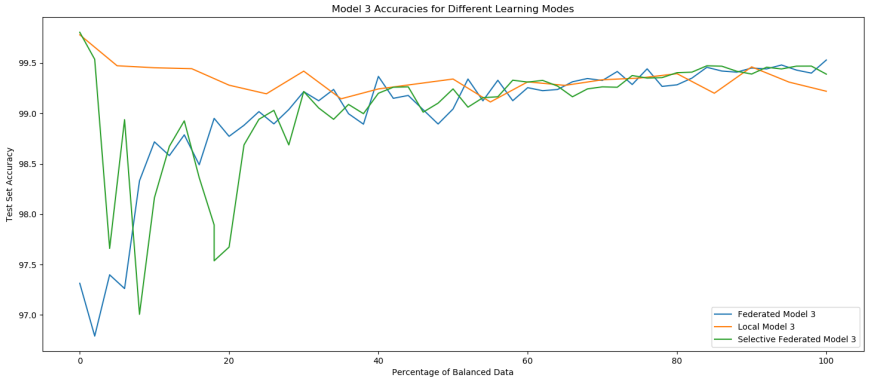

Figure 7.3: Model 3 Accuracy for Different Learning Modes

7.1 Centralized Learning Baseline

These are the test accuracy for each model in centralized training where all the data is at one location and there is no unbalance in the distribution of the training or test set.

Model	Accuracy (percent)
1	96.40
2	99.12
3	99.54

Table 7.1: Centralized Training Test Accuracies

7.2 Discussion

By using a deeper model with more convolution, batch norm layers, and wider fully connected layers in model 3, we can improve the robustness of the federated learning to non-iid data. However, the architecture in Model 3 does not even train on the Raspberry Pi 4 because of memory constraints. Federated learning must consider

the constraints of edge devices of limited memory, computation and power. By using selective federated learning, we are able to achieve great test accuracy by using simpler models that train in less time and fit the computational and memory requirements of a edge device while still being robust to non-iid data.

It is also notable that while model 2 and model 3 have similar performance in the centralized setting, 99.12 percent vs 99.54 percent, model 3 is much more robust to unbalanced data in federated training, 84.32 percent vs 97.31 percent for a 0 percent balanced data distribution.

Chapter 8

Conclusion

In this paper we examine theoretically and experimentally on the effects non-IID training data on Federated Learning with non-IID test sets. Previous papers [3] had examine the effects of non-IID data but with a unified test set and used models which were quite large for edge devices. In this paper we find that while deeper and wider models are more robust to non-IID data distributions in federated learning, the ones with performance comparable to local learning do not fit the memory and computational constraints of edge devices such as Raspberry Pi 4. We proposed selective federated learning which has the performance of local learning for extremely unbalanced data and has the performance for federated learning for almost balanced to balanced data.

8.1 Future Work

While our experiments show that selective federated learning is suitable for non-IID datasets, it may also be suitable for data which is distributed among nodes that are systematically different. In other words, the feature that explains the difference between nodes is not found in the inputs, so training samples from different nodes

have the same input but different output. In the future, we would like to experiment by grouping systematically different nodes using selective grouping to obtain better prediction accuracies.

Bibliography

[1] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V., "How to backdoor federated learning,"CoRR, vol. abs/1807.00459, 2018. [Online].Available: http://arxiv.org/abs/1807.00459

[2] Cao, V., Chu, K., Le-Khac, N., Kechadi, M., Laefer, D., and Truong-Hong, L., "Toward a new approach for massive lidar data processing," in2015 2ndIEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM), July 2015, pp. 135–140.

[3] Konecny, J., McMahan, H. B., Yu, F. X., Richt arik, P., Suresh, A. T., and Bacon, D., "Federated learning: Strategies for improving communicationefficiency,"CoRR, vol. abs/1610.05492, 2016. [Online]. Available: http://arxiv.org/abs/1610.05492

[4] Konecny, J., McMahan, B., and Ramage, D., "Federated optimization: Distributed optimization beyond the datacenter,"CoRR, vol. abs/1511.03575,2015. [Online]. Available: http://arxiv.org/abs/1511.03575

[5] Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z., "On the convergence of fedavg on non-iid data," 2019.

[6] McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A., "Federated learn-

ing of deep networks using model averaging,"CoRR, vol. abs/1602.05629,2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[7] Smith, V., Chiang, C., Sanjabi, M., and Talwalkar, A., "Federated multi-task learning,"CoRR, vol. abs/1705.10467, 2017. [Online]. Available:http://arxiv.org/abs/1705.10467

[8] Stich, S. U., "Local sgd converges fast and communicates little," 2018.

[9] Stich, S. U., Cordonnier, J.-B., and Jaggi, M., "Sparsified sgd with memory," 2018.

[10] Wang, J. and Joshi, G., "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms,"CoRR, vol.abs/1808.07576, 2018. [Online]. Available: http://arxiv.org/abs/1808.07576

[11] Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., and Chan, K., "When edge meets learning: Adaptive control for resource-constraineddistributed machine learning,"CoRR, vol. abs/1804.05271, 2018. [Online]. Available: http://arxiv.org/abs/1804.05271

[12] Woodworth, B., Wang, J., Smith, A., McMahan, B., and Srebro, N., "Graph oracle models, lower bounds, and gaps for parallel stochastic optimization,"2018.

[13] Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., and Beaufays, F., "Applied federated learning: Improving google keyboardquery suggestions,"CoRR, vol. abs/1812.02903, 2018. [Online]. Available: http://arxiv.org/abs/1812.02903

[14] Yu, H., Yang, S., and Zhu, S., "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deeplearning," 2018.

[15] Zhang, Y., Duchi, J. C., and Wainwright, M., "Comunication-efficient algorithms for statistical optimization," 2012.

[16] Zhou, F. and Cong, G., "On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization,"CoRR,vol. abs/1708.01012, 2017. [Online]. Available: http://arxiv.org/abs/1708.01012

Vita

Jiahan Liu was born in Guangzhou, China. After completing his work at Klein High School, Houston, Texas in 2014, he entered the University of Texas at Austin in Ausitn, TX. He enrolled in the Intergrated BSEE/MSE program at the University of Texas at Austin in 2017. He will be graduating in December, 2019 and working in industry in California.

Permanent Address: 8422 Glenn Elm Dr.
Spring, TX 77379

This thesis was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.