The Thesis Committee for Jiahan Liu

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

Federated Learning Model Complexity vs Robustness to non-IID data and Selective Federated Learning

Supervising Committee:

Christine Julien, Supervisor

Jonathan Valvano

Haris Vikalo

Federated Learning Model Complexity vs Robustness to non-IID data and Selective Federated Learning

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

Federated Learning Model Complexity vs Robustness to non-IID data and Selective Federated Learning

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 address the challenges of data ownership, privacy, computational power, and data storage. Previous federated learning research has addressed the effect of non independent and identically distributed data on federated learning [6]. Meanwhile, local models may have better performance if the test set is also non-IID [7]. However, there may be insufficient data on a node to train a local model for every node; hence the purpose of federated learning.

This research is the first, to our knowledge, to consider model performance on both a global test set and non-IID test set. Our experiments provide a original finding in that federated learning is only robust to non-IID data with constraints on the width and depth of a neural network. There is a tradeoff, however, between

model complexity and feasibility of training the model on edge devices. Thus, we propose selective federated learning algorithm which greatly allows simpler models that fit on edge devices to be robust to highly non-IID data. For non-IID test sets, we prove that a converged federated model may converge to weights which do not provide the optimal local loss for an arbitrary chosen number of training samples on each node. Additionally, this 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. For the experiments, we deployed federated learning library for multiple devices, Jetson Nano, Raspberry Pi, Macbook Pro, and Linux server and provide hardware benchmarks.

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. in 2015[4] that 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 optimizing weights locally on each node then aggregating the weights on a centralized server before updating all the local models with the new weights to start another round of training. We consider this one round of training between communications. Rounds of training continue until model convergence. Federated learning addresses of data ownership because the raw data never 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 engineer the data from the weights; hence federated learning is a potential paradigm that 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 to application. While it's true that models can have millions or billions of parameters, raw data can be even larger; for example, 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 leveraged 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 bottlenecked 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 participation of the devices in the aggregation step needs to be synchronized. The theoretical challenges involve the convergence of the federated model and robustness to non-IID data. While Xi et al. have proven the convergence of FedAvg on non-IID Data [5], Smith et al. has shown that federated models may underperform models trained using only local data [7]. However, in practice, nodes may not have enough data to train it's own local node; hence, the purpose of federated learning.

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 IID and non-IID. Testing with a non-IID test is significant because the nodes that collect non-IID training data may keep encountering non-IID data during inference time. 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 a non-IID test set, 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 for federated learning to find that neural network models trained in the federated setting are only robust to non-IID data once the network is wide or deep enough. The key finding from our experiments is that a single layer 30-neuron neural network can achieve achieve 96.40% accuracy on MNIST but only 33.2% in the federated setting on completely unbalanced data. This was missed in the Federated Learning of Deep Networks using Model Averaging paper [6] where the smallest model tested was a 2 layer hidden neural network that was 200 neurons wide. This makes complexity vs feasibility on edge devices an even more significant tradeoff. To improve the performance of simpler models, we propose a selective federated learning. 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 percentages from the paper averaged over ten experiments:

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 percentages and standard deviation for 10 experiments (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 simple models in the federated setting for non-IID data.

Chapter 2

Literature Review

We cover four key papers that lead up to the ideas present in this thesis. Federated Learning of Deep Networks using Model Averaging [6] is the first detailed paper that conducts experiments to examine effects of non-IID training data. Federated Learning: Strategies for Improving Communication Efficiency [3] is a systems paper that provided the foundation for how federated learning was implemented in this thesis. Virgina Smith's paper, Federated Multi-Task Learning [7] was the first paper to observe that local models could outperform federated global models which motivated our experiments. Finally, On the Converge of FedAvg on Non-IID Data [5] was the first proof on the convergence of federated averaging with realistic constraints and is the prerequisite for the proof in in chapter 5.

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

McMahan's 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 which uses the

weighted average for the aggregation step which is the key step in federated learning. In figure 2.1 is the algorithm presented by the paper.

Algorithm 1: Federated Averaging

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 Algorithm

McMahan's paper shows empirically that FederatedAveraging is robust to non-iid data by showing in experiments which 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.

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

Bonawitz's 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 to improve communication costs, *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.

We implement a simpler version of the federated learning presented here. Our paper focuses on the performance aspect of federated learning in terms of model accuracy 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. proposes multi-task learning for federated learning on non-iid 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.

Smith's paper led to the experiments for this thesis. It indirectly suggests that the test set should be distributed the same as the training set which the experiments form the FederatedAveraging 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 multilayer 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 datasets. There is a tradeoff, however, as complex models may not fit the constraints of edge devices.

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

Federated learning must be deployable to edge devices of varying computation and storage capabilites. In our implementation, we require that the selective federated learning algorithm be deployable on network with both CPU-only device as well as GPU-devices. In our experiments, we deployed 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 limit the time it takes to complete one round of training to model the fact that training should only occur on personal devices when charging.

To ensure our experiments are generalize, we followed current machine learning practices. We random shuffle the MNIIST dataset between experiments. At the time of this thesis, ReLu is a popular activation function and Kaiming He is a popular initialization for it. We use Kaiming He initialization with ReLu non-linearity for the weights. The results of this experiment assume ideal device participation. Federated Learning of Deep Networks by Model Averaging [6] also makes this assumption.

As with the experiments in Federated Averaging of Deep Networks by Model Averaging [6], the majority experiments were conducted on servers 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 networking was added to service the device to server communication.

3.2 Dataset Division

In each experiment, we partition the MNIST training set into partitions to represent different degrees of non-IID (unbalanced). Then we assigned nodes to a unique partition and distributed the data in each partition to it's nodes. We performed two sets of experiments. For our unbalanced test experiments we partitioned the test set to create unbalanced test sets in the same fashion we created the training set. For our balanced tests, we used the global test set of 10,000 images provided by the original MNIST dataset (Balanced Tests)

3.2.1 Training Set Division

Ours partitions can be distinguished by the percentage with which they contain balanced data. In table 3.1 the data partitions are described. We first remove 10,000 images from the original MNIST training set to create a validation set. The term random data refers to MNIST images which have been randomly assigned, this represents the balanced data.

	Partition 1	Partition 2	Partition 3
N% Balanced	N% Random Data (100-N)% Labels 0-3	N% Random Data (100-N)% Labels 4-6	N% Random Data (100-N)% Labels 7-9

Table 3.1: Data Partition

For example, to create datasets for 9 nodes, we would divide partition 1 equally among nodes 1-3, divide partition 2 equally among nodes 4-6, and divide partition 3 equally among nodes 7-9. Likewise, the test set can be made non-IID.

3.3 Software Library and Design

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

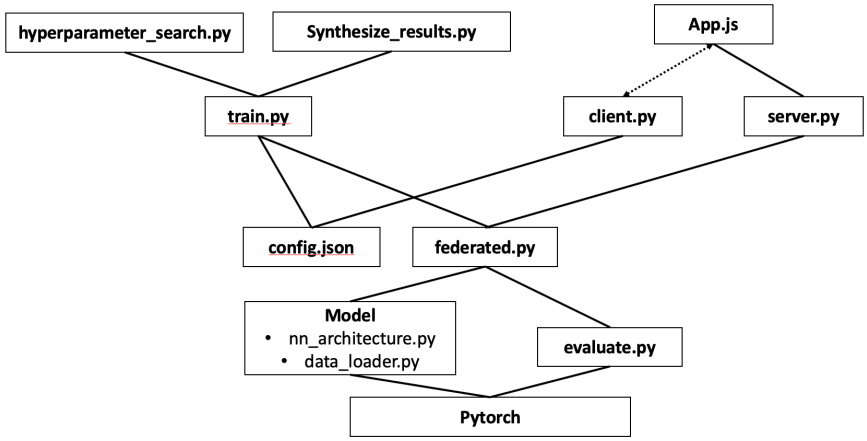

Figure 3.1: Software Design

3.3.1 Federated Learning

The software library is built using the Pytorch, a popular python library for machine learning. Federated.py contains all the code to perform a single round of federated learning in both the experiments as well as the deployed test setup. It allows creation of a local and global classes to represent or be deployed on local nodes and the aggregating server respectively. The data loaders and model architecture are con-

figurable and passed in as parameters to allow for flexibility in future experiments. All configurations for the hyperparameters, results file, networking were refactored out and put inside a json formatted configuration file. The programs are also setup to use CPU or GPU to do the training through the program args.

The bulk of the experiments was conducted in synthesize_results.py and train.py which reduce the networking and communication time of federated learning to communication through shared memory. Model parameters are passed in memory and different objects from the Federated.py classes are created to represent each local node and the aggregation server.

3.3.2 Hyperparameter Search

To perform the hyperparameter search, the hyperparameter_search.py program uses random search to train models using different learning rates and batch sizes. 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 finer resolution 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 manually chooses 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.

3.3.3 Deployed Version

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 running node.js 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 the server has received all the HTTP request and files, 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.

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

3.4 Model Complexity

The amount of data needed to train a neural network depends on model complexity, and as neural networks get larger they need more data to train. Since we are varying the distribution of data, it is also important to take the effects of model

complexity into consideration. We conduct our experiments on three models of different complexity. We would like to see the effects of this on federated learning with data distributed non-IID.

Single Layer ReLu Model

This is our simplest model. 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.

Four Layer Convolutional and ReLu Model

This is our second most simplest model. 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 and. The first hidden layer has a drop out of 0.5. The output is generated using log softmax.

Six Layer Extra Wide Convolutional and ReLu Model

This is our most complex model. The convlution block has three 3x3x2 convolutional layers of stride 1 and padding 1. 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 6272, 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.

3.4.1 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. Then we performed random search for the batch size and learning rate. For each search, we train for N epochs where for epochs N+1 to N+20 there was no improvement in the validation accuracy. 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 Constraints

Although our experiments show that non-IID data can be fixed by introducing more complex models, models that are trained on the edge have real computation and memory constraints. From an engineering perspective, one must trade off model complexity vs edge device computation, memory capabilities and power consumption. The selective federated algorithm proposed in this paper, makes the tradeoff easier and boosts the performance of simpler models for non-IID data. We devote this section to describing the hardware constraints associated with federated learning to see why selective federated learning is valuable.

We deployed and tested federated learning onto three devices, the Raspberry Pi 4 with 4GB of memory, Nvidia Jetson Nano, and Macbook Pro. We either ran the training to competition or until error. The Raspberry Pi 4 was the least powerful of the three devices. On the Raspbery Pi 4, both the Single Layer ReLu Model and Four Layer Convolutional and ReLu Model training was able to run to completion while training the Six Layer Extra Wide Convolutional and ReLu Model cause the Raspberry Pi 4 to crash. We measure the real, system, and user times and provide them in tables 4.2, 4.4. The real time is the total time it takes to train the model. The system time is the time which the operating system spends making system calls,

page swap or other tasks in kernel mode. The user time is the time spent running the user program.

In addition to running experiments on the local models, we calculate the number of multiply accumulate (MAC) operations for federated learning to do the backpropgation part of the training as a reference for compatibility with future devices. This is the dominant part of the training process. The MACs described in tables 4.1, 4.3, 4.5 are approximate calculations and only include the operations required to calculate the gradient of the loss for each parameter. The computations not accounted for in the calculations ones associated with the dropout, batchnorm layer, and max pooling layers.

In 4.1, 4.3, 4.5, the activation shape together with the layer allows us to calculate the number of parameters. To obtain the number of parameters for a fully connected layer denoted in the tables as FC1, we multiply the number of input activations by the number of output activations plus one for the bias. To obtain the number of parameters in a convolutional layer we multiply by the activation shape height by width and then we add one. The number of parameters in the current, and next layers allow us to calculate the number of MACs.

4.1 Number of Multiply And Accumulate

4.1.1 Single Layer ReLu Model Computation Requirements

In the Single Layer ReLu Model, backpropagation on a single image takes 23821 MACs. For 13333 images, this would take over 300,000,000 MACs. Both federated learning and selective federated learning works on the Raspberry Pi 4, and we timed the average training time per round to complete training (21 rounds). The calculations for the MAC are shown in table 4.1 and the software timings are shown in table 4.2.

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

Table 4.1: Single Layer ReLu Model

Type	Time (Seconds)
Real	57.90
User	83.42
System	40.56

Table 4.2: Software time for Single Layer ReLu Model

4.1.2 Four Layer Convolutional and ReLu Model Computation Requirements

In the Four Layer Convolutional and ReLu Model, training on one image takes 21852 MACs. For 13333 images, this would take over 291,000,000 MACs. This model takes less MACs to train because it is deep instead of wider. Training computation scales linearly with depth but is proportional to the width of a neural network. On the Raspberry Pi 4, we time the average training time per round to complete training (83 rounds). The calculations for the MAC are shown in table 4.3 and the software timings are shown in table 4.4.

Layer	Activation Shape	Number of Parameters	Number of MACs
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: Four Layer Convolutional and ReLu Model Computation Requirements

Type	Time (Seconds)
Real	169.98
User	503.14
System	133.86

Table 4.4: Software time for Four Layer Convolutional and ReLu Model

4.1.3 Six Layer Extra Wide Convolutional and ReLu Model Computation Requirements

In the Six Layer Extra Wide Convolutional and ReLu Model, each round of communication takes 904188 MACs. For 13333 images, we would have over 12,000,000,000 MACs. In the runtime of the training, the Raspberry Pi error as and threw an memory error stating that a python library function could not allot 1GB of memory. The calculations for the MACs are shown in table 4.5.

Layer	Activation Shape	Number of Parameters	Number of MACs
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: Six Layer Extra Wide Convolutional and ReLu Model MACs

Chapter 5

Federated Convergence of Global vs Local Losses

For non-IID dataset, it is reasonable to expect the test set should also be distributed non-IID with the same percentage of unbalanced data. For a non-IID test set, the local loss may be a better proxy for local accuracy than the global loss. In a federated model in which the global loss converges to a minimum, the local loss not be at the minimum for every node. 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

Here we proposed selective federated averaging to boost the performance of simple federated learning non-IID data. In extremely non-IID data, complex models do well in the federated setting. Local models also do well cannot be trained if there is insufficient data on each node. Selective federated averaging partitions 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 nodes first trains their own local model. In the selective grouping process, the local models uses the validation set $V = \{v_1, v_2, v_3...\}$ to generate a selection vector by the selection function given in figure 6.1. For every pair of nodes, a, b are in the same group if the percentage of overlap in their selection vector meets a similarity threshold. The similarity threshold is calculated as a hyperparameter using a seperate validation set. In our experiments, we split the validation set in half. Finally, a separate federated model is trained using federated average on all the nodes in each group. The end result is that there is a model for every 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 N+1 rounds of communication. The extra round is required communicate the validation vector to the central server for group selection.

Chapter 7

Experimental Results

We perform two sets of experiments. The first calculates test accuracy with the 10,000 image test set from MNIST to represent the performance of a model exposed to all a test set of classes during inference time. The second calculates test accuracy with unbalanced partitioned test set as described in section 3.2.1. In each set of experiments, we test all three models from section 3.4. We then display the accuracy on the graph represents in which the x-axis is the percentage of balanced data in the training set. We also provide the centralized accuracy for each model architecture as a benchmark.

A key finding in our experiments that was not found in previous research is that robustness of federated learning models requires a model certain depth or width. In the Federated Learning of Deep Networks using Model Averaging paper [6], the smallest model tested was a 2 layer hidden neural network that was 200 neurons wide. We found that a 1 layer network with 30 neurons wide can achieve 96.40% accuracy in the centralized setting but only 33.21% accuracy in the federated setting. There is, however, a tradeoff between model complexity vs computation, storage and power constraints of an edge device. The Six Layer Extra Wide Convolutional and ReLu Model in the federated setting, which is robust to a completely unbalanced

data distribution, cannot run on the Raspberry Pi 4. There are many embedded systems which run slow CPU's and have less memory than a Raspberry Pi 4. Thus we found edge device capabilities to be a constraint.

7.1 Experiments: Test Set of Original 10,000 MNIST Images

In this set experiments, ordinary federated averaging had the best performance. Wider or deeper models performed better in the federated setting when given unbalanced data. Six Layer Extra Wide Convolutional and ReLu Model Accuracy for IID Test Set is almost completely robust to non-balanced data. In our experiments, models of all complexity perform poorly if trained locally with 0% balanced data. Models of all complexity cannot learn to recognize images that it does not see. Selective federated averaging performs poorly if it's tested on a global dataset because it only creates federated models for similar nodes whose collective dataset is unbalanced. For federated, selective federated and local models, performance increases rapidly as the amount of balanced data increases. For federated learning models trained on 0% balanced data, the performance increases from 32.66% accuracy to 84.32% accuracy to 97.31% accuracy as the model's width and depth increases. Figures 7.4, 7.5, 7.6 show the test accuracy vs percentage of unbalanced data in the training set.

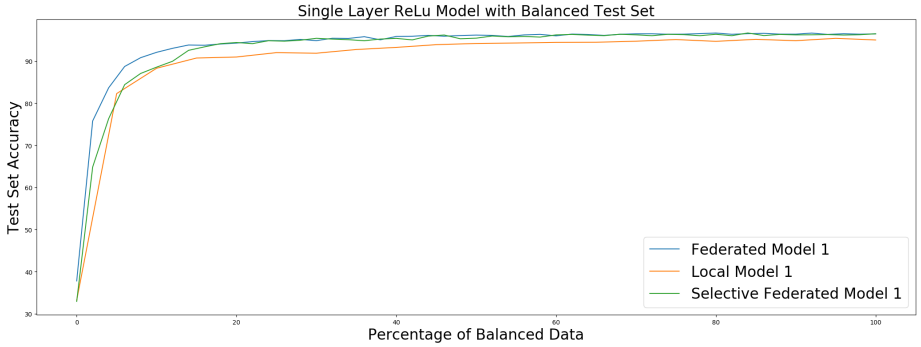

Figure 7.1: Single Layer ReLu Model Accuracy for IID Test Set

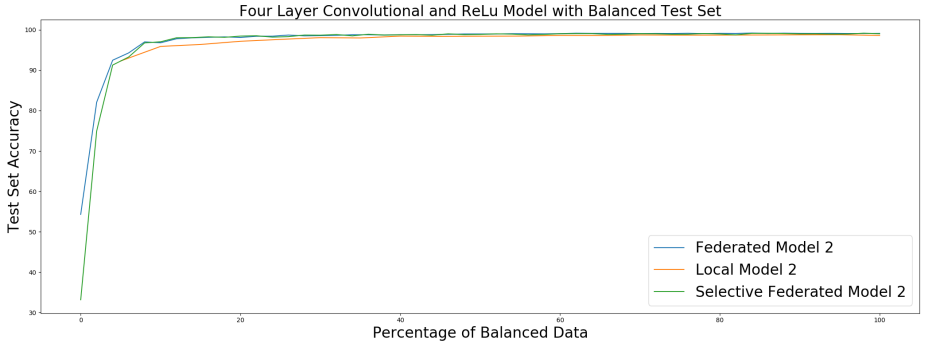

Figure 7.2: Four Layer Convolutional and ReLu Model Accuracy for IID Test Set

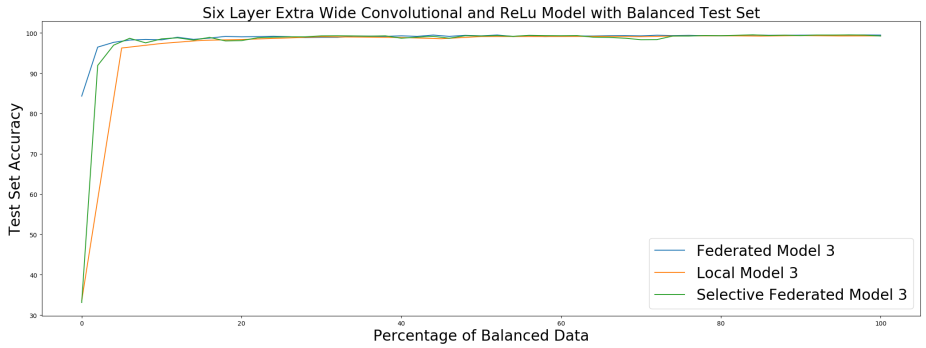

Figure 7.3: Six Layer Extra Wide Convolutional and ReLu Model Accuracy for IID Test Set

7.2 Experiments: Test Set from N% Balanced Partitioned

In these experiments, we distribute the test set data so that each test set contained the same percentage of unbalanced data as the training set. This is an important test not covered in the Federated Learning of Deep Networks using Model Averaging paper [6] because if an edge device encounters unbalanced data during training time, it is reasonable to expect the edge device to encounter unbalanced data during testing time. Simpler models using selective federated averaging a model perform better than simple models using federated averaging. Selective federated averaging has the performance of local models but since it aggregates multiple node's datasets together, it is more likely to have sufficient for training.

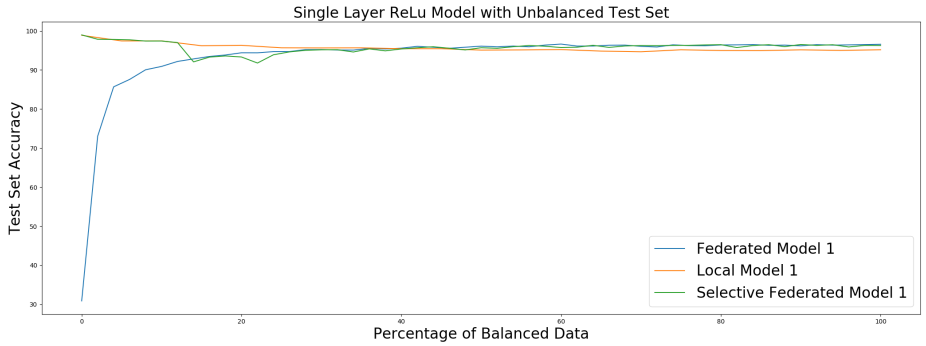

Figure 7.4: Single Layer ReLu Model Accuracy for non-IID Test Set

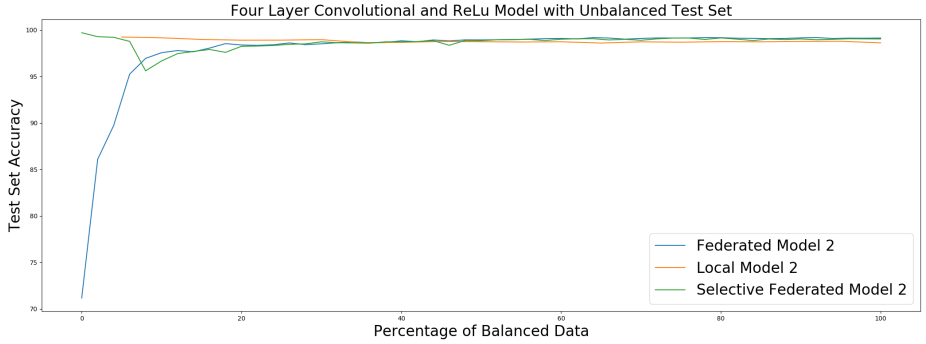

Figure 7.5: Four Layer Convolutional and ReLu Model Accuracy for non-IID Test Set

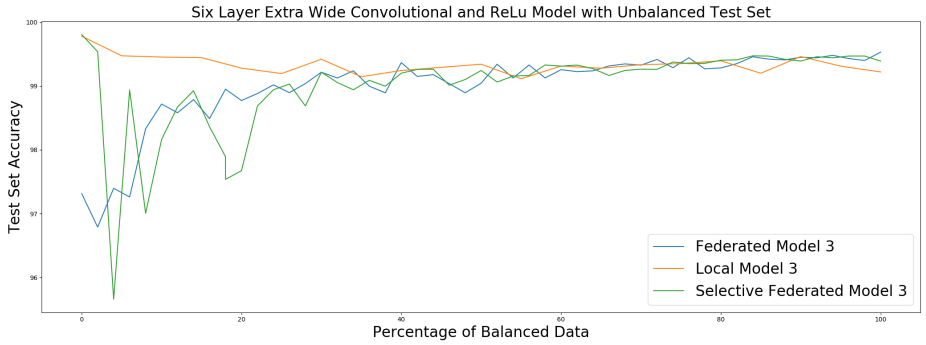

Figure 7.6: Six Layer Extra Wide Convolutional and ReLu Model Accuracy for non-IID Test Set

7.3 Centralized Learning Baseline

These are the baselines for each of the models trained the entire training set.

Model	Accuracy (percent)
1	96.40
2	99.12
3	99.54

Table 7.1: Centralized Training Test Accuracies

Chapter 8

Conclusion

In this paper we examine theoretically and experimentally the effects non-IID training data on Federated Learning with both IID and non-IID test sets. We found that model complexity is an significant constraint in federated learning performance. In the federated setting, however, there is a tradeoff between model complexity and hardware constraints. In our experiments, we find that the model that performs the best in the federated setting cannot be deployed to the Raspberry Pi 4, a common edge device. Additionally for non-IID test sets, we propose selective federated averaging, which is more robust to non-IID data than federated averaging. However given the rapid growth of the performance of edge nodes, one can predict complex Federated Learning models will be possible.

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 accuracy. Other grouping algorithms such as k-means clustering should also be considered.

A further question is whether models trained by federated learning and centralized variants of stochastic gradient descent learn the same mode. Do they produce model with similar parameters? Do the models make similar predictions on a common validation set? If not, what are the factors that determine whether a federated model and centralized model is similar. Is model complexity a factor?

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.