THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Adaptive Partial Training for Model-Heterogeneous Federated Learning

JASON SWOPE
SPRING 2024

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Mehrdad Mahdavi
Assistant Professor of Computer Science
Thesis Supervisor

Mohamed Almekkawy
Associate Research Professor of Computer Science
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

Federated Learning (FL) has increasingly become an area of interest within Machine Learning (ML) recently for its ability to combine the performance of multiple devices. Model-Heterogeneous FL in particular allows for the clients to train a larger model than each individual device could train individually by dropping out specific neurons from the global model. This allows even low-performance devices to contribute to training even when the device would otherwise would not be able to contribute under traditional Model-Homogeneous FL. The state of the art method for sub-model extraction is FedRolex, which systematically steps through the available neurons. In addition to model-heterogeneity, another major factor in the performance of FL is the level of data-heterogeneity between the devices. This study investigates the performance of Model-Heterogeneous methods FedRolex and FedDropout at differing levels of dropout, data-heterogeneity, and synchronization, and compares their performance with the Model-Homogeneous method FedAvg. In addition, three new methods are proposed to tackle the problem: FedStack, FedCover, and FedMinOccurances. The performance of FedDropout falls below the performance of any of the other methods, and FedMinOccurances shows inferior performance with high model heterogeneity.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we introduce the basics behind machine learning, in particular neural networks. We gradually increase the scope of the problem investigated in this study through distributed machine learning, federated learning, and finally model-heterogeneous federated learning. This chapter discuss the optimization problems for each of these sections and the approaches used to find the approximate solution of these problems.

1.1 Machine Learning

Machine learning (ML) is a subset of Artificial Intelligence aimed at creating a network of neurons, or neural network (NN), that accomplishes a task such as classifying a set of images or trying to predict an unknown value based on a set of known input values. In this sense, neural networks learn an unknown function $h : X \rightarrow Y$ that maps an input x to an output y where $d = (x, y)$ is a data point in dataset D, which is sampled from source distribution $\mathcal{D}$. The function $h \sim \mathcal{H}$ where $\mathcal{H}$ is the hypothesis space for a given neural network. Neural networks are trained by taking samples from the input distribution with known outputs, such as an image of a car which is know to be the image of a car, calculating what the neural network function believes the output should be and then adjusting the function to improve its performance. This is called training the neural network, and the set of samples used for this are called the training dataset. This is repeated for several iterations, or epochs, until the neural network reaches the point where it no longer improves from each epoch to the next.

1.1.1 Deep Neural Networks

A deep neural network (DNN) is a type of NN that consists of a series of several fully-connected layers which contain a set of neurons that each receives the outputs of the previous layer and passes its own output to each neuron in the next layer. The first layer is called the input layer, which is given the values of the input of one sample, so it must consist of a number of neurons equal to the number of input values for one sample. The final layer is the output layer and consists of the value(s) the neural network has calculated as the output. Every layer between the input layer and output layer are called the hidden layers. The basic structure of a DNN is shown in Figure 1.1.

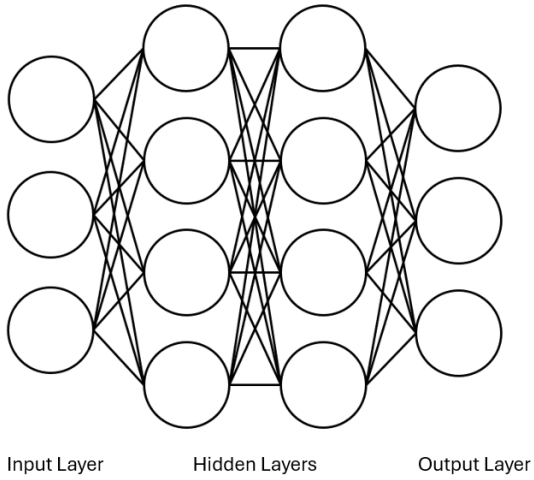

Figure 1.1: A Visualization of the structure of a DNN

Each neuron is passed a set of values, which is a vector x with length m, where m is the number of neurons in the previous layer. To calculate the value the neuron will pass to the next layer, the neuron multiplies the input vector with a weight vector w, also with length m that denotes the importance of values received from the neurons in the previous layer, and adds a bias term b that allows for the value to be shifted, $wx + b$. To parrallelize the computation, the current layer of n neurons can be calculated simultaneously where w is a $n \times m$ matrix, and b is a vector of length n. The result of calculating $wx + b$ is an output vector of length n, which is the output of the current layer.

1.1.2 Minimization of Empirical Risk

The true risk of a neural network is how well the network is able to map the input to the output. This is measured by a true risk function $\ell(h, d)$, which is the loss for one data point d. The true risk function

$$\mathcal{L}(h) = \mathbb{E}_{d\sim\mathcal{D}}[\ell(h, d)]. \tag{1.1}$$

However, it is impossible to practically calculate the expected value over the entire distribution – notably because $\mathcal{D}$ and h cannot be defined easily – so we use the cumulative loss over a set of samples D taken from distribution $\mathcal{D}$. This means the neural network can only find the *empirical* risk, which is the risk associated with the training samples. The empirical risk

$$\widehat{\mathcal{L}}(h) = \sum_{d\in D\sim\mathcal{D}} \ell(h, d). \tag{1.2}$$

This empirical risk must be minimized, which presents us with an optimization problem. The optimal model is defined

$$h_* = \arg\min_{h\in\mathcal{H}} \widehat{\mathcal{L}}(h). \tag{1.3}$$

1.1.3 Gradient Descent

To minimize the empirical risk, the network calculates the gradient of the loss function with respect to the network weights, w_t. The gradient will point in the direction of steepest ascent within the loss function, and the magnitude will represent the steepness. To minimize the loss function, the neural network will take a "step" in the opposite direction of the gradient with a step size equal to the magnitude of the gradient multiplied by η, the learning rate. This step is in the direction of steepest descent of the loss function, which decreases the value of the loss function as rapidly as possible. The weights are updated such that

$$\begin{aligned} w_{t+1} &= w_t - \eta \nabla \mathcal{L}(w_t) \\ &= w_t - \frac{\eta}{\|D\|} \sum_{d \in D \sim \mathcal{D}} \nabla \ell(w_t, d). \end{aligned} \tag{1.4}$$

1.2 Distributed Machine Learning

Distributed Machine Learning (DML) is a branch of ML in which training is split between several devices, which are called clients. The training data is split into local datasets $D = \{D_1, D_2, ...D_m\}$ where $D_i \sim \mathcal{D}$. Each client calculates a local update to the neural network weights using the local dataset. The local updates

$$\Delta w_{t,i} = \frac{1}{\|D_i\|} \sum_{d_i \in D_i \sim \mathcal{D}} \nabla \ell(w_t, d_i) \tag{1.5}$$

are calculated for each client i, and are aggregated at the server to create an update to the global server model

$$w_{t+1} = w_t - \eta \sum_{i=1}^{m} p(i) \Delta w_{t,i}. \tag{1.6}$$

This server update is a weighted average of the m local client updates, where the weights are given by $p(i)$ for each client i, and $\sum_{i=1}^{m} p(i) = 1$. The local true risk in DML is

$$\mathcal{L}(h) = \mathbb{E}_{d_i \sim \mathcal{D}}[\ell(h, d_i)] \tag{1.7}$$

for each client i, while the local empirical risk is found using only D_i, the client's shard of the dataset D. The local empirical risk is

$$\widehat{\mathcal{L}}_i(h) = \sum_{d_i \in D_i \sim \mathcal{D}} \ell(h, d_i) \tag{1.8}$$

for each client i. DML also adjusts the optimization problem to account for the aggregation of the local updates. The optimal model minimizes a weighted average of the empirical risk functions for each client, such that

$$h_* = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{m} p(i) \widehat{\mathcal{L}}_i(h) \tag{1.9}$$

Learning on multiple clients parallelizes the learning of the neural network, which speeds up the learning process to a pace unachievable by any of the client machines individually. However, the aggregation and averaging of the model weights is expensive, particularly for large machine learning models. The weights are transmitted over a communication network which is a major bottleneck of DML, so efficiently communicating these large matrices is an important field of study [1].

1.2.1 Synchronous Distributed Machine Learning

The computers used for each client in DML are likely not identical, so the performance will be different on each machine. In addition, even identical clients may take different amounts of time to complete the training depending on the dataset passed to it and other computational constraints such as background processes. Using traditional DML, if the clients do not complete training at the same time, those that finish quickly will be waiting idly for the rest of the clients to complete their training. This is called synchronous DML, which can be very inefficient. One strategy to mitigate the inefficiency is to approximate the performance of each client's computer and split the dataset to attempt to have the training on each to finish at approximately the same time. However, the speed of this approach relies on the accuracy of the approximation of performance of the clients. Another approach is asynchronous distributed machine learning which does not wait for all of the clients to finish training before it aggregates the server model. This will reduce the amount of waiting, but the weights that the models are training on will not be the same, introducing an added level of complexity [2].

1.3 Federated Learning

Federated Learning is similar DML; however, the datasets for each of the clients are not assumed to be sampled from the same source distribution. Rather, the local datasets $\{D_1, D_2, ..., D_m\}$ are sampled from distributions $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_m\}$ such that $\mathcal{D}_1 \neq \mathcal{D}_2 \neq ... \neq \mathcal{D}_m$. If the data is sampled from the same distribution, it is identically and independently distributed (IID), which means the gradient in the loss function created on each client will be approximately the same [3]. If the datasets are not IID, training using separate datasets will result in differing gradients for the loss function of each client, called gradient drift. Because the local datasets are sampled from different distributions, the local updates are

$$\Delta w_{t,i} = \frac{1}{\|D_i\|} \sum_{d_i \in D_i \sim \mathcal{D}_i} \nabla \ell(w_t, d_i) \tag{1.10}$$

for each client i. The local true risk is

$$\mathcal{L}(h) = \mathbb{E}_{d_i \sim \mathcal{D}_)}[\ell(h, d_i)] \tag{1.11}$$

for each client i, and the local empirical risk is

$$\widehat{\mathcal{L}}_i(h) = \sum_{d_i \in D_i \sim \mathcal{D}_i} \ell(h, d_i) \tag{1.12}$$

for each client i.

The more training that occurs without synchronizing, or the more epochs completed per communication round, the greater these differences will be, and the more difficult the optimization problem becomes [2]. This results in a higher lower-bound on the loss and a lower upper-bound on the accuracy achievable for any particular neural network.

One major advantage of FL compared to distributed machine learning is that the data is not shared between clients, so this allows private data to be used to train the neural network with substantially fewer privacy concerns [4].

1.4 Model-Heterogeneous Federated Learning

Model-heterogeneous federated learning is a branch of FL where different clients train different models. The objective of model-heterogeneous FL is to train a global model that is larger than the clients can store in memory. This is accomplished by removing some of the neurons in each of the sub-models passed to each of the clients. Therefore, each client has a different function $h_i \sim \mathcal{H}_i$, where $\mathcal{H}_i$ is the hypothesis space of the reduced neural network. The client models are created by applying a dropout mask

$$m_{i,l,j} \triangleq \begin{cases} 1 & \text{if the neuron is selected for client } i \\ 0 & \text{otherwise} \end{cases} \tag{1.13}$$

for each neuron j for each layer l for each client i.The size of the sub-models for each of the clients is smaller than the server model, so low-performance machines can help train the server model, even if they do not have a large amount of memory [5]. However, the removal of some neurons introduces gradient drift because only part of the model is being trained and each sub-model disconnects the neurons that are selected for one client from the neurons that are not selected. This reduces the achievable performance and learning speed of the global model [6].

The local weight update

$$\Delta\widehat{w}_{t,i} = \frac{1}{\|D_i\|} \sum_{d_i \in D_i \sim \mathcal{D}_i} \nabla \ell(\widehat{w}_{t,i}, d_{i,n}), \tag{1.14}$$

where $\widehat{w}_{t,i} \subseteq w_t$ are the weights for the sub-model passed to client i in communication round t. The global model update is

$$w_{t+1} = w_t - \eta \sum_{i=1}^{m} p(i) \Delta\widehat{w}_{t,i}, \tag{1.15}$$

which is similar to the equation for FL, except only the weights present in each client contribute to the new server weights. In the case where a neuron is not present in any client, the original server weights are used. The sub-models each have differing true and empirical risk functions

$$\mathcal{L}(h) = \mathbb{E}_{d_i \in D_i \sim \mathcal{D}_\rangle}[\ell(h_i, d_i)] \tag{1.16}$$

for each client i, and

$$\widehat{\mathcal{L}}(h_i) = \sum_{d_i \in D_i \sim \mathcal{D}_\rangle} \ell(h_i, d_i) \tag{1.17}$$

for each client i, which depend on the different sub-model functions. The optimal server model

$$h_* = arg \min_{h \in \mathcal{H}} \sum_{i=1}^{m} p(i) \widehat{\mathcal{L}}(h_i) \tag{1.18}$$

minimizes the sum of the local empirical risks.

The state-of-the-art method for Model-Heterogeneous FL is FedRolex[7], which "rolls" through the neurons, advancing one step for each client or round of synchronization. One of the keys to FedRolex's performance is its ability to ensure that as many neurons appear in at least one client as possible [8]. Neurons that do not appear in any client is not training, so it cannot contribute anything to the performance of the model. FedRolex's systematic approach to selecting neurons increases the number of neurons included in the sub-models compared to FedDropout [9], which randomly selects the neurons for each client.

Chapter 2

Methodology

This chapter will focus on the details of the implementation of Model Heterogeneous Federated Learning used for this study. It will discus which dataset was chosen and explain the reasoning behind this choice, as well as detail how a customizable level of data heterogeneity was induced among the client datasets. The model chosen for the study will be explained, and a description of how the server model weights were updated is also included. Finally, this chapter details the description of each method of selecting neurons tested and their implementations.

2.1 Dataset

The MNIST[10] dataset was used for this study. MNIST is a relevant dataset for machine learning and poses a difficult yet achievable problem for deep neural networks. The FedRolex study uses the CIFAR-10 and CIFAR-100 datasets [11] which were considered for this study. MNIST consists of 70000 28x28 grayscale images separated evenly into 10 classes of 7000 images, while CIFAR-10 and CIFAR-100 contain 60000 32x32x3 colored images split evenly into 10 classes of 6000 images and 100 classes of 600 images respectively. The larger colored images of the CIFAR datasets are a challenge better suited to convolutional neural networks (CNNs). This study focuses on the performance of DNNs, so MNIST was better suited for the model architecture chosen.

DML and FL require the dataset to be split differently than traditional ML. Each client requires both training and testing data, and the server also requires testing data. To accomplish this, the server test data is selected first, and then the rest of the data is distributed among the clients. The data given to each client is then randomly partitioned into training and test data.

2.1.1 Data Heterogeneity

One consideration of FL is that the distribution of data on each client is not necessarily the same. The level of heterogeneity of the distributions on each client will impact the convergence of the server network. The level of data heterogeneity is one of the main parameters investigated in this study

To induce heterogeneity in the local client datasets is to limit the classes passed to each of the clients. This means each client may not have access to data from all 10 classes, so the resulting datasets are not IID. The number of classes that are passed to each client is a measure of the level of data heterogeneity between the clients. For the purposes of this study, the levels selected for investigation were 2, 4, 6, 8, and 10 classes per client. This gave a representative scale to study the effect data heterogeneity will have on the performance of the server model.

Separating the data was accomplished by first partitioning the dataset into server and client data. The data used for the server data is the 10000 testing images of the MNIST dataset. Separating the client data consisted of first generating random indices that indicated which classes would be present on each client. The entire client dataset was split by class, and each class was split randomly by the number of clients that the class occurs on. These sub-class partitions are then combined so that the correct classes occur on each client. These client datasets are then randomly partitioned into training and test data using a 70-30 split.

Algorithm 1 Federated Learning Dataset Partitioning

procedure PARTITIONDATASET(*MNISTTrainData, numClassesPerClient, numClients*)
    allClientData ← *MNISTTrainData*
    classIndices ← GetClassIndices(*numClientsPerClient*)
    splitClientData ← ParitionClientData(*allClientData, classIndices*)
    return*splitClientData*

Algorithm 2 Getting Indices of Classes to be Allocated to each Client

procedure GETCLASSINDICES(*numClassesPerClient*)
    for client **do**
        indices ← *numClassesPerClient* randomly sampled integers between 0 and 9
        add *indices* to *classIndices*[*client*]
    return *classIndices*

Algorithm 3 Partitioning Client Data

procedure PARITIONCLIENTDATA(*allClientData, classIndices*)
    dataSplitByClass ← *allClientData* split by class label
    for class **do**
        classOccurance ← number of occurrences of *class* in *classIndices*
        dataSplitByClass[class] ← *dataSplitByClass[class]* split into*classOccurance* partitions
    for client **do**
        for class in classIndices[client] **do**
            add next element of *dataSplitByClass[class]* to *clientData[client]*
    return*clientData*

2.2 Model

This study also aims to understand the performance of the different methods for model-heterogeneous FL. To be able to measure the performance of each method, a simple deep neural network (DNN) was selected for this task. The Neural network chosen consists of a flatten layer to convert the 2-dimensional image to a 1-dimensional input. This is followed by two fully-connected layers, the first containing 128 neurons and the second containing 32, and a final output layer of 10 neurons. PyTorch was used to implement this neural network due to its power and modularity, and the Adam optimizer was used also due to it's strong performance when compared to a traditional SGD optimizer.

In Federated Learning, each client trains its copy of the model individually for a set number of epochs before the models are aggregated and synchronized. With less synchronizations, the training rounds are longer, so the gradients diverge more than with more frequent synchronizations. This results in the model weights converging slower, and the loss function having a higher floor. However, synchronizing the models require the model parameters to be communicated, aggregated, and communicated back to the clients. This synchronization cost is non-trivial, particularly for large neural networks. Selecting the number of epochs per round is a precarious balance between speed and performance of the model. This study tests two levels of synchronization: 40 rounds with 5 epochs per round, and 200 rounds with only 1 epoch per round.

The hyperparameters used for training were consistent across the levels of model and data heterogeneity, but only differed in the rate of decay of the learning rate between the levels of synchronization. An initial learning rate of 1e-5 was chosen as it was found to be a suitable learning rate for the set of parameters investigated. An exponential learning rate decay of 0.95 was used for the low synchronization level of 40 rounds with 5 epochs per round because it best suited for this synchronization level. Similarly, an exponential decay of 0.99 was used for the high synchronization level of 200 rounds with 1 epoch per round.

Algorithm 4 Server Training

```
1:  procedure TRAINSERVER(MNISTTestData, MNISTTrainData)
2:      serverModel ← The Model
3:      testDataset ← MNISTTestData
4:      clientDatasets ← PartitionDataset(MNISTTrainData
5:      for round do
6:          clientNeuronIndices ← GetNeurons() ▷ GetNeurons depends on the selection method
7:          clientModels ← serverModel with only neurons specified in clientNeuronIndices
8:          for client do
9:              for epoch do
10:                 Train clientModels[client] using train data of clientDatasets[client]
11:                 Test clientModels[client] using test data of clientDatasets[client]
12:         serverModel ← Average of clientModels weights
13:         Test serverModel using test data of testDataset
```

2.2.1 Model Heterogeneity

The challenge with model-heterogeneous FL is trimming the server model to create smaller models that will be trained by each client. The dimensions of the input and outputs must remain unchanged for the model to be able to function properly, but the number of neurons in each intermediate layer is reduced. The proportion of neurons selected for each client represents the level of model-heterogeneity, where larger proportion of neurons selected represents less heterogeneity. This study will investigate 4 levels of model heterogeneity, 0.25, 0.5, 0.75, and 1. The method in which the neurons to keep are selected is the main focus of this study. The methods this study investigates are FedAvg, FedDropout, FedRolex, and 3 new methods proposed to improve performance beyond FedRolex: FedCover, FedStack, and FedMinOccurances.

2.2.2 Client Network Selection Methods

FedAvg is the benchmark method as it is not model-heterogeneous method. The entire server model is trained on each client and then the weights from each client are averaged to create the new server model. FedAvg is the same as selecting all of the neurons to be kept, or a model-heterogeneity level of 1.

FedDropout is the baseline method of model-heterogeneous FL. It randomly selects the neurons to be kept for each client. Each neuron has a p_k probability of being selected, so the mask for FedDropout is

$$m_{i,l,j} \triangleq \begin{cases} 1 & p_k \\ 0 & (1-p_k) \end{cases} \tag{2.1}$$

where the mask is divided by (p_k) to maintain an equal weighting of the neurons when aggregating to calculate the server weights.

The state-of-the-art method of selecting neurons is FedRolex, which takes a slice of the neurons for each client and takes a 'step' for each round. This results in a sliding window of neurons selected, which systematically "rolls" through all of the neurons. This ensures that the neurons get an approximately equal amount of training, which was identified as a key factor in the performance of model-heterogeneous FL because it means all of the neurons are contributing to the accuracy of the model and not fouling the model with contributions from untrained weights. The key hyperparameter in FedRolex is the size of the step taken in each round. This study attempts 4 sizes of steps, represented as the proportion of the neurons present in each layer. The first two steps investigated were 0.2 and 0.5, but two of the proposed methods, FedCover and FedStack, are special cases of FedRolex. FedCover attempts to spread out the overlaps of the models by using a step equal to the inverse of the number of clients. In this study, which uses 10 clients, the steps are 0.1. This minimizes the number of clients each neuron will be present in, which leads to a more even distribution of learning. FedStack attempts to cover as many new neurons as possible by using a step of size 1. This "stacks" the neurons of each consecutive round on the round prior. This aims to evenly train the neurons by ensuring neurons not used in the previous round will appear in the next round. However, the original FedRolex study suggests that step size has an unclear, non-linear effect on

the model performance[7]. The mask for FedRolex is

$$m_{i,l,j} \triangleq \begin{cases} 1 & \text{if the neuron is in the range [start + (step} * \text{round), start + (step} * \text{round) + windowWidth]} \\ 0 & \text{o.w.} \end{cases} \tag{2.2}$$

FedMinOccurances introduces an entirely new approach where the neurons are sampled from a distribution that prioritizes neurons that did not occur in the most recent round of training. This approach attempts to incorporate neurons that did not train in the previous round while also including some randomness. Due to the nature of FedRolex, each neuron only trains with the neurons that have similar indices. The randomness within FedMinOccurances aims to avoid this repeating pattern so that each neuron can train in a larger number of sub-models while training the neurons more evenly than FedDropout. FedMinOccurances mask for the first round is the same as FedDropout. If $n_{l,j}$ is the numbner of clients neuron l, j occured in the previous round FedMinOccurances's mask

$$m_{i,l,j} \triangleq \begin{cases} 1 & \frac{N+1-n_{l,j}}{\sum_j N+1-n_{l,j}} \\ 0 & 1 - \frac{N+1-n_{l,j}}{\sum_j N+1-n_{l,j}} \end{cases} \tag{2.3}$$

Chapter 3

Results and Discussion

3.1 Parameters

To evaluate the performance of each method, the model was trained with each combination of parameters. The first parameter is Data Heterogeneity, evaluated by the number of classes present on each client, which was tested at five levels: 2, 4, 6, 8, and 10 classes per client. The second parameter is Model Heterogeneity, represented by the proportion of neurons selected for each client, which was tested at four levels: 0.25, 0.5, 0.75, and 1. The final parameter is the level of synchronization, represented by the number of epochs before the client networks were aggregated at the server, which was tested at two levels: 40 rounds with 5 epochs per round, and 200 rounds with 1 epoch per round. Each of these was evaluated at three separate seeds to ensure that random chance does not impact the performance of the methods when comparing across combinations of parameters

There are some important disclaimers about these parameters. Data and Model Heterogeneity are not hyperparameters that can be tuned, but are rather external factors determined by the computational and data resources available, as well as the model desired. In addition, while synchronization could be considered a hyperparameter, the loss of efficiency associated with higher synchronization encourage lower synchronization, so it is better classified as a parameter defined by the environment the model is training in. Because these parameters are more frequently out of the control of the developer training the model, it is important to understand how the performance of each of these method varies in each scenario to determine whether these methods are best suited for a specific application.

3.2 Seed 1

The two aspects of the training of the model that are most important in determining the performance of the model are the speed of convergence and the final performance achieved by each method. One aspect to note is that FedAvg is included in every graph; however, it is only an accurate comparison to the performance of the other methods when the model heterogeneity is 1. FedAvg is included in each graph because it is a good benchmark for the methods under perfect conditions. When the model heterogeneity level is 1, all of the methods performed equally as all of the neurons were included in each of the clients, and there is no variation between the methods.

3.2.1 Low Synchronization

Low synchronization in this study is represented by 40 rounds of training each consisting of 5 epochs per round. Figure 3.1 shows the plots of Accuracy vs Epochs for Seed 1's low synchronization tests, which were measured as the server's accuracy sampled at each synchronization of the client models. The first item to note is that along the bottom row, all of the methods perform the same. This aligns with the expected performance as all of the neurons were selected for each client, so there is no variation between any of the methods.

In many of the plots in the lower right corner, there is not much variation between the methods even when comparing to FedAvg. This is expected as the differences in the methods only marginal in the slightly sub-optimal conditions. The higher levels of model and data heterogeneity are where the methods' performance diverge more. However, there are not many meaningful patterns in the

performance of each method. The first aspect to note is that FedDropout performs considerable worse than the other methods, particularly in the more extreme combinations of parameters. FedRolex with a 0.2 overlap appears to perform worse when there is high model heterogeneity and low data heterogeneity. One concerning trend is the oscillating nature of some of the FedRolex-based methods in the more extreme situations. The overall accuracies of each of the methods do trend upward, but these fluctuations are likely due to the nature of the repeating coverage of certain neurons in each of the clients.

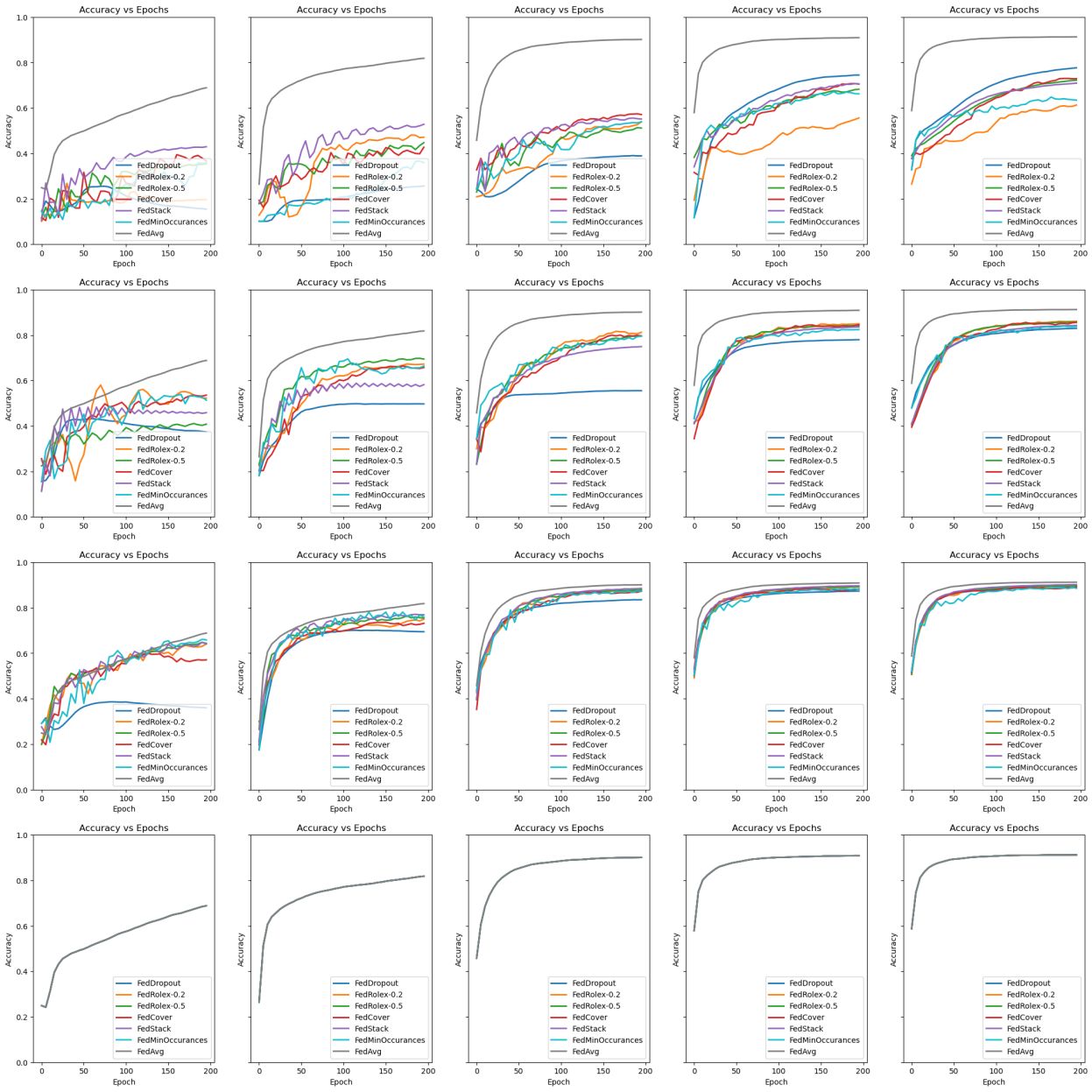

Figure 3.1: Seed 1 Accuracy Summary with Low Synchronization

Figure 3.2 shows the Loss vs Epochs of Seed 1's low synchronization tests. The trends observed in the accuracy plots of Figure 3.1 also appear in Figure 3.2 as well. FedDropout performs

substantially worse than the other methods in instances with high data and model heterogeneity. In addition, there is a concerning trend with FedDropout where the loss is increasing, even while the accuracy increases. Figure 3.3 showcases this counterintuitive phenomenon, which contradicts the expectation of the test data loss strictly decreasing until the model begins to overfit the training data. In addition, FedMinOccuranes shows a slightly worse performance in comparison to the rest of the FedRolex-based methods with high model heterogeneity, which is likely due to its proximity to FedDropout. When FedMinOccurances has only a few neurons selected per round, the number of neurons that did not occur in the previous round is very high and all of these neurons share the same probability. In practice, this almost reduces FedMinOccurances to FedDropout.

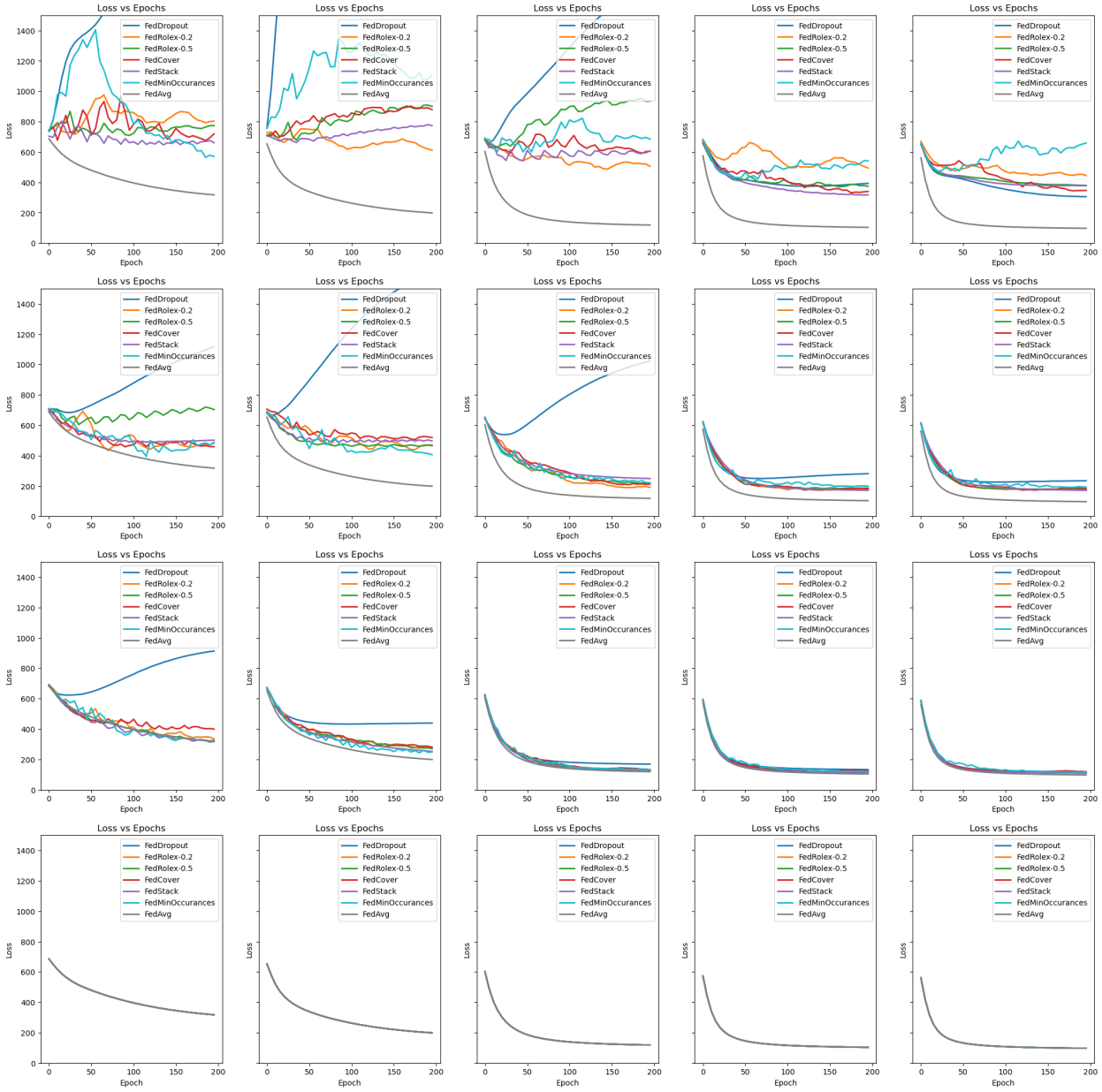

Figure 3.2: Seed 1 Loss Summary with Low Synchronization

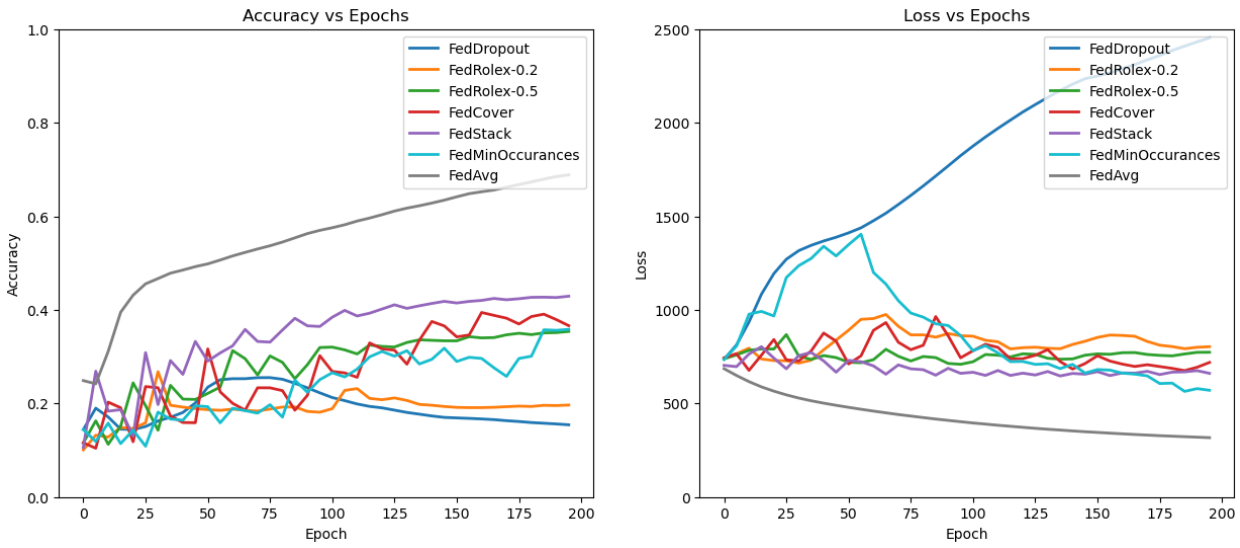

Figure 3.3: Seed 1 Method Comparison low synchronization, 2 classes per client, 0.25 neruons selected

3.2.2 High Synchronization

In the high synchronization tests, 200 rounds each with a single epoch were used to train each model. The expected performance is higher when training with high synchronization because the client models do not deviate as far from each other as when there are more epochs conducted before aggregating the weights of the clients.

Figure 3.4 shows the accuracy vs epoch plots for each combination of parameters tested, and very similar trends start to appear in the high synchronization tests to those that appear in the low synchronization tests. FedDropout routinely performs worse than the other methods, particularly in the more extreme combinations of parameters, and FedRolex with a 0.2 step performs poorly in high model heterogeneity and low data-heterogeneity situations. Many of the FedRolex-based methods perform very similarly and do not have a lot of variation between them.

FedMinOccurances has a trend to start quite poorly but has a sharp rise in performance part of the way through training, which did appear weakly in the low synchronization tests but appears much more dramatically in these instances. Interestingly, FedMinOccurances performs considerably better than the rest of the other methods with 2 classes per client and 0.25 of the neurons selected for each client, which is shown in the top leftmost plot, which contradicts its relatively poor performance with low synchronization in the same scenario.

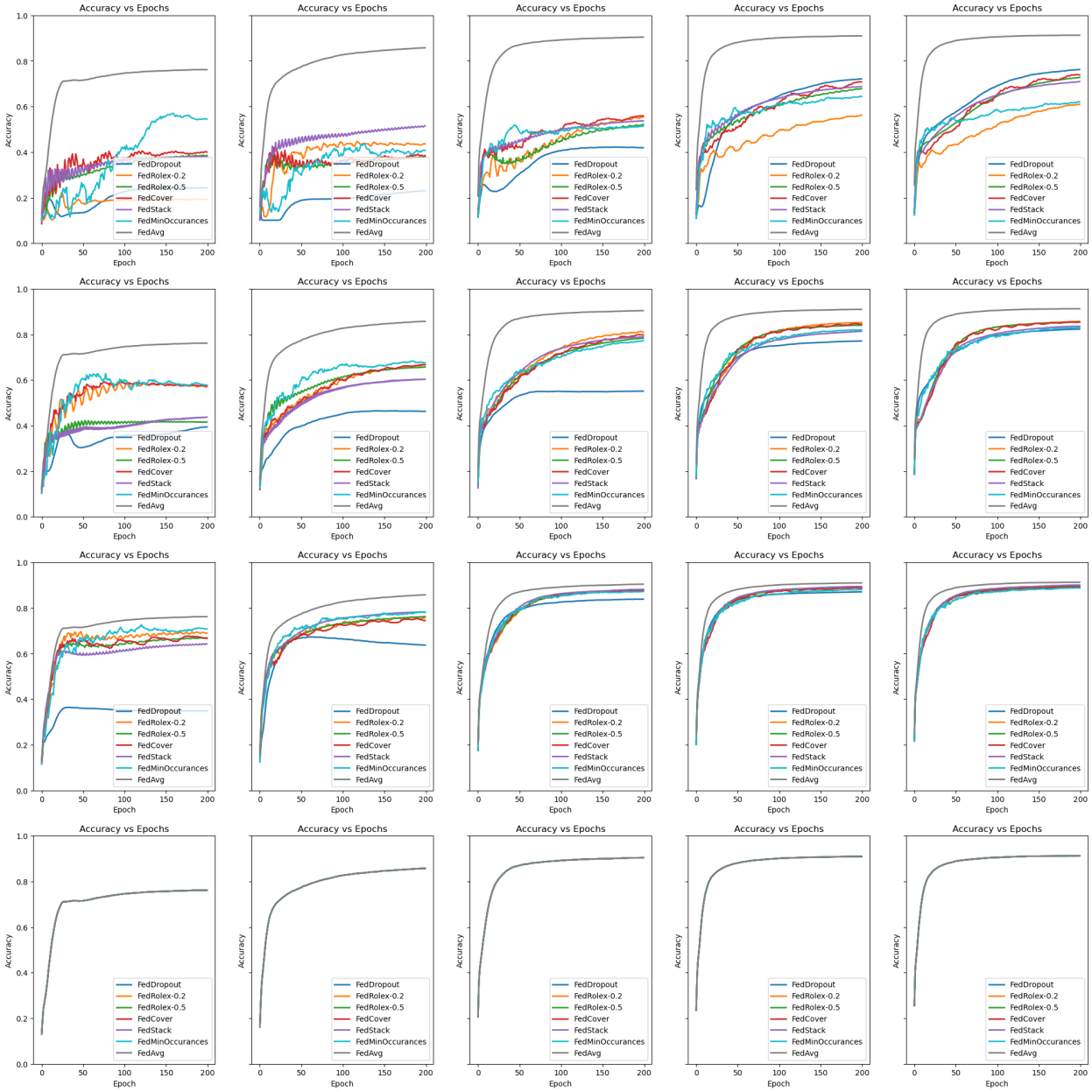

Figure 3.4: Seed 1 Accuracy Summary with High Synchronization

Figure 3.5 displays the plots of loss over each of the epochs, which again follows some previous trends. FedDropout has an increasing loss with the more extreme situations, which continues to follow the patterns established by previous tests, and FedMinOccurances performs slightly worse than the rest of the methods with high model heterogeneity. The anomalous situation with FedMinOccurances also shows in the plot with high data and model heterogeneity.

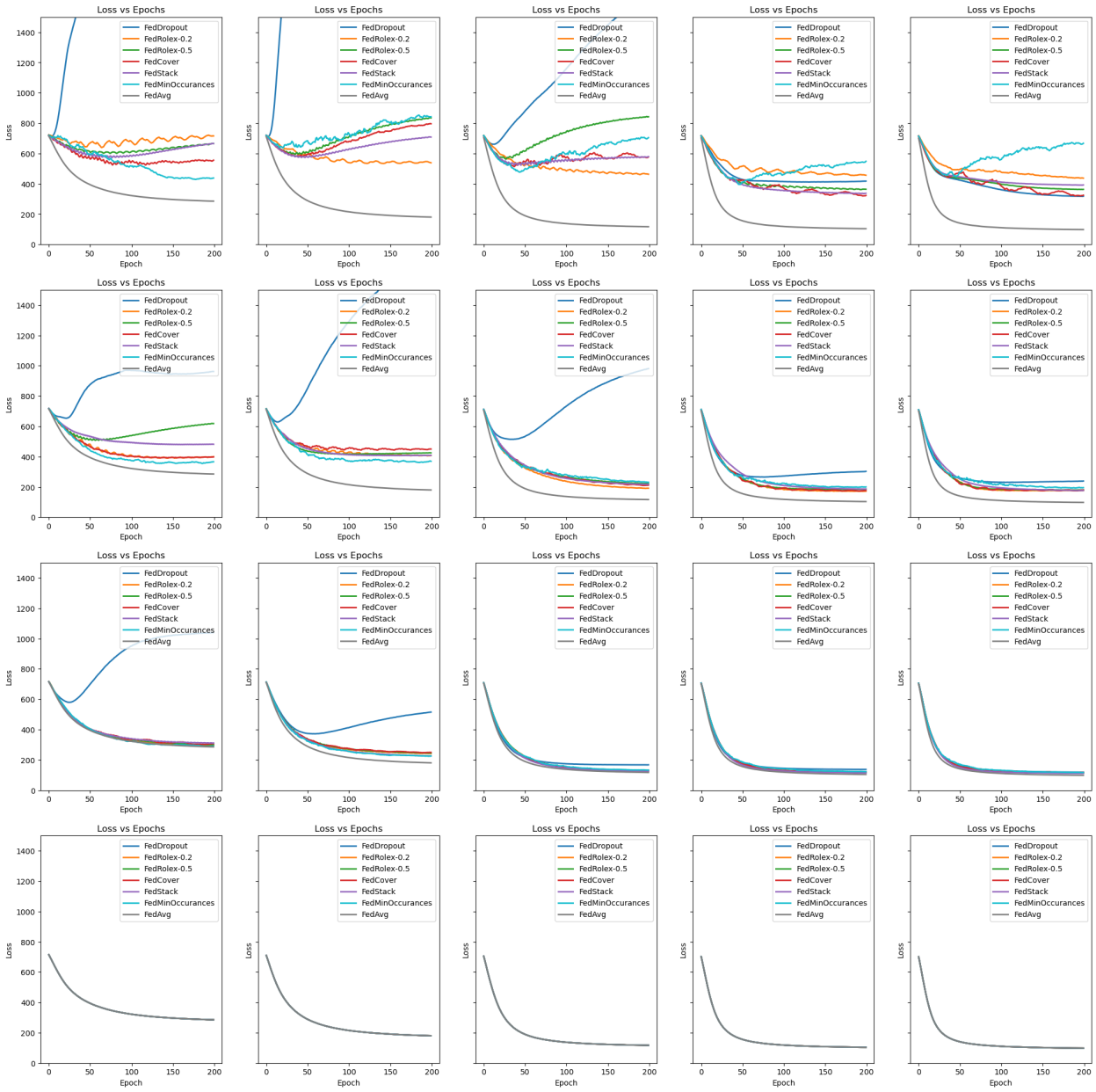

Figure 3.5: Seed 1 Loss Summary with High Synchronization

3.3 Seeds 2 and 3

The plots for Seeds 2 and 3 are attached in A.1 and A.2. The major patterns which appear in Seed 1 continue in Seed 2. However, the situational and anomalous situations were not replicated, so they were the result of the seed rather than the algorithm. Seeds 2 and 3 did also have its own alomalies depending on the unique combination of environmental factors. Some underlying patterns that arose were that FedDropout performs the worst of all methods in most situations, and FedMinOccurances performs poorly om instances with particularly high model heterogeneity. Unexpectedly the 0.2 step size FedRolex does not perform as well as the other step sizes in with

high model heterogeneity, despite a step size of 0.1 (FedCover) and 0.5 performing approximately equally. In addition, some instances of FedRolex have oscillating performance, which is likely due to the repetition of each part of the server model occurring in the sub-models.

3.4 Average Performance

The individual seeds provide a good perspective into the case-by-case performance of each of the methods. In comparison, if we want to look at the larger picture, averaging the performances of the seeds should provide a more holistic perspective of each method. This section takes the maximum performance achieved by each model in each of the combinations of hyperparameters, and averages these performances over each of the seeds.

3.4.1 Low Synchronization

Table 3.1 shows the low synchronization average results across the 3 seeds. FedMinOccurance shows a strong performance with high data heterogeneity; however, there was an anomalous instance with high model and data heterogeneity in seed 1. In addition, it appears that beyond FedMinOccurance, the FedRolex-based methods performed well with lower data heterogeneity.

Classes Per Client	Method	Proportion of Neurons Per Client			
		0.25	0.50	0.75	1.0
2 Classes	FedAvg				75.11
	FedDropout	21.03	30.55	38.34	75.11
	FedRolex-0.2	25.14	50.35	67.29	75.11
	FedRolex-0.5	38.96	45.09	66.51	75.11
	FedCover	32.61	50.10	66.39	75.11
	FedStack	33.32	41.97	65.91	75.11
	FedMinOccurances	**39.29**	**53.85**	**68.84**	75.11
4 Classes	FedAvg				86.88
	FedDropout	31.08	45.79	62.72	86.88
	FedRolex-0.2	41.10	67.75	80.79	86.88
	FedRolex-0.5	47.19	66.69	80.37	86.88
	FedCover	45.46	68.24	80.39	86.88
	FedStack	**48.26**	58.02	80.98	86.88
	FedMinOccurances	42.49	**68.84**	**81.07**	86.88
6 Classes	FedAvg				90.45
	FedDropout	43.39	61.33	85.27	90.45
	FedRolex-0.2	50.01	**82.59**	88.18	90.45
	FedRolex-0.5	60.42	77.90	88.13	90.45
	FedCover	**60.58**	82.33	**88.20**	90.45
	FedStack	54.38	76.76	88.12	90.45
	FedMinOccurances	43.41	79.93	87.63	90.45
8 Classes	FedAvg				90.96
	FedDropout	70.42	78.19	87.64	90.96
	FedRolex-0.2	60.25	**85.04**	89.19	90.96
	FedRolex-0.5	**70.68**	82.20	89.16	90.96
	FedCover	70.31	84.95	89.05	90.96
	FedStack	69.46	80.75	**89.23**	90.96
	FedMinOccurances	59.60	83.01	88.41	90.96
10 Classes	FedAvg				91.37
	FedDropout	76.51	82.52	88.98	91.37
	FedRolex-0.2	61.65	**86.12**	89.61	91.37
	FedRolex-0.5	**73.38**	84.29	89.75	91.37
	FedCover	72.89	85.98	89.51	91.37
	FedStack	73.24	82.61	**89.93**	91.37
	FedMinOccurances	59.36	83.46	88.90	91.37

Table 3.1: Low Synchronization Average Accuracy

3.4.2 High Synchronization

Table 3.1 shows the high synchronization average results across the 3 seeds. FedCover consistently performed the best or near to the best among the methods tested, particularly with low data heterogeneity. The small step size of FedCover and many rounds associated with more synchronization likely combined to create this strong performance. In addition, in most instances there was very limited difference between the high and low synchronization average performances with each combination of parameters. Perhaps an example with even lower synchronization should be tested to investigate the performance in those situations.

Classes Per Client	**Method**	**Proportion of Neurons Per Client**			
		0.25	0.50	0.75	1.0
2 Classes	FedAvg				68.84
	FedDropout	21.17	30.70	36.28	68.84
	FedRolex-0.2	31.87	46.58	62.15	68.84
	FedRolex-0.5	37.93	40.32	62.66	68.84
	FedCover	29.14	**48.72**	60.26	68.84
	FedStack	**37.35**	41.04	61.43	68.84
	FedMinOccurances	30.74	48.07	**64.78**	68.84
4 Classes	FedAvg				85.21
	FedDropout	31.80	45.56	68.18	85.21
	FedRolex-0.2	43.39	67.89	80.68	85.21
	FedRolex-0.5	**49.84**	64.79	80.86	85.21
	FedCover	44.95	67.30	78.91	85.21
	FedStack	47.72	59.18	**81.04**	85.21
	FedMinOccurances	39.17	**69.11**	80.84	85.21
6 Classes	FedAvg				90.43
	FedDropout	45.70	60.69	85.18	90.43
	FedRolex-0.2	51.51	82.56	88.31	90.43
	FedRolex-0.5	61.27	77.55	**88.36**	90.43
	FedCover	**63.01**	**82.69**	87.43	90.43
	FedStack	54.84	75.88	88.32	90.43
	FedMinOccurances	48.13	81.85	88.00	90.43
8 Classes	FedAvg				90.96
	FedDropout	71.08	78.29	87.76	90.96
	FedRolex-0.2	60.85	85.18	89.36	90.96
	FedRolex-0.5	70.67	82.20	89.22	90.96
	FedCover	**85.11**	**88.51**	**90.96**	90.96
	FedStack	70.58	81.54	89.29	90.96
	FedMinOccurances	63.60	83.92	88.75	90.96
10 Classes	FedAvg				91.37
	FedDropout	77.23	83.08	89.10	91.37
	FedRolex-0.2	61.74	85.94	89.84	91.37
	FedRolex-0.5	73.58	84.66	89.81	91.37
	FedCover	**73.66**	**86.23**	89.11	91.37
	FedStack	72.89	82.94	**89.95**	91.37
	FedMinOccurances	61.31	84.46	89.26	91.37

Table 3.2: High Synchronization Average Accuracy

Chapter 4

Conclusion

This study investigates the performance of state-of-the-art methods for model-heterogeneous federated learning and introduces and evaluates new methods of selecting neurons for the client sub-models. The new methods perform approximately equal to FedRolex in most situations. FedCover and FedStack propose limitations to FedRolex by specifying the step size as the inverse of the number of clients and 1 respectively. FedMinOccurances randomly samples the neurons using a distribution that prioritizes neurons that occurred less in the previous round, which aims to evenly distribute training among the neurons while introducing slight randomness to combat the cyclical increases and decreases in accuracy and loss observed in FedRolex. However, FedMinOccurances also performed equally to FedRolex in most situations, but on average performed worse in high model heterogeneity situations.

Additional study of the methods investigated needs to be conducted to confirm the results of this study. A larger sample of tests is needed to definitively examine the true performance of each method, including tests with lower synchronization. Additionally, other methods could be studied such as methods that prioritize neurons that have smaller changes in weights, or methods based on entirely different approaches such as elastic averaging SGD instead of local-update SGD [2], or a dynamic pruning approach similar to FedDP [6].

Bibliography

[1] Baohao Liao, Yan Meng, and Christof Monz. Parameter-efficient fine-tuning without introducing new latency, 2023.

[2] Gauri Joshi. *Optimization algorithms for distributed machine learning*. Springer, 2023.

[3] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. 2018.

[4] Sixing Yu, J. Pablo Muñoz, and Ali Jannesari. Federated foundation models: Privacy-preserving and collaborative learning for large models, 2023.

[5] JaeYeon Park and JeongGil Ko. Fedhm: Practical federated learning for heterogeneous model deployments. *ICT Express*, 2023.

[6] Sixing Yu, Phuong Nguyen, Ali Anwar, and Ali Jannesari. Heterogeneous federated learning using dynamic model pruning and adaptive gradient, 2023.

[7] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction, 2023.

[8] Hanhan Zhou, Tian Lan, Guru Venkataramani, and Wenbo Ding. Every parameter matters: Ensuring the convergence of federated learning with dynamic heterogeneous models reduction, 2023.

[9] Dingzhu Wen, Ki-Jun Jeon, and Kaibin Huang. Federated dropout – a simple approach for enabling federated learning on resource constrained devices, 2022.

[10] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[11] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[12] Weiming Zhuang, Chen Chen, and Lingjuan Lyu. When foundation model meets federated learning: Motivations, challenges, and future directions, 2024.

[13] Nikhil Kandpal, Brian Lester, Mohammed Muqeeth, Anisha Mascarenhas, Monty Evans, Vishal Baskaran, Tenghao Huang, Haokun Liu, and Colin Raffel. Git-theta: A git extension for collaborative development of machine learning models, 2023.

[14] Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. Fwdllm: Efficient fedllm using forward gradient, 2024.

[15] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Federated learning versus classical machine learning: A convergence comparison, 2021.

[16] Yuyang Deng, Mohammad Mahdi Kamani, Pouria Mahdavinia, and Mehrdad Mahdavi. Distributed personalized empirical risk minimization. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 70812–70846. Curran Associates, Inc., 2023.

Appendices

A.1 Seed 2 Results

A.1.1 Low Synchronization

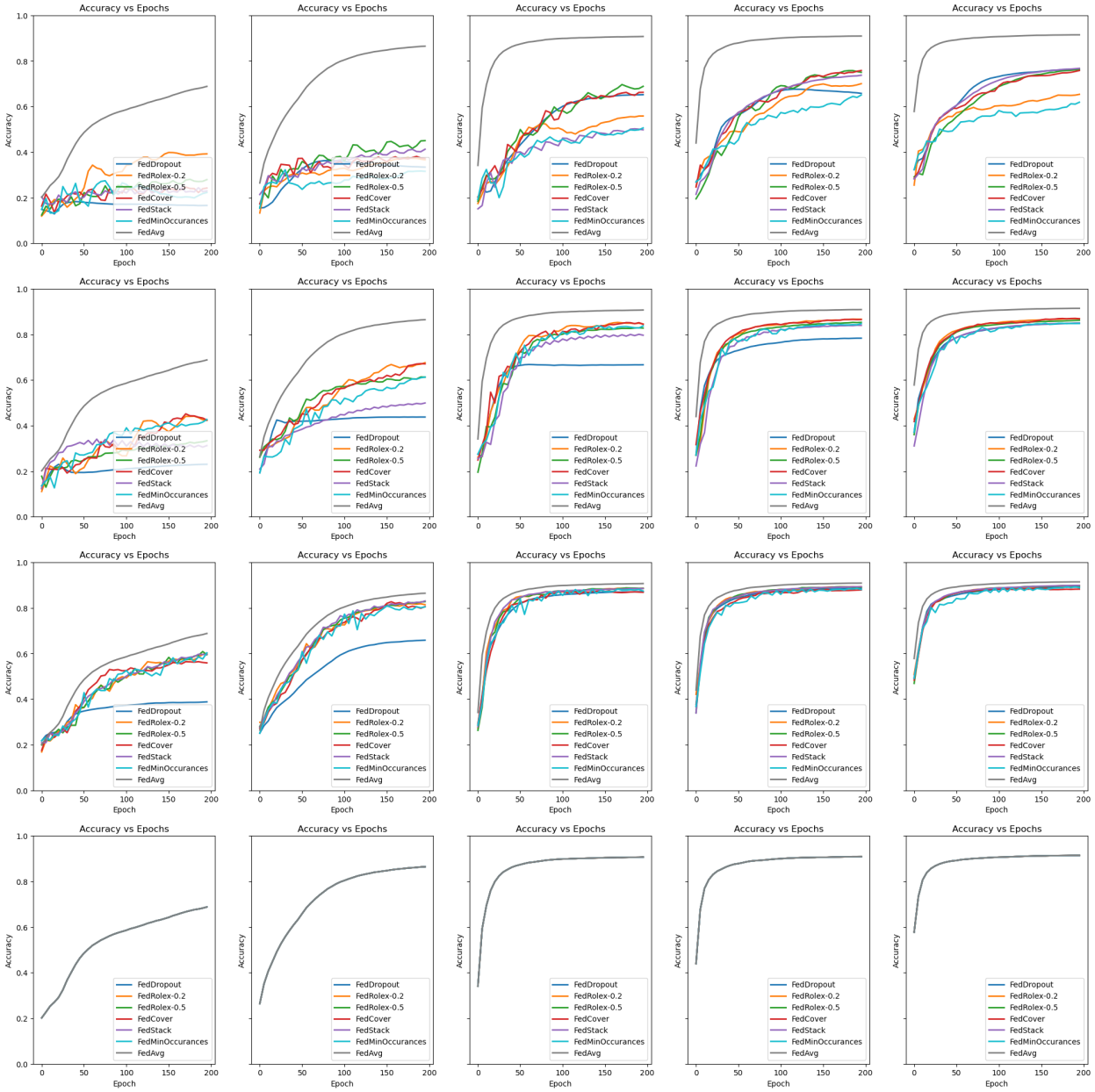

Figure A.1.1: Seed 2 Accuracy Summary with Low Synchronization

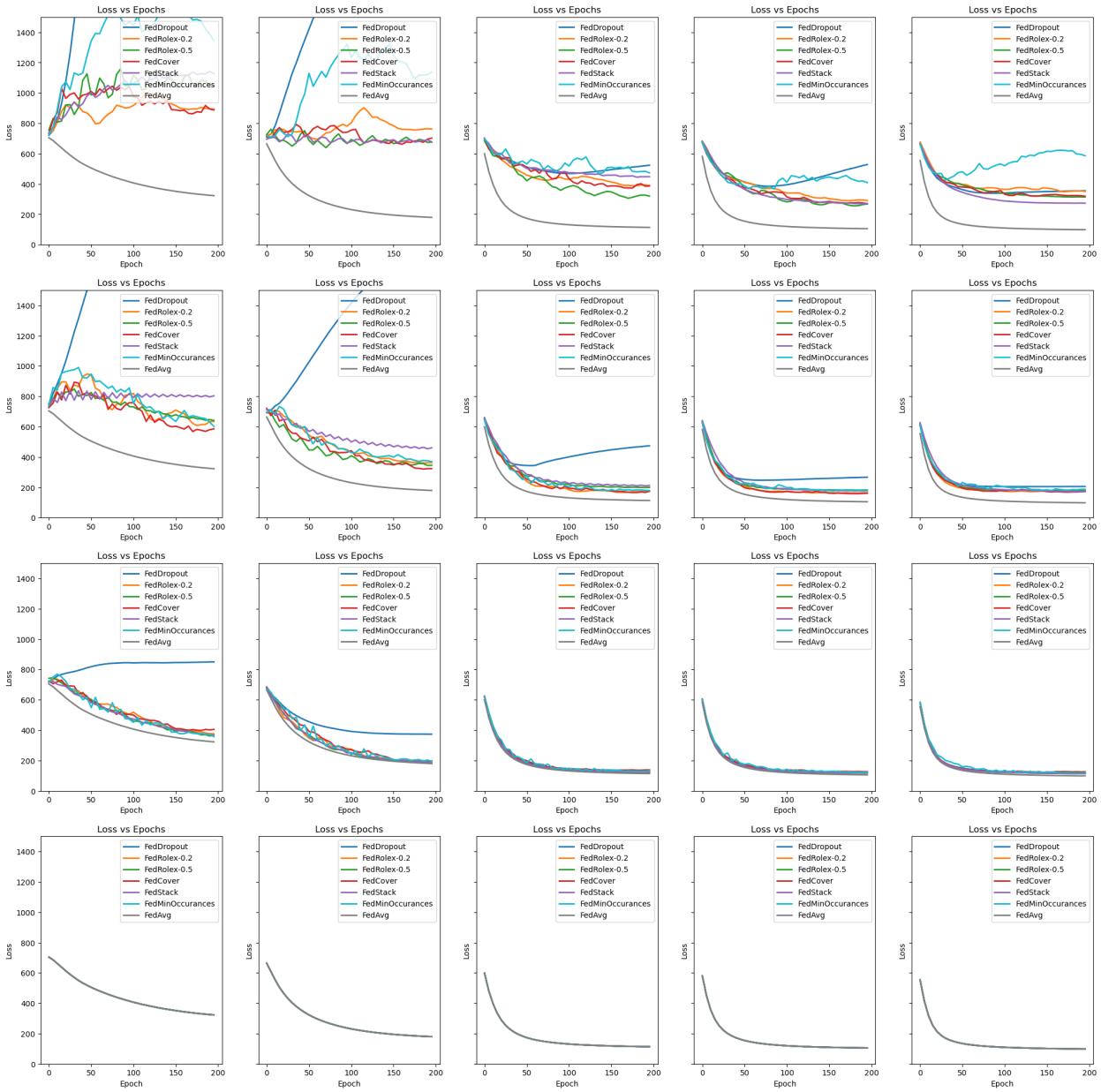

Figure A.1.2: Seed 2 Loss Summary with Low Synchronization

A.1.2 High Synchronization

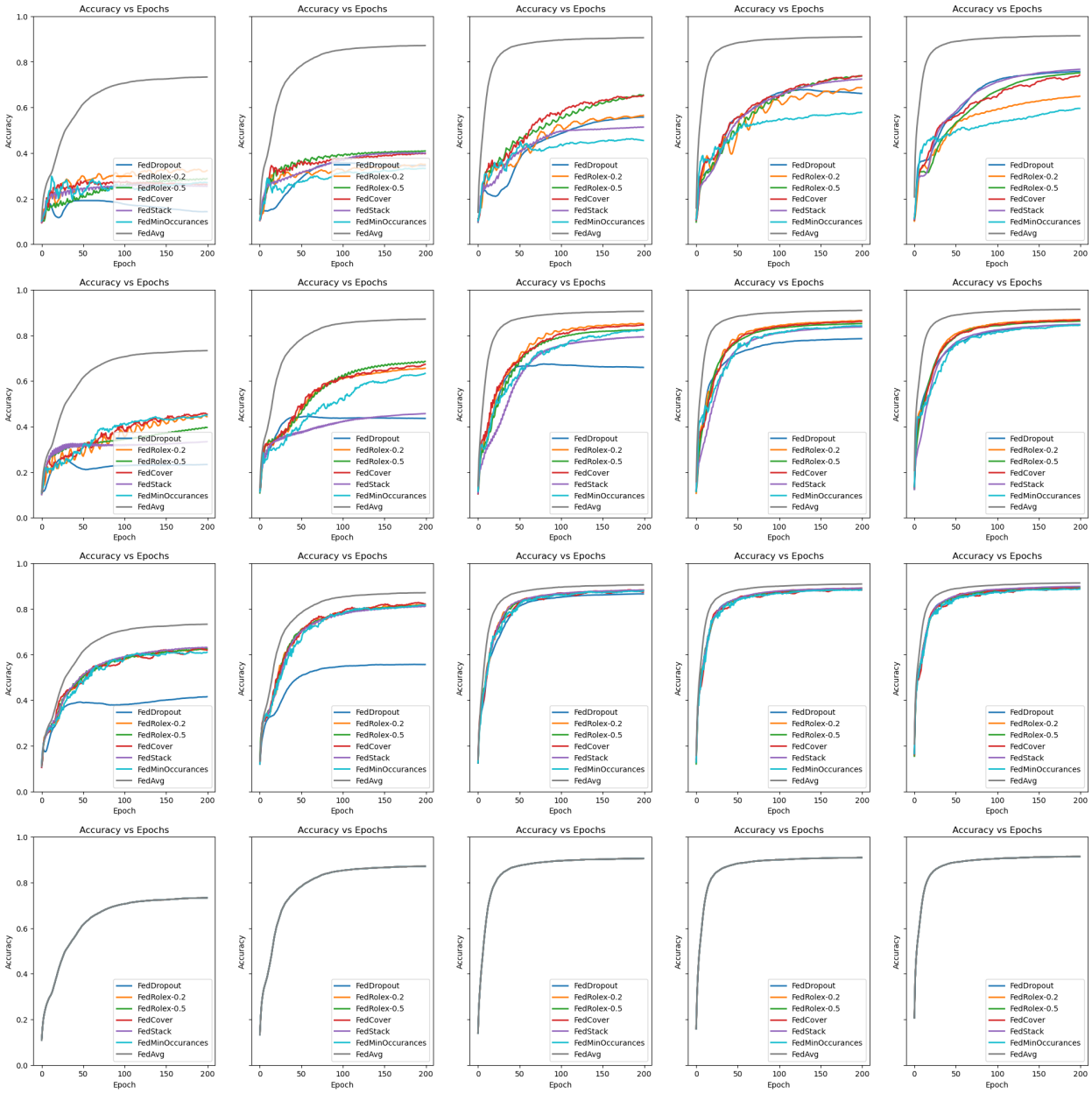

Figure A.1.3: Seed 2 Accuracy Summary with High Synchronization

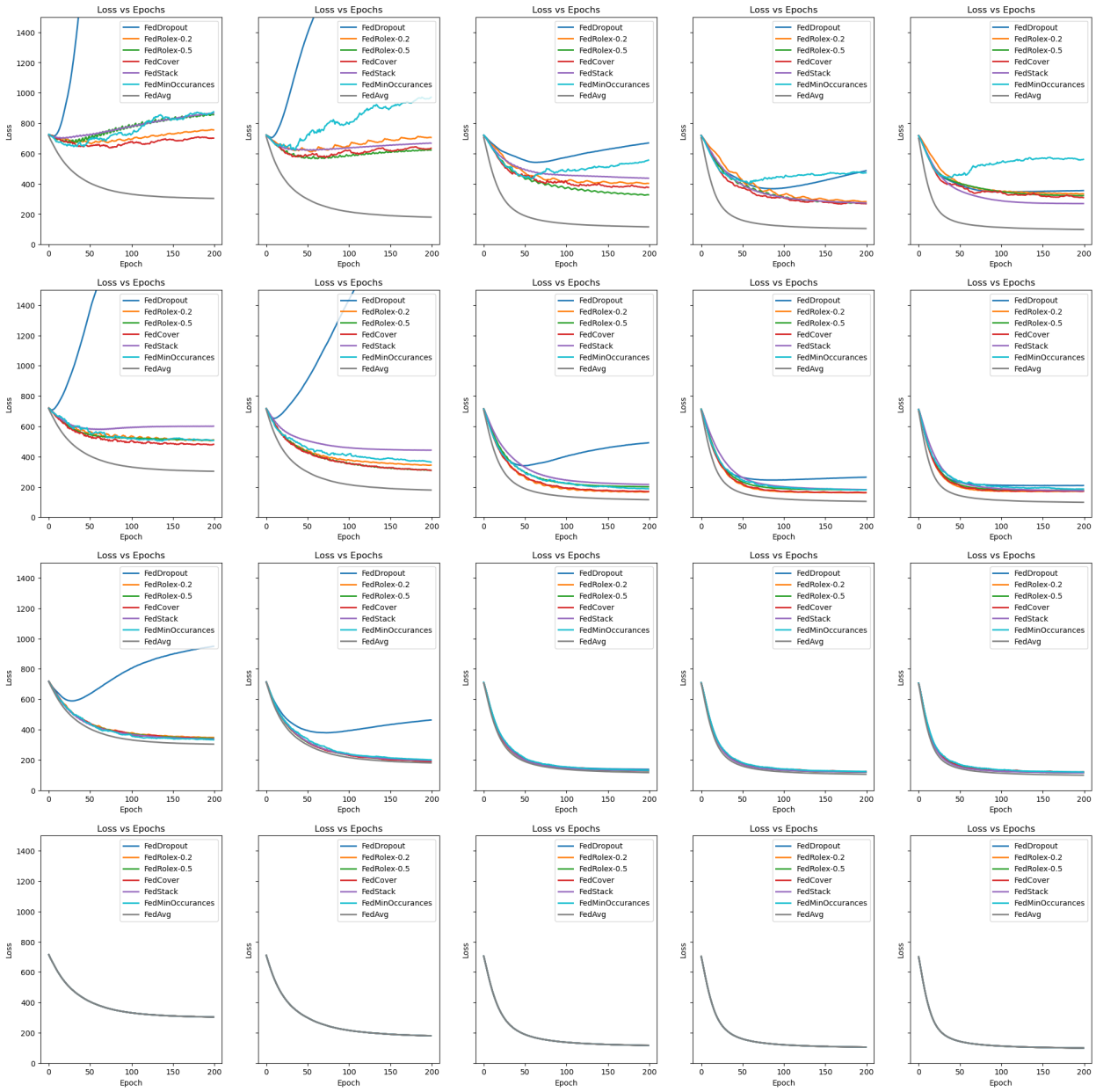

Figure A.1.4: Seed 2 Loss Summary with High Synchronization

A.2 Seed 3 Results

A.2.1 Low Synchronization

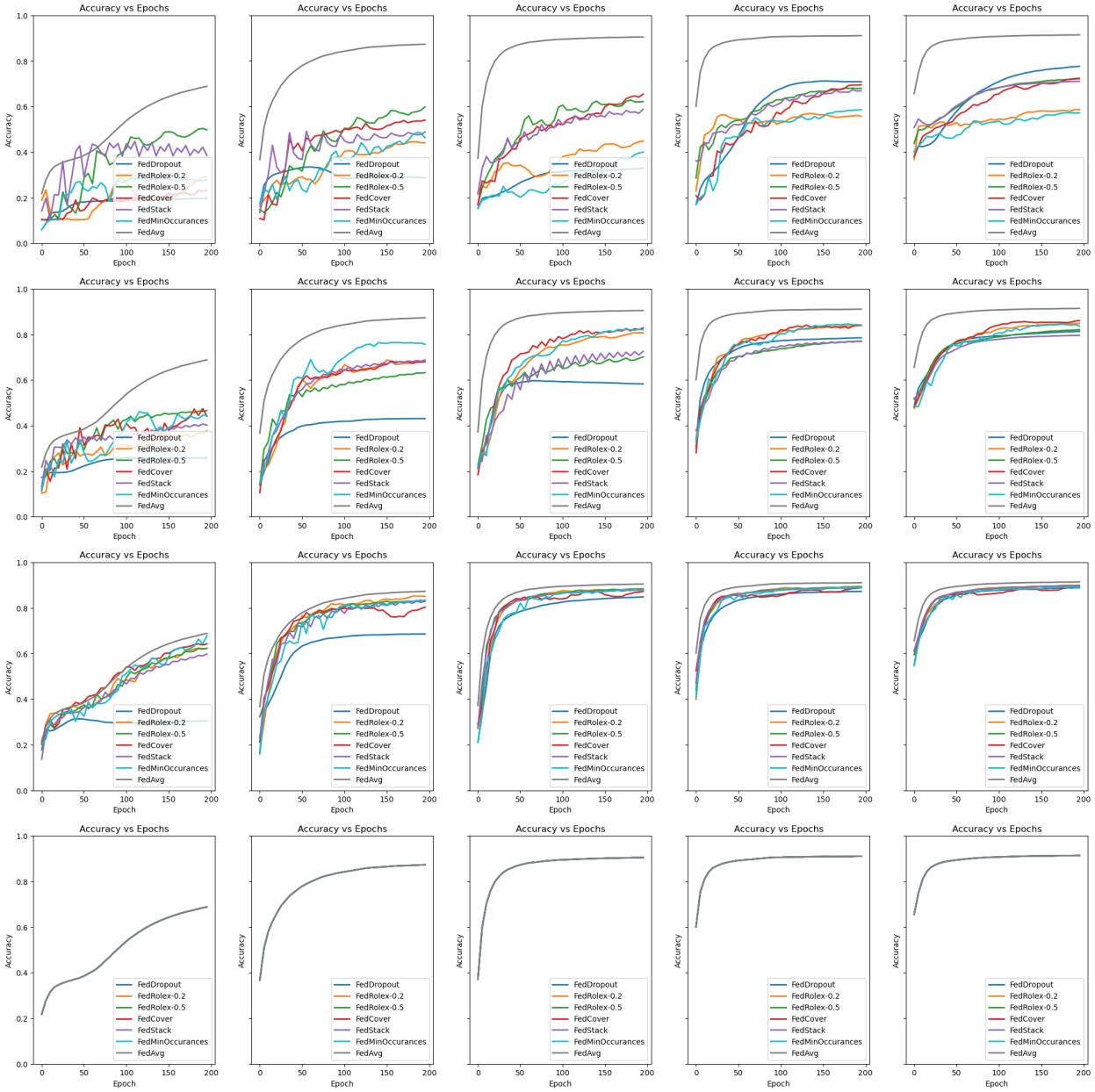

Figure A.2.1: Seed 3 Accuracy Summary with Low Synchronization

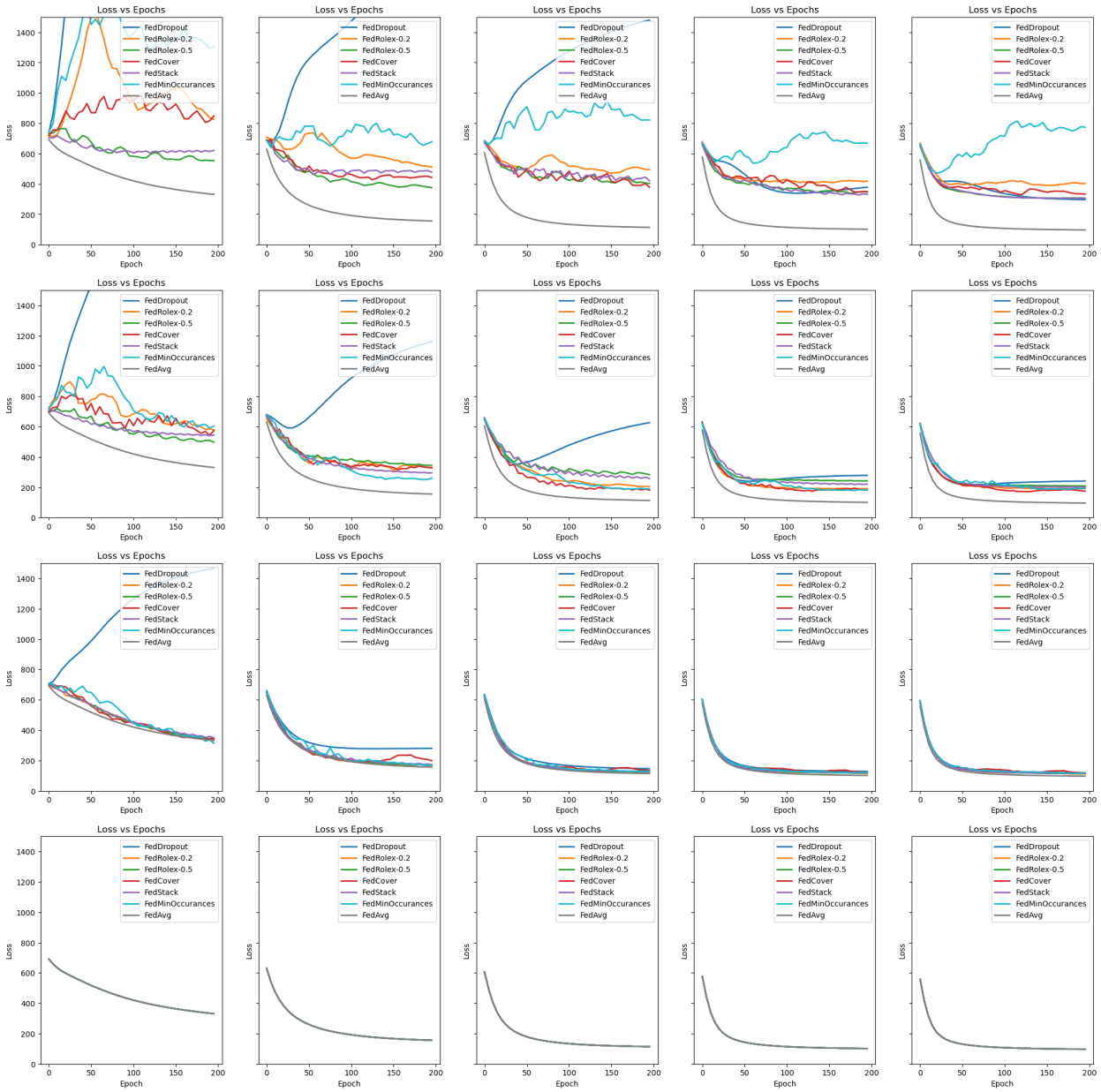

Figure A.2.2: Seed 3 Loss Summary with Low Synchronization

A.2.2 High Synchronization

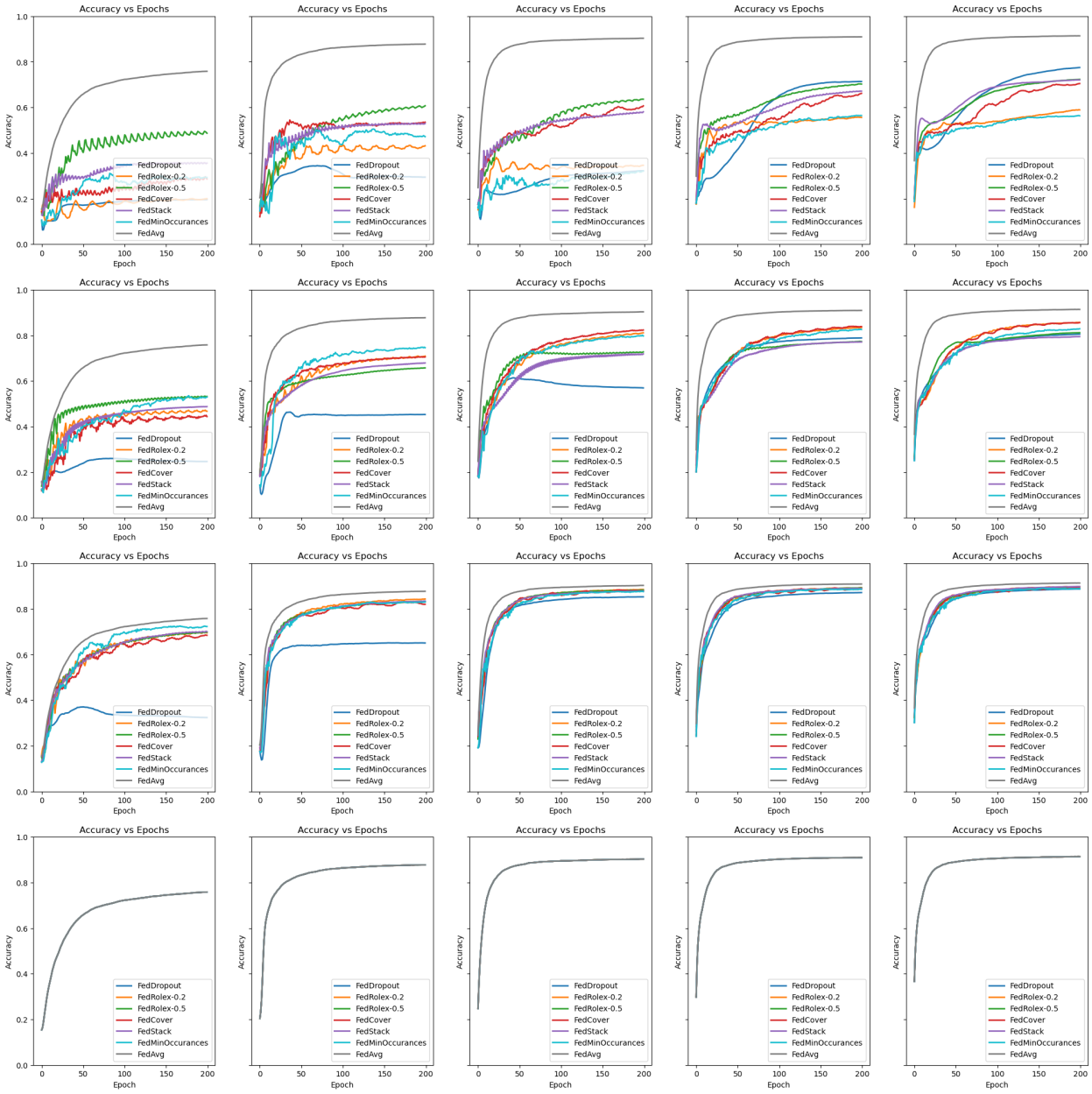

Figure A.2.3: Seed 3 Accuracy Summary with High Synchronization

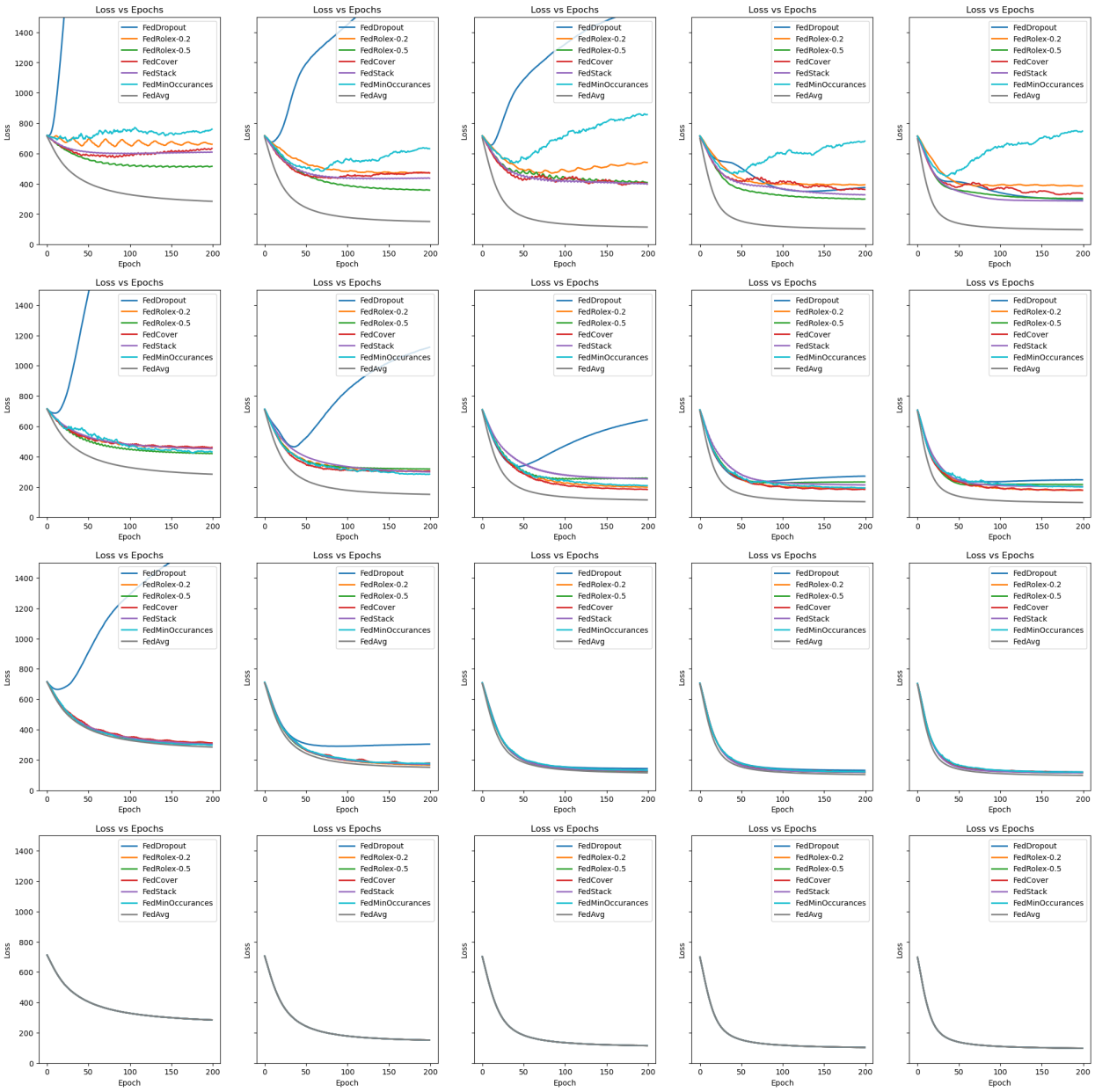

Figure A.2.4: Seed 3 Loss Summary with High Synchronization

Jason Swope

EDUCATION

The Pennsylvania State University, University Park Expected Graduation: May 2024
Schreyer Honors College
Bachelor of Science in Computer Science

TECHNICAL SKILLS

Java, Python, C, C++, PyTorch, TensorFlow, Git, Latex, Object Oriented Design, Unix/Linux, Computer Vision, Verilog, SQL, JDBC, HTML, MIPS Assembly Language

RELEVANT COURSES

Machine Learning and Algorithmic AI	Artificial Intelligence
Fundamentals of Computer Vision	Introduction to Neural Networks
Object Oriented Programming	Introduction to Systems Programming
Introduction to the Theory of Computation	Data Structures and Algorithms
Operating Systems Design & Construction	Technical Writing

EXPERIENCE

AI/ML Engineering Intern May 2023 – August 2023
Lockheed Martin Space – King of Prussia, PA
- Trained and tested machine learning models on timeseries data
- Developed systems for evaluating the performance of each model using relevant metrics
- Worked in a live team environment following an agile methodology

Calculus III Grader August 2021 – December 2022
The Pennsylvania State University – University Park, PA
- Grade assignments and provide individual feedback to students for three sections of Calculus of Several Variables
- Coordinate with professors regarding expectations and areas of improvement for the students

Customer Service Associate June 2022 - August 2022
Lowe's – Warrington, PA
- Delivered quality customer services and maintained store organization
- Assisted customers and staff in moving purchases safely and loading contractor orders

Research Intern September 2018 - May 2020
Baruch S. Blumberg Institute and Conifer Point Pharmaceuticals – Doylestown, PA
- Conducted research and collaborated with established scientists and software developers
- Calculated and studied clusters of water molecules on interaction surfaces of proteins to be applied in drug discovery
- Used convolutional neural networks to visually distinguish between healthy cells and cells infected with Hepatitis B

HONORS & AWARDS

- The Evan Pugh Scholar Senior Award – Top 0.5% of the senior class Spring 2023
- President Sparks Award – GPA of 4.00 as a sophomore Spring 2022
- President Walker Award – GPA of 4.00 as a freshman Spring 2021
- Order of the Eastern Star Educational Scholarship Spring 2022
  - Members and relatives who embody good character and strive for improvement
- Masonic Education Scholarship Summer 2020
  - Members and relatives committed to community service pursuing higher education

LEADERSHIP & ACTIVITIES

- Penn State Robotics Club August 2021 – May 2023
- Nittany Data Labs - Data Sciences August 2021 - December 2022
- Google Developer Student Club, Penn State Chapter August 2021 - December 2022
- Action Karate, 3rd Degree Black Belt September 2007 - July 2020