Analysis Report

Ghazal Ebrahimi

Overview

This document explores the non-IID data challenge in federated learning by following the steps outlined in the assignment. It begins with an explanation of the problem, followed by detailed descriptions of the implementation for each task. The final section presents and analyzes the experimental results. All source code is available on GitHub, along with a `.yml` environment file and a `README.md` containing setup instructions and usage guidelines.

Introduction

Federated learning is a machine learning paradigm where multiple clients collaboratively train a model using their own local datasets without sharing raw data. Most federated learning algorithms assume that data is independent and identically distributed (IID) across clients. However, in real-world scenarios, this assumption rarely holds. In practice, client datasets are often non-IID, meaning they are not drawn from the same distribution and may not be representative of the overall data distribution.

The non-IID data problem is one of the most significant challenges in federated learning, as it can greatly reduce the performance of global models. One of the foundational algorithms in this field is Federated Averaging (FedAvg), introduced by McMahan et al. in their paper *"Communication-Efficient Learning of Deep Networks from Decentralized Data."* FedAvg works by averaging the parameters of locally trained client models, with each client's contribution weighted by the size of its dataset.

In this assignment, we simulate a federated learning setup with two clients using the MNIST training dataset. We create two subsets of the dataset with imbalanced label distributions to represent a scenario of high label distribution skew. A custom model is trained independently on each client's subset, and then a simplified version of the FedAvg algorithm is applied by averaging the saved model weights.

To address the potential limitations of standard FedAvg under label imbalance, we propose and implement two modifications to the aggregation method. Finally, we evaluate all trained models, present the experimental results, compare model performances, and provide a comprehensive analysis of the impact of label skew and aggregation strategy on model effectiveness.

Implementation

I first downloaded and saved the MNIST training dataset. Then, I created a highly skewed non-IID split of the dataset into two clients:

- Client 1 will have 80% of digits 0-4 and 20% of digits 5-9
- Client 2 will have 20% of digits 0-4 and 80% of digits 5-9

To do that, I iterated over each class, and using the given weights for class distribution, I randomly selected a subset of data points for each class to assign to one client. The remaining data points from that class were then assigned to the other client. This way, I ensured that each class was split between the clients according to the specified proportions, resulting in a controlled label-skewed distribution across the two clients. Figure 1 shows the data distribution across classes for each resulting split.

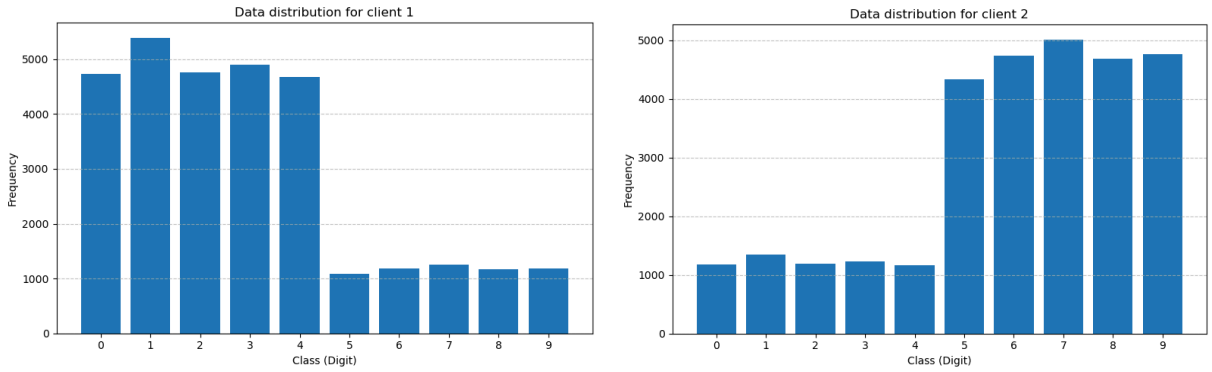

Figure 2. Clients datasets class distributions

For the next step, I implemented a simple CNN model consisting of one convolutional layer followed by a pooling layer and two fully connected layers, as shown in the below code snippet:

```
self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3) # 1 × 28 × 28 -> 16 x 28 x 28
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2) # -> 16 x 13 x 13
        self.fc1 = nn.Linear(2704, 128) # 2704 -> 128
        self.fc2 = nn.Linear(128, 10)  # -> 10
```

I then implemented a training script that accepts a client ID as an input argument and trains an instance of the classifier on the corresponding client-specific data split prepared in the previous step. I used the Adam optimizer and the cross entropy loss for training. Models are trained for 15 epochs using a batch size of 64, and the resulting model parameters are saved in corresponding `.pth` files. Using this script, I trained the model separately on both client datasets and saved the learned parameters. Additionally, I trained the same model architecture on the original MNIST training dataset without any label skew or client split. This model serves as a baseline for comparison and is referred to as the *base model* throughout the rest of the document. The plots in Figure 2 illustrate loss and accuracy curves for clients 1 and 2 during training.

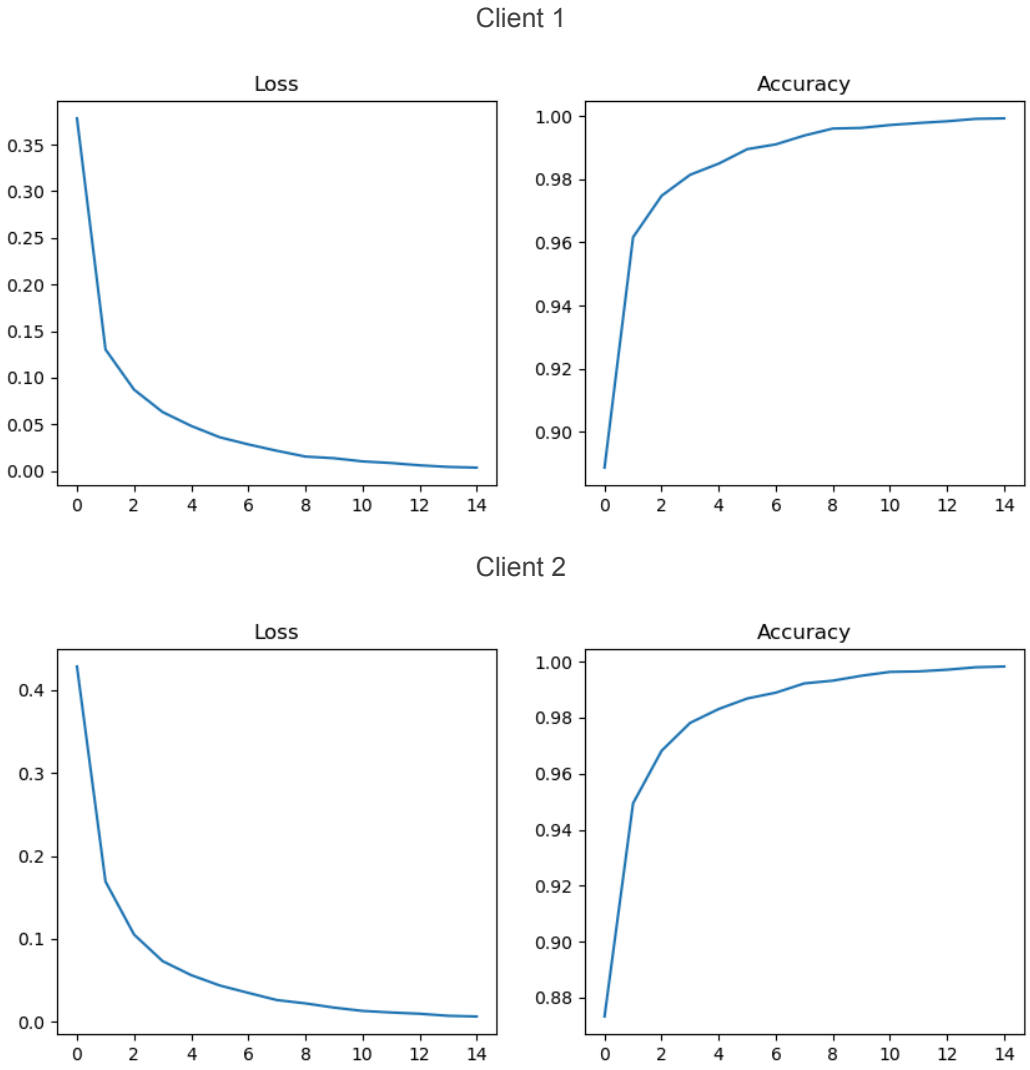

Figure 2. Loss and accuracy curves for training over the number of epochs

Next, I implemented a simplified version of the standard FedAVG algorithm based on the following aggregation rule provided in the document:

$$\theta_{t+1} = (n_1/n) \times \theta_1 + (n_2/n) \times \theta_2 + \ldots + (n_k/n) \times \theta_k$$

Since this setup involves only 2 clients with an equal number of data points, I simplified the weighted averaging to:

$$\theta = 0.5 \times \theta_1 + 0.5 \times \theta_2$$

I loaded the trained model parameters from each client, verified parameter compatibility, and performed element-wise averaging across all parameters. The resulting global model parameters were saved in a separate file for later evaluation.

To specifically address the label distribution skew challenge in this assignment and potentially improve the global model's performance, I explored two modified versions of the FedAvg algorithm:

Approach 1: Averaging All Layers Except Classifier

Instead of averaging the entire model, I averaged only the feature extraction layers and selected the classifier layer from one of the clients. This avoids degrading class-specific representations learned by each client.

```
for key in state_dict1:
        if "fc2" in key:
            # pick from better client
            averaged_state_dict[key] = state_dict1[key].clone()
        else:
            averaged_state_dict[key] = 0.5 * state_dict1[key] + 0.5 * state_dict2[key]
```

Approach 2: Class-wise Weighted Averaging for Classifier

For the classifier layer, I used a custom averaging strategy where the weights for each class output are averaged, with more weight given to the client that saw that class more frequently. For instance, weights for classes 0–4 were biased toward Client 1, and weights for classes 5–9 were biased toward Client 2.

```
for i in range(weight.size(0)):  # 10 output classes
        if i < split_index:
            w = w_client1  # more trust in client 1
        else:
            w = 1 - w_client1  # more trust in client 2

        weight[i] = w * fc1.weight[i] + (1 - w) * fc2.weight[i]
        bias[i] = w * fc1.bias[i] + (1 - w) * fc2.bias[i]
```

In the following section, the experimental results and model evaluations are presented, followed by a discussion and comparison of the different approaches implemented in this assignment.

Results and Analysis

In this section, the experimental results based on the previous steps are presented. Tables 1 and 2 display the classification accuracy of each client model on its own training split and on the other client's training split, respectively. Table 3 presents the accuracy of all models on the shared MNIST test dataset. Following the tables, confusion matrices are provided to further

illustrate the performance of the client models and the standard FedAvg model on the shared test data. The rest of the document analyzes these results in detail.

All results are based on models trained for 15 epochs. To ensure that overfitting did not distort the analysis, I also trained the models for 5 epochs and compared the outcomes. The results and conclusions remained consistent, indicating that the number of training epochs did not significantly impact the overall trends or insights.

As shown in Table 1, both clients achieve very high accuracy on their respective training splits, demonstrating that each model has effectively learned the distribution of its local data. However, Table 2 shows that when evaluated on the other client's data, each model exhibits a performance bias. Specifically, Client 1 performs better on classes 0–4, while Client 2 performs better on classes 5–9. This outcome is consistent with expectations, as Client 1's data is skewed toward the first half of the digit classes, and Client 2's data is skewed toward the second half.

Table 1. Accuracy (%) on own training split

Model	0	1	2	3	4	5	6	7	8	9	Total
Client 1	99.98	100	100	99.98	99.96	99.82	100	100	99.91	99.58	99.96
Client 2	99.92	100	99.83	99.02	99.91	99.88	99.96	100	99.91	99.94	99.90

Table 2. Accuracy (%) on the other client’s training split

Model	0	1	2	3	4	5	6	7	8	9	Total
Client 1	99.58	99.48	98.99	98.86	98.89	96.38	98.06	96.63	95.53	93.68	96.70
Client 2	96.90	98.11	95.20	92.84	96.34	97.88	99.15	99.12	98.80	98.91	96.47

Moving on to Table 3, which presents the models' performance on the shared MNIST test dataset, we observe that both client models generalize reasonably well to the global distribution. However, each model shows a slight bias toward the classes that were more frequent in its respective training data. Specifically, Client 1 performs better on digits 0–4, while Client 2 performs better on digits 5–9. This indicates that the label distribution skew continues to influence the performance of the local models, even when evaluated on a balanced test set.

Table 3. Accuracy (%) on shared test set (MNIST test dataset)

Model	0	1	2	3	4	5	6	7	8	9	Total
Client 1	99.80	99.65	99.03	99.70	99.19	97.42	96.97	96.89	96.10	92.96	97.80
Client 2	97.14	97.97	95.54	94.85	96.54	98.99	98.75	99.03	98.36	98.61	97.56
FedAvg 0	98.88	99.65	97.19	98.81	99.39	97.20	97.91	98.54	97.84	95.54	98.12
FedAvg 1	98.16	98.68	95.16	92.08	97.56	96.52	98.43	99.51	98.56	93.95	96.87
FedAvg 2	99.29	99.47	98.26	98.32	99.08	97.98	97.29	98.44	96.71	96.73	**98.18**

The last three rows of Table 3 show the accuracies for three models trained using different versions of the FedAvg algorithm. The first row corresponds to the standard FedAvg implementation, where the global model is created by taking a weighted average of the parameters from the two client models. This version achieves a total accuracy of **98.12**%, which is notably high, and the per-class accuracies do not reveal any strong bias toward a specific group of digits.

Figures 3 and 4 present the confusion matrices for the client models and the standard FedAvg model, respectively. Both client models exhibit high true positive rates across most classes. However, Client 1's confusion matrix reveals a tendency to misclassify more examples as digits 0–4, while Client 2 shows the opposite pattern, with more examples misclassified as digits 5–9. These trends further confirm the impact of the label distribution skew in the clients' training data.

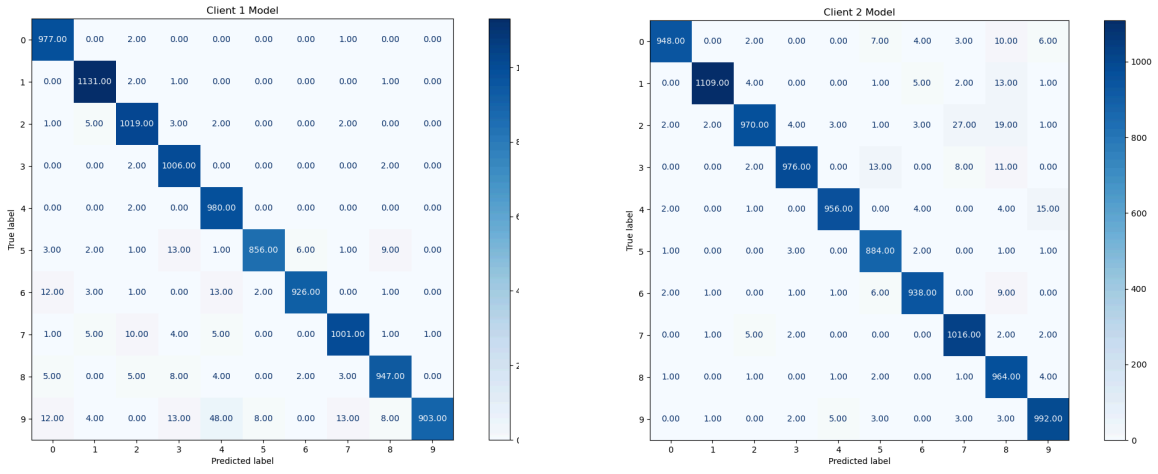

Figure 3. Confusion Matrices of locally trained models on shared test dataset

As shown in Figure 4, the confusion matrix of the standard FedAvg model displays a high rate of true positives, which aligns with the overall evaluation accuracy. Unlike the clients' matrices, the misclassification frequency here is more evenly distributed across classes, indicating a lower degree of label bias. One notable exception is the cell at row 9 and column 4, which shows a value of 43—a relatively high number compared to other cells. This suggests that the digit 9 is frequently misclassified as 4. The likely reason is the visual similarity between these two digits and their shared features, which appears to be unrelated to the biases present in the client models.

Figure 4. Confusion Matrix of standard FedAvg model on shared test dataset

Therefore, based on the results, the standard FedAvg algorithm effectively aggregates the learnings of the local models despite the label distribution skew in the datasets. The resulting model is both highly accurate and relatively unbiased.

Although standard FedAvg proved to be effective in this case, the algorithm, by definition, does not directly address label distribution skew and primarily focuses on quantity skew. It simply averages the parameters of the local models without considering label imbalance. The method assumes that all local models are optimizing toward the same global objective; however, this assumption breaks down under non-IID label distributions. In such cases, each client is learning from a different data distribution, leading to local models with divergent objectives. As a result, their parameters point in different directions in the parameter space, and the aggregated global model may perform worse than the individual local models. This issue, known in the literature as client drift, becomes more pronounced with increasing label skew and can compromise the effectiveness of FedAvg. To explore whether this could be improved, I implemented two slightly modified versions of FedAvg.

In the first approach, the classifier layer of the global model adopts the parameters from the better-performing client model, while the rest of the parameters are aggregated using the

standard FedAvg method—by averaging the corresponding parameters from both clients. This approach is based on the observation that the classifier layer is particularly sensitive to label distribution skew, as it directly maps extracted features to class predictions. When each client is exposed primarily to different subsets of labels, their classifier layers become biased toward those classes. Averaging these layers risks diluting both clients' specialized knowledge. By averaging only the feature extraction layers and preserving the classifier from one client, this method avoids merging incompatible outputs and maintains effective class-specific mappings.

Despite initial expectations, this approach did not improve model performance and even slightly degraded it. A possible explanation is that, in this case, the label skew does not lead to significantly divergent parameter spaces—as evidenced by the strong performance of the standard FedAvg model. Therefore, discarding one client's learned classifier parameters may have weakened the model without offering a meaningful benefit in terms of mitigating client drift.

In the second approach, rather than averaging all classifier weights equally, the method gives more influence to the client with greater representation for each class. Since Client 1 performs better on digits 0–4 and Client 2 on digits 5–9, soft-averaging allows each client to contribute more to the classes they specialize in. This preserves useful class-specific knowledge and results in a more balanced and accurate classifier. According to the experiments, this approach produced the most accurate model overall, outperforming Client 1, Client 2, and both of the other aggregated models on the MNIST test dataset, with a classification accuracy of **98.18**%.

Conclusion

Label distribution skew is a critical challenge that can severely impact the performance of federated models if left unaddressed; however, there is a growing set of techniques designed to mitigate its effects. FedAvg is a known and effective technique for addressing non-IID challenges and was shown to be effective in some label distribution skew scenarios, as demonstrated in the course of this assignment. However, this algorithm simplifies the problem of non-IID datasets and only focuses on quantity skew. This limitation might negatively affect the performance of the aggregated model in extreme cases of label imbalance. There are other techniques to modify the FedAvg algorithm in ways that address label distribution skew as well. In this assignment, two modifications to FedAvg were explored but did not show significant impacts.