Dependable AI Assignment-3

Susim Mukul Roy - B20AI043

Question 1

Dataset

We have to operate on 3 datasets, namely MNIST, Coloured MNIST and SVHN. The MNIST dataset has 50k training samples and 10k testing samples. The Coloured MNIST also has the same no. of training and testing samples except that the images are now coloured, that is, in RGB. The SVHN dataset has 73257 digits for training and 26032 digits for testing. Each of the three datasets has 10 classes of images which are digits in the range of 0-9. Hence the task is image classification.

We are to select 10k samples(with 1000 samples from each class) as our training data for each type of dataset and 5k samples(with 500 samples from each class) as our testing data for each type of dataset. This is to ensure that our data is IID.

Approach

I used the Resnet34 architecture with pretrained weights as the backbone for classification with just changing the classification head. Next, we perform image classification in a federated learning setup and perform the aggregation using Federated Averaging (FedAvg). For our optimizer, we used Stochastic Gradient Descent (SGD). For our federated setup, we choose a fixed pool of 3 clients, and the same clients are sampled at every round. The number of rounds is 15 and the number of epochs per round is 5. The server is then aggregated after the clients are trained per round. We describe our results in the following sections, answering each part of the assignment paper.

Configurations

The configurations used in solving are:

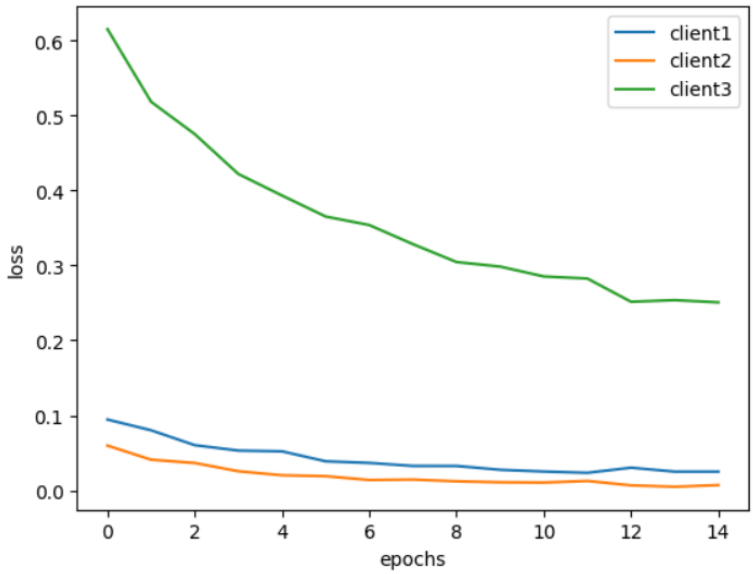
- Batch size: 64 for each dataloader
- Epochs: 5
- No. of rounds: 15

- Loss-function: Cross-Entropy loss
- Optimizer: SGD with lr = 0.01
- Client 1: MNIST
- Client 2: Coloured MNIST
- Client 3: SVHN

Results

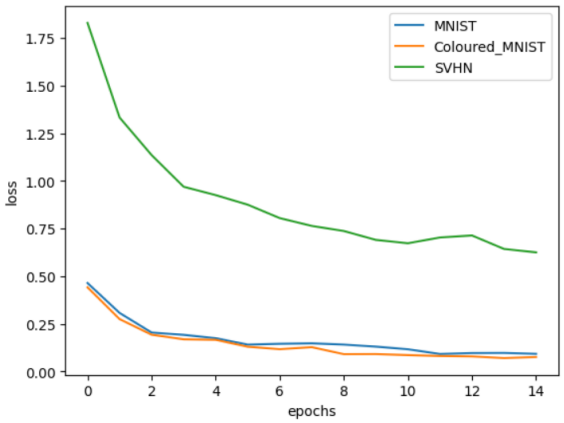
Visualization

In order to visualise the change in the loss values on the client side, I provide the following visualization:

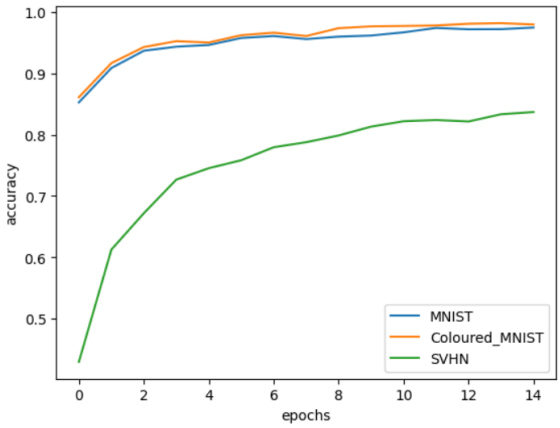


Thus, we see that the loss decreases on the clients over the epochs.

In order to visualise the change in the loss values and the increase of the accuracies on the server side, I provide the following visualization:



loss curve



accuracy curve

Thus, we see that the average accuracy of the central server is increasing over the epochs which shows that it is learning properly.

Numerical Data

The class-wise accuracy on the client side are:

- Client 1

```
tensor([0.9818, 0.9845, 0.9673, 0.9621, 0.9793, 0.9636, 0.9708, 0.9663, 0.9879,
        0.9689])
```

- Client 2

```
tensor([0.9896, 0.9720, 0.9881, 0.9857, 0.9739, 0.9775, 0.9959, 0.9778, 0.9801,
        0.9741])
```

- Client 3

```
Class-wise accuracy of client 3
100%| 313/313 [00:04<00:00, 70.49it/s]tensor([0.0020, 0.9779, 0.0100, 0.1708, 0.1548, 0.0096, 0.0085, 0.0373, 0.2786,
        0.0144])
```

The class-wise accuracy on the server side are:

- MNIST

```
tensor([0.9938, 0.9885, 0.9572, 0.9759, 0.9798, 0.9574, 0.9801, 0.9601, 0.9796,
        0.9619])
```

- Coloured-MNIST

```
tensor([0.9864, 0.9806, 0.9793, 0.9861, 0.9839, 0.9690, 0.9886, 0.9742, 0.9817,
        0.9937])
```

- SVHN

```
class-wise accuracy of server for SVHN
100%|██████████| 313/313 [00:04<00:00, 65.78it/s]tensor([0.9223, 0.8860, 0.8542, 0.7056, 0.8553, 0.8696, 0.8198, 0.8804, 0.8455,
        0.7982])
```

This is an interesting observation in case of SVHN. On the client side, we see very low accuracies. However, the server side has high accuracies. This is because the client is a standalone system and it is only learning from SVHN dataset. However, the server is learning from MNIST and coloured MNIST too. Hence, the server knows the RGB forms of the labels as it has high accuracies on MNIST and coloured-MNIST. Thus, due to this additional knowledge on the server side, we see an increased accuracy on SVHN on the server side.

The overall-accuracy and the corresponding confusion matrix on the client sides are:

- Client 1
  - Accuracy: 97.42%
  - Confusion Matrix

```
tensor([[494.,   0.,   0.,   0.,   1.,   0.,   1.,   0.,   0.,   0.],
        [  0., 526.,   0.,   2.,   2.,   0.,   0.,   2.,   0.,   0.],
        [  3.,   2., 505.,   1.,   1.,   2.,   4.,   1.,   0.,   0.],
        [  1.,   0.,   0., 500.,   0.,   6.,   0.,   0.,   2.,   1.],
        [  1.,   3.,   1.,   0., 502.,   0.,   2.,   1.,   5.,   9.],
        [  0.,   2.,   4.,   5.,   1., 454.,   0.,   1.,   0.,   0.],
        [  5.,   1.,   1.,   3.,   0.,   2., 483.,   1.,   0.,   0.],
        [  2.,   8.,   6.,   1.,   0.,   4.,   0., 465.,   0.,   1.],
        [  1.,   0.,   2.,   4.,   0.,   3.,   1.,   1., 467.,   1.],
        [  0.,   0.,   3.,   2.,   4.,   1.,   0.,   2.,   2., 475.]])
```

- Client 2
  - Accuracy: 98.08%
  - Confusion Matrix

```
tensor([[466.,   0.,   0.,   0.,   0.,   1.,   2.,   0.,   1.,   0.],
        [  4., 482.,   0.,   0.,   1.,   2.,   1.,   2.,   0.,   0.],
        [  0.,   1., 502.,   3.,   0.,   1.,   0.,   1.,   1.,   0.],
        [  1.,   0.,   2., 497.,   0.,   3.,   0.,   1.,   2.,   2.],
        [  1.,   3.,   1.,   0., 515.,   0.,   1.,   0.,   1.,   5.],
        [  1.,   0.,   1.,   2.,   0., 472.,   5.,   2.,   0.,   2.],
        [  0.,   0.,   0.,   0.,   0.,   3., 514.,   0.,   0.,   0.],
        [  1.,   4.,   3.,   0.,   0.,   1.,   0., 495.,   1.,   2.],
        [  0.,   0.,   0.,   2.,   0.,   2.,   2.,   0., 492.,   5.],
        [  1.,   1.,   1.,   1.,   6.,   0.,   0.,   1.,   2., 469.]])
```

- Client 3
  - Accuracy: 18.42%
  - Confusion Matrix

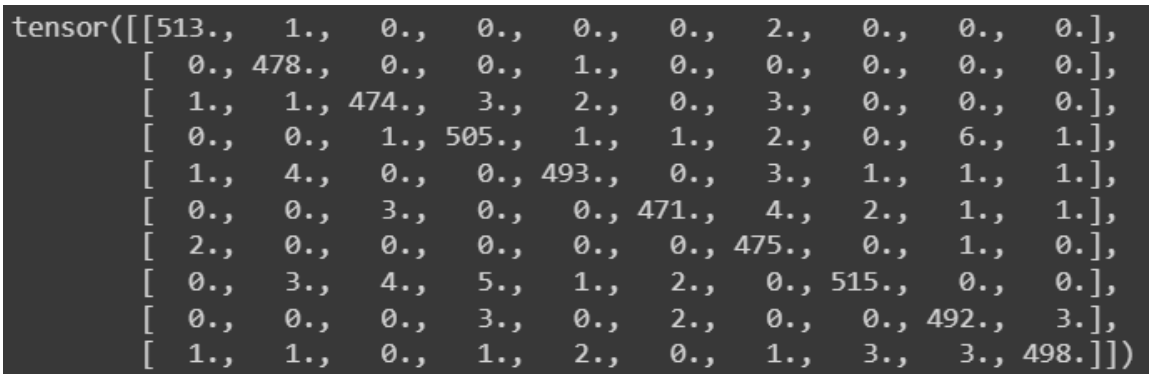
```
tensor([[  2., 453.,   0.,  37.,   2.,   0.,   0.,   0.,   6.,   0.],
        [  0., 557.,   0.,   1.,   2.,   0.,   0.,   2.,   0.,   0.],
        [  0., 280.,   5., 137.,  56.,   0.,   0.,   0.,  27.,   0.],
        [  0., 364.,   0.,  95.,  28.,   0.,   0.,   0.,  46.,   0.],
        [  0., 330.,   1.,  53.,  85.,   0.,   0.,   0.,   8.,   0.],
        [  0., 221.,   0., 213.,  40.,   5.,   1.,   0.,  17.,   0.],
        [  0., 312.,   0.,  94.,  50.,   0.,   5.,   1.,  28.,   0.],
        [  0., 426.,   0.,  58.,   2.,   0.,   0.,  16.,   4.,   0.],
        [  0., 278.,   0.,  33.,  24.,   0.,   0.,   0., 144.,   0.],
        [  0., 310.,   4.,  55.,  29.,   1.,   0.,   2.,  43.,   7.]])
```

The dataset-wise accuracy, confusion matrix and the overall accuracy on the server side are:

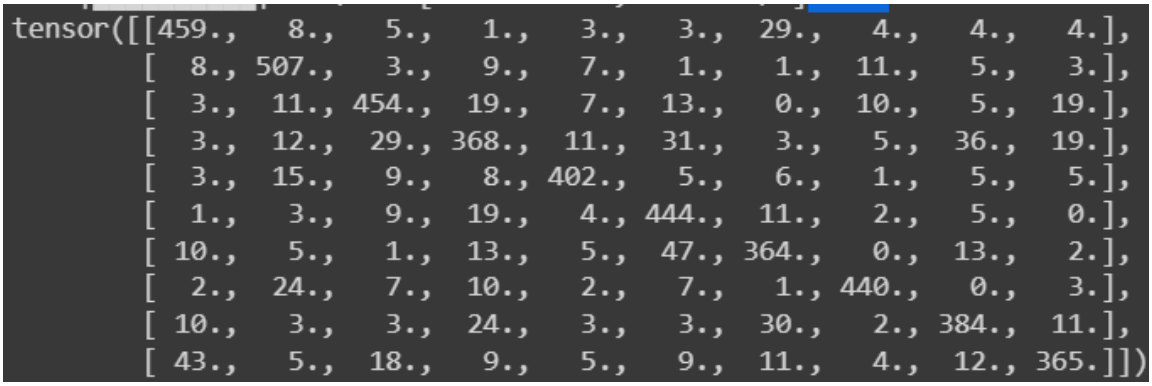
- MNIST
  - Accuracy: 96.88%
  - Confusion Matrix

```
tensor([[511.,   0.,   0.,   0.,   1.,   0.,   1.,   0.,   1.,   0.],
        [  2., 483.,   0.,   3.,   2.,   0.,   5.,   1.,   0.,   0.],
        [  0.,   2., 471.,   4.,   4.,   3.,   7.,   5.,   2.,   0.],
        [  1.,   0.,   6., 466.,   0.,   5.,   0.,   0.,   3.,   2.],
        [  1.,   2.,   1.,   0., 499.,   0.,   0.,   0.,   1.,   7.],
        [  1.,   0.,   8.,   4.,   0., 464.,   5.,   2.,   0.,   1.],
        [  4.,   1.,   1.,   2.,   1.,   1., 474.,   0.,   1.,   0.],
        [  0.,   4.,   7.,   2.,   1.,   2.,   0., 475.,   0.,   1.],
        [  2.,   0.,   1.,   3.,   0.,   2.,   3.,   0., 492.,   3.],
        [  2.,   2.,   3.,   1.,   4.,   2.,   1.,   2.,   4., 509.]])
```

- Coloured-MNIST
  - Accuracy: 98.28%
  - Confusion Matrix



- SVHN
  - Accuracy: 83.74%
  - Confusion Matrix



The overall accuracy is **92.96%.**

Mathematical Explanation

Federated Averaging (FedAvg) is an algorithm used in federated learning toaggregate model updates from multiple clients. In FedAvg, each client trains themodel locally on their own dataset, and then sends the model updates to the server.The server then aggregates these updates to produce a new global model. Insteadof simply averaging the updates, FedAvg uses a weighted average based on

thenumber of local updates performed by each client. It is formally given by the following:

Learning rate η total samples n , total clients K, samples on a client n_K, client fraction C

- In a round t:
  - The central server broadcasts current model w_t to each client, each client k computes gradient g_k = $\nabla F_k(w_t)$, on it's local data.
  - Each client k computes for E epoch: $w_{t+1}^k \leftarrow w_t - \eta g_k$
  - The central server performs aggregation: $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$
  - Suppose B is the local mini batch size, number of updates on client k in each round: $u_k = E\frac{\eta_k}{B}$.

Thus, the formula for FedAvg is:

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

where,

- w_{t+1} is the updated model at time step $t+1$.
- k indexes the K clients.(computed via the client fraction C)
- n_k is the number of local updates performed by client k
- $n = \sum_{k=1}^K n_k$ is the total number of local updates.
- $w_{k,t+1}$ is the updated model at time step $t+1$ on client k.

Diagramatic Explanation

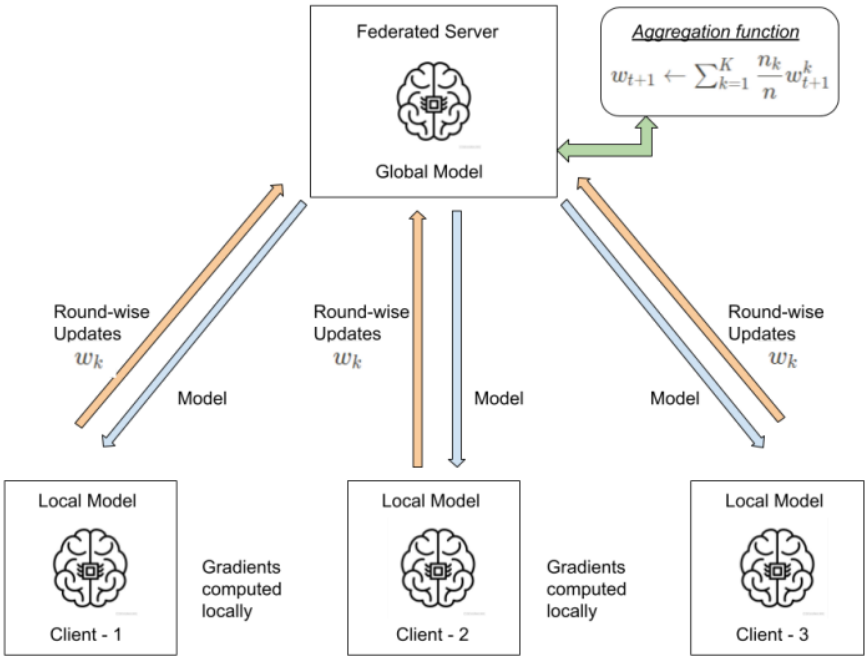
I have shown a diagrammatic version of the experiments, using a diagram that I made from scratch.

Federated learning is a distributed machine learning paradigm in which multiple devices collaboratively train a machine learning model without the need to upload their private data to a centralized server. Instead, each device trains the model locally on its own data, and only sends its updated model parameters to a central server, which aggregates them to generate a new global model. This allows the participating devices to jointly learn a model while maintaining their data privacy, as their data is never shared or exposed to other devices or the central server.

The federated learning algorithm typically involves the following steps:

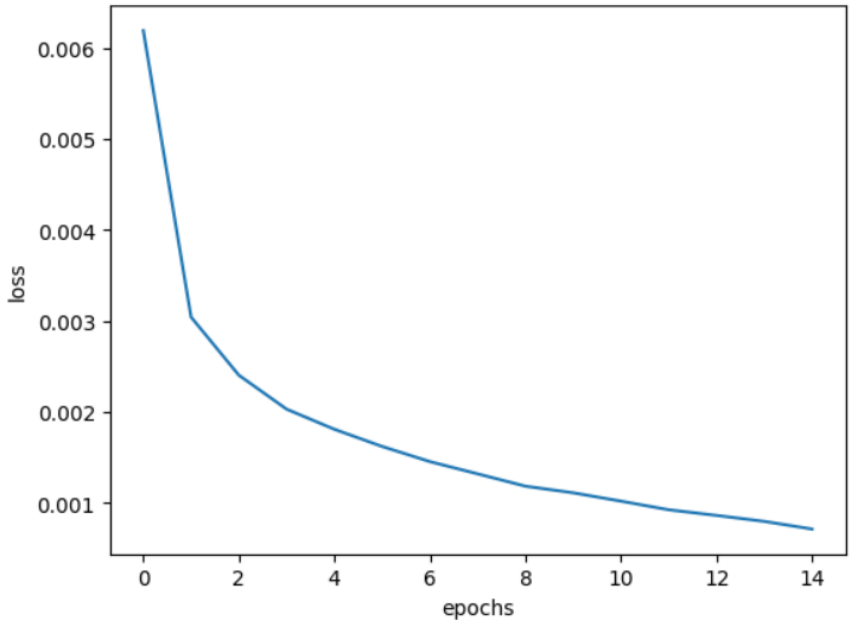
1. **Client selection:** A subset of devices (clients) is selected to participate in the federated learning process. The selection can be random or based on certain criteria, such as the device's computational power or the quality of its data.
2. **Model initialization:** A global model is initialized, which serves as the starting point for the federated learning process.
3. **Client model training:** Each selected client downloads the current global model and trains it on its local data. The training can be performed using any standard machine learning algorithm, such as stochastic gradient descent (SGD). The client's local training process is typically limited in time and resources, to ensure that all clients can participate equally in the federated learning process.
4. **Model aggregation:** After training, each client sends its updated model parameters (e.g., weight matrices) to the central server. The server aggregates the model updates using a predefined aggregation function, such as averaging or weighted averaging, to generate a new global model. The server then sends the new global model back to the clients, which use it as the starting point for the next round of training.
5. **Repeat:** Steps 3-4 are repeated for a predefined number of rounds, until the global model converges or reaches a satisfactory level of accuracy.

The diagram is as follows:



Federated vs Non-Federated Approach

On training the combined dataset in a non-federated approach, I obtain the following loss curve:



Thus, we see that the decrease in the loss is smoother than the previous case of FedAvg. On testing this model on the combined test dataset, I obtained a classification accuracy of **95.10%**.

Analysis & Conclusion

Thus, we observe that on training the model in a non-federated manner gives more classification accuracy than training in a federated manner. This is can be due to loss of information during aggregation. While training in a non-federated manner, all of our data is intermixed and hence there is no bias in which type of data is being used to train the network. However, in federated learning, the data feature distribution is not the same across the clients although the labels might be same. This may lead to slightly different gradient magnitude and directions in the different clients. Hence, we see a slight difference in the accuracies of federated vs non-federated learning.