Federated Learning

Dependable AI Assignment-3

Kwanit Gupta, B19EE046[a]

[a]*Indian Institute of Technology, Jodhpur*

May, 2023

Note

I went with the 1st Question, of having 3 different clients (naming MNIST, CMNIST and SVHN). I utilized Pytorch Lightning and concept of Federated Averaging (summing up layer-wise weights and biases) while taking a fair assumption of same model across all clients. Although due to pre-processing misinterpretations and information loss, results didn't came as expected despite changing the model types.

---

1. Question-1

Implement any federated learning algorithm of your choice using Pytorch with MNIST dataset on first client model, Coloured MNIST dataset on second client model, and SVHN dataset on third client model. Take 10,000 samples (i.e., 1000 per class) for each dataset in training set on their respective client and 5000 (i.e., 500 per class) samples of each dataset for a test set.

- Perform (0-9) digit classification task using federated setup by performing aggregation at the central server.
- Report the class-wise accuracy results for all three datasets at their respective client side and at the central server also. Report overall classification Accuracy and Confusion Matrix.
- Write the mathematical explanation of the function used to perform the aggregation at the central server.
- Write the detailed explanation of the federated learning algorithm with the diagrammatic representation used for the above solution.
- Compare the results of overall accuracy in federated setup with the baseline results calculated by combining all the datasets and training in non-federated setup. Do you observe any decrease/increase in accuracy for both the setups? State your answer with proper reasoning.

1.1. Part-1 (Pre-Processing Pipeline)

Since the input paradigm of the Pytorch-Lightning's model would remain same across all of client variables, I built the following pipeline:-

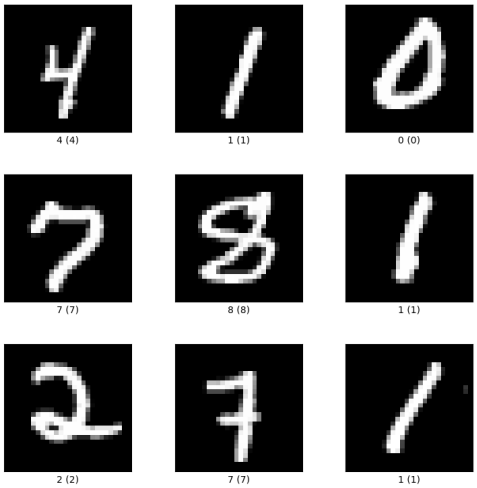


Figure 1: MNIST Dataset

- For MNIST Dataset, I resized it to 32 x 32 and then concatenated into 3 duplicate tensors together.
- For SVHN Dataset, I resized it to 32 x 32 only (since it was already a 3-channeled RGB image).
- For CMNIST Dataset, I resized it to 32 x 32, add a randomly initialized color pixel (independent of class distribution).

Figure 2: SVHN Dataset

Seeing the fact that the SVHN dataset constituted multiple digits into a single image, it was trivially expected to perform poor in this (and that too into 32 x 32). Also, it was required to capture 1000 images per class and 500 images per class for training and testing purposes per clients, respectively.

So, I did that using customized hand-written functions since svhn and mnist differ in their torchvision dataset calls. Later on, I utilized IterableDataset and ConcatDataset utility classes of pytorch to concatenate all client datasets for performing comparitive analysis of federated model over vanilla ones.

1.2. Part-2 Choice of Models and Federated Aggregation

I tried different model variants for the purpose of comparitive study, as following:-

- GoogLeNet: GoogLeNet is a deep convolutional neural network (CNN) architecture developed by Google, which won the ILSVRC in 2014. It includes the Inception module, which uses multiple filters of different sizes to extract features from the input image.
- MobileNet-V3 Large: MobileNet-V3 Large is a mobile-friendly CNN architecture designed for resource-constrained devices. It uses a combination of depthwise separable convolutions, linear bottleneck layers, and squeeze-and-excitation blocks to achieve high accuracy with low computational cost.
- SqueezeNet 1.0: SqueezeNet 1.0 is a lightweight CNN architecture that achieves high accuracy with fewer parameters than traditional CNNs. It uses 1x1 convolutions to reduce the number of parameters and a combination of 3x3 and 1x1 convolutions to extract features from the input image.
- ShuffleNet-V2 X0.5: ShuffleNet-V2 X0.5 is a CNN architecture that achieves high accuracy with low computational cost by using channel shuffling and pointwise group convolutions. It uses residual connections and a pyramid feature fusion module to improve performance.
- ConvNext Tiny: ConvNext Tiny is a small and efficient CNN architecture designed for mobile and embedded devices. It uses a combination of depthwise separable convolutions and 1x1 convolutions to reduce computational cost.
- RegNet Y-400MF: RegNet Y-400MF is a CNN architecture that achieves state-of-the-art performance on ImageNet with a small number of parameters. It uses a combination of group convolutions, regularized training, and progressive learning to improve performance.
- ResNet-34: ResNet-34 is a deep CNN architecture that includes residual connections to enable training of very deep neural networks. It achieved state-of-the-art performance on ImageNet in 2015 and is widely used as a backbone network for many computer vision tasks.
- Swin Transformer: Swin Transformer is a CNN architecture that uses self-attention mechanisms to improve feature extraction and achieve high accuracy on ImageNet. It uses a hierarchical structure and shifted windows to process large images efficiently.
- DenseNet-121: DenseNet-121 is a deep CNN architecture that uses dense connections to connect every layer to every other layer in a feedforward manner. It achieves high accuracy with fewer parameters than traditional CNNs and is widely used as a backbone network for many computer vision tasks.

No matter which type of model I wanna apply hereby, the main concept of Federated Aggregation (I changed optimizer from SGD to Adam, unfortunately it gave me bad results at the end) will remain same, as following:-

- Let us consider a federated learning setting where there are N clients with their local models $w_1, w_2, \ldots, w_N$, and a central server that aggregates their updates.

- Each client i trains its local model w_i using its local data $\mathcal{D}_i$, which is a subset of the overall data $\mathcal{D}$, by minimizing a local loss function $f_i(w)$:

$$w_i^* = \text{argmin}_{w \in W} f_i(w) \tag{1}$$

where $\mathcal{W}$ is the parameter space.

- To update the global model, the central server receives the updated model parameters from each client i in a round-robin fashion, and performs the aggregation of the model parameters. For parameter aggregation, we consider a weighted average of the model parameters of all the clients. In each round of communication, the central server calculates the weighted average of the models as follows:

$$w^+ = \sum_{i=1}^{N} \alpha_i w_i \tag{2}$$

where α_i is the weight of client i, and $\sum_{i=1}^{N} \alpha_i = 1$.

- The weight of each client is calculated based on its contribution to the global model and is determined using a weighting function. For example, the weight of each client can be proportional to the amount of data it has or its computation power.

- The central server then broadcasts the updated global model w^+ to all the clients. This process is repeated for multiple rounds until convergence. To sum the models, the central server adds the models of each client as follows:

$$w' = \sum_{i=1}^{N} w_i \tag{3}$$

and to take the mean, the central server divides the sum by the number of clients:

$$w_{t+1} = w_t + N_1 w' \tag{4}$$

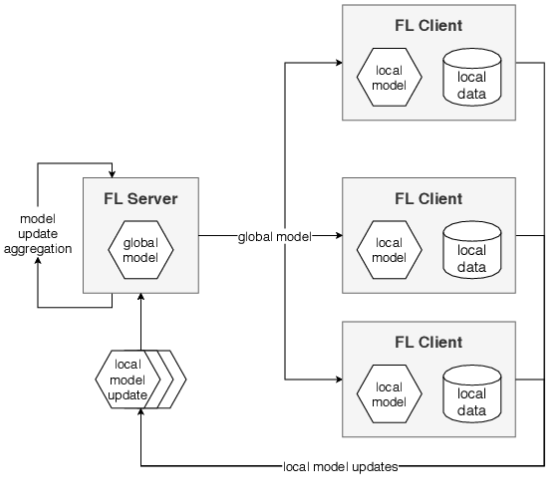
The central server can also perform other types of aggregation, such as the geometric mean, the harmonic mean, or the median, depending on the specific use case.

1.3. Part-3 What is Federated Learning at all?

Federated learning is a machine learning approach that allows multiple clients to collaboratively train a model without sharing their local data. Instead, each client trains a model on its local data and sends the model updates to a central server.

The central server aggregates the updates from all clients to update the global model. The updated global model is then sent back to the clients, who repeat the process of training the model on their local data and sending the updates to the server.

The federated learning algorithm can be broken down into the following steps:



high-level overview of FL, detailing the steps performed in a

Figure 3: Flowchart for Federated Aggregation

- Initialization: The central server initializes the global model, which is then sent to all clients.

- Client Training: Each client trains the model on its local data using stochastic gradient descent (SGD) or another optimization algorithm. The client then computes the model update by subtracting its local model parameters from the global model parameters.

- Model Update: The client sends the model update to the central server. The server aggregates the model updates from all clients by taking the weighted average of the updates.

- Global Model Update: The central server updates the global model by adding the aggregated model updates to the current global model.

- Model Distribution: The updated global model is sent back to all clients, who use it to start the next round of training.

This process continues until the global model converges to a solution or a stopping criterion is reached.

1.4. Part-4 Evaluation Metrics and Results

For Evaluation Metrics (Link Hereby for Notebook), I used the following:-

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{7}$$

$$\text{support}(y) = \sum_{i=1}^{n} 1(y_i = y) \tag{8}$$

Class-wise metrics and Confusion Matrices were outputted as well, to get clarity of the model's aggregation process.

Model	*TAcc.*	*MAcc.*	*WAcc.*
GoogLeNet	93.6%	94%	93%
MobileNet-V3	96%	96%	96%
SqueezeNet 1.0	10%	2%	18%
ShuffleNet-V2	80.06%	81%	80%
ConvNext Tiny	10%	2%	18%
RegNet Y-400mf	94.38%	94%	94%
ResNet-34	91.7%	91%	92%
Swin Transformer	10%	2%	18%
DenseNet-121	95.2%	95%	95%

Table 1: MNIST's own Accuracies

Model	*TAcc.*	*MAcc.*	*WAcc.*
GoogLeNet	9.42%	3%	16%
MobileNet-V3	85.06%	86%	84%
SqueezeNet 1.0	10%	2%	18%
ShuffleNet-V2	92%	92%	92%
ConvNext Tiny	10%	2%	18%
RegNet Y-400mf	92.5%	92%	93%
ResNet-34	10.24%	3%	17%
Swin Transformer	10%	2%	18%
DenseNet-121	88.02%	87%	89%

Table 2: Colored MNIST's own Accuracies

1.5. Observations, Limitations and Scope of Improvements

One of the main reasons why "10,2,18" were common numbers in metric tables hereby, was because of shortcomings of SVHN as following:-

Model	*TAcc.*	*MAcc.*	*WAcc.*
GoogLeNet	10.4%	3%	18%
MobileNet-V3	14.76%	11%	19%
SqueezeNet 1.0	10%	2%	18%
ShuffleNet-V2	10.44%	5%	16%
ConvNext Tiny	10%	2%	18%
RegNet Y-400mf	10%	2%	18%
ResNet-34	9.68%	5%	15%
Swin Transformer	10%	2%	18%
DenseNet-121	12.96%	7%	19%

Table 3: SVHN's own Accuracies

Model(Class)	*mnist*	*cmnist*	*svhn*
GoogLeNet (3rd)	10%	10%	10%
MobileNet-V3 (5th)	10%	10%	10%
SqueezeNet 1.0 (1st)	10%	10%	10%
ShuffleNet-V2 (3-5th)	9.98%	10.02%	9.98%
ConvNext Tiny (7th)	10%	10%	10%
RegNet Y-400mf (1st)	10%	10%	10%
ResNet-34 (1st)	10%	10%	10%
Swin Transformer (7th)	10%	10%	10%
DenseNet-121 (2nd)	10%	10%	10%

Table 4: Central Server's Accuracies on different datasets

- Variability in image quality: The SVHN dataset contains images of street numbers taken from Google Street View, and as such, the images have a high degree of variability in terms of brightness, contrast, and occlusion. This variability can make it challenging for CNNs to learn discriminative features and patterns in the images.
- Small image size: The SVHN images are only 32 x 32 pixels, which is relatively small compared to other image datasets like ImageNet. This small size can make it difficult for CNNs to extract meaningful features from the images, especially when there is a high degree of variability in the images.
- Large number of classes: The SVHN dataset has 10 classes, which is a relatively large number of classes for a 32 x 32 image. This can make it challenging for CNNs to learn discriminative features and patterns that can distinguish between the different classes.

Model(Class)	*TAcc.*	*MAcc.*	*WAcc.*
MobileNet-V3 (10th)	10%	2%	18%
Swin Transformer (10th)	10%	2%	18%

Table 5: Model Accuracies for combined Dataloaders

- Class imbalance: The SVHN dataset has a class imbalance, with some classes having many more examples than others. This can make it difficult for CNNs to learn to recognize the minority classes, which can lead to lower overall classification performance.
- Overfitting: The small size of the SVHN images and the variability in image quality can make it easy for CNNs to overfit to the training set. This can result in poor generalization performance on the test set.
- Limited amount of training data: The SVHN dataset has 73,257 training images, which is a relatively small amount of data compared to other image datasets like ImageNet. This limited amount of data can make it challenging for CNNs to learn robust representations that generalize well to new images.

With the Preprocessing pipeline too, there were shortcomings as following:-

- Incorrect resizing of images: Resizing images to a smaller size can result in loss of information, and resizing to a larger size can introduce noise and artifacts. Incorrect resizing can lead to distorted images that are difficult for the CNN to classify.
- Poor image normalization: Normalizing images to a common scale and range is crucial for CNNs. If images are not normalized properly, it can affect the CNN's ability to learn and generalize from the data.
- Inconsistent image quality: The quality of the images in the dataset can vary, with some images being blurry or poorly lit. Inconsistent image quality can make it difficult for the CNN to learn patterns and generalize from the data.
- Insufficient data augmentation: Data augmentation techniques such as rotation, flipping, and scaling can help increase the size of the dataset and improve the CNN's ability to generalize. Insufficient data augmentation can result in overfitting and poor classification performance.
- Incorrect data labeling: Incorrect or inconsistent labeling of the data can affect the CNN's ability to learn and generalize from the data. Inconsistent labeling can also make it difficult to compare results across different experiments and datasets.
- Lack of preprocessing techniques: Various preprocessing techniques such as denoising, edge detection, and contrast enhancement can improve the quality of the images and make it easier for the CNN to learn and classify. The lack of these techniques can result in poor classification performance.