|  |  |  |  |
| --- | --- | --- | --- |
|  |  | DP | No DP |
| **CIFAR-10** | Acc Test | 58.06 | **72.08** |
| Acc Train | 61.11 | **97.64** |
| ε | **49.99** | inf |
| Time | 28201.90 s | **385.38 s** |
| **DermaMNIST** | Acc Test | **71.86** | 68.46 |
| Acc Train | 72.47 | **90.89** |
| ε | **49.99** | inf |
| Time | 2883.29 s | **52.59 s** |
| **PneumoniaMNIST** | Acc Test | **86.77** | 83.93 |
| Acc Train | 97.96 | **99.96** |
| ε | **49.99** | inf |
| Time | 2047.42 s | **35.00 s** |
| **BloodMNIST** | Acc Test | **84.34** | 58.09 |
| Acc Train | 87.07 | **94.08** |
| ε | **49.99** | inf |
| Time | 5395.57 s | **90.73 s** |
| **OrganCMNIST** | Acc Test | 79.36 | **85.49** |
| Acc Train | 90.14 | **99.05** |
| ε | **49.99** | inf |
| Time | 5989.60 s | **98.28 s** |

**Task 1 a)**

The code provided implements a robust training pipeline for a neural network model using PyTorch, with the added ability to apply Differential Privacy (DP) for improved privacy preserving mechanisms. The process starts with the definition of data transformations, such as tensor conversion and normalization. Key parameters related to differential privacy, including `MAX\_GRAD\_NORM`, `EPSILON` and `DELTA`, are set to control the level of privacy during training.

The `make\_private` function introduces differential privacy into the model, optimizer and data loader. This function uses a privacy engine (`privacy\_engine`) to ensure that the training process complies with privacy constraints.

The training function (`train`) iterates through the specified number of epochs, updates the model parameters using backpropagation and monitors the training progress, including the average loss, accuracy and privacy budget (ε). The training time is also tracked.

A test function (`test`) evaluates the model on a separate test data set and provides insights into the generalization performance.

The entire training pipeline is orchestrated by the "pipeline" function, which initializes the model and optimizer, corrects the model with ModuleValidator and trains two models (one with and one without DP).

The CIFAR-10 dataset is used for training and testing, with data loading configured via PyTorch's DataLoader. The ResNet18 model architecture is used for image classification with 10 output classes, and the RMSprop optimizer with a given learning rate (LR) is used for updating the model parameters during training.

Table 1: Accuracies, Privacy Budgets and Training Times for Task 1a) and b)

**Task 1 b)**

For the datasets ‘PneumoniaMNIST’ and ‘OrganCMNIST’ I had to change the number of input channels of the first convolutional layer to one.

The introduction of noise into the training process in DP, designed to safeguard individual data points, heightens the complexity of optimization. This is noticeable in the training time, which is significantly longer with DP than without.

After 20 epochs the privacy budget is 49.99 in all four cases. It seems, that in all four cases the same amount of privacy is provided.

Furthermore, the training accuracy without DP is higher in all four cases than with DP. The noise introduced for privacy protection in DP may hinder the model's ability to fit the training data perfectly, leading to lower training accuracy. DP's focus on protecting individual data points can potentially compromise the model's ability to memorize the training set.

However, the training with DP apparently generalizes more than the training without DP, since in 3 of four datasets (DermaMNIST, PneumoniaMNIST, BloodMNIST) the accuracy of the test set with DP is higher than that without DP. That could be since DP encourages models to learn more robust features that are less likely to overfit to the training data. This suggests that the models trained with DP are more effective at making predictions on new, unseen examples.

Overall, the ‘PneumoniaMNIST’ dataset performs the best while taking relatively little time to train.

**Task 2 a)**

We ran a simulation for federated learning with 3 clients, we used a simple CNN implementation consisting of 4 convolution layers and one full connected layer.

We created a load dataset function to load the ChestMnist dataset in a way to create a list of *trainloaders* and *valloaders* with *length = number of clients*, the function also returns the testloader which is used at the server-side to evaluate the training

The metrics were aggregated using a simple weighted average function.

The evaluation was done at the client side with a small portion of the validation dataset (referred to as distributed in the table) and on the server-side with the test dataset (referred to as centralized in the table)

We tried this setup and documented the results while varying the batch size and learning rate as follows

* Batch size = 32 and learning rate = 1e-3
* Batch size = 8 and learning rate = 1e-3
* Batch size = 64 and learning rate = 1e-3
* Batch size = 32 and learning rate = 1e-1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Round | Batch\_size=32  L\_R = 1e-3 | Batch\_size=8  L\_R = 1e-3 | Batch\_size=64  L\_R = 1e-3 | Batch\_size=32  L\_R = 1e-1 |
| Loss ↓ Distributed | 1 | 0.052 | 0.211 | **0.0261** | 0.0504 |
| 2 | 0.048 | 0.202 | **0.0243** | 0.0504 |
| 3 | 0.047 | 0.195 | **0.0241** | 0.0506 |
| Loss ↓ Centralized | *0* | *0.060* | *0.241* | *0.030* | *0.061* |
| 1 | 0.054 | 0.219 | **0.026** | 0.0524 |
| 2 | 0.050 | 0.210 | **0.025** | 0.0524 |
| 3 | 0.049 | 0.202 | **0.024** | 0.0525 |
| Accuracy ↑ Distributed | 1 | 12.59 | 12.594 | **12.6** | 12.59 |
| 2 | 13.96 | 12.594 | **20.3** | 12.59 |
| 3 | 24.28 | 15.874 | **24.8** | 12.59 |
| Accuracy ↑ Centralized | *0* | *3.25* | *12.035* | *0.51* | *2.23* |
| 1 | 12.03 | 12.035 | **12.04** | 12.035 |
| 2 | 13.40 | 12.035 | **19.74** | 12.035 |
| 3 | 23.81 | 15.517 | **24.54** | 12.035 |

Results, as expected, are better with larger batch size, the model is capable of converging faster, Comparing batch size of 64 batch size of 8, the accuracy with the first one was capable to jump from 12% to 19.8% in one round, while with batch size 8 accuracy remained 12% for 2 rounds.

The same is true for the learning rate with a high learning rate (last column); the model didn’t converge at all, and as shown, loss and accuracy are not changing round after the other.