Dependable Artificial Intelligence

Federated Learning

TEAM MEMBERS:
Rutuja Janabandhu [B21AI017]
Khushi Maheshwari [B21AI052]

INTRODUCTION

Federated learning has emerged as a powerful paradigm for training machine learning models on decentralized data without the need to centralize sensitive information. In this project, we delve into the realm of federated learning by implementing and comparing three prominent algorithms: Federated Averaging (FedAvg), Federated Proximal (FedProx), and Federated Adaptive Gradient (FedAdaGrad).

The objective of this study is to evaluate the performance of these federated learning algorithms on a heterogenous dataset. Unlike homogeneous datasets where all clients have similar distributions, heterogeneous datasets pose additional challenges due to variations in data characteristics across clients. By conducting a comparative analysis, we aim to gain insights into how these algorithms handle heterogeneity and their impact on model convergence, communication efficiency, and overall performance.

DATASET

The CIFAR-10 dataset is a widely used benchmark in computer vision and machine learning research. It consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The classes are as follows:Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck.

The dataset is split into 50,000 training images and 10,000 test images, making it suitable for tasks such as image classification. Each image is represented in RGB format, with pixel values ranging from 0 to 255.

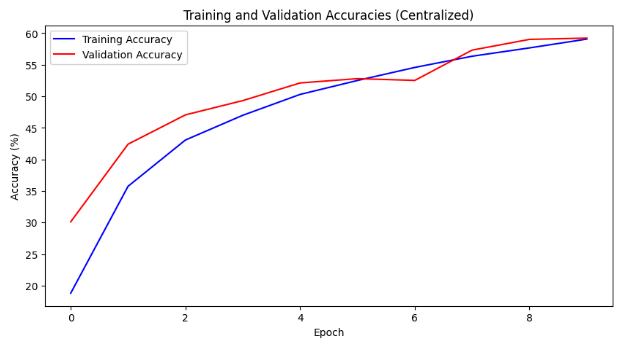
For the federated learning assignment comparing FedAvg, FedProx, and FedAdaGrad on a heterogenous dataset, the CIFAR-10 dataset provides a diverse set of images across multiple categories.

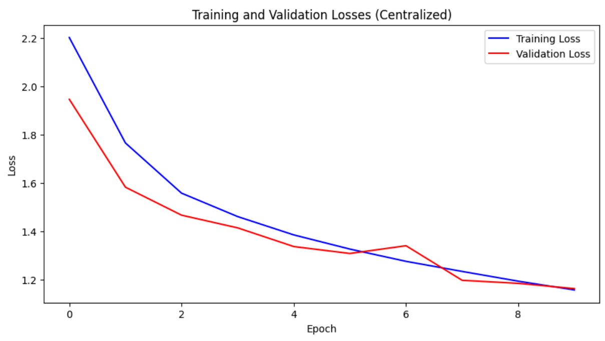
We Implemented a convolutional neural network (CNN) model using PyTorch for image classification on the CIFAR-10 dataset

Centralized Learning

Centralized learning, also known as centralized training or centralized machine learning, refers to the traditional approach where all data used for training a machine learning model is collected and stored in a centralized location

First we train the model in the centralized manner by taking all the data to the server and find the losses and accuracies for epochs.
Plots for centralized learning algorithm:





We will further train the model using federated algorithms and compare the results with this.

Federated Learning Algorithms

For this Assignment we implemented three Federated learning algorithms, i.e., Federated Averaging (FedAvg), Federated Proximal (FedProx), and Federated Adaptive Gradient (FedAdaGrad).

Federated Averaging (FedAvg)

Federated Averaging (FedAvg) is a fundamental federated learning algorithm designed to train a global model using decentralized data from multiple clients or devices. FedAvg leverages the power of decentralized data while ensuring model convergence and preserving data privacy. By averaging model updates across clients, it mitigates the effects of data heterogeneity and improves the robustness of the global model. Additionally, FedAvg allows for scalable and efficient training in distributed environments without compromising data security.

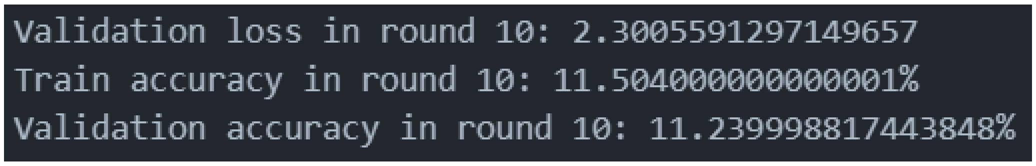
We implemented FedAvg where:
The FedAvgClientUpdate class responsible for updating the model on client devices and implementing the federated averaging algorithm. It also includes functions to split the dataset for federated learning (split_dataset), perform federated training (federated_training), and evaluate federated learning rounds (federated_learning_round).

The split_dataset function divides the CIFAR-10 dataset into multiple subsets to simulate a federated learning scenario. Each subset represents data available to individual clients in a federated setup.

The federated_training function orchestrates federated training using the FedAvg algorithm. It involves selecting a fraction of clients for each round, updating their models locally, aggregating their weights using federated averaging, and evaluating the aggregated model on a validation set.

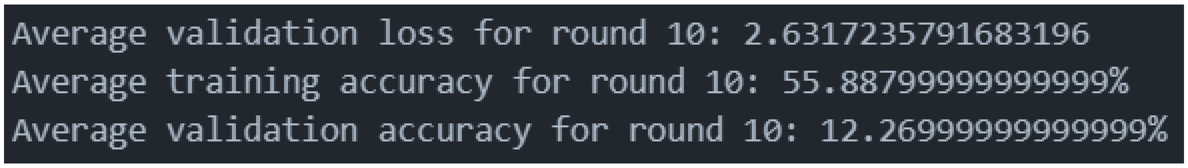
The CustomCIFAR10 class creates a custom dataset using CIFAR-10 data and implements non-IID sampling (cifar_non_iid_sampling). Non-IID sampling ensures that each client's data is diverse and representative, crucial for realistic federated learning simulations.

The federated_learning_round function conducts a single round of federated learning, where selected clients update their models locally, and the server aggregates their weights to update the global model. It evaluates the model's performance on a validation set after each round.
The Accuracy for IID Dataset came out to be:



```
Validation loss in round 10: 2.3005591297149657
Train accuracy in round 10: 11.504000000000001%
Validation accuracy in round 10: 11.239998817443848%
```

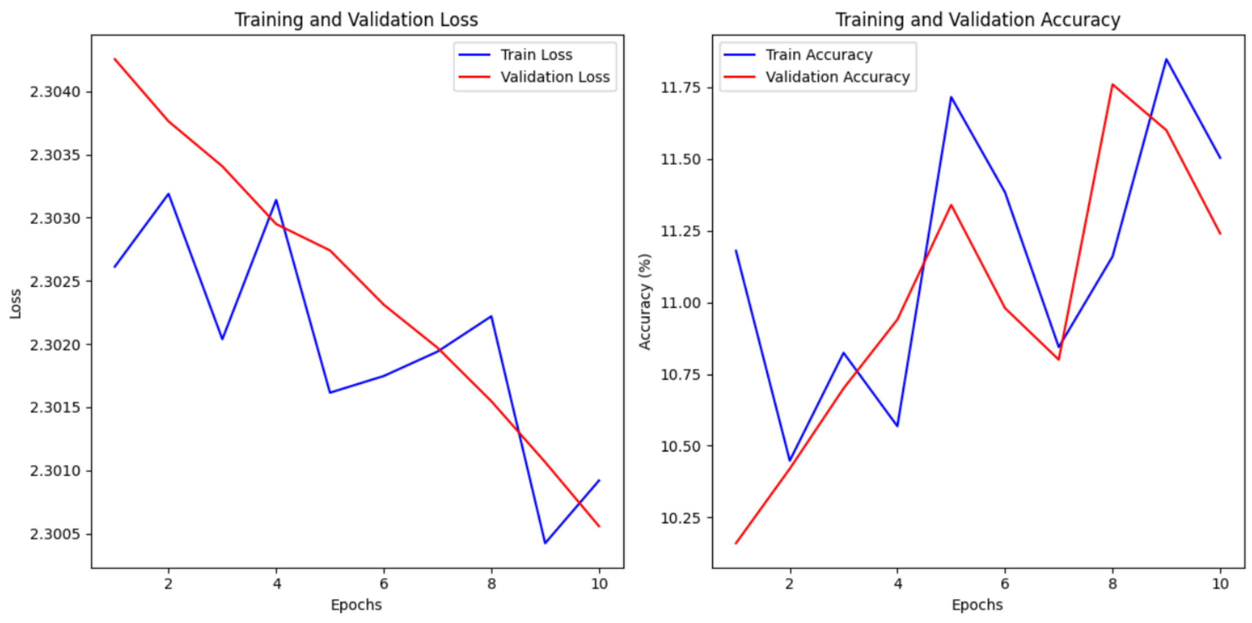
The Accuracy for Non-IID Dataset came out to be:



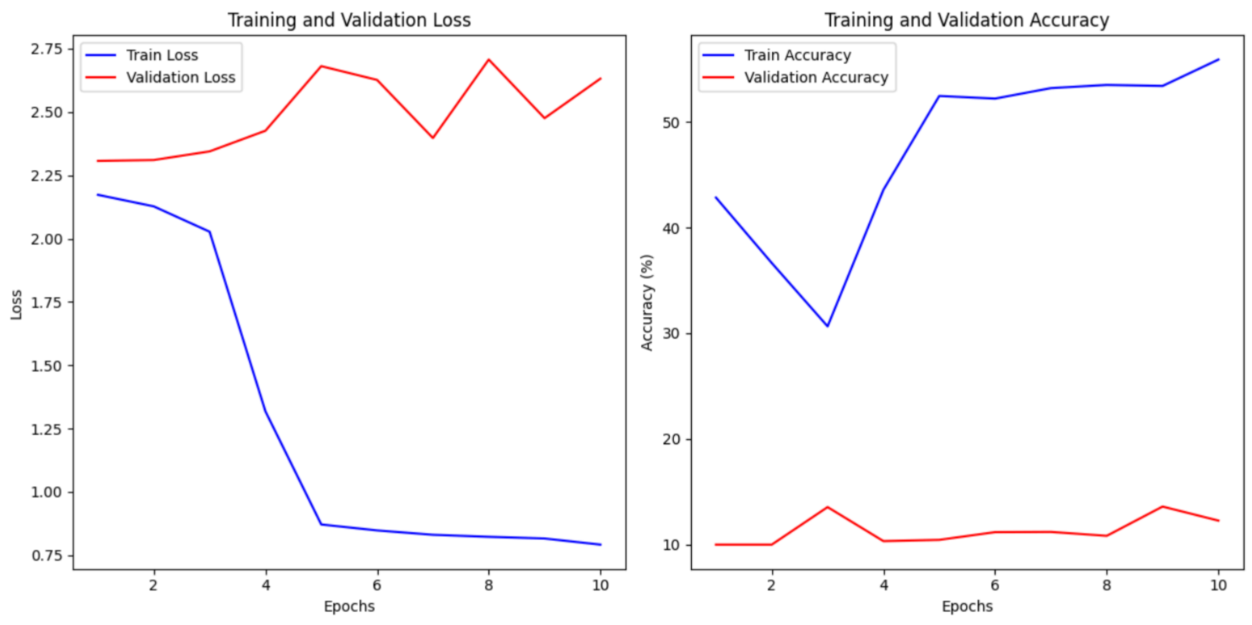
```
Average validation loss for round 10: 2.6317235791683196
Average training accuracy for round 10: 55.887999999999999%
Average validation accuracy for round 10: 12.269999999999999%
```

Plotting Training and Validation Metrics:
Finally, the plot_metrics function to visualize the training and validation losses as well as accuracies over epochs during federated training. These plots help track the model's learning progress and generalization performance in a federated learning setting.

Plots for (IID Dataset)



Plots for (Non-IID Dataset)



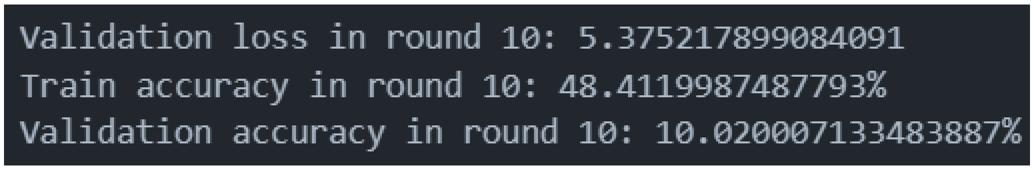
Federated Proximal (FedProx)

Federated Proximal (FedProx) is a variant of federated learning that introduces a proximal term to the optimization objective, aiming to improve model robustness and convergence in the presence of non-iid (non-independent and identically distributed) data across clients. FedProx addresses the challenge of data heterogeneity by incorporating a regularization term that penalizes large model parameter deviations from previous rounds.

We implemented FedProx where:
This class handles the client-side operations for the Federated Proximal (FedProx) algorithm. It includes methods for updating the model (update_model), computing the loss function with a federated proximal term (loss_function), and performing federated averaging (federated_average).

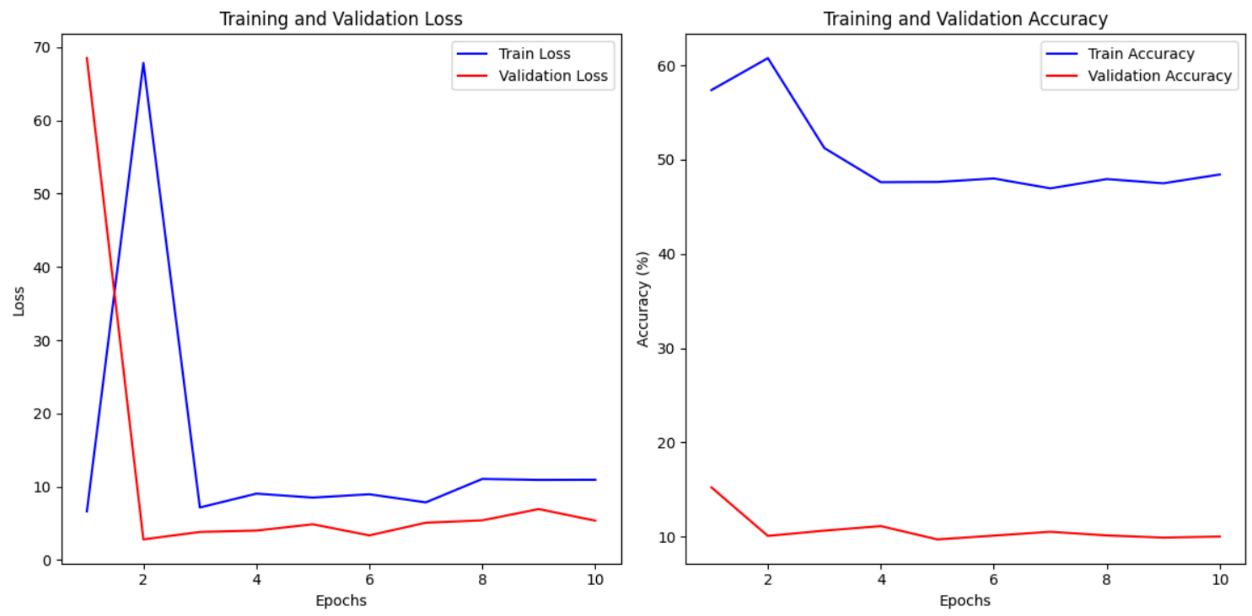
This class is an alternative implementation of the FedProxClientUpdate class, with similar functionalities but different variable names and coding style.

The train_with_fedprox function orchestrates the training process using the FedProx algorithm. It involves selecting a subset of clients for each round, updating their models locally with proximal regularization, aggregating their updates using federated averaging, and evaluating the aggregated model on a validation set.
The Accuracy came out to be:



```
Validation loss in round 10: 5.375217899084091
Train accuracy in round 10: 48.4119987487793%
Validation accuracy in round 10: 10.020007133483887%
```

Plotting Training and Validation Metrics:
The plot_metrics function visualizes the training and validation losses, as well as the training and validation accuracies, over epochs during federated training. These plots provide insights into the model's learning progress and generalization performance under the FedProx algorithm.

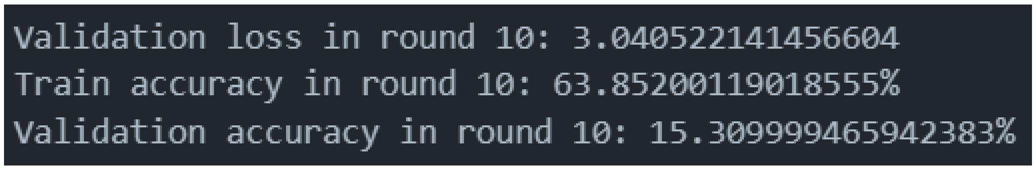


Federated Adaptive Gradient (FedAdaGrad)

Federated Adaptive Gradient (FedAdaGrad) is a federated learning algorithm that incorporates adaptive learning rates for individual model parameters across clients. FedAdaGrad's adaptive learning rate mechanism allows it to adapt to the varying importance of model parameters across clients and data distributions.

We implemented FedAdaGrad:
The FedAdagradClientUpdate class handles the client-side operations for the Federated Adagrad (FedAdagrad) algorithm. It includes methods for updating the client model (update_model) using Adagrad optimization and calculating the loss function with a federated proximal term (calculate_loss).

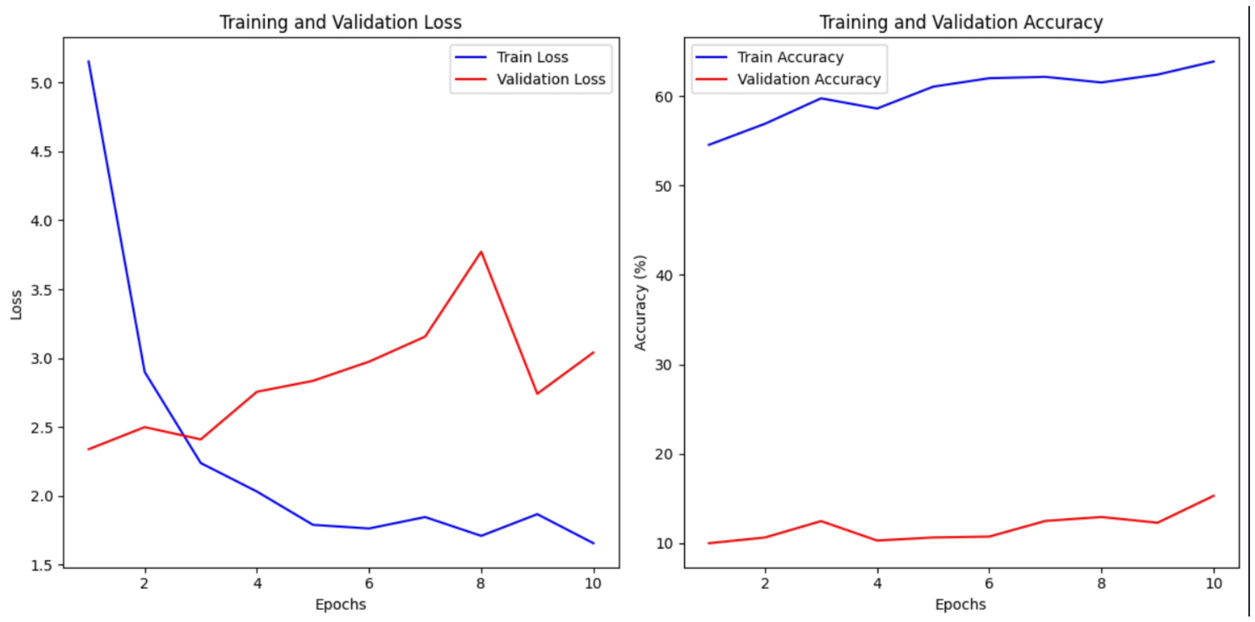
The train_with_fedadagrad function orchestrates the federated training process using the FedAdagrad algorithm. It involves selecting a subset of clients for each round, updating their models using Adagrad optimization, aggregating their updates using federated averaging, and evaluating the aggregated model on a validation set. The Accuracy came out to be:



```
Validation loss in round 10: 3.040522141456604
Train accuracy in round 10: 63.85200119018555%
Validation accuracy in round 10: 15.309999465942383%
```

Plotting Training and Validation Metrics:

The plot_metrics function visualizes the training and validation losses, as well as the training and validation accuracies, over epochs during federated training using FedAdagrad. These plots provide insights into the model's learning progress and generalization performance under the FedAdagrad algorithm.



It includes classes and functions for client updates, federated averaging, training orchestration, and result visualization, offering a comprehensive view of FedAdagrad-based federated learning experimentation.

Comparison between the three algorithms (FedAvg, FedProx and FedAdaGrad)

Each algorithm has its unique characteristics and trade-offs in terms of convergence speed, communication efficiency, and robustness to client heterogeneity.

FedAvg (Federated Averaging):
FedAvg is a basic federated learning algorithm that averages model updates from all clients to update the global model. It is simple and easy to implement, making it a good baseline for comparison. However, it may suffer from issues like communication bottlenecks and non-IID data distribution, impacting its performance on heterogeneous datasets.

FedProx (Federated Proximal):

FedProx extends FedAvg by adding a proximal term to the loss function, encouraging model weights to stay close to the previous global model. This helps mitigate the effects of non-IID data and reduces communication overhead. FedProx is effective when dealing with non-IID data and achieving better convergence compared to FedAvg.

FedAdagrad (Federated Adaptive Gradient):
FedAdagrad incorporates adaptive learning rates for each parameter based on the historical gradients. It adapts to varying data distributions across clients and helps in training models efficiently on non-IID data. FedAdagrad can handle non-IID data well and converge faster compared to traditional methods like FedAvg.

Comparison between Federated Learning and Centralized Learning

Federated learning and centralized learning represent two contrasting approaches to machine learning model training, each with its strengths and weaknesses.

Federated learning

- Federated learning has gained prominence due to its ability to train models across decentralized data sources while preserving data privacy and security.
- This approach is particularly beneficial in scenarios where data privacy is paramount, such as healthcare, finance, and sensitive user information.
- By keeping data local and only sharing model updates, federated learning minimizes the risk of exposing raw data to unauthorized parties, addressing concerns related to data governance and regulatory compliance.

Centralized Learning

- Centralized learning, which aggregates data into a single location for model training, is often favored for its simplicity and ease of implementation.
- It leverages powerful computing resources to process large datasets efficiently, leading to potentially higher model performance and faster convergence.
- Centralized learning is well-suited for scenarios where data consolidation is feasible, such as in well-connected environments with abundant computing resources and less stringent privacy requirements.

The choice between federated and centralized learning depends on various factors, including the sensitivity of the data, regulatory constraints, network bandwidth limitations, computational resources, and scalability requirements.

Federated learning shines in privacy-sensitive domains and distributed environments but may require specialized algorithms and face challenges with heterogeneous data sources.

Centralized learning offers straightforward implementation and superior performance with consolidated data but raises concerns about data privacy, scalability, and potential single points of failure.
