The federative learning model-

The main functionality that is being represented in the model is the working function of federative learning, there by creating a server and also clients.

As the federative learning model suggests the user privacy by training the model locally on the user part updating the global model, here we connect the server and client and the model calculating the accuracy-

STEP BY STEP PROCEDURE

The above part explains the procedural working of the code.for the the practical implementation-

1.unzip the code file from the zip.

2.open command prompt.

3.redirect the path to the existing folder.

4.run the server by the command line-python server1.py

5.after the server shows amessage to connect the client ,open another command prompt and redirect to the existing folder.

6. now run the client with the command line -python client1.py

7.now repeat the same procedure to connect another client ,with yet another command prompt and with same command line-python client1.py

8.after connecting both the clients to the server the command prompt shows the calculated losses

(for any issues or confusion pls refer the vedieo)

Server part code:

1.Importing the necessary modules:

socket: This module provides low-level network communication capabilities.

torch and torch.nn: These modules are part of PyTorch, usually used in deep learning framework.

datasets and transforms from torchvision: These modules provide utilities for working with datasets and image transformations of the mnist data set in this case.

2. Defining the model architecture:

Defines a neural network model using PyTorch's nn.Module class.

Here there are two parts precisely constructor and foreward pass,of which the constructor part sows the relation between the layers input and output features in the case of self.fc1 = nn.Linear(784, 128) is 784 is the input pixels and 128 is output neurons concting between three layers

Further, The model consists of three fully connected layers (nn.Linear) with ReLU activation functions (torch.relu).

3. Loading the MNIST dataset:

Loads the MNIST dataset, which consists of handwritten digit images and their corresponding labels.

The dataset is transformed into tensors and normalized using mean 0.5 and standard deviation 0.5.

4.Creating the model, criterion, and optimizer:

The loss function (nn.CrossEntropyLoss) is defined

The line optimizer = torch.optim.SGD(model.parameters(), lr=0.01) initializes the optimizer used for training the model. In this case, it creates an instance of the stochastic gradient descent (SGD) optimizer from the torch.optim module.

5.Defining the server training function:

This function performs the training on a batch of data.

It sets the model to training mode, computes the forward pass, computes the loss, performs backward pass (gradient computation), and updates the model's parameters.

6.Performing federated learning:

The average loss is returned.

This function performs federated learning by iterating over a specified number of epochs.

It calls the train\_on\_batches function for each client in data\_loaders to train on their respective data.

The loss is aggregated and printed for each client and epoch.

7.Creating a socket for server-client communication:

Creates a socket using the socket module to establish a network connection with clients.

It binds the socket to a specific host and port and sets it to listen for incoming client connections, which in this case is modified for two clients but can be modified to any number of clients based on the requirement.

8.Accepting connections from clients:

This loop accepts connections from clients using the accept() method of the server socket.

It prints the address of each connected client and stores the client sockets in a list.

9.Splitting the trainset into two client datasets:

The trainset is split into two subsets, client1\_dataset and client2\_dataset, using torch.utils.data.Subset.

The split is based on the length of the trainset divided by 2,as we have conection list of 2 clients.

10. Creating data loaders for each client dataset:

Data loaders are created for each client dataset using torch.utils.data.DataLoader.

The data is loaded in batches of size 64, and the order of the data is shuffled (shuffle=True).

11. Creating a data loader dictionary for clients:

A dictionary is created to map client names to their respective data loaders.

12. Performing federated learning on the server:

The federated\_learning function is called with the model, loss criterion, optimizer, data loaders, and the number of epochs.

This initiates the federated learning process where the server coordinates the training of the model on client data.

13. Saving the trained model:

Saves the pre trained model.

14. Closing the sockets:

The client sockets and the server socket are closed to release the network resources.

Client code part:

1.Importing the necessary modules:

socket: This module provides low-level network communication capabilities.

torch and torch.nn: These modules are part of PyTorch, usually used in deep learning framework.

datasets and transforms from torchvision: These modules provide utilities for working with datasets and image transformations of the mnist data set in this case.

2. Defining the model architecture:

Defines a neural network model using PyTorch's nn.Module class.

Here there are two parts precisely constructor and foreward pass,of which the constructor part sows the relation between the layers input and output features in the case of self.fc1 = nn.Linear(784, 128) is 784 is the input pixels and 128 is output neurons concting between three layers

Further, The model consists of three fully connected layers (nn.Linear) with ReLU activation functions (torch.relu).

3. Loading the MNIST dataset:

Loads the MNIST dataset, which consists of handwritten digit images and their corresponding labels.

The dataset is transformed into tensors and normalized using mean 0.5 and standard deviation 0.5.

4.Creating a socket for server-client communication:

Creates a socket using the socket module to establish a network connection with clients.

It binds the socket to a specific host and port and sets it to listen for incoming client connections, which in this case is modified for two clients but can be modified to any number of clients based on the requirement.

5. Sending and receiving messages between client and server:

The client sends a message to the server using send(). The message is converted to bytes using b"..."

The client receives a response from the server using recv(). The received data is stored in message.

The received message is then decoded from bytes to a string using decode() and printed.

6. Defining the model and optimizer:

The loss function (nn.CrossEntropyLoss) is defined

The line optimizer = torch.optim.SGD(model.parameters(), lr=0.01) initializes the optimizer used for training the model. In this case, it creates an instance of the stochastic gradient descent (SGD) optimizer from the torch.optim module.

7.Training the model on the client:

Trains the model on the local dataset (trainset) for 15 epochs.

It iterates over the images and labels in the dataset, performs the forward pass, computes the loss, computes gradients, and updates the model parameters using the optimizer.

8. Sending and receiving model weights between client and server:

The client sends a message to the server indicating that it wants to send the model weights.

The client receives the model weights from the server.

The received model weights are loaded into the local model using model.load\_state\_dict().

9. Closing the socket:

Finally, the client socket is closed to end the communication with the server.