Lab Assignment 7

Problem Statement:

Write a program to implement an Artificial Neural Network (ANN) to classify the MNIST dataset of handwritten digits.

Hints for Solution:

Framework: You can use PyTorch to solve this assignment. PyTorch provides an efficient way to handle data, define models, and perform training and evaluation on datasets.

Pseudocode Outline:

- Step 1: Set up the MNIST dataset with training and testing splits.
- Step 2: Define a simple neural network model with:
  - One input layer that takes in flattened 28x28 images (784 inputs).
  - One to three hidden layer with a suitable number of neurons (e.g., 512, 256, 128).
  - One output layer with 10 neurons (one for each digit).
- Step 3: Define the loss function and optimizer.
- Step 4: Implement a training loop:
  - Load each batch of images and labels.
  - Forward pass: Compute predictions and calculate the loss.
  - Backward pass: Update model weights based on loss.
- Step 5: Evaluate model accuracy on test data.

Helpful PyTorch Functions:

- Data Loading:
  - Use `torchvision.datasets.MNIST` to download and load the dataset.
  - Use `torch.utils.data.DataLoader` to handle batch processing and shuffling.
- Model Definition:
  - Use `torch.nn.Linear` to create fully connected (dense) layers.
  - Use `torch.nn.ReLU` for activation between layers.
- Training:
  - Use `torch.optim.RMSprop` or `torch.optim.ADAM` to define the optimizer.
  - Use `torch.nn.CrossEntropyLoss` as the loss function.
- Evaluation:
  - Use `torch.max` to find the predicted label for each image.

This approach will help you train a neural network to classify handwritten digits with a basic ANN structure.

In [1]:

```python
### WRITE CODE HERE ###
import torch
import torch.nn as nn
import torch.optim as optim

import torchvision
from torchvision import datasets, transforms

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [2]:

```python
# hyper parameters
batch_size = 128
learning_rate = 0.001
num_epochs = 10

# download the dataset
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])
train_dataset = datasets.MNIST(root='./data', train=True,
                                    transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False,
                                      transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset,
                                    batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=
                                          batch_size, shuffle=False)

class SimpleANN(nn.Module):
    def __init__(self, input_size, hidden_sizes, num_classes):
        super(SimpleANN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_sizes[0])
        self.fc2 = nn.Linear(hidden_sizes[0], hidden_sizes[1])
        self.fc3 = nn.Linear(hidden_sizes[1], hidden_sizes[2])
        self.fc4 = nn.Linear(hidden_sizes[2], num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.relu(out)
        out = self.fc3(out)
        out = self.relu(out)
        out = self.fc4(out)
        return out
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.
gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.
gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [00:00<00:00, 15.9MB/s]
```

```
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.
gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.
gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 481kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.g
z
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.g
z to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.43MB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.g
z
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.g
z to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 2.76MB/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

In [3]:

```python
model = SimpleANN(28*28, [512, 256, 128], 10).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    train_correct = 0
    train_total = 0
    train_loss = 0.0
    for i, (images, labels) in enumerate(train_loader):
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        train_loss += loss.item()

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()
```

```
    train_accuracy = 100 * train_correct / train_total
    avg_train_loss = train_loss / len(train_loader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_train_loss:.4f},
            Training Accuracy: {train_accuracy:.2f}%")
```

```
Epoch [1/10], Loss: 0.3561, Training Accuracy: 89.19%
Epoch [2/10], Loss: 0.1495, Training Accuracy: 95.44%
Epoch [3/10], Loss: 0.1095, Training Accuracy: 96.53%
Epoch [4/10], Loss: 0.0856, Training Accuracy: 97.34%
Epoch [5/10], Loss: 0.0715, Training Accuracy: 97.71%
Epoch [6/10], Loss: 0.0608, Training Accuracy: 98.03%
Epoch [7/10], Loss: 0.0554, Training Accuracy: 98.22%
Epoch [8/10], Loss: 0.0480, Training Accuracy: 98.42%
Epoch [9/10], Loss: 0.0441, Training Accuracy: 98.59%
Epoch [10/10], Loss: 0.0405, Training Accuracy: 98.71%
```

In [4]:

```
model.eval()
with torch.no_grad():
    correct = 0
    total = 0

    for images, labels in test_loader:
        images = images.reshape(-1, 28*28).to(device)
labels = labels.to(device)
outputs = model(images)
_, predicted = torch.max(outputs.data, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()
print(f'Test Accuracy of the model: {100 * correct / total} %')
```

```
Test Accuracy of the model: 100.0 %
```

In [ ]: