MNIST dataset, preparation and setup

Elena Pashkova

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

```
# Define the transformations for the dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
```

```
# Download the MNIST training and test datasets
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
```

Had some errors while dowloading MNIST dataset. However, it looks like PyTorch successfully switched to a backup URL (from Amazon S3), and the dataset was downloaded and extracted. So, this is fine as the dataset is correctly loaded despite the 403 errors.

```
from torch.utils.data import DataLoader
```

```
# Data loaders
train_loader = DataLoader(dataset=train_dataset, batch_size=100, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=100, shuffle=False)
```

Defining the Model (MLP)

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        # Fully connected layers
        self.fc1 = nn.Linear(28*28, 512)  # MNIST images are 28x28 pixels
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)  # 10 classes for digits 0-9
        self.relu = nn.ReLU()
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):
        x = x.view(-1, 28*28)  # Flatten the image into a vector
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.softmax(self.fc3(x))
        return x
```

```
# Instantiate the model
model = MLP()
```

Print model summary: Let's print the summary of the model to check the architecture and the number of parameters.

```
from torchsummary import summary
```

```
# Summary of the model
summary(model, (1, 28, 28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 512]         401,920
              ReLU-2                  [-1, 512]               0
            Linear-3                  [-1, 256]         131,328
              ReLU-4                  [-1, 256]               0
            Linear-5                   [-1, 10]           2,570
```

```
        LogSoftmax-6                  [-1, 10]               0
================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 2.04
Estimated Total Size (MB): 2.06
----------------------------------------------------------------
```

Model Mid-Summary: The model is a Multilayer Perceptron (MLP) consisting of three fully connected layers:

Layer 1 (Linear-1): This is a fully connected layer with 512 output neurons. It takes the flattened 28x28 MNIST image (784 input features). It has 401,920 trainable parameters.

Activation (ReLU-2): A Rectified Linear Unit (ReLU) activation function follows Layer 1 to introduce non-linearity. It doesn't have trainable parameters.

Layer 2 (Linear-3): This layer has 256 output neurons. It takes the 512-dimensional input from the previous layer. It has 131,328 trainable parameters.

Activation (ReLU-4): Another ReLU activation function follows Layer 2 to introduce non-linearity.

**Layer 3 (Linear-5):* The final fully connected layer maps the 256-dimensional output from Layer 2 to 10 output neurons (one for each class in the MNIST dataset). It has 2,570 trainable parameters.

Output Layer (LogSoftmax-6): A LogSoftmax layer converts the 10 outputs into log probabilities for classification.

The total number of trainable parameters is 535,818, and the total estimated size of the model is 2.06 MB. This architecture is well-suited for MNIST classification tasks.

Set up Loss Function and Optimizer

```
import torch.optim as optim

print(dir(torch.optim))
```

```
['ASGD', 'Adadelta', 'Adagrad', 'Adam', 'AdamW', 'Adamax', 'LBFGS', 'NAdam', 'Optimizer', 'RAdam', 'RMSprop', 'Rprop', 'SGD', 'SparseAda
```

```
loss_fn = nn.CrossEntropyLoss()  # Loss function for classification

# Optimizer: Adam optimizer with a learning rate of 0.001
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Training the Model

```
def train(model, train_loader, optimizer, loss_fn, num_epochs):
    for epoch in range(num_epochs):
        model.train()
        total_loss = 0
        for images, labels in train_loader:
            optimizer.zero_grad()  # Clear previous gradients
            outputs = model(images)  # Forward pass
            loss = loss_fn(outputs, labels)  # Compute loss
            loss.backward()  # Backpropagation
            optimizer.step()  # Update weights

            total_loss += loss.item()
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss/len(train_loader):.4f}")

# Train the model for 10 epochs
train(model, train_loader, optimizer, loss_fn, num_epochs=15)
```

```
Epoch [1/15], Loss: 0.3157
Epoch [2/15], Loss: 0.1364
Epoch [3/15], Loss: 0.1023
```

```
Epoch [4/15], Loss: 0.0793
Epoch [5/15], Loss: 0.0651
Epoch [6/15], Loss: 0.0602
Epoch [7/15], Loss: 0.0507
Epoch [8/15], Loss: 0.0447
Epoch [9/15], Loss: 0.0407
Epoch [10/15], Loss: 0.0343
Epoch [11/15], Loss: 0.0334
Epoch [12/15], Loss: 0.0310
Epoch [13/15], Loss: 0.0280
Epoch [14/15], Loss: 0.0269
Epoch [15/15], Loss: 0.0225
```

Evaluate the Model

```
# Evaluation Function: Evaluates the model on the test set
def evaluate(model, test_loader):
    model.eval()  # Set the model to evaluation mode
    total_correct = 0
    total_samples = 0
    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total_samples += labels.size(0)
            total_correct += (predicted == labels).sum().item()

    accuracy = 100 * total_correct / total_samples
    print(f'Accuracy: {accuracy:.2f}%')

# Evaluate the model on the test data
evaluate(model, test_loader)
```

```
Accuracy: 97.48%
```

Summary: In this project, I implemented a Multilayer Perceptron (MLP) using PyTorch to classify the MNIST dataset, which contains images of handwritten digits. I first normalized the dataset to ensure faster and more stable learning. The model architecture consists of three fully connected layers with 512, 256, and 10 neurons, respectively, along with ReLU activation functions. For the output, I used a LogSoftmax function.

I used the Adam optimizer with a learning rate of 0.001 and CrossEntropyLoss as our loss function. The training process ran for 15 epochs, and the training loss steadily decreased over time, indicating the model was learning effectively. After training, the model achieved an accuracy of 97.48% on the test dataset, which is a strong result for a simple feed-forward network.

Conclusions: The MLP model performed well on the MNIST dataset with a test accuracy of 97.48%, demonstrating that it is well-suited for this classification task. However, a limitation of this model is its reliance on fully connected layers, which may not be optimal for image data compared to convolutional neural networks (CNNs). Potential improvements could include adding dropout for regularization or switching to a CNN architecture to improve performance further on more complex datasets.