PyTorch Practical Exercises

Practical exercises to help solidify the concepts of using PyTorch to build neural networks:

Exercise 1: Setting Up PyTorch

1. **Task:** Install PyTorch on your local machine or cloud environment
   - **Command:** Use the appropriate command from [PyTorch's official site]() to install the latest version compatible with your setup.
   - For Google Colab:

     ```
     bash
     Copy code
     !pip install torch torchvision
     ```

2. **Task:** Verify the installation by importing PyTorch and checking the version.
   - **Code:**

     ```
     python
     Copy code
     import torch
     print(torch.__version__)
     ```

Exercise 2: Building a Simple Neural Network

1. **Task:** Define a simple feedforward neural network using `nn.Module`.
   - **Code:**

     ```
     python
     Copy code
     import torch.nn as nn

     class SimpleNN(nn.Module):
         def __init__(self):
             super(SimpleNN, self).__init__()
             self.fc1 = nn.Linear(784, 128)  # Input layer
             self.fc2 = nn.Linear(128, 64)   # Hidden layer
             self.fc3 = nn.Linear(64, 10)    # Output layer

         def forward(self, x):
             x = torch.relu(self.fc1(x))
             x = torch.relu(self.fc2(x))
             x = torch.softmax(self.fc3(x), dim=1)
             return x

     # Instantiate the network
     model = SimpleNN()
     print(model)
     ```

Exercise 3: Training the Neural Network

1. **Task:** Train the neural network on a simple dataset, such as MNIST.
   - **Code:**

```
python
Copy code
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms

# Load the MNIST dataset
transform = transforms.Compose([transforms.ToTensor()])
trainset = datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,
batch_size=32, shuffle=True)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(1, 6):  # 5 epochs
    running_loss = 0.0
    for images, labels in trainloader:
        # Flatten the images into vectors
        images = images.view(images.shape[0], -1)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)

        # Compute loss
        loss = criterion(outputs, labels)

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

        # Update running loss
        running_loss += loss.item()

    print(f"Epoch {epoch}, Loss:
{running_loss/len(trainloader)}")
```

Exercise 4: Evaluating the Model

1. **Task:** Evaluate the trained model on a validation set.
   - **Code:**

```
python
Copy code
testset = datasets.MNIST(root='./data', train=False,
download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset,
batch_size=32, shuffle=False)

correct = 0
```

```
total = 0

with torch.no_grad():
    for images, labels in testloader:
        images = images.view(images.shape[0], -1)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy: {100 * correct / total}%")
```

Exercise 5: Advanced Practice

1. **Task:** Implement a Convolutional Neural Network (CNN) using PyTorch and train it on the CIFAR-10 dataset.

The CIFAR dataset is a popular image dataset used for machine learning and computer vision tasks. There are two versions of the CIFAR dataset:

1. **CIFAR-10**: Consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class.
2. **CIFAR-100**: Consists of 60,000 32x32 color images in 100 classes, with 600 images per class.

Accessing the CIFAR Dataset in PyTorch

In PyTorch, you can easily access the CIFAR-10 or CIFAR-100 dataset using the `torchvision.datasets` module, which provides a straightforward way to load and preprocess these datasets.

Here's an example of how to load the CIFAR-10 dataset:

```
python
Copy code
import torch
import torchvision
import torchvision.transforms as transforms

# Define a transformation to apply to the images
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Download and load the training dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
shuffle=True)

# Download and load the test dataset
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)
```

```
testloader = torch.utils.data.DataLoader(testset, batch_size=32,
shuffle=False)

# Classes in CIFAR-10
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
'ship', 'truck')
```

Accessing CIFAR-100:

If you want to work with the CIFAR-100 dataset, you can simply replace `CIFAR10` with `CIFAR100` in the code above:

```
python
Copy code
trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
download=True, transform=transform)
testset = torchvision.datasets.CIFAR100(root='./data', train=False,
download=True, transform=transform)
```

This will load the CIFAR-100 dataset, which has 100 classes instead of 10.

2. **Task:** Implement transfer learning by fine-tuning a pretrained model like ResNet on a custom dataset.