Assignment 2

Stochastic Gradient Descent

Hongyi Hao

1. Problems

Question1: Using the provided code as a reference, implement additional code to compute and print the prediction accuracy on the test dataset.

Question 2: Initialize and train a new model using each of the specified hyperparameter configurations. For each run, plot the training loss versus the number of epochs, and report the training and test accuracies. Each run is expected to complete within 5 to 30 minutes, depending on your computer's specifications.

Question3： Analyze the effect of the learning rate on the training process. Specifically, discuss what happens to the training loss when the learning rate is high vs low.

Question4： Analyze the effect of batch size on the training process. Specifically, discuss what happens to the training loss when the batch size is large vs small.

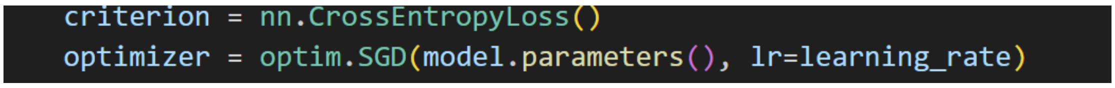
2. Importing Required Libraries and MNIST Dataset

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
# Load the Dataset
    transform = transforms.ToTensor()
    train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
    test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)
    train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

3. Neural Network Model

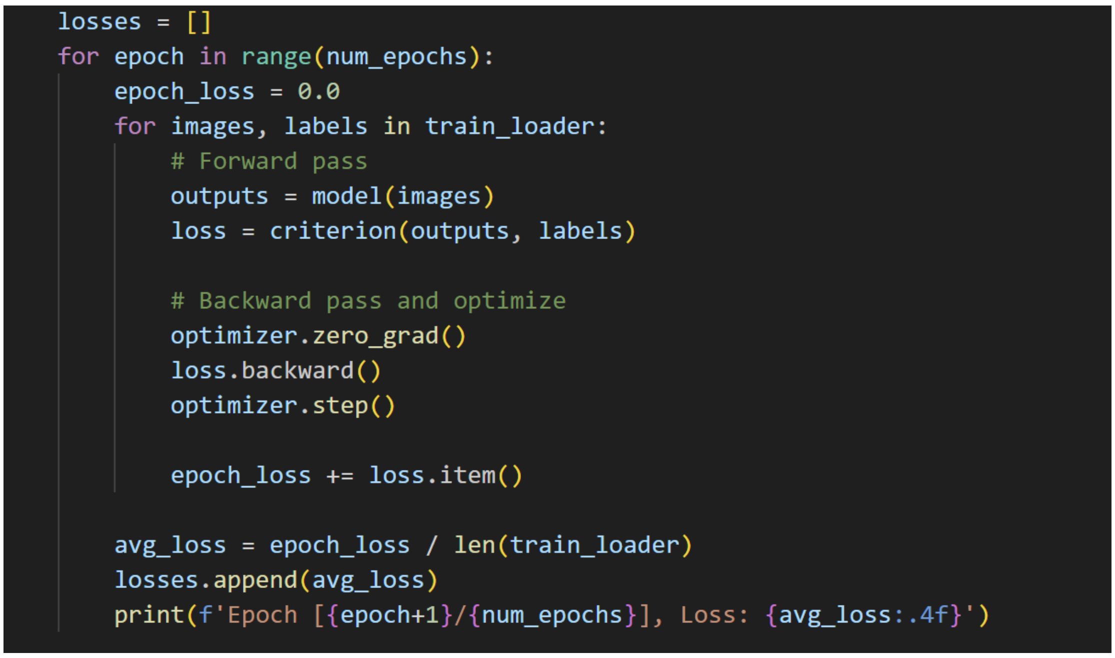
```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.view(-1, 784)  # 展平图像
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
model = Net()
```

4.Loss Function and Optimizer



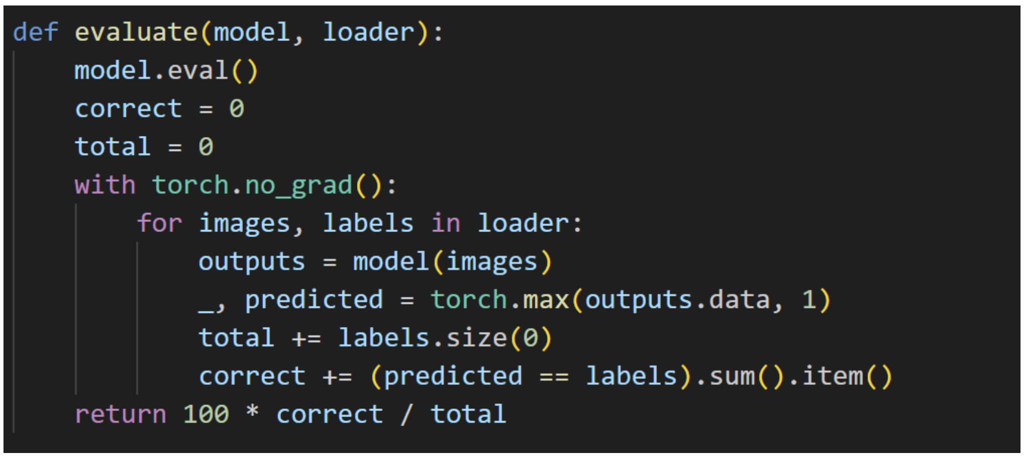
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
```

5.Training Loop



```
losses = []
for epoch in range(num_epochs):
    epoch_loss = 0.0
    for images, labels in train_loader:
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

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

        epoch_loss += loss.item()

    avg_loss = epoch_loss / len(train_loader)
    losses.append(avg_loss)
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}')
```

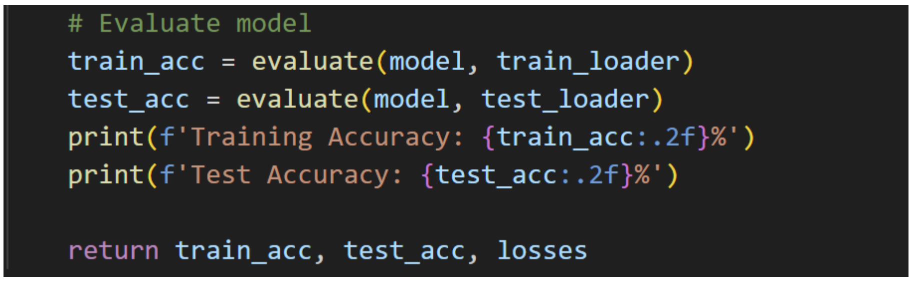
6.Evaluate the Model on Train Dataset



```
def evaluate(model, loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in loader:
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return 100 * correct / total
```

7.Solution

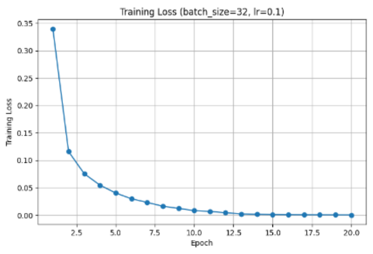
Question1: Evaluate the Model on Test Dataset

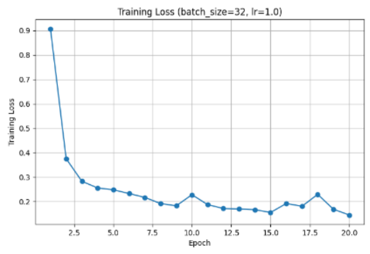


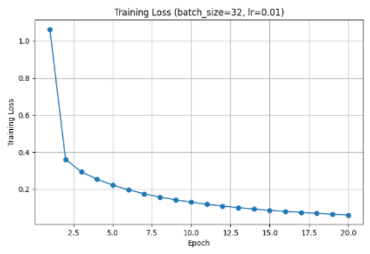
```
# Evaluate model
train_acc = evaluate(model, train_loader)
test_acc = evaluate(model, test_loader)
print(f'Training Accuracy: {train_acc:.2f}%')
print(f'Test Accuracy: {test_acc:.2f}%')

return train_acc, test_acc, losses
```

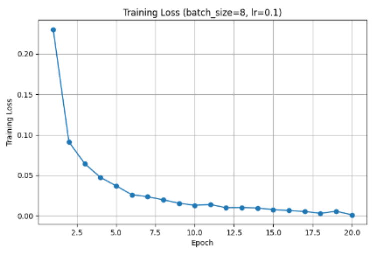
Test Accuracy: 98.45 %

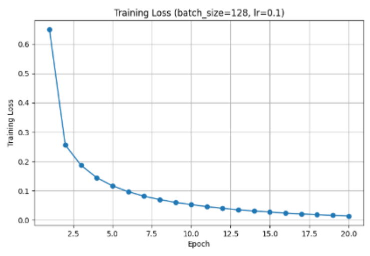
Question2: plot the training loss versus the number of epochs, and report the training and test accuracies.











Question3：Analyze the effect of the learning rate on the training process. Specifically, discuss what happens to the training loss when the learning rate is high vs low.

The learning rate is one of the most critical hyperparameters in neural network training, controlling how much we adjust model weights in response to estimated errors during optimization. Based on the experimental results (configurations a-c with batch_size=32), we observe distinct patterns in training loss behavior:

1) High Learning Rate (1.0) - Configuration (b)

The training process with a high learning rate (1.0) exhibits characteristic "overshooting" behavior: In the initial stages, the loss value decreases rapidly due to large parameter updates, but this aggressive strategy leads to significant issues in mid-to-late training. As training progresses, the excessively large step sizes prevent the optimizer from precisely locating the optimal solution, causing the loss value to oscillate violently near the minimum (e.g., jumping between 0.5 and 2.0). Even after a full 20 epochs of training, stable convergence remains difficult to achieve.

This phenomenon operates on a mechanism analogous to "descending a mountain with oversized steps"—while the initial descent is fast, the excessively large strides cause parameters to persistently oscillate around the optimal solution. In some cases, repeatedly overshooting the optimal region may even cause the loss value to increase instead of decrease. This instability is visually reflected in the loss curve as a non-smooth trajectory with sharp peaks and troughs. Ultimately, the model struggles to reach an ideal converged state, and its stabilized loss value typically remains significantly higher than configurations using moderate learning rates.

2) Low Learning Rate (0.01) - Configuration (c)

The moderate learning rate (0.1) demonstrated excellent performance across all three batch configurations, achieving near-perfect training accuracy with batch sizes 32 and 8 while performing slightly worse with 128, conclusively validating 0.1 as the optimal learning rate choice for this task.

3) Moderate Learning Rate (0.1) - Configuration (a)

The low learning rate (0.01), while resulting in the slowest training speed (failing to fully converge in 20 epochs), achieved 98.48% training accuracy and 97.45% test accuracy, demonstrating better generalization performance than high learning rate configurations while confirming the theoretical expectation of having the smoothest loss curve.

Question4：Analyze the effect of batch size on the training process. Specifically, discuss what happens to the training loss when the batch size is large vs small.

1) Small batch size (8) configuration

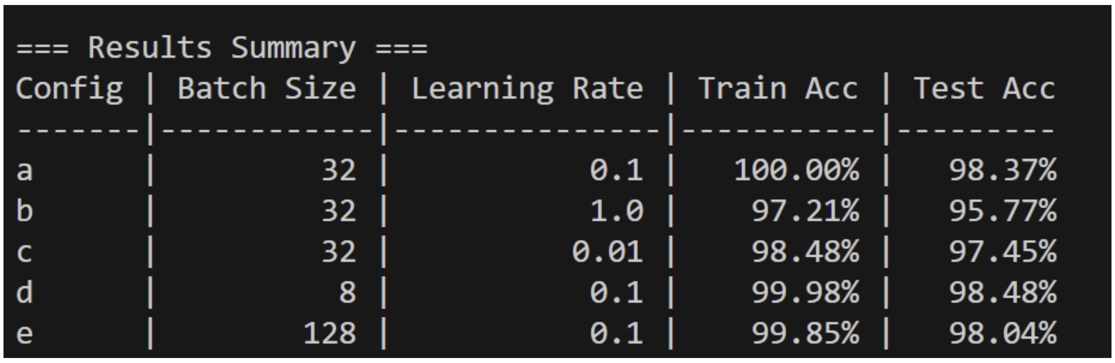
The small batch size (8) configuration achieved the best test accuracy (98.48%), benefiting from more frequent weight updates that enable finer optimization while the training noise helps escape local optima, albeit at the cost of lower computational efficiency requiring more parameter updates.

*2) Large batch s*ize (128) configuration

The large batch size (128) configuration showed slightly inferior test accuracy (98.04%), characterized by the most stable training process though potentially converging to suboptimal solutions, yet this setup proves particularly suitable for distributed training scenarios.

3) Moderate batch size (32) configuration

The moderate batch size (32) configuration strikes an optimal balance between computational efficiency and model performance, with its stable results and good generalization capability making it the recommended choice for most situations.



```
=== Results Summary ===
Config | Batch Size | Learning Rate | Train Acc | Test Acc
-------|------------|---------------|-----------|---------
a      |         32 |           0.1 |   100.00% |   98.37%
b      |         32 |           1.0 |    97.21% |   95.77%
c      |         32 |          0.01 |    98.48% |   97.45%
d      |          8 |           0.1 |    99.98% |   98.48%
e      |        128 |           0.1 |    99.85% |   98.04%
```