Assignment_01

July 29, 2024

[32]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from torchvision import datasets, transforms, utils
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import torchvision
```

[33]:
```
batch_size = 64
learning_rate = 0.001
num_epochs = 10
```

[34]:
```
# Transformations for the training and test sets
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Load the training and test sets
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)

# Data loaders
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,␣
 ↪batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,␣
 ↪batch_size=batch_size, shuffle=False)
```

[35]:
```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 512)
        self.fc2 = nn.Linear(512, 256)
```

```
        self.fc3 = nn.Linear(256, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)  # Flatten the input tensor
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = SimpleNN()
```

[36]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

[37]:
```
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/
 ↪{len(train_loader)}], Loss: {loss.item():.4f}')
```

```
Epoch [1/10], Step [100/938], Loss: 0.4324
Epoch [1/10], Step [200/938], Loss: 0.2779
Epoch [1/10], Step [300/938], Loss: 0.1745
Epoch [1/10], Step [400/938], Loss: 0.3732
Epoch [1/10], Step [500/938], Loss: 0.1378
Epoch [1/10], Step [600/938], Loss: 0.1605
Epoch [1/10], Step [700/938], Loss: 0.0519
Epoch [1/10], Step [800/938], Loss: 0.0654
Epoch [1/10], Step [900/938], Loss: 0.1549
Epoch [2/10], Step [100/938], Loss: 0.0188
Epoch [2/10], Step [200/938], Loss: 0.2365
Epoch [2/10], Step [300/938], Loss: 0.2074
Epoch [2/10], Step [400/938], Loss: 0.1378
Epoch [2/10], Step [500/938], Loss: 0.0787
Epoch [2/10], Step [600/938], Loss: 0.1287
Epoch [2/10], Step [700/938], Loss: 0.1627
Epoch [2/10], Step [800/938], Loss: 0.1103
Epoch [2/10], Step [900/938], Loss: 0.0332
Epoch [3/10], Step [100/938], Loss: 0.1013
Epoch [3/10], Step [200/938], Loss: 0.2451
```

```
Epoch [3/10], Step [300/938], Loss: 0.1702
Epoch [3/10], Step [400/938], Loss: 0.0564
Epoch [3/10], Step [500/938], Loss: 0.0173
Epoch [3/10], Step [600/938], Loss: 0.1185
Epoch [3/10], Step [700/938], Loss: 0.0497
Epoch [3/10], Step [800/938], Loss: 0.1140
Epoch [3/10], Step [900/938], Loss: 0.0877
Epoch [4/10], Step [100/938], Loss: 0.0301
Epoch [4/10], Step [200/938], Loss: 0.0219
Epoch [4/10], Step [300/938], Loss: 0.0899
Epoch [4/10], Step [400/938], Loss: 0.0430
Epoch [4/10], Step [500/938], Loss: 0.0570
Epoch [4/10], Step [600/938], Loss: 0.0689
Epoch [4/10], Step [700/938], Loss: 0.0468
Epoch [4/10], Step [800/938], Loss: 0.1556
Epoch [4/10], Step [900/938], Loss: 0.2259
Epoch [5/10], Step [100/938], Loss: 0.0333
Epoch [5/10], Step [200/938], Loss: 0.0400
Epoch [5/10], Step [300/938], Loss: 0.0720
Epoch [5/10], Step [400/938], Loss: 0.0721
Epoch [5/10], Step [500/938], Loss: 0.1190
Epoch [5/10], Step [600/938], Loss: 0.0520
Epoch [5/10], Step [700/938], Loss: 0.1079
Epoch [5/10], Step [800/938], Loss: 0.0072
Epoch [5/10], Step [900/938], Loss: 0.0529
Epoch [6/10], Step [100/938], Loss: 0.0278
Epoch [6/10], Step [200/938], Loss: 0.0268
Epoch [6/10], Step [300/938], Loss: 0.0037
Epoch [6/10], Step [400/938], Loss: 0.0069
Epoch [6/10], Step [500/938], Loss: 0.0694
Epoch [6/10], Step [600/938], Loss: 0.0162
Epoch [6/10], Step [700/938], Loss: 0.2042
Epoch [6/10], Step [800/938], Loss: 0.0583
Epoch [6/10], Step [900/938], Loss: 0.0597
Epoch [7/10], Step [100/938], Loss: 0.0568
Epoch [7/10], Step [200/938], Loss: 0.0214
Epoch [7/10], Step [300/938], Loss: 0.0451
Epoch [7/10], Step [400/938], Loss: 0.0650
Epoch [7/10], Step [500/938], Loss: 0.1262
Epoch [7/10], Step [600/938], Loss: 0.0297
Epoch [7/10], Step [700/938], Loss: 0.2784
Epoch [7/10], Step [800/938], Loss: 0.1150
Epoch [7/10], Step [900/938], Loss: 0.0616
Epoch [8/10], Step [100/938], Loss: 0.1569
Epoch [8/10], Step [200/938], Loss: 0.0280
Epoch [8/10], Step [300/938], Loss: 0.1508
Epoch [8/10], Step [400/938], Loss: 0.0498
Epoch [8/10], Step [500/938], Loss: 0.0062
```

```
Epoch [8/10], Step [600/938], Loss: 0.0985
Epoch [8/10], Step [700/938], Loss: 0.0136
Epoch [8/10], Step [800/938], Loss: 0.1082
Epoch [8/10], Step [900/938], Loss: 0.0105
Epoch [9/10], Step [100/938], Loss: 0.0636
Epoch [9/10], Step [200/938], Loss: 0.1002
Epoch [9/10], Step [300/938], Loss: 0.0073
Epoch [9/10], Step [400/938], Loss: 0.0068
Epoch [9/10], Step [500/938], Loss: 0.0019
Epoch [9/10], Step [600/938], Loss: 0.0952
Epoch [9/10], Step [700/938], Loss: 0.0419
Epoch [9/10], Step [800/938], Loss: 0.1478
Epoch [9/10], Step [900/938], Loss: 0.0293
Epoch [10/10], Step [100/938], Loss: 0.0355
Epoch [10/10], Step [200/938], Loss: 0.0069
Epoch [10/10], Step [300/938], Loss: 0.0697
Epoch [10/10], Step [400/938], Loss: 0.1980
Epoch [10/10], Step [500/938], Loss: 0.0027
Epoch [10/10], Step [600/938], Loss: 0.0102
Epoch [10/10], Step [700/938], Loss: 0.1465
Epoch [10/10], Step [800/938], Loss: 0.0089
Epoch [10/10], Step [900/938], Loss: 0.0344
```

[38]:
```python
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print(f'Accuracy of the model on the 10000 test images: {100 * correct /␣
↪total}%')
```

```
Accuracy of the model on the 10000 test images: 97.67%
```

[38]:

[38]:

[38]: