

H5P1

November 1, 2024

```
[ ]: from torchvision import datasets
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
import torch
from torch import nn
import numpy as np
import matplotlib.pyplot as plt

[ ]: # Load the USPS dataset
train_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),
    ↪train=True)
test_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),
    ↪train=False)

# Create DataLoaders for training and testing
train_loader = DataLoader(train_data, batch_size=1024, shuffle=True)
test_loader = DataLoader(test_data, batch_size=len(test_data), shuffle=False)

[ ]: # Define the MLP model with 2 hidden layers, both with 128 units
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.mlp = nn.Sequential(
            nn.Linear(16 * 16, 128), # Input layer to first hidden layer
            nn.ReLU(),
            nn.Linear(128, 128),     # First hidden layer to second hidden
            ↪layer
            nn.ReLU(),
            nn.Linear(128, 128),     # Second hidden layer
            nn.ReLU(),
            nn.Linear(128, 10)       # Output layer for 10 classes
        )

    def forward(self, X):
        return self.mlp(self.flatten(X))
```

```

model = MLP().to('cuda')
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.5)

```

```

[ ]: %%time
epochs = 1000
CE = torch.zeros((epochs))
Training = torch.zeros((epochs))
Test = torch.zeros((epochs))
for epoch in range(epochs):
    cumulative_accuracy = 0
    cumulative_loss = 0
    for X, Y in train_loader:
        X, Y = X.to('cuda'), Y.to('cuda')
        out = model(X)
        loss = loss_fn(out, Y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        cumulative_loss += loss.item()
        cumulative_accuracy += (out.argmax(axis=1) == Y).sum().item()
    CE[epoch] = cumulative_loss / len(train_loader)
    Training[epoch] = cumulative_accuracy / len(train_data)
    with torch.no_grad():
        for Xt, Yt in test_loader:
            Xt, Yt = Xt.to('cuda'), Yt.to('cuda')
            test_out = model(Xt)
            test_accuracy_epoch = (test_out.argmax(axis=1) == Yt).sum().item() /
↪ len(test_data)
        Test[epoch] = test_accuracy_epoch
    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch + 1}/{epochs} | Loss: {CE[epoch]:.4f} | Training_
↪Accuracy: {Training[epoch]:.4f} | Test Accuracy: {Test[epoch]:.4f}")

```

```

Epoch 100/1000 | Loss: 0.0631 | Training Accuracy: 0.9815 | Test Accuracy:
0.9268
Epoch 200/1000 | Loss: 0.1229 | Training Accuracy: 0.9624 | Test Accuracy:
0.9033
Epoch 300/1000 | Loss: 0.0545 | Training Accuracy: 0.9824 | Test Accuracy:
0.9178
Epoch 400/1000 | Loss: 0.0751 | Training Accuracy: 0.9772 | Test Accuracy:
0.9203
Epoch 500/1000 | Loss: 0.0304 | Training Accuracy: 0.9938 | Test Accuracy:
0.9178
Epoch 600/1000 | Loss: 0.0187 | Training Accuracy: 0.9945 | Test Accuracy:
0.9297
Epoch 700/1000 | Loss: 0.0057 | Training Accuracy: 0.9989 | Test Accuracy:
0.9273

```

Epoch 800/1000 | Loss: 0.7398 | Training Accuracy: 0.7594 | Test Accuracy: 0.7260
Epoch 900/1000 | Loss: 0.2131 | Training Accuracy: 0.9317 | Test Accuracy: 0.8660
Epoch 1000/1000 | Loss: 0.0930 | Training Accuracy: 0.9728 | Test Accuracy: 0.9113
CPU times: user 12min 38s, sys: 1.7 s, total: 12min 39s
Wall time: 12min 44s

```
[ ]: epochs_range = np.arange(1, epochs + 1)
      CE_np = CE.cpu().numpy()
      Training_np = Training.cpu().numpy()
      Test_np = Test.cpu().numpy()

      plt.figure(figsize=(10, 5))
      plt.plot(epochs_range, CE_np, label="Cross Entropy Loss")
      plt.xlabel("Epoch")
      plt.ylabel("Loss")
      plt.title("Cross Entropy Loss over Epochs")
      plt.legend()
      plt.show()

      plt.figure(figsize=(10, 5))
      plt.plot(epochs_range, Training_np, label="Training Accuracy")
      plt.plot(epochs_range, Test_np, label="Test Accuracy")
      plt.xlabel("Epoch")
      plt.ylabel("Accuracy")
      plt.title("Training and Test Accuracy over Epochs")
      plt.legend()
      plt.show()
```

