ROHAN NYATI

500075940

R177219148

BATCH – 5 (Ai & Ml)

Experiment -2

Steps to Implement in PyTorch -

- ➔ Import PyTorch
- ➔ Initialize Hyper-parameters
- ➔ Download MNIST Dataset
- ➔ Load the Dataset
- ➔ Build the Feedforward Neural Network
- ➔ Instantiate the FNN
- ➔ Enable GPU
- ➔ Choose the Loss Function and Optimizer
- ➔ Training the FNN Model
- ➔ Testing the FNN Model
- ➔ Save the trained FNN Model for future use

```
# Import torch

import torch
import torch.nn as nn
import torchvision
import matplotlib.pyplot as plt

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Running on '{device}'")
```

```
# Downloading dataset
```

```python
from torchvision import datasets

train_data = datasets.MNIST(
    root = 'data',
    train = True,
    download = True,
)

test_data = datasets.MNIST(
    root = 'data',
    train = False,
    download = True,
)

# Loading dataset

train_labels = train_data.targets
test_labels = test_data.targets
train_data = train_data.data
test_data = test_data.data

print(f"train_data.shape:{train_data.shape} test_data.shape:{test_data.shape}")

# Exploring data

plt.imshow(train_data[0], cmap='gray')
plt.show()

# Set hyperparameters

n_train_sample = train_data.shape[0]
n_test_samples = test_data.shape[0]

image_size = train_data.shape[-1]
input_size = image_size * image_size
hidden_layer_size = 512
n_classes = 10

n_iters = 50
```

```python
batch_size = 2000
lr = 0.001
```

```python
# Transform data

train_data = train_data.to(torch.float32)/255.
test_data = test_data.to(torch.float32)/255.
train_data = train_data.reshape(-1, input_size)
test_data = test_data.reshape(-1, input_size)

# Store all data on gpu

train_data = train_data.to(device)
test_data = test_data.to(device)
train_labels = train_labels.to(device)
test_labels = test_labels.to(device)

# Making model

class FFNN(nn.Module):

   def __init__(self, input_size, hidden_layer_size, n_classes):
       super(FFNN, self).__init__()
       self.input_size = input_size
       self.hidden_layer_size = hidden_layer_size
       self.n_classes = n_classes
       # Adding layers
       self.l1 = nn.Linear(input_size, hidden_layer_size)
       self.l1_activation = nn.ReLU()
       self.l2 = nn.Linear(hidden_layer_size, n_classes)

   def forward(self, x):
       z = self.l1(x)
       a = self.l1_activation(z)
       z2 = self.l2(a)
       return z2

# Making model and training
```

```
ffnn = FFNN(input_size, hidden_layer_size, n_classes)
ffnn.to(device)

# Loss and Optimizer

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(ffnn.parameters(), lr=lr)
```

```

# Training

losses = []
test_losses = []

for epoch in range(n_iters):
    avg, c = 0, 0
    for idx in range(0, n_train_sample, batch_size):
        # Get images and labels for current epoch
        batch_x = train_data[idx:idx+batch_size]
        batch_labels = train_labels[idx:idx+batch_size]
        # Forward pass
        predictions = ffnn.forward(batch_x)
        # Compute Loss
        loss = criterion(predictions, batch_labels)
        losses.append(float(loss))
        avg, c = avg+loss, c+1
        # Backprop
        optimizer.zero_grad() # Reset gradients
        loss.backward() # Recompute gradients
        optimizer.step() # Update weights
        print(f"Epoch:{epoch+1}\tBatch:{idx}\tLoss:{loss}")
    # Output loss every 100 epochs
    if ((epoch + 1) % 5) == 0:
        print(f"Epoch:{epoch+1}\tAverageLoss:{(avg/c)}")
        # Compute loss on test set
        predictions = ffnn.forward(test_data)
        test_loss = criterion(predictions, test_labels)
        test_losses.append(float(test_loss))
```

```python
# Plot losses

plt.plot(losses)
plt.title("Training Loss")
plt.show()

# Plot test losses

plt.plot(test_losses)
plt.title("Test Losses")
plt.show()

# Finally compute loss on both sets once more

predictions = ffnn.forward(train_data)
train_loss = criterion(predictions, train_labels)
print(f"Loss on train set: {train_loss}")

predictions = ffnn.forward(test_data)
test_loss = criterion(predictions, test_labels)
print(f"Loss on test set: {test_loss}")

# Save model for future use

torch.save(ffnn.state_dict(), "./ffnn_mnist.torch")
```

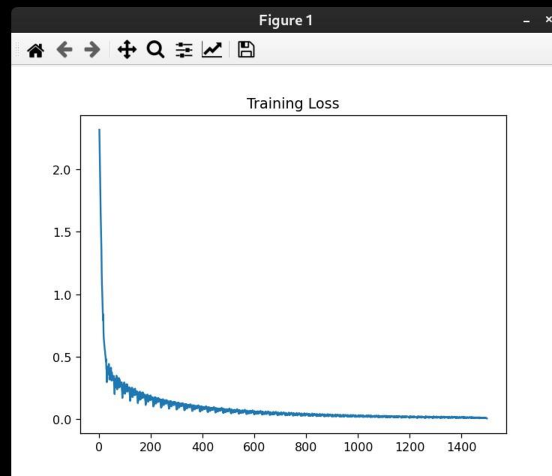
OUTPUT AND SCREENSHOTS


matplotlib 601.3 B/s 699.3 B/s
Fri Jan 28 14:22:41
65 %
/run/me/f/S/w/UPESx162/SEM VI/N/2 master ↑1 ?1 python mnist.py
Running on 'cuda'
train_data.shape:torch.Size([60000, 28, 28]) test_data.shape:torch.Size([10000, 28, 28])
Figure 1
0
5
10
15
20
25

matplotlib ↓ 560.0 B/s ↑ 830.0 B/s Fri Jan 28 14:23:32 65 %

pyth st.py

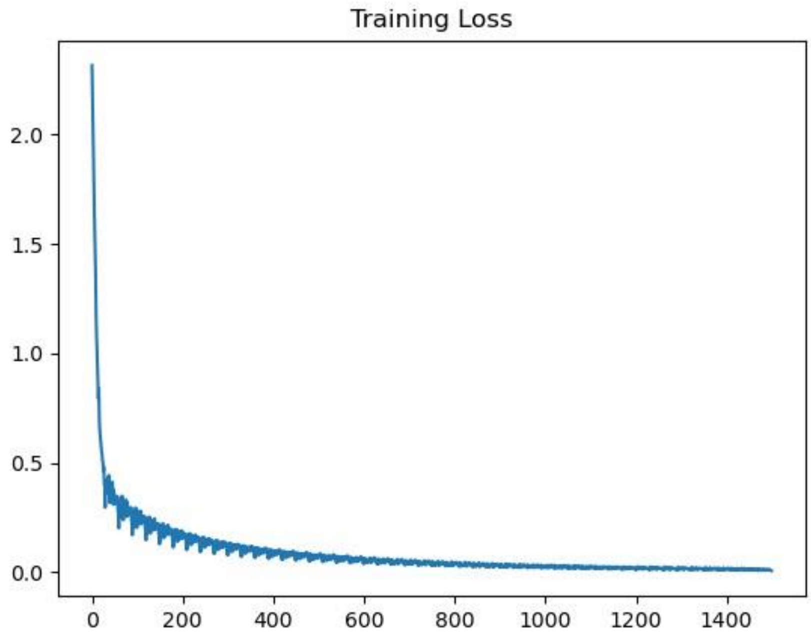
```
Epoch:50        Batch:8000      Loss:0.010021898895502090
Epoch:50        Batch:10000     Loss:0.009307419881224632
Epoch:50        Batch:12000     Loss:0.007356675807386637
Epoch:50        Batch:14000     Loss:0.009302264079451561
Epoch:50        Batch:16000     Loss:0.011390299536287785
Epoch:50        Batch:18000     Loss:0.008497527800500393
Epoch:50        Batch:20000     Loss:0.012792868539690971
Epoch:50        Batch:22000     Loss:0.010484099388122559
Epoch:50        Batch:24000     Loss:0.008942842483520508
Epoch:50        Batch:26000     Loss:0.011973166838288307
Epoch:50        Batch:28000     Loss:0.00843243207782507
Epoch:50        Batch:30000     Loss:0.009519902057945728
Epoch:50        Batch:32000     Loss:0.008697958663105965
Epoch:50        Batch:34000     Loss:0.012340042740106583
Epoch:50        Batch:36000     Loss:0.012927193194627762
Epoch:50        Batch:38000     Loss:0.009917577728629112
Epoch:50        Batch:40000     Loss:0.009576226584613323
Epoch:50        Batch:42000     Loss:0.010907260701060295
Epoch:50        Batch:44000     Loss:0.009496862068772316
Epoch:50        Batch:46000     Loss:0.010957431048154831
Epoch:50        Batch:48000     Loss:0.010440832003951073
Epoch:50        Batch:50000     Loss:0.012056056410074234
Epoch:50        Batch:52000     Loss:0.00968592707067728
Epoch:50        Batch:54000     Loss:0.00933942198753357
Epoch:50        Batch:56000     Loss:0.0079934848472476
Epoch:50        Batch:58000     Loss:0.0057856556959450245
Epoch:50        AverageLoss:0.010274291038513184
```

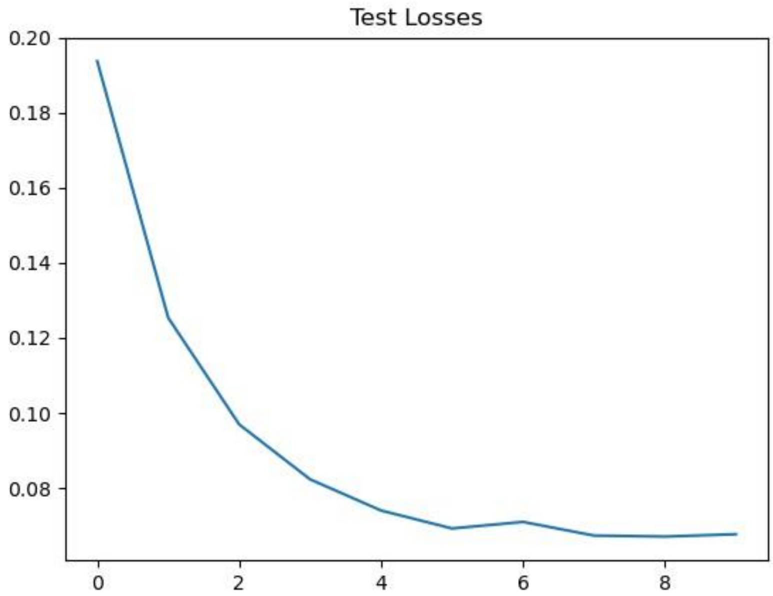


Terminal ↓ 996.7 B/s ↑ 2.0 KB/s Fri Jan 28 14:23:52 65 %

felix@thunderchild:/run/media/felix/SIGM k/UPESx162/SEM VI/Neural Networks Lab/2

```
Epoch:50        Batch:12000     Loss:0.007356675807386637
Epoch:50        Batch:14000     Loss:0.009302264079451561
Epoch:50        Batch:16000     Loss:0.011390299536287785
Epoch:50        Batch:18000     Loss:0.008497527800500393
Epoch:50        Batch:20000     Loss:0.012792868539690971
Epoch:50        Batch:22000     Loss:0.010484099388122559
Epoch:50        Batch:24000     Loss:0.008942842483520508
Epoch:50        Batch:26000     Loss:0.011973166838288307
Epoch:50        Batch:28000     Loss:0.00843243207782507
Epoch:50        Batch:30000     Loss:0.009519902057945728
Epoch:50        Batch:32000     Loss:0.008697958663105965
Epoch:50        Batch:34000     Loss:0.012340042740106583
Epoch:50        Batch:36000     Loss:0.012927193194627762
Epoch:50        Batch:38000     Loss:0.009917577728629112
Epoch:50        Batch:40000     Loss:0.009576226584613323
Epoch:50        Batch:42000     Loss:0.010907260701060295
Epoch:50        Batch:44000     Loss:0.009496862068772316
Epoch:50        Batch:46000     Loss:0.010957431048154831
Epoch:50        Batch:48000     Loss:0.010440832003951073
Epoch:50        Batch:50000     Loss:0.012056056410074234
Epoch:50        Batch:52000     Loss:0.00968592707067728
Epoch:50        Batch:54000     Loss:0.00933942198753357
Epoch:50        Batch:56000     Loss:0.0079934848472476
Epoch:50        Batch:58000     Loss:0.0057856556959450245
Epoch:50        AverageLoss:0.010274291038513184
Loss on train set: 0.009162847883999348
Loss on test set: 0.06766894459724426
/run/me/f/S/w/UPESx162/SEM VI/N/2 > master ↑1 ?1              ✔ < 1m 16s
```


Training Loss
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400


Test Losses
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0
2
4
6
8