Experiment 4 BUILD A SIMPLE FEED FORWARD
14.8.2025 NEURAL NETWORK TO RECOGNIZE HANDWRITTEN CHARACTER

Aim: To design and implement a simple feedforward neural network that can classify handwritten characters

Objective: To design and implement a FNN including forward propagation, activation functions and backpropagation for training and understand it

Pseudocode:

→ Load the Dataset
- Normalize pixel values

→ Initialize the Network
- Define input layer size
- Define hidden layer size
- Define output layer size
- Initialize weights and biases randomly.

→ Define activation functions

→ Do forward and backward propagation

→ Training Loop:
For number of epochs:
- Forward propagate all training samples
- Backpropagate errors
- update weights and biases

→ Evaluate Model:

→ Output:

Notes:

→ Device configuration is done so that code can automatically decide whether to run on CPU or GPU.

→ MNIST images have pixel values from 0 to 255 so normalization rescales pixels to a smaller range

→ MNIST contains 60,000 training data and 10,000 testing images

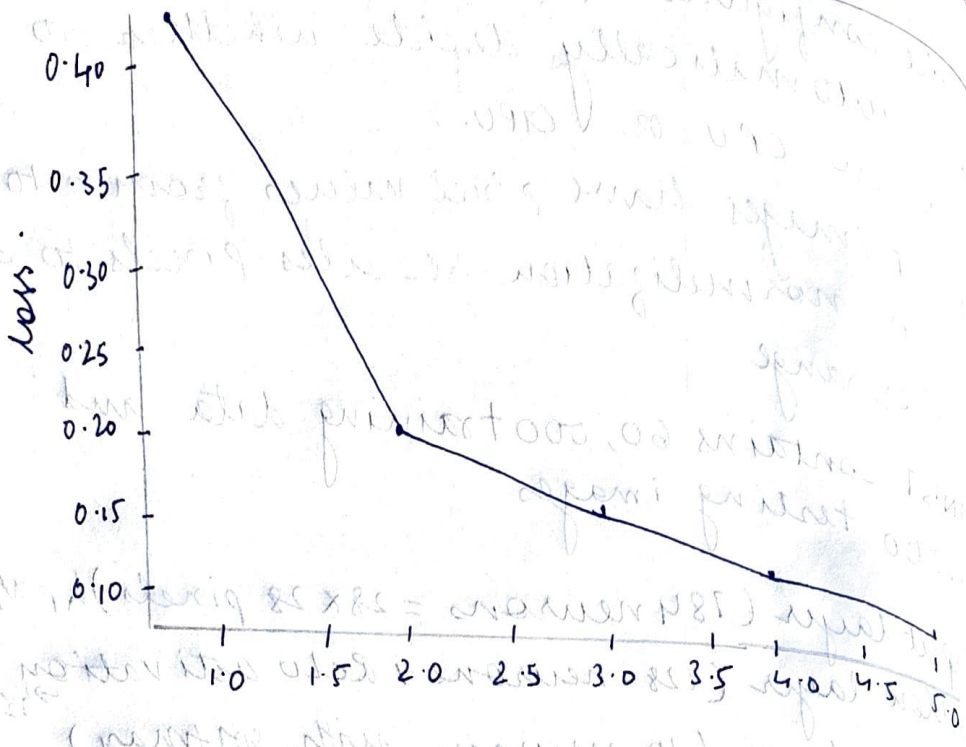
→ Input layer (784 neurons = 28×28 pixels) $h_1 = W_1 x + b_1$
Hidden layer (128 neurons + ReLU activation) $\Rightarrow h_2 = W_2 a_1 + b_2$
Output layer (10 neurons with softmax)
↳ $z = W_3 a_2 + b_3$
(logit)

→ Logit refer to the raw, unnormalized outputs from the final layer of a model before any activation function is applied.

→ Input image → Flatten - fc1
Apply ReLU → non-linear activation
fc2 → Apply ReLU
fc3 → get logits
Pass logits to crossentropy → compare with true labels to compute loss

Observation

Epoch [1/5] : Loss : 0.4134
Epoch [2/5] ; Loss : 0.1975
Epoch [3/5] , Loss : 0.1445
Epoch [4/5] , Loss : 0.1153
Epoch [5/5] , Loss : 0.0971
Test accuracy : 97.06%.



Result: Successfully implemented and evaluated using PyTorch.

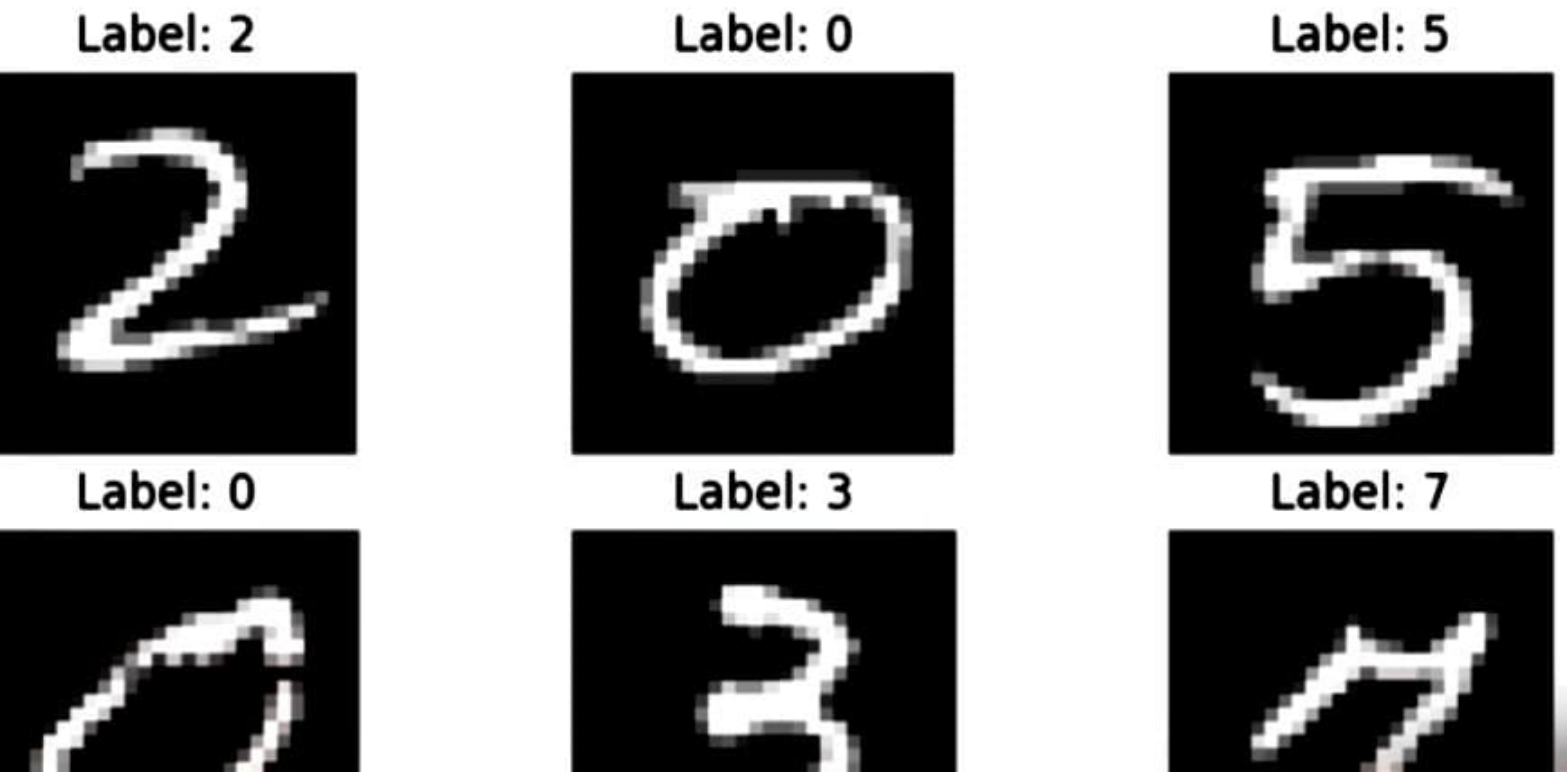
14/8/25

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

# 1. Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 2. Transform: normalize pixel values
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# 3. Load MNIST dataset
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)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False)

# 4. Build a simple Feed Forward Neural Network
class FFN(nn.Module):
    def __init__(self):
        super(FFN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)  # input -> hidden
        self.fc2 = nn.Linear(128, 64)     # hidden -> hidden
        self.fc3 = nn.Linear(64, 10)      # hidden -> output

    def forward(self, x):
        x = x.view(-1, 28*28)  # flatten image
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
```

```python
model = FFN().to(device)

# 5. Loss and optimizer
criterion = nn.CrossEntropyLoss()
c

# 6. Training Loop
epochs = 5
for epoch in range(epochs):
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.4f}")

# 7. Testing Loop
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

```
    Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.
4.127 nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-c
u12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvidia-cusparse-cu12-12.3.1.170 nvidia-nccl-cu12-2.21.5 nvidia-n
vjitlink-cu12-12.4.127
Epoch [1/5], Loss: 0.4134
Epoch [2/5], Loss: 0.1975
Epoch [3/5], Loss: 0.1445
Epoch [4/5], Loss: 0.1153
Epoch [5/5], Loss: 0.0977
Test Accuracy: 97.06%
```

```python
import matplotlib.pyplot as plt
import torchvision

# Get one batch of images
examples = iter(train_loader)
images, labels = next(examples)

# Plot first 6 images
plt.figure(figsize=(8, 4))
for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(images[i][0], cmap='gray')  # images[i][0] because MNIST is grayscale
    plt.title(f"Label: {labels[i].item()}")
    plt.axis('off')
plt.show()
```



```
plt.plot(epochs, loss_values, marker='o')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss Curve")
plt.grid(True)
plt.show()
```