RA2311047010044

INDEX

NAME K·D·D·SaiAbhiram SUBJECT Deep learning lab obdb

STD. DIV. ROLL NO. SCHOOL

SR. NO.	PAGE NO.	TITLE	DATE	TEACHER'S SIGN / REMARKS
1		Analyze and exploring the deep learning Platforms	24-07-25	
2		Implement a classifier using open-Source data set	7/8/25	7/11/25
3.		Study of the classifiers with respect to Statistical parameters	7/8/25	14/8/25
4.		Build a simple feed forward neural network to recognize hand written character	14/8/25	14/8/25
5.		Study of activation function and its role	28-8-25	
6.		Implement gradient descent & back Propagation in deep neural network,	09-9-25	9/9
7.		Build a CNN model to classify Cats & dogs images	16-9-25	23/9/25

14-08-25

4. Build a simple feed forward neural network to recognize hand written character.

Aim: to design and implement a simple feed forward neural network using open source dataset to recognize hand written character.

~~Algorithm~~:

Objective:

1. To load and preprocess the MNIST dataset for neural network input
2. to build feed forward network model with hidden layers
3. to train the model using gradient descent optimizer and sparse cross-entropy loss
4. Evaluate the trained model on test data and measure its accuracy
5. to predict the accuracy of a image of hand written character.

Pseudo Code:

Start

load MNIST dataset

Pattern each image from 28x28 to 784 features

normalize pixels values to range [0,1]

Output :

training

Epoch	Accuracy	Loss
1	0.8748	0.4363
2	0.9649	0.1134
3	0.9775	0.0715
4	0.9827	0.0548
5	0.9861	0.0431

accuracy : 0.9698

loss : 0.0963

testing:

accuracy : 97.46%

Create a sequential neural network

layer 1 : dense (128 neurons, Relu activation)
layer 2 : dense (64 neurons, Relu activation)
output layer: dense (10 neurons, softmax activation)

Compile model :-

optimizer = stochastic gradient descent
loss = sparse categorical crossentropy
metric = accuracy

train model on training data for epochs
evalute model on testing datay

observation :-

→ the loss decrease with each showing that model is learning

→ Accuracy improve stedily during learning

Result :- successfully built a simple feed forward neural network to recognize handwritten characters and accuracy while testing is 97.46%

```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

# Step 1: Transformations
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Step 2: Load MNIST dataset
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(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# Step 3: Define Feedforward Neural Network
class FeedforwardNN(nn.Module):
    def __init__(self):
        super(FeedforwardNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)
        self.relu = nn.ReLU()
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
```

```python
# Step 4: Loss and optimizer
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# To store training history
train_losses = []
train_accuracies = []

# Step 5: Train the model
for epoch in range(5):
    model.train()
    total_loss = 0
    correct, total = 0, 0

    for images, labels in train_loader:
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        _, predicted = torch.max(output.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    avg_loss = total_loss / len(train_loader)
    accuracy = 100 * correct / total
    train_losses.append(avg_loss)
    train_accuracies.append(accuracy)

    print(f"Epoch {epoch+1}, Loss: {avg_loss:.4f}, Accuracy: {accuracy:.2f}%")
```

```python
# Step 6: Evaluate model on test data
model.eval()
correct, total = 0, 0
with torch.no_grad():
    for images, labels in test_loader:
        output = model(images)
        _, predicted = torch.max(output.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Final Test Accuracy: {100 * correct / total:.2f}%")

# Step 7: Visualization
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(train_losses, marker='o')
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.subplot(1,2,2)
plt.plot(train_accuracies, marker='o')
plt.title("Training Accuracy per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")

plt.show()
```

```
Epoch 1, Loss: 0.3909, Accuracy: 88.39%
Epoch 2, Loss: 0.1871, Accuracy: 94.38%
Epoch 3, Loss: 0.1394, Accuracy: 95.76%
Epoch 4, Loss: 0.1122, Accuracy: 96.63%
Epoch 5, Loss: 0.0970, Accuracy: 96.95%
Final Test Accuracy: 96.50%
```