EX 1 THREE-LAYER NEURAL NETWORK FROM SCRATCH

DATE:

Problem Statement:

Design and implement a three-layer neural network from scratch using Python. Train the network using the backpropagation algorithm with appropriate activation and loss functions. Apply the model to recognize handwritten digits using the MNIST dataset.

Objectives:

1. Understand the structure and working of a basic feedforward neural network.
2. Implement forward propagation and backpropagation from scratch.
3. Use the sigmoid activation function and cross-entropy loss.
4. Train the model on the MNIST dataset and evaluate the loss over epochs.
5. Visualize training performance and perform predictions on sample images.

Scope:

This experiment demonstrates the core principles behind neural networks and training them using backpropagation. It is foundational for understanding deep learning and how more complex models (like CNNs or Transformers) build upon this architecture.

Tools and Libraries Used:

1. Python 3.x
2. NumPy
3. Matplotlib
4. TensorFlow (for dataset loading)
5. scikit-learn (OneHotEncoder)

Implementation Steps:

Step 1: Import Necessary Libraries

```
import numpy as np import
matplotlib.pyplot as plt from
tensorflow.keras.datasets import
mnist
from sklearn.preprocessing import OneHotEncoder
```

Step 2: Load and Preprocess the Dataset

```
(X_train, y_train), (_, _) =
mnist.load_data() X_train =
X_train.reshape(-1, 784) / 255.0 encoder =
OneHotEncoder(sparse_output=False)
y_train =
encoder.fit_transform(y_train.reshape(-1,
1))
```

Step 3: Initialize the Network

```
input_size, hidden_size, output_size = 784,
64, 10 W1 = np.random.randn(input_size,
hidden_size) * 0.01 b1 = np.zeros((1,
hidden_size))
W2 = np.random.randn(hidden_size, output_size) * 0.01
b2 = np.zeros((1, output_size))
```

Step 4: Define Activation and Loss Functions

```
sigmoid = lambda x: 1 / (1 + np.exp(-x))
sigmoid_deriv = lambda x: x * (1 - x) loss_fn =
lambda y, y_hat: -np.mean(y * np.log(y_hat + 1e-
8))
```

Step 5: Train the Model

```
epochs, lr = 10,
0.1 losses = [] for
epoch in
range(epochs):
   total_loss = 0     for i
in
range(X_train.shape[0])
:
x = X_train[i:i+1]
y = y_train[i:i+1]

      z1 = x @
W1 + b1
a1 =
sigmoid(z1)
z2 = a1 @ W2
+ b2        a2 =
sigmoid(z2)
loss =
loss_fn(y, a2)
      total_loss += loss

      dz2 =
a2 - y
dW2 = a1.T
@ dz2
```

```
        db2 = dz2

        dz1 = (dz2 @ W2.T) * sigmoid_deriv(a1)
        dW1 = x.T @ dz1
        db1 = dz1

        W2 -=
    lr * dW2
    b2 -= lr *
    db2
    W1 -= lr *
    dW1
        b1 -= lr * db1
      losses.append(total_loss / X_train.shape[0])
      print(f"Epoch {epoch+1}, Loss: {losses[-1]:.4f}")
```

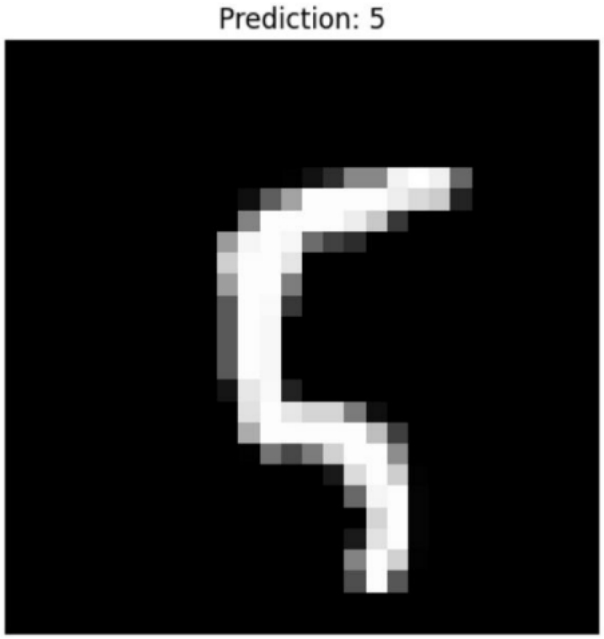
Step 6: Visualize Training Loss

```
plt.plot(losses)
plt.title("Training
Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```

Step 7: Predict a Sample Digit

```
def predict(img):
   img = img.reshape(1,
784) / 255.0     a1 =
sigmoid(img @ W1 + b1)
a2 = sigmoid(a1 @ W2 +
b2)
   return np.argmax(a2)

idx = 100
plt.imshow(X_train[idx].reshape(28, 28),
cmap='gray') plt.title(f"Prediction:
{predict(X_train[idx])}") plt.axis('off')
plt.show()
```



Conclusion:

This experiment successfully demonstrates the manual implementation of a basic threelayer neural network. Through backpropagation and weight updates using gradient descent, the model learns to classify handwritten digits. It emphasizes key deep learning concepts such as feedforward computation, loss minimization, and numerical stability in gradient calculations.