Machine Learning
Assignment 10
Aakanksha Darekar
202200733
09
A1

Implement a single-layer perceptron using the MNIST dataset.

Code:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

import seaborn as sns

from sklearn.preprocessing import LabelBinarizer


# Load MNIST data (using a more reliable method)

def load_mnist():

    from sklearn.datasets import fetch_openml

    mnist = fetch_openml('mnist_784', version=1, as_frame=False)

    X, y = mnist.data, mnist.target.astype(np.uint8)

    return X[:60000], y[:60000], X[60000:], y[60000:]  # Standard train-test split


class SingleLayerPerceptron:

    def __init__(self, input_size, output_size, learning_rate=0.01):

        self.weights = np.random.randn(input_size, output_size) * 0.01

        self.bias = np.zeros(output_size)

        self.learning_rate = learning_rate

        self.loss_history = []

        self.accuracy_history = []
```

```
def softmax(self, z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def forward(self, x):
    return self.softmax(np.dot(x, self.weights) + self.bias)

def compute_loss(self, y_true, y_pred):
    m = y_true.shape[0]
    return -np.sum(y_true * np.log(y_pred + 1e-10)) / m  # Cross-entropy

def train(self, x, y, epochs=10, batch_size=32):
    n = x.shape[0]
    lb = LabelBinarizer()
    y_onehot = lb.fit_transform(y)

    plt.figure(figsize=(12, 5))

    for epoch in range(epochs):
        # Shuffle and batch
        indices = np.random.permutation(n)
        for i in range(0, n, batch_size):
            batch = indices[i:i+batch_size]
            x_batch, y_batch = x[batch], y_onehot[batch]

            # Forward pass
            preds = self.forward(x_batch)

            # Backpropagation
```

Threshold

Calculate loss and predict

normal training

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train.reshape(-1, 784) / 255.0,
x_test.reshape(-1, 784) / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test)

# Build model
model = Sequential([Dense(10, activation='softmax',
input_shape=(784,))])
model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train model
model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_data=(x_test, y_test))

# Evaluate model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
```

```
                error = preds - y_batch
                grad_w = x_batch.T @ error / batch_size
                grad_b = np.mean(error, axis=0)

                # Update weights
                self.weights -= self.learning_rate * grad_w
                self.bias -= self.learning_rate * grad_b
```

If not error

```
            # Track progress
            train_pred = np.argmax(self.forward(x), axis=1)
            acc = np.mean(train_pred == y)
            loss = self.compute_loss(y_onehot, self.forward(x))

            self.loss_history.append(loss)
            self.accuracy_history.append(acc)

            # Visualize first 5 epochs
            if epoch < 5:
                self.visualize_weights(epoch)

            print(f"Epoch {epoch+1}/{epochs} - Loss: {loss:.4f}, Accuracy: {acc:.4f}")

        # Plot training curves
        self.plot_training_curves()

    def visualize_weights(self, epoch):
        plt.figure(figsize=(10, 5))
        for i in range(10):
            plt.subplot(2, 5, i+1)
```

```
            plt.imshow(self.weights[:, i].reshape(28, 28), cmap='viridis')
            plt.title(f"Digit {i}")
            plt.axis('off')
        plt.suptitle(f"Learned Weight Visualizations (Epoch {epoch+1})")
        plt.tight_layout()
        plt.show()

    def plot_training_curves(self):
        plt.figure(figsize=(12, 5))
        plt.subplot(1, 2, 1)
        plt.plot(self.loss_history)
        plt.title("Training Loss")
        plt.xlabel("Epoch")
        plt.ylabel("Cross-Entropy Loss")

        plt.subplot(1, 2, 2)
        plt.plot(self.accuracy_history)
        plt.title("Training Accuracy")
        plt.xlabel("Epoch")
        plt.ylabel("Accuracy")
        plt.tight_layout()
        plt.show()

    def evaluate(self, x_test, y_test):
        y_pred = np.argmax(self.forward(x_test), axis=1)
        acc = np.mean(y_pred == y_test)
        print(f"\nTest Accuracy: {acc:.4f}")

        # Confusion matrix
```

```
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(10, 8))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
        plt.title("Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("True")
        plt.show()

        # Show some example predictions
        self.show_examples(x_test, y_test, y_pred)

    def show_examples(self, x, y_true, y_pred, num_examples=10):
        indices = np.random.choice(len(x), num_examples)
        plt.figure(figsize=(15, 3))
        for i, idx in enumerate(indices):
            plt.subplot(1, num_examples, i+1)
            plt.imshow(x[idx].reshape(28, 28), cmap='gray')
            color = 'green' if y_pred[idx] == y_true[idx] else 'red'
            plt.title(f"P:{y_pred[idx]}\nT:{y_true[idx]}", color=color)
            plt.axis('off')
        plt.suptitle("Example Predictions (Green=Correct, Red=Wrong)")
        plt.tight_layout()
        plt.show()

# Main execution
if __name__ == "__main__":
    # Load data
    X_train, y_train, X_test, y_test = load_mnist()
```

```
# Normalize pixel values

X_train = X_train / 255.0

X_test = X_test / 255.0


# Initialize and train perceptron

perceptron = SingleLayerPerceptron(input_size=784, output_size=10, learning_rate=0.1)

print("Starting training...")

perceptron.train(X_train, y_train, epochs=20, batch_size=128)


# Evaluate

perceptron.evaluate(X_test, y_test)
```

Output:

```
Epoch 3/20 - Loss: 0.3304, Accuracy: 0.9085
Epoch 4/20 - Loss: 0.3178, Accuracy: 0.9120
Epoch 5/20 - Loss: 0.3086, Accuracy: 0.9144
Epoch 6/20 - Loss: 0.3025, Accuracy: 0.9159
Epoch 7/20 - Loss: 0.2970, Accuracy: 0.9179
Epoch 8/20 - Loss: 0.2940, Accuracy: 0.9184
Epoch 9/20 - Loss: 0.2906, Accuracy: 0.9193
Epoch 10/20 - Loss: 0.2862, Accuracy: 0.9200
Epoch 11/20 - Loss: 0.2840, Accuracy: 0.9209
Epoch 12/20 - Loss: 0.2816, Accuracy: 0.9217
Epoch 13/20 - Loss: 0.2796, Accuracy: 0.9230
Epoch 14/20 - Loss: 0.2781, Accuracy: 0.9221
Epoch 15/20 - Loss: 0.2763, Accuracy: 0.9233
Epoch 16/20 - Loss: 0.2747, Accuracy: 0.9235
Epoch 17/20 - Loss: 0.2748, Accuracy: 0.9237
Epoch 18/20 - Loss: 0.2723, Accuracy: 0.9250
Epoch 19/20 - Loss: 0.2719, Accuracy: 0.9252
Epoch 20/20 - Loss: 0.2704, Accuracy: 0.9257

Test Accuracy: 0.9241
PS C:\Users\sanke\OneDrive\Documents\Project\Python\Assignments>
```

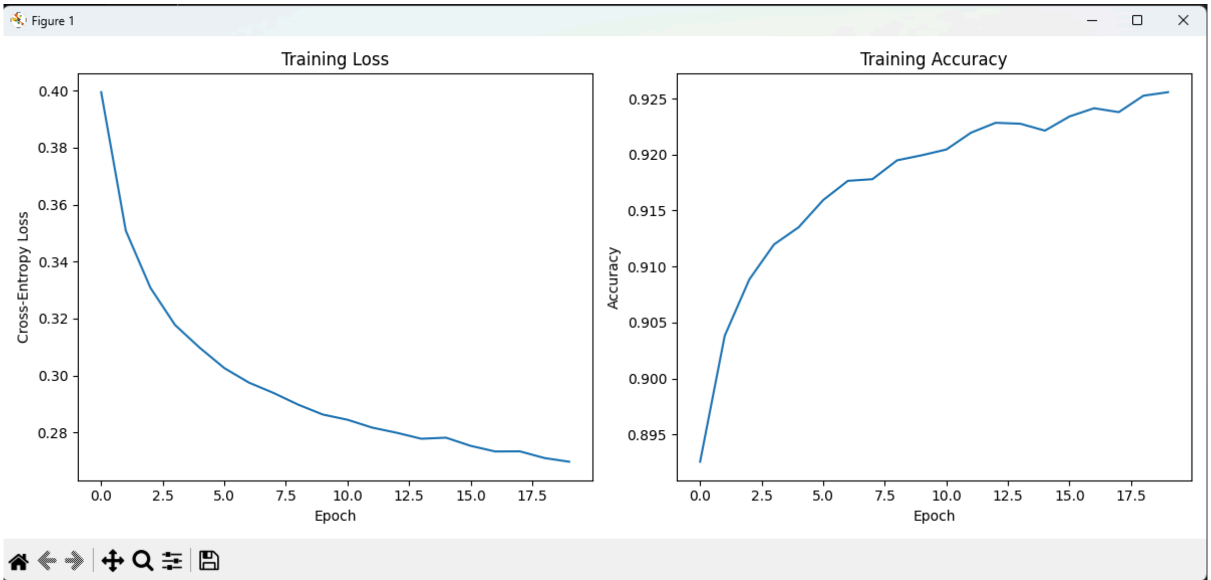


Figure 1
Training Loss
Cross-Entropy Loss
Epoch
Training Accuracy
Accuracy
Epoch

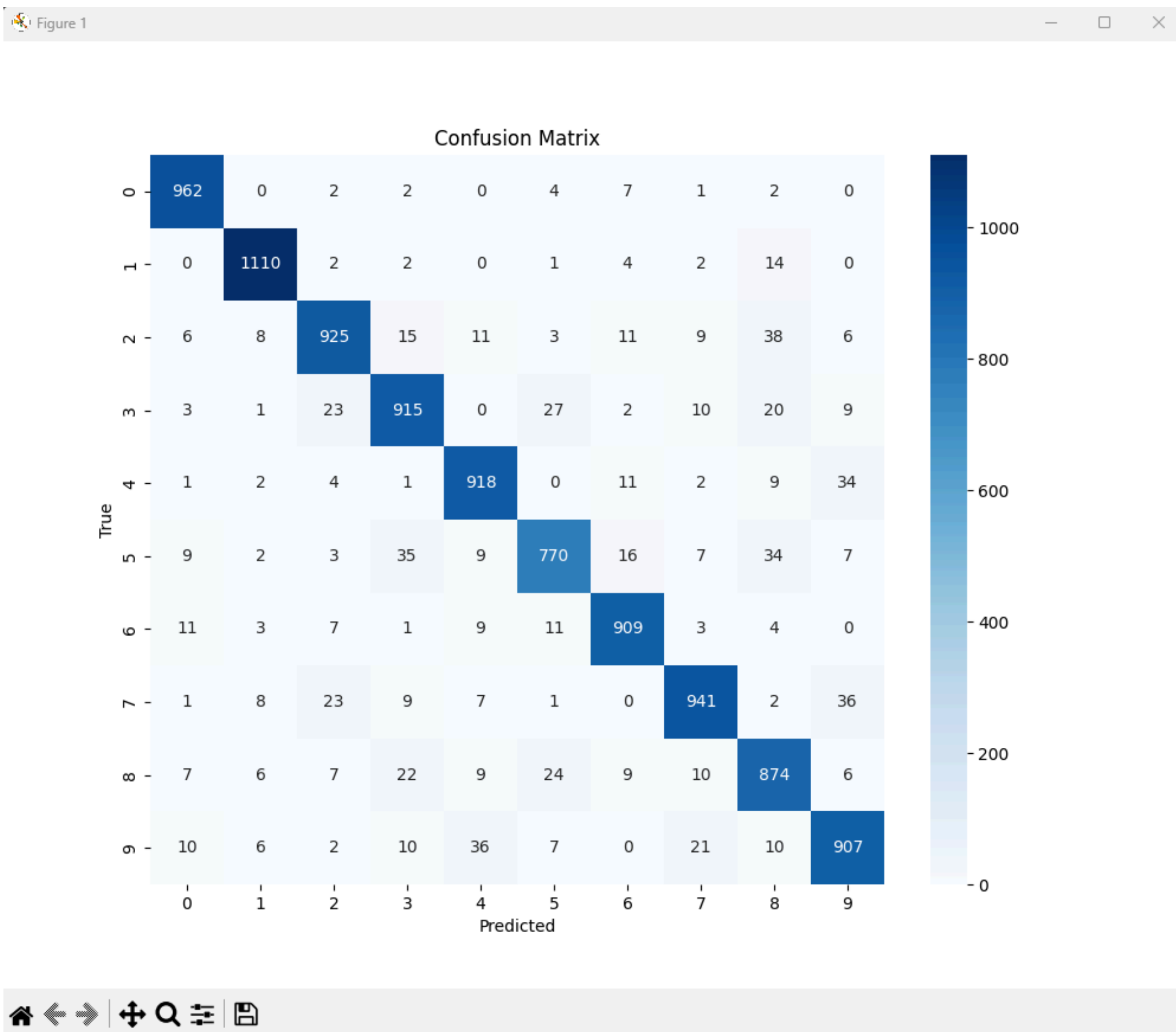
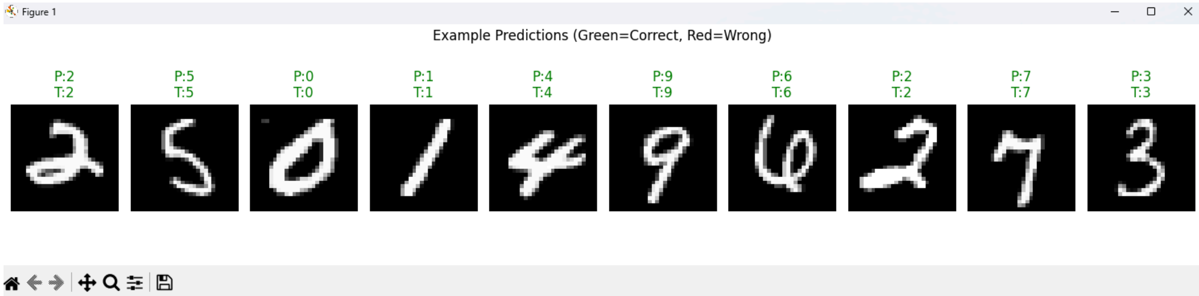
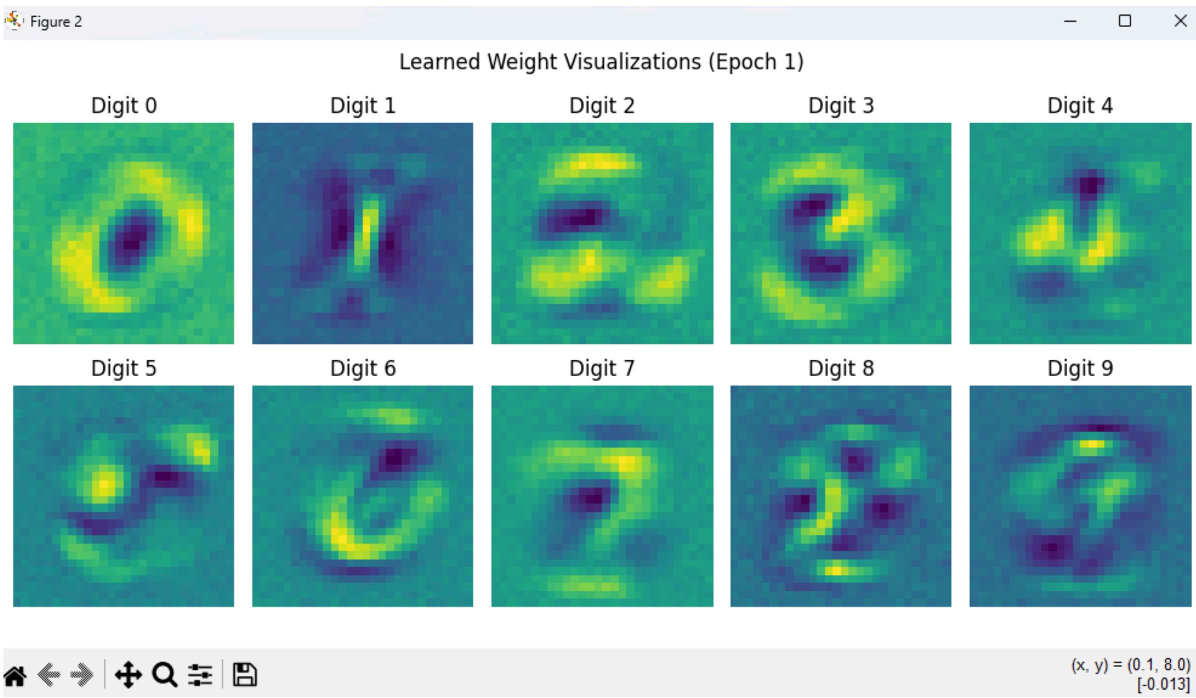


Figure 1
Confusion Matrix
True
Predicted
962 0 2 2 0 4 7 1 2 0
0 1110 2 2 0 1 4 2 14 0
6 8 925 15 11 3 11 9 38 6
3 1 23 915 0 27 2 10 20 9
1 2 4 1 918 0 11 2 9 34
9 2 3 35 9 770 16 7 34 7
11 3 7 1 9 11 909 3 4 0
1 8 23 9 7 1 0 941 2 36
7 6 7 22 9 24 9 10 874 6
10 6 2 10 36 7 0 21 10 907



Epoch 1:



Epoch 2:

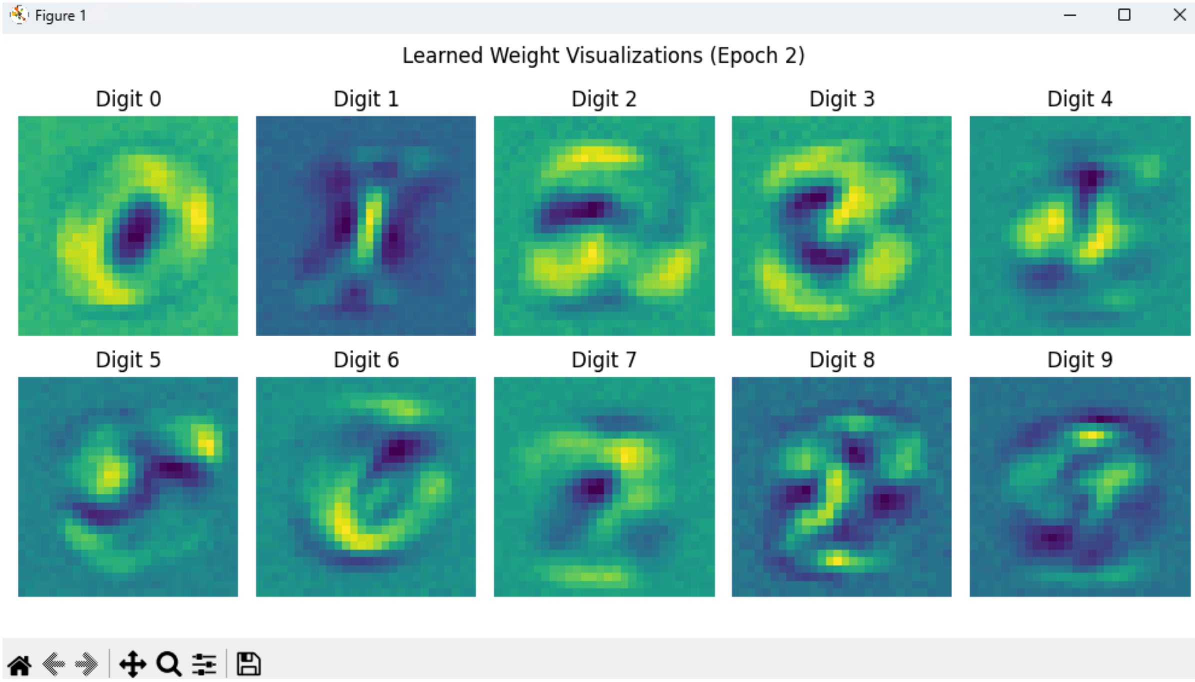


Figure 1
Learned Weight Visualizations (Epoch 2)
Digit 0
Digit 1
Digit 2
Digit 3
Digit 4
Digit 5
Digit 6
Digit 7
Digit 8
Digit 9

Epoch 3:

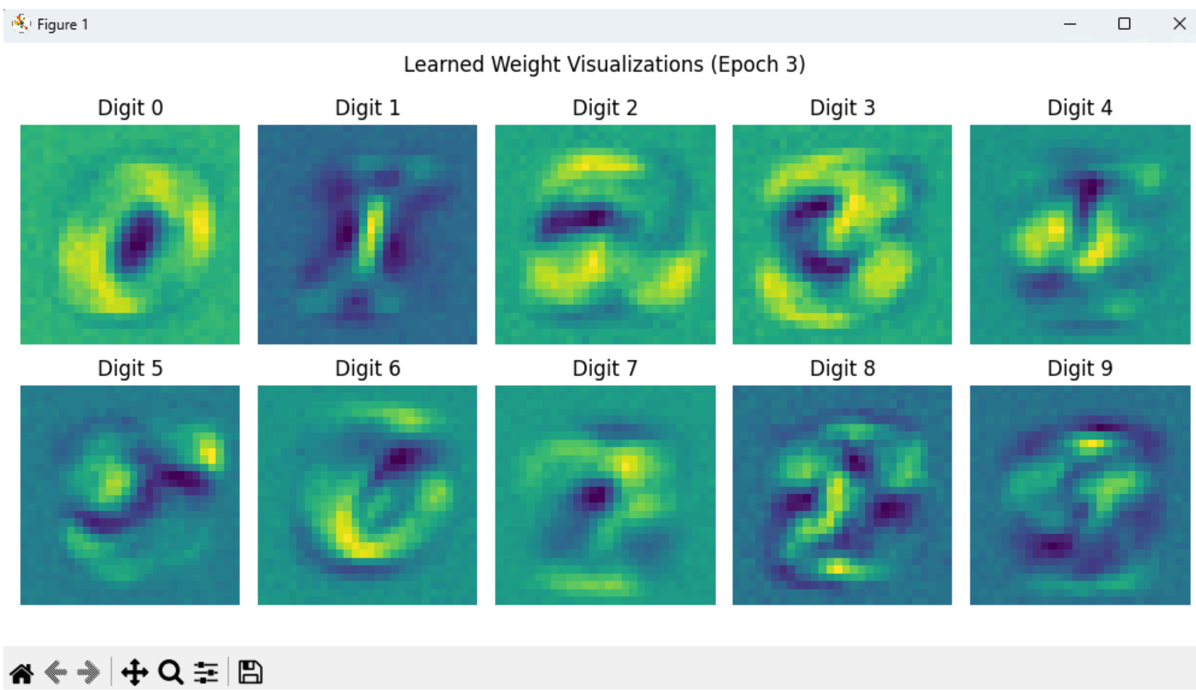


Figure 1
Learned Weight Visualizations (Epoch 3)
Digit 0
Digit 1
Digit 2
Digit 3
Digit 4
Digit 5
Digit 6
Digit 7
Digit 8
Digit 9