Name: Muhammad Moiz Ahmad

Roll.no: BSCE-22029

Task 1: Neural Network for MNIST Classification

1. Introduction

This task involves implementing a fully connected neural network from scratch to classify MNIST handwritten digits. The model includes two hidden layers with different activation functions and utilizes mini-batch gradient descent for optimization.

2. Dataset Preprocessing

- **Dataset Download**: The MNIST dataset was loaded and prepared.
- **Normalization**: Pixel values were scaled to [0,1].
- **Flattening**: 28×28 images were converted into 784-dimensional vectors.

```
def load_idx_images(filename):
    with open(filename, 'rb') as f:
        magic, num, rows, cols = struct.unpack(">IIII", f.read(16))
        images = np.frombuffer(f.read(), dtype=np.uint8).reshape(num, rows * cols)
    return images / 255.0  # Normalize pixel values
```

- **One-hot Encoding**: Labels were converted into one-hot vectors.

```
# Convert labels to one-hot encoding
Tabnine | Edit | Test | Explain | Document
def one_hot_encode(y, num_classes=10):
    return np.eye(num_classes)[y]

y_train = one_hot_encode(y_train)
y_test = one_hot_encode(y_test)
```

- **Data Splitting**: 80% for training, 10% for validation, 10% for testing.

```
# Split into Train (80%), Validation (10%), Test (10%)
split1 = int(0.8 * len(x_train))
split2 = int(0.9 * len(x_train))
x_train, x_val, x_test = x_train[:split1], x_train[split1:split2], x_test
y_train, y_val, y_test = y_train[:split1], y_train[split1:split2], y_test
print(f"Train Samples: {x_train.shape}, Validation Samples: {x_val.shape}, Test Samples: {x_test.shape}")
```

3. Implemented Functions

Activation Functions

- **Sigmoid**: σ(x)=11+e−x\sigma(x) = \frac{1}{1 + e^{-x}}

```
Tabnine | Edit | Test | Explain | Document
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

Tabnine | Edit | Test | Explain | Document
def sigmoid_derivative(x):
    return x * (1 - x)
```

- **ReLU**: f(x)=max(0,x)f(x) = \max(0, x)

```
Tabnine | Edit | Test | Explain | Document
def relu(x):
    return np.maximum(0, x)

Tabnine | Edit | Test | Explain | Document
def relu_derivative(x):
    return (x > 0).astype(float)
```

- **Softmax**: σ(z)i=ezi∑jezj\sigma(z)_i = \frac{e^{z_i}}{\sum_{j} e^{z_j}}

```
Tabnine | Edit | Test | Explain | Document
def softmax(x):
    exp_x = np.exp(x - np.max(x))  # Avoid overflow
    return exp_x / exp_x.sum(axis=1, keepdims=True)
```

Loss Function

- **Cross-Entropy Loss**: L=−∑yilog(yi^)L = -\sum y_i \log(\hat{y_i})

```
# Cross-entropy loss function
Tabnine | Edit | Test | Explain | Document
def cross_entropy_loss(y_true, y_pred):
    return -np.mean(np.sum(y_true * np.log(y_pred + 1e-9), axis=1))
```

Forward and Backward Propagation

- Forward propagation calculates activations through layers.

- Backpropagation computes gradients for weight updates.

Gradient Descent

- Mini-batch gradient descent was implemented to update weights and biases.

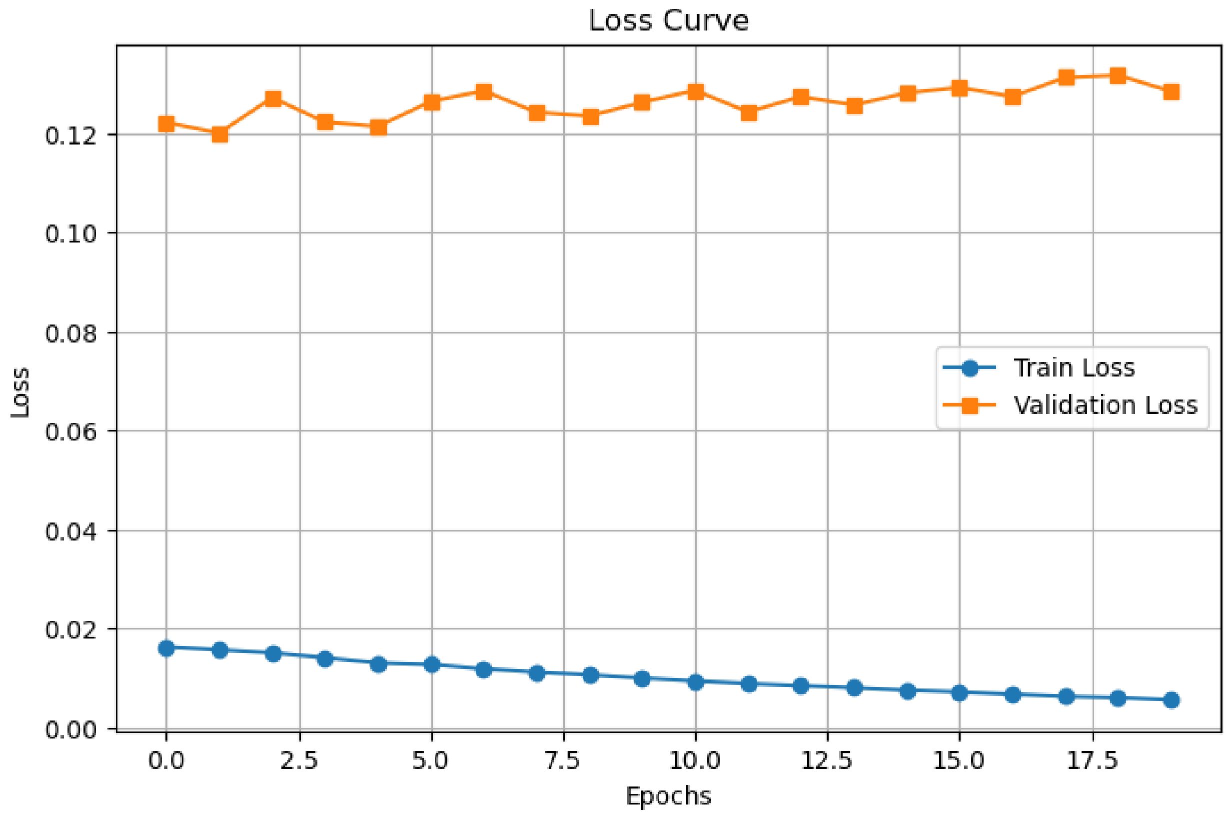
```
# Initialize weights and biases
np.random.seed(42)
input_size, hidden1_size, hidden2_size, output_size = 784, 128, 64, 10

w1 = np.random.randn(input_size, hidden1_size) * 0.01
b1 = np.zeros((1, hidden1_size))
w2 = np.random.randn(hidden1_size, hidden2_size) * 0.01
b2 = np.zeros((1, hidden2_size))
w3 = np.random.randn(hidden2_size, output_size) * 0.01
b3 = np.zeros((1, output_size))
```

4. Model Training

- The model was trained for 20 epochs.

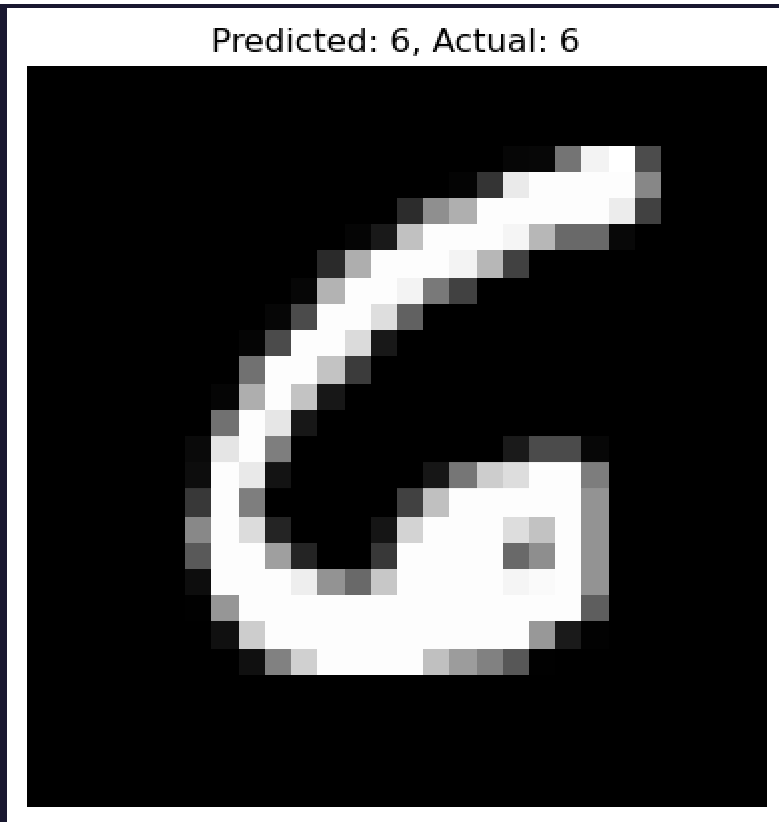
```
# Hyperparameters
learning_rate = 0.1
epochs = 20
batch_size = 64
```

- Loss curves were plotted for training, validation, and test sets.

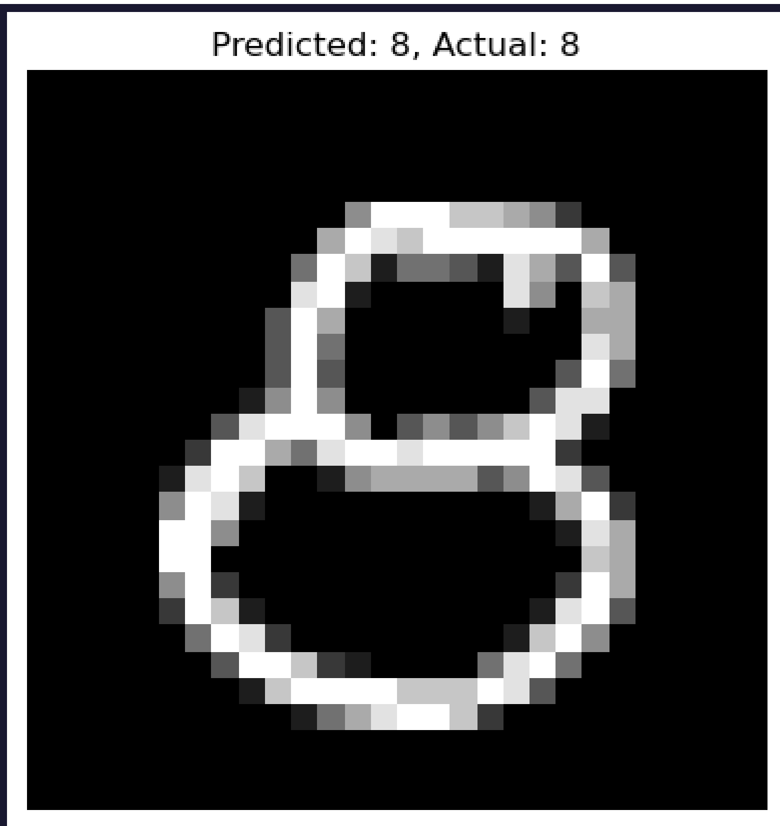


5. Testing and Evaluation

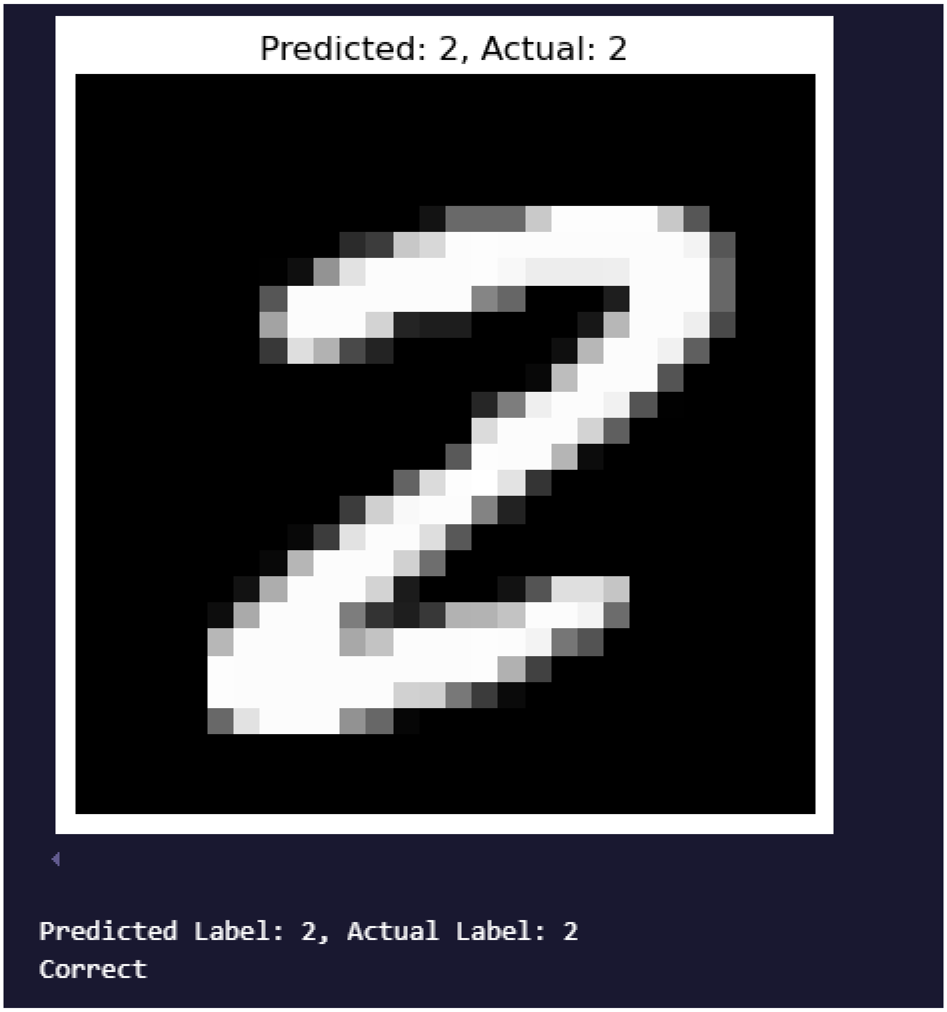
- A random test image was classified.




```
Predicted Label: 6, Actual Label: 6
Correct
```




```
Predicted Label: 8, Actual Label: 8
Correct
```



- Results included predicted vs actual labels and correctness.

6. Observations

- Sigmoid activation led to vanishing gradients in deep layers.
- ReLU improved gradient propagation.
- Loss curves showed convergence after multiple epochs.

Task 2: Support Vector Machine (SVM) for Iris Classification

1. Introduction

This task involved implementing an SVM classifier from scratch using gradient descent to classify Iris flowers (Setosa and Versicolor).

2. Dataset Preprocessing

- Used only Setosa (0) and Versicolor (1) classes.
- Selected **Petal Length** and **Petal Width** as features.
- Converted labels to {-1, 1}.
- Split data into training and testing sets.

```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. Implemented Functions

Hinge Loss Function

- L=∑max(0,1−y(w·x+b))+1C||w||2L = \sum \max(0, 1 - y(w \cdot x + b)) + \frac{1}{C} ||w||^2

Gradient Descent Optimization

- Updated weights and biases using gradients.

4. Experiments with Regularization Parameter (C)

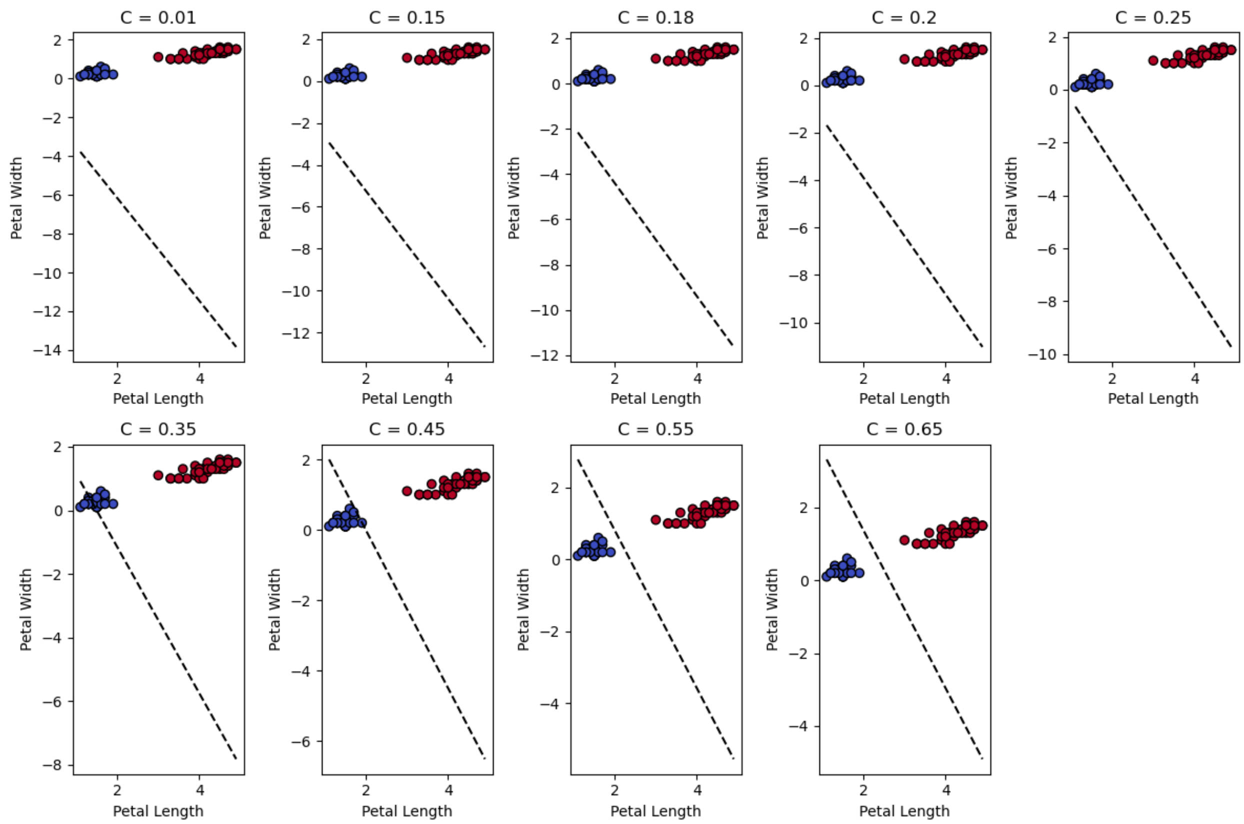
- Multiple models were trained with different C values.

```
# Train and evaluate SVM for different C values
C_values = [0.01, 0.15,0.18,0.20,0.25,0.35,0.45,0.55,0.65]
plt.figure(figsize=(12, 8))
for i, C in enumerate(C_values, 1):
    svm = SVM(C=C, lr=0.01, epochs=1000)
    svm.fit(X_train, y_train)
```

- Decision boundaries were plotted.

5. Observations

- Smaller C resulted in larger margins but more misclassifications.
- Larger C resulted in stricter margins and overfitting.
- The best C value is 0.465 balance between accuracy and generalization.



6. Evaluation

- The final model was tested on the test set.
- Accuracy was calculated and analyzed.

Conclusion

- Implementing a neural network from scratch reinforced understanding of forward and backward propagation.
- Mini-batch gradient descent was effective in optimizing the model.
- SVM experiments highlighted the impact of regularization on classification.
- The project demonstrated the importance of hyperparameter tuning for optimal results.

HANDWRITTEN ARE GIVEN BELOW