Ben Ganon - 318731007

Omri Ben Hemo - 313255242

Report: Implementation and Evaluation of a Multi-Layer Artificial Neural Network

1. Introduction

This report presents the implementation and evaluation of a multi-layer artificial neural network (ANN) for classifying handwritten digits from the MNIST dataset. The work extends the code from Chapter 11 of "Machine Learning with PyTorch and Scikit-Learn" by Raschka et al. (2022), modifying the original single hidden layer model to incorporate two hidden layers. The performance of this modified model is compared with both the original single-layer model and a fully connected ANN implemented in PyTorch.

Initial evaluation:

The evaluation metrics for the one-layer default mlp:

```
Epoch: 001/030 | Train MSE: 0.05 | Train Acc: 76.91% | Valid Acc: 77.52%
Epoch: 002/030 | Train MSE: 0.03 | Train Acc: 85.79% | Valid Acc: 86.48%
Epoch: 003/030 | Train MSE: 0.02 | Train Acc: 88.02% | Valid Acc: 88.90%
Epoch: 004/030 | Train MSE: 0.02 | Train Acc: 89.35% | Valid Acc: 90.22%
Epoch: 005/030 | Train MSE: 0.02 | Train Acc: 90.03% | Valid Acc: 90.72%
Epoch: 006/030 | Train MSE: 0.02 | Train Acc: 90.58% | Valid Acc: 91.20%
Epoch: 007/030 | Train MSE: 0.02 | Train Acc: 91.07% | Valid Acc: 91.54%
Epoch: 008/030 | Train MSE: 0.02 | Train Acc: 91.51% | Valid Acc: 91.96%
Epoch: 009/030 | Train MSE: 0.01 | Train Acc: 91.68% | Valid Acc: 92.18%
Epoch: 010/030 | Train MSE: 0.01 | Train Acc: 92.01% | Valid Acc: 92.54%
Epoch: 011/030 | Train MSE: 0.01 | Train Acc: 92.20% | Valid Acc: 92.76%
Epoch: 012/030 | Train MSE: 0.01 | Train Acc: 92.41% | Valid Acc: 92.92%
Epoch: 013/030 | Train MSE: 0.01 | Train Acc: 92.59% | Valid Acc: 93.12%
Epoch: 014/030 | Train MSE: 0.01 | Train Acc: 92.84% | Valid Acc: 93.28%
Epoch: 015/030 | Train MSE: 0.01 | Train Acc: 92.94% | Valid Acc: 93.52%
Epoch: 016/030 | Train MSE: 0.01 | Train Acc: 92.95% | Valid Acc: 93.12%
Epoch: 017/030 | Train MSE: 0.01 | Train Acc: 93.21% | Valid Acc: 93.70%
Epoch: 018/030 | Train MSE: 0.01 | Train Acc: 93.31% | Valid Acc: 93.74%
Epoch: 019/030 | Train MSE: 0.01 | Train Acc: 93.51% | Valid Acc: 94.00%
Epoch: 020/030 | Train MSE: 0.01 | Train Acc: 93.52% | Valid Acc: 93.98%
Epoch: 021/030 | Train MSE: 0.01 | Train Acc: 93.59% | Valid Acc: 93.88%
Epoch: 022/030 | Train MSE: 0.01 | Train Acc: 93.79% | Valid Acc: 93.94%
Epoch: 023/030 | Train MSE: 0.01 | Train Acc: 93.85% | Valid Acc: 94.06%
Epoch: 024/030 | Train MSE: 0.01 | Train Acc: 93.94% | Valid Acc: 94.26%
Epoch: 025/030 | Train MSE: 0.01 | Train Acc: 94.06% | Valid Acc: 94.38%
...
Epoch: 027/030 | Train MSE: 0.01 | Train Acc: 94.22% | Valid Acc: 94.48%
Epoch: 028/030 | Train MSE: 0.01 | Train Acc: 94.32% | Valid Acc: 94.56%
Epoch: 029/030 | Train MSE: 0.01 | Train Acc: 94.43% | Valid Acc: 94.58%
Epoch: 030/030 | Train MSE: 0.01 | Train Acc: 94.44% | Valid Acc: 94.52%
```

2. Implementation

The implementation follows these key steps:

A local copy of the ch11.ipynb file was created and modified to include two hidden layers.

```
class TwoLayerNN:
    def __init__(self, input_layer_size, hidden_layer1_size, hidden_layer2_size,
output_layer_size, learning_rate=0.01):
        # Initialize weights and biases
        self.weights_input_to_hidden1 = np.random.randn(input_layer_size,
hidden_layer1_size) * 0.01
        self.bias_hidden_layer1 = np.zeros((1, hidden_layer1_size))

        self.weights_hidden1_to_hidden2 = np.random.randn(hidden_layer1_size,
hidden_layer2_size) * 0.01
        self.bias_hidden_layer2 = np.zeros((1, hidden_layer2_size))

        self.weights_hidden2_to_output = np.random.randn(hidden_layer2_size,
output_layer_size) * 0.01
        self.bias_output_layer = np.zeros((1, output_layer_size))

        self.learning_rate = learning_rate

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def one_hot_encode(self, labels, num_classes=10):
        return np.eye(num_classes)[labels]

    def forward(self, input_data):
        hidden_layer1_input = np.dot(input_data, self.weights_input_to_hidden1) +
self.bias_hidden_layer1
        hidden_layer1_activation = self.sigmoid(hidden_layer1_input)

        hidden_layer2_input = np.dot(hidden_layer1_activation,
self.weights_hidden1_to_hidden2) + self.bias_hidden_layer2
        hidden_layer2_activation = self.sigmoid(hidden_layer2_input)

        output_layer_input = np.dot(hidden_layer2_activation,
self.weights_hidden2_to_output) + self.bias_output_layer
        output_layer_activation = self.sigmoid(output_layer_input)

        self.hidden_layer1_activation = hidden_layer1_activation
        self.hidden_layer2_activation = hidden_layer2_activation
```

```
        return output_layer_activation

    def backward(self, input_data, target_output, output_layer_activation):
        # Compute error
        output_error = output_layer_activation - target_output
        hidden_layer1_activation = self.hidden_layer1_activation
        hidden_layer2_activation = self.hidden_layer2_activation
        # Backpropagation
        output_layer_gradient = output_error *
self.sigmoid_derivative(output_layer_activation)
        gradient_weights_hidden2_to_output = np.dot(hidden_layer2_activation.T,
output_layer_gradient)
        gradient_bias_output_layer = np.sum(output_layer_gradient, axis=0,
keepdims=True)

        hidden_layer2_gradient = np.dot(output_layer_gradient,
self.weights_hidden2_to_output.T) *
self.sigmoid_derivative(hidden_layer2_activation)
        gradient_weights_hidden1_to_hidden2 = np.dot(hidden_layer1_activation.T,
hidden_layer2_gradient)
        gradient_bias_hidden_layer2 = np.sum(hidden_layer2_gradient, axis=0,
keepdims=True)

        hidden_layer1_gradient = np.dot(hidden_layer2_gradient,
self.weights_hidden1_to_hidden2.T) *
self.sigmoid_derivative(hidden_layer1_activation)
        gradient_weights_input_to_hidden1 = np.dot(input_data.T,
hidden_layer1_gradient)
        gradient_bias_hidden_layer1 = np.sum(hidden_layer1_gradient, axis=0,
keepdims=True)

        self.weights_hidden2_to_output -= self.learning_rate *
gradient_weights_hidden2_to_output
        self.bias_output_layer -= self.learning_rate * gradient_bias_output_layer

        self.weights_hidden1_to_hidden2 -= self.learning_rate *
gradient_weights_hidden1_to_hidden2
        self.bias_hidden_layer2 -= self.learning_rate *
gradient_bias_hidden_layer2

        self.weights_input_to_hidden1 -= self.learning_rate *
gradient_weights_input_to_hidden1
        self.bias_hidden_layer1 -= self.learning_rate *
gradient_bias_hidden_layer1
```

```
        return gradient_weights_hidden2_to_output, gradient_bias_output_layer,
gradient_weights_hidden1_to_hidden2, gradient_bias_hidden_layer2,
gradient_weights_input_to_hidden1, gradient_bias_hidden_layer1
```

```
model_two_layers = TwoLayerNN(
    input_layer_size=28*28,
    hidden_layer1_size=100,
    hidden_layer2_size=100,
    output_layer_size=10,
)
```

Input Layer: 784 neurons (flattened 28x28 images)

Hidden Layer 1: 100 neurons, sigmoid activation

Hidden Layer 2: 100 neurons, sigmoid activation

Output Layer: 10 neurons, sigmoid activation

The model was trained using the MNIST dataset with a 70%-30% train-test split.

```
# 70% train, 30% test
X_train, X_test, y_train, y_test = train_test_split(
    X_all, y_all, test_size=0.30, random_state=123, shuffle=True)
```

```
def train_two_layer_nn(model, X_train, y_train, X_test, y_test,
                       num_epochs=num_epochs, learning_rate=0.1, minibatch_size=100):
    # One-hot encode the labels
    y_train = model.one_hot_encode(y_train)
    y_test = model.one_hot_encode(y_test)
    epoch_loss = []
    epoch_train_acc = []
    epoch_valid_acc = []

    for e in range(num_epochs):
        minibatch_gen = minibatch_generator(X_train, y_train, minibatch_size)
```

```
        for X_mini, y_mini in minibatch_gen:
            # Forward pass
            a_out = model.forward(X_mini)

            # Backward pass
            dW_out, dB_out, dW_h2, dB_h2, dW_h1, dB_h1 = \
                model.backward(X_mini, y_mini, a_out)
```

Training was performed for 30 epochs using the manual batch SGD, with simple MSE loss.

3. Performance Evaluation

The model's performance was assessed based on Macro-AUC

Training accuracy and validation accuracy over epochs

```
[Epoch 01] Test Macro AUC: 0.573
Epoch: 001/030 | Train MSE: 0.09 | Train Acc: 9.97% | Valid Acc: 9.48%
[Epoch 02] Test Macro AUC: 0.604
Epoch: 002/030 | Train MSE: 0.09 | Train Acc: 10.51% | Valid Acc: 10.20%
[Epoch 03] Test Macro AUC: 0.629
Epoch: 003/030 | Train MSE: 0.09 | Train Acc: 11.16% | Valid Acc: 11.47%
[Epoch 04] Test Macro AUC: 0.658
Epoch: 004/030 | Train MSE: 0.09 | Train Acc: 10.04% | Valid Acc: 9.86%
[Epoch 05] Test Macro AUC: 0.698
Epoch: 005/030 | Train MSE: 0.09 | Train Acc: 9.97% | Valid Acc: 9.48%
[Epoch 06] Test Macro AUC: 0.711
Epoch: 006/030 | Train MSE: 0.09 | Train Acc: 19.11% | Valid Acc: 19.11%
[Epoch 07] Test Macro AUC: 0.729
Epoch: 007/030 | Train MSE: 0.08 | Train Acc: 29.71% | Valid Acc: 29.57%
[Epoch 08] Test Macro AUC: 0.817
Epoch: 008/030 | Train MSE: 0.07 | Train Acc: 39.52% | Valid Acc: 39.05%
[Epoch 09] Test Macro AUC: 0.888
Epoch: 009/030 | Train MSE: 0.06 | Train Acc: 56.51% | Valid Acc: 56.62%
[Epoch 10] Test Macro AUC: 0.936
Epoch: 010/030 | Train MSE: 0.04 | Train Acc: 75.51% | Valid Acc: 75.13%
[Epoch 11] Test Macro AUC: 0.961
Epoch: 011/030 | Train MSE: 0.03 | Train Acc: 83.01% | Valid Acc: 82.80%
[Epoch 12] Test Macro AUC: 0.971
Epoch: 012/030 | Train MSE: 0.02 | Train Acc: 86.76% | Valid Acc: 86.42%
[Epoch 13] Test Macro AUC: 0.976
...
[Epoch 29] Test Macro AUC: 0.994
Epoch: 029/030 | Train MSE: 0.01 | Train Acc: 95.63% | Valid Acc: 94.82%
[Epoch 30] Test Macro AUC: 0.994
Epoch: 030/030 | Train MSE: 0.01 | Train Acc: 95.82% | Valid Acc: 95.03
```

The validation accuracy reached 95.03% after 30 epochs, indicating a well-generalized model.

4. Comparison with Baseline Models

The modified two-layer model outperforms the single-layer model by approximately 1% in validation accuracy.

5. Conclusion

Final results:

```
One Layer Perceptron Micro-AUC: 0.996
Two Layer Perceptron Micro-AUC: 0.994
Two Layer Perceptron PyTorch Micro-AUC: 0.996
```

6. Submission Details

The modified code is available at https://github.com/Ben-Ganon/AutoML_Ex3.git

This report and the necessary files are packaged as per submission requirements.