MNIST Neural Network Implementation

Josue Rolando Naranjo Sieiro
Student of Computer Science, University of Havana, Cuba

November 23, 2024

Abstract

This report summarizes the implementation and results of neural networks with varying layers to classify handwritten digits from the MNIST dataset. The experiments include a single-layer neural network, a two-layer neural network, and a three-layer neural network.

1 Introduction

The MNIST dataset is a well-known dataset for handwritten digit classification. This project implements three neural network architectures to classify the digits: a single-layer neural network, a two-layer neural network, and a three-layer neural network. The goal is to compare the performance and training time of these architectures.

2 Single Layer Neural Network

The single-layer neural network consists of one hidden layer and an output layer. The implementation is provided in the `NN1Layer.py` script. The following functions are used:

2.1 Functions

- **one_hot_encode(y)**: Converts labels to one-hot encoded vectors.
- **init_weights(input_size, hidden_size, output_size)**: Initializes weights and biases.
- **sigmoid(z)**: Sigmoid activation function.
- **sigmoid_derivative(a)**: Derivative of the sigmoid function.
- **relu(z)**: ReLU activation function.
- **relu_derivative(z)**: Derivative of the ReLU function.

- **softmax(z)**: Softmax activation function.
- **forward_propagation(X, W1, b1, W2, b2)**: Performs forward propagation.
- **compute_cost(A2, Y)**: Computes the cost using categorical cross-entropy.
- **backward_propagation(X, Y, cache, W2)**: Performs backward propagation.
- **update_parameters(W1, b1, W2, b2, gradients, learning_rate)**: Updates weights and biases using gradient descent.
- **train(X, Y, input_size, hidden_size, output_size, epochs, learning_rate)**: Trains the neural network.
- **predict(X, parameters)**: Makes predictions using the trained model.

2.2 Results

The single-layer neural network achieved an accuracy of 92.28% on the test set.

3 Two Layer Neural Network

The two-layer neural network consists of two hidden layers and an output layer. The implementation is provided in the `NN2Layer.py` script. The following functions are used:

3.1 Functions

- **one_hot_encode(y)**: Converts labels to one-hot encoded vectors.
- **init_weights(layer_sizes)**: Initializes weights and biases.
- **relu(z)**: ReLU activation function.
- **relu_derivative(z)**: Derivative of the ReLU function.
- **softmax(z)**: Softmax activation function.
- **forward_propagation(X, parameters)**: Performs forward propagation.
- **compute_cost(AL, Y, parameters, lambd)**: Computes the cost with L2 regularization.
- **backward_propagation(Y, cache, parameters, lambd)**: Performs backward propagation.
- **update_parameters(parameters, gradients, optimizer_params)**: Updates weights and biases using the Adam optimizer.

- **initialize_optimizer_params(parameters)**: Initializes parameters for the Adam optimizer.
- **get_mini_batches(X, Y, batch_size)**: Generates mini-batches for training.
- **train(X, Y, layer_sizes, epochs, lambd, batch_size)**: Trains the neural network.
- **predict(X, parameters)**: Makes predictions using the trained model.

3.2 Results

The two-layer neural network achieved an accuracy of 97.71% on the test set.

4 Three Layer Neural Network

The three-layer neural network consists of three hidden layers and an output layer. The implementation is provided in the `NN3Layer.py` script. The following functions are used:

4.1 Functions

- **one_hot_encode(y)**: Converts labels to one-hot encoded vectors.
- **init_weights(layer_sizes)**: Initializes weights and biases.
- **relu(z)**: ReLU activation function.
- **relu_derivative(z)**: Derivative of the ReLU function.
- **softmax(z)**: Softmax activation function.
- **forward_propagation(X, parameters)**: Performs forward propagation.
- **compute_cost(AL, Y, parameters, lambd)**: Computes the cost with L2 regularization.
- **backward_propagation(Y, cache, parameters, lambd)**: Performs backward propagation.
- **update_parameters(parameters, gradients, optimizer_params)**: Updates weights and biases using the Adam optimizer.
- **initialize_optimizer_params(parameters)**: Initializes parameters for the Adam optimizer.
- **get_mini_batches(X, Y, batch_size)**: Generates mini-batches for training.

- **train(X, Y, layer_sizes, epochs, lambd, batch_size)**: Trains the neural network.
- **predict(X, parameters)**: Makes predictions using the trained model.

4.2 Results

The three-layer neural network achieved an accuracy of 97.70% on the test set, but with a significantly longer training time compared to the two-layer neural network.

5 Conclusion

The experiments demonstrate that increasing the number of layers in a neural network can improve accuracy, but it also increases the training time. The two-layer neural network provided the best balance between accuracy and training time, achieving an accuracy of 97.71%.

6 References

- MNIST dataset: `http://yann.lecun.com/exdb/mnist/`
- Neural Network implementation: `NN1Layer.py`, `NN2Layer.py`, `NN3Layer.py`

- Pre-trained model: `trained`$_{m}$*odel.pkl*