CS460G MACHINE LEARNING
PROJECT 3 – MULTILAYER PERCEPTRONS WRITEUP

Part 1: Binary Classification using Multilayer Perceptrons

Implementation Choices:

- Pixel values (feature values) are normalized within the range of 0 and 1 by:
  $x_i = \frac{x_i}{255}$ with x_i is the i-th pixel value and 255 is the maximum pixel value.
- The multilayer perceptron model for this part with a learning rate 0.01 has 1 hidden layer with 5 hidden nodes per layer, 1 output node whose value would classify the data in the following way:
  + If output node's value (activation at output layer) is greater than 0.5, the classification would be 1.
  + Otherwise, the classification would be 0.
- The activation function used in this model is the sigmoid function $g(x) = \frac{1}{1+e^{-x}}$ and its derivative is $g'(x) = g(x) * (1 - g(x))$
- At each layer of input layer and hidden layers, a bias node with the value of 1 is added. These biases are not added to the vectors that hold the nodes' values, but the weights for the bias nodes are managed separately from the matrices holding the weights of interior nodes (*weights_h*) and output nodes (*weights_o*) by 2 matrices, *biases_h* (weights of bias nodes associated with interior layers) and *biases_o* (weights of bias nodes associated with output layer).
- The weight values of all nodes (including interior, output, and bias nodes) are randomly initialized in the range of -1 and 1 using *numpy.random.uniform(-1, 1, {MATRIX_SIZE}).*
- The model uses forward pass process and backpropagation (stop before reaching the input layer) before updating its weights. The weights are updated using stochastic gradient descent method (updating weights after every example that was looped through).
- The model is trained with *'mnist_train_0_1.csv'* in one epoch as it converges.
- After training, the model would use the updated weights of interior, output, and bias nodes to generate predictions. The step to generate a prediction is exactly the same as forward pass process, except that the weights in this case would help generating activation values that corresponding to the classifications using the method mentioning above.

Accuracy:

I ran the code file ***neuralNetwork.py*** and calculate the accuracy rate of the model by taking the total number of correct predictions divided by the total number of examples generated from the test data *'mnist_test_0_1.csv'*:

Learning rate {ALPHA}	Hidden layers	Nodes per Hidden layer {NUM_HIDDEN_LAYER_NODES}	Output nodes {NUM_OUTPUT_NODES}	Accuracy rate	Side-by-side (Prediction, Key) Comparison File
0.01	1	5	1	0.9947990543735225 ≈ 99%	classified_mnist_test_0_1.csv

Bonus: Multiclass Classification using Multilayer Perceptrons

Implementation Choices:

- The implementation of this part is almost identical to the previous part, except for some changes of the representation for class label and
- Pixel values (feature values) are also normalized as the previous part, but with an addition of expanding class labels into matrices of the size [{NUM_OUTPUT_NODES} x 1] = [5 x 1] with all the entries equal 0 except for the n-th entry with n is the class label. So, for example, if a class label is 3, then the corresponding matrix would be: $\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}$. Then, the model continues doing calculations for activations using these new matrices instead of the original [1 x 1] for each example.
- The multilayer perceptron model for this part with a learning rate 0.1 has 1 hidden layer with 10 hidden nodes per layer, 5 output nodes (since the class label values are within the range of 0 and 4) whose value would classify the data in the following way:
  + The activation matrix for output layer would then also be [{NUM_OUTPUT_NODES} x 1] = [5 x 1]. The classification is then would be the index of the maximum entry of the output activation matrix, which should be within the range of 0 and 4.
- The activation function used in this model is also the sigmoid function $g(x) = \frac{1}{1+e^{-x}}$ and its derivative is $g'(x) = g(x) * (1 - g(x))$
- The biases are also added exactly like the previous part as well as the random initializations of weights and update weights after every example (after forward pass and backpropagation).
- The model is trained with *'mnist_train_0_4.csv'* in one epoch as it converges.

Accuracy:

I ran the code file ***bonusNeuralNetwork.py*** and calculate the accuracy rate of the model by taking the total number of correct predictions divided by the total number of examples generated from the test data *'mnist_test_0_4.csv'*:

Learning rate {ALPHA}	Hidden layers	Nodes per Hidden layer {NUM_HIDDEN_LAYER_NODES}	Output nodes {NUM_OUTPUT_NODES}	Accuracy rate	Side-by-side (Prediction, Key) Comparison File
0.1	1	10	5	$0.9696438995913602 \approx 96\%$	classified_mnist_test_0_4.csv