Programming Assignment 1 Summary Report

For this assignment, a two-layer perceptron was implemented to solve the parity problem. The perceptron used the backpropagation algorithm to minimize error and learn weights. The source program can be tested by running: “python main.py”

Training Data

I created the training data in “data_gen.py” by generating every combination of 4 binary values, then creating a “desired labels” array by counting the number of 1’s in the combination.

Perceptron Representation

The perceptron was implemented in “network.py” and represented as a class by 4 numpy matrices - w1, b1, w2, and b2. These weights and biases were used to perform the forward pass and were updated during training. To calculate the activation functions during the forward and backward passes, the sigmoid and sigmoid derivative were implemented as helper functions in “activation.py”. Similarly, the error and error derivative were implemented as helper functions in “error.py”

Backpropagation

The backpropagation implementation first started with a forward pass, which was used to save the outputs of each neuron (y_i) and to calculate the error. From here, I calculated δ_i for each layer, which was used to update weights and biases.

Momentum

Momentum was implemented by saving the previous gradients and using them as part of calculating the next gradient update.

Results

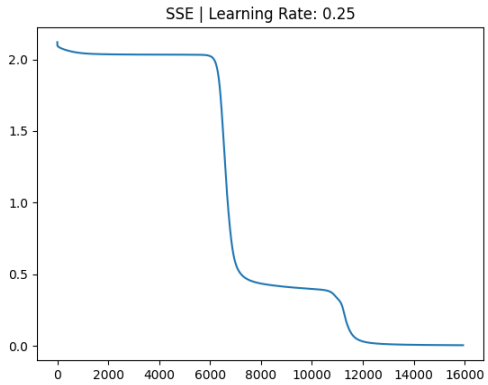
To stop training, the network is used to predict each of the possible inputs to verify the absolute error threshold. Also, the error for each training procedure was plotted.

Below are the results for the perceptron without momentum, as well as the error plot for the learning rate that converged in the fewest number of epochs. As I varied the learning rate η, the number of epochs ranged from around 15,942 (η=0.25) to 122,288 (η=0.3).

Perceptron Performance (No Momentum)

Learning Rate: 0.05 Epoch: 71900 Time: 57.2976279258728
Learning Rate: 0.1 Epoch: 31632 Time: 25.331719875335693
Learning Rate: 0.15 Epoch: 21384 Time: 16.97755765914917
Learning Rate: 0.2 Epoch: 16848 Time: 13.299903631210327
Learning Rate: 0.25 Epoch: 15942 Time: 12.662356615066528
Learning Rate: 0.3 Epoch: 122288 Time: 97.09324979782104
Learning Rate: 0.35 Epoch: 119770 Time: 94.40694689750671
Learning Rate: 0.4 Epoch: 91806 Time: 73.31185913085938

Learning Rate: 0.45 Epoch: 25410 Time: 20.261873960494995
Learning Rate: 0.5 Epoch: 21398 Time: 17.199246644973755



Below are the results for the perceptron with momentum, as well as the error plot for the learning rate that converged in the fewest number of epochs. As I varied the learning rate η, the number of epochs ranged from around 2,633 (η=0.5) to 163,855 (η=0.05). Compared to previously, adding momentum generally decreased the number of epochs required significantly, except for 2 cases (η=0.05 and η=0.1) where the epochs increased.

Network Performance with Momentum

Learning Rate: 0.05 Epoch: 163855 Time: 149.08933329582214
Learning Rate: 0.1 Epoch: 76101 Time: 69.05570888519287
Learning Rate: 0.15 Epoch: 2694 Time: 2.451601028442383
Learning Rate: 0.2 Epoch: 7373 Time: 6.698744058609009
Learning Rate: 0.25 Epoch: 4464 Time: 4.113528490066528
Learning Rate: 0.3 Epoch: 3673 Time: 3.3698880672454834
Learning Rate: 0.35 Epoch: 3232 Time: 2.982557535171509
Learning Rate: 0.4 Epoch: 2949 Time: 2.7328431606292725
Learning Rate: 0.45 Epoch: 2760 Time: 2.536674976348877
Learning Rate: 0.5 Epoch: 2633 Time: 2.4305837154388428