EE 456: Introduction to Neural Networks

11/12/2023

Ian Deal

Overview:

This mini-project consisted of constructing a multi-layer perceptron neural network with weights trained by using the back propagation algorithm. The perceptron takes in a 2-dimensional vector and has a 1-dimensional output and is trained to classify which bi-polar class the vector belongs to. Using the back propagation algorithm allows the perceptron to learn a non-linear decision-boundary between these two bipolar classes.

Implementation:

I divided both datasets into their respective training and testing datasets such that the training dataset is 80% of the original dataset and the testing set is the remaining 20%. Elements from the original dataset are chosen at random for the training set. These datasets are located within the Data/ folder.

I initialize two weight matrices, W1: a (3,20) matrix that connects the bias and the input 2-dimensional vector to the hidden layer of neurons, and W2: a (21,1) matrix that connects the hidden layer of neurons to the single output neuron. The weights in these matrices are initialized as a random distribution with a mean of zero and a standard deviation set to the reciprocal of the square root of the number of connections in the weight matrix. This heuristic is used to ensure that the weight values are not too large that their values overflow, and not too small that the back propagation model does not get stuck in a shallow divot on the error surface.

To start the weight training process, I utilize a stochastic update process which randomly selects 1 sample vector from the training dataset and is used to compute the forward pass of the model. With each iteration, a new random sample is chosen.

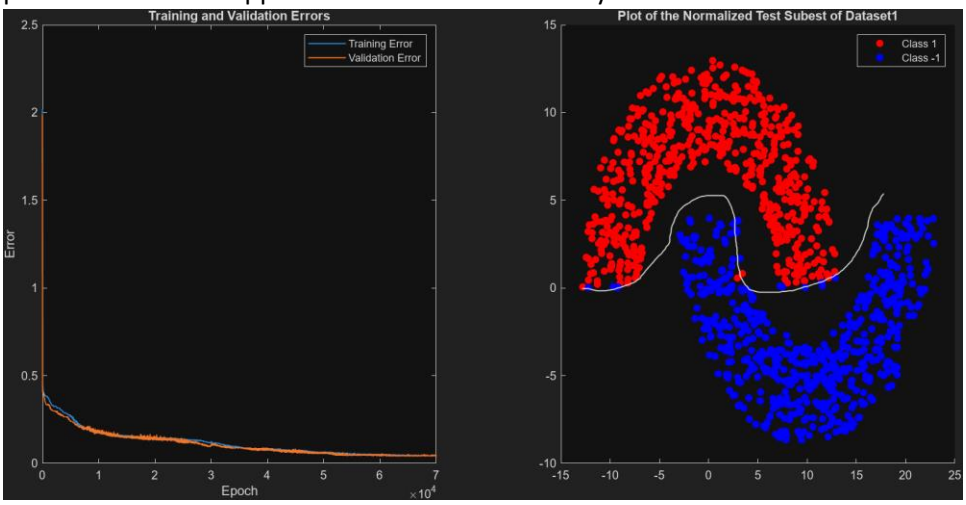
To make the training and testing process easier to follow, I abstracted the forward pass operations for training and testing into their own separate functions. The training forward pass function saves and returns the state of the hidden layer and output layer before and after the tanh activation function is applied to the neurons. The test forward pass function only returns the post tanh applied outputs and is designed to process a batch input for simpler validation testing. After each epoch of 1000 iterations, the testing forward pass is called with the training dataset and testing datasets, the mean squared error is computed between these output vectors and their corresponding target vectors.

The weights are then updated following the back propagation algorithm, an initial sigma value is computed by taking the difference between the desired output and the computed output from the forward pass, this value is multiplied by sech(hyperbolic secant) of the y_in value (pre activation output value). The sech function is used as it is the derivative of the tanh activation function used in the model. This sigma value is used to update the output layer weights and is used to calculate the error information values of the hidden neurons. After each training iteration, the eta (learning rate) value is linearly decreased by a value 7.5%.

Heuristics: setting initial weights, stochastic update in training to maximize information content of the network.

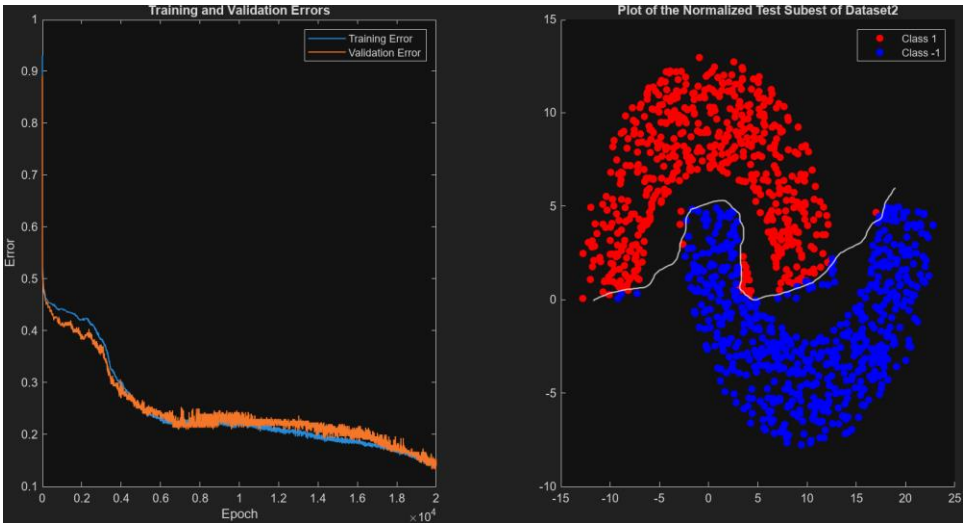
Results:

Dataset1: training error, validation error vs. epochs and a plot of the testing set with corresponding class predictions and an approximate decision boundary drawn in white:



An overall error rate of about 0.04.

Dataset2: training error, validation error vs. epochs and a plot of the testing set with a decision boundary drawn in white:



An overall error rate of about 0.15.

The overall error rates for both datasets are very small, showing that the network is able to accurately distinguish between the two classes. In both plots, the training error and validation error seem to converge to roughly the same error rate, suggesting that the model is not overfitting to the training

data. In the error plot for Dataset1, the training error is shown to be slightly higher than the validation error for most of the training epochs, this reaffirms that the model is not overfitting to the training data, likely due to the large variety of samples in the training dataset. This same trend can be noticed to a lesser degree in the error plots for Dataset2. In this plot the validation error tends to fluctuate much more than the training error, suggesting that the model is slightly overfitting to specific features in the training dataset. The resulting decision boundaries on the test datasets are very clearly non-linear and can more accurately separate the two classes than any piecewise-linear function could.