Report

Goal of this lab is to build a model that correctly predicts the ouput of a 7 bit odd parity detector.

We first build the training data set.The training set size is 128 x 7. Every row contains an input vector of length 7 which is an array of 1s and -1s. We will take all possible 128 combinations of data here.

We build a network with 1 input layer with 7 perceptrons, a hidden layer with 12 perceptrons and output layer with 1 perceptron.

We next train the weights of the 2 layers of neurons. In the input layer we have 7 input neurons. After adding the bias term, we get 8 input neurons. The hidden layer has 12 neurons. Therefore the dimension of the layer 1 weights w1 matrix is 8x12. These 12 hidden neurons along with the bias perceptron are linked to one output neuron. Hence the layer 2 w2 weight matrix is of dimension 13x1.

In every epoch we feed all the 128 training data (shuffled after every epoch) to the input neurons. These propogate to the output neuron.

Then the error observed in the output layer is fed back through the inner layers. While we use the activation function during the feed forward process to calculate the output of the next layer, we use the derivative of the activation function during back propagation. These two processes can be visualized as inverse processses.

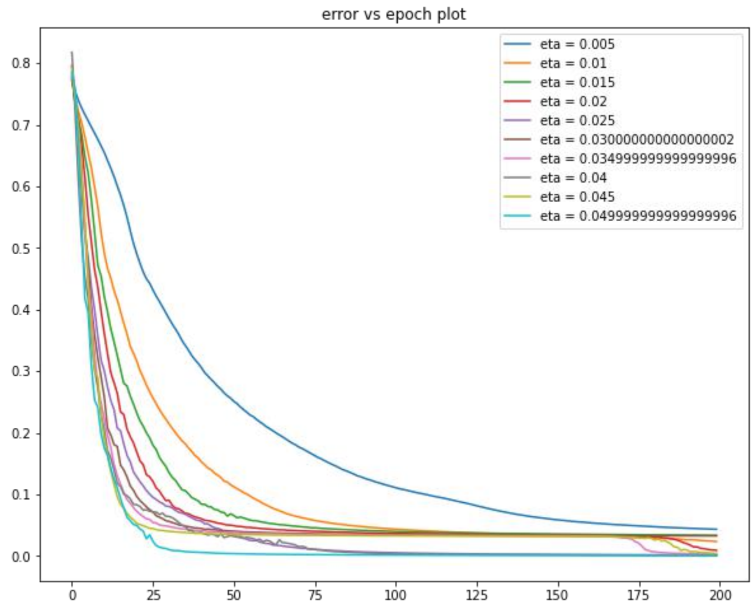
At the end of every epoch we average the squared output error observed for all 128 samples. Running this over 1000 epochs will give us an idea of the distribution of the training error.

The learning rate of the algorithm decides how soon the gradient descent converges or even converges at all.

Results

Effect of learning rate:

Convergence for eta =0.005 achieved at epoch 109
Convergence for eta =0.01 achieved at epoch 55
Convergence for eta =0.015 achieved at epoch 37
Convergence for eta =0.02 achieved at epoch 28
Convergence for eta =0.025 achieved at epoch 25
Convergence for eta =0.030000000000000002 achieved at epoch 20
Convergence for eta =0.034999999999999996 achieved at epoch 17
Convergence for eta =0.04 achieved at epoch 17
Convergence for eta =0.045 achieved at epoch 14
Convergence for eta =0.049999999999999996 achieved at epoch 15

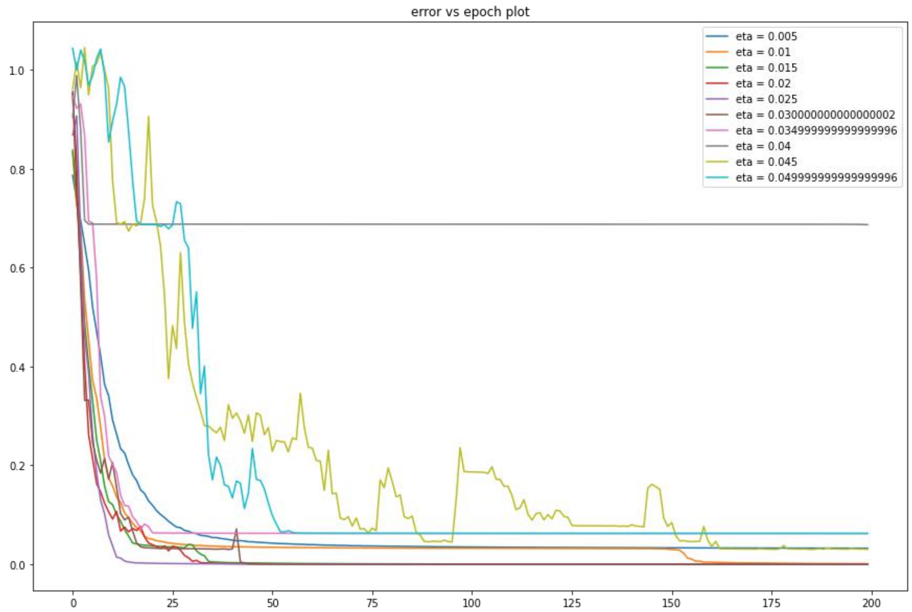


As can be seen from the output as learning rate increases, the algorithm converges faster. But as eta nears 0.05, the number of epochs needed becomes equal.

Effect of Momentum

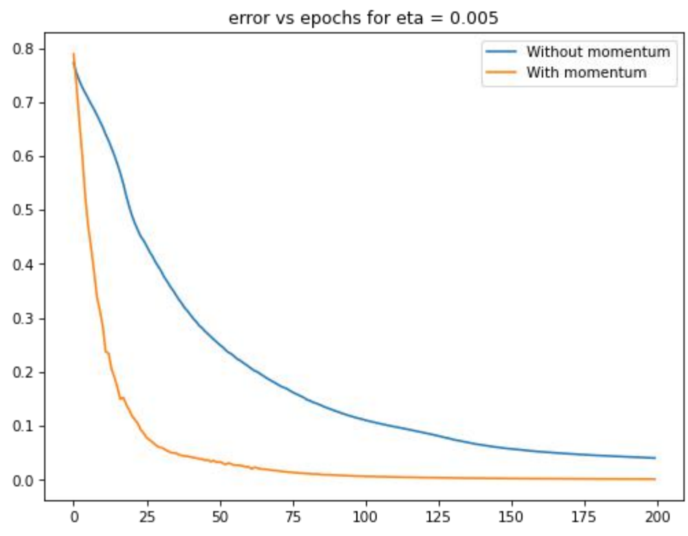
Adding the momentum term makes things interesting here. Even though momentum term helps speed up the convergece for larger eta values, we see that it is not true for all learning rate values here.

Convergence for eta =0.005 achieved at epoch 23
Convergence for eta =0.01 achieved at epoch 14
Convergence for eta =0.015 achieved at epoch 12
Convergence for eta =0.02 achieved at epoch 10
Convergence for eta =0.025 achieved at epoch 9
Convergence for eta =0.030000000000000002 achieved at epoch 13
Convergence for eta =0.034999999999999996 achieved at epoch 15
Convergence for eta =0.045 achieved at epoch 67
Convergence for eta =0.049999999999999996 achieved at epoch 50



Let us look at learning rate 0.005 with momentum 0.8 closely.

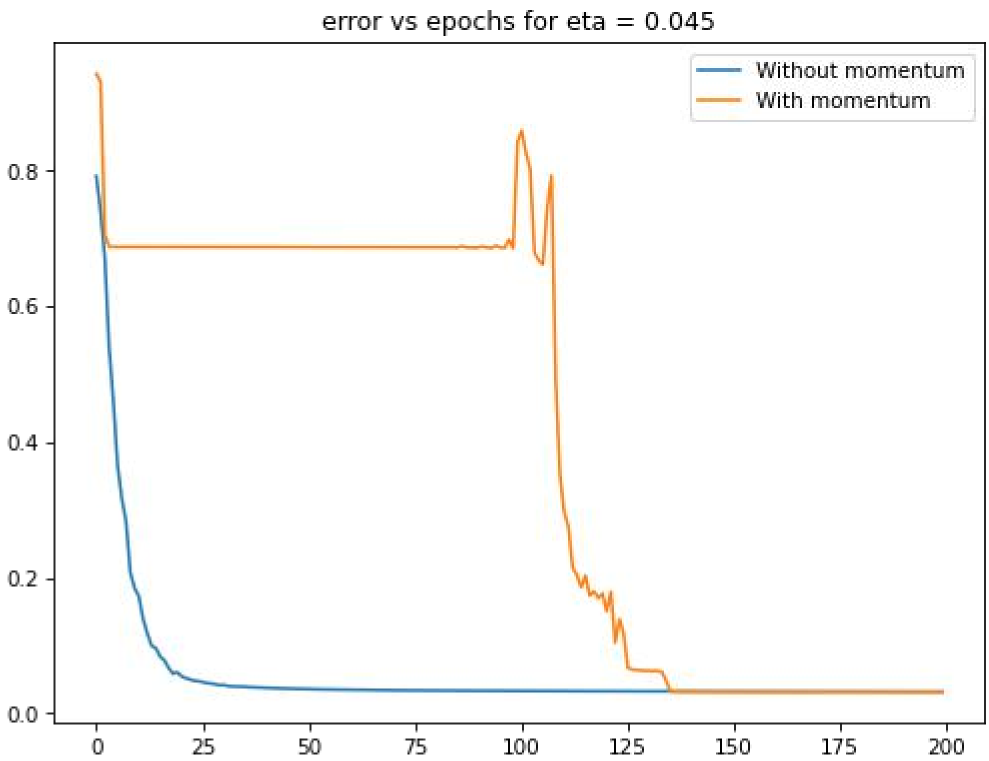
Convergence for eta =0.005 achieved at epoch 109
Convergence for eta =0.005 achieved at epoch 23



Here adding the momentum term has helped us accelerate learning.

Let us look at learning rate 0.045 with momentum

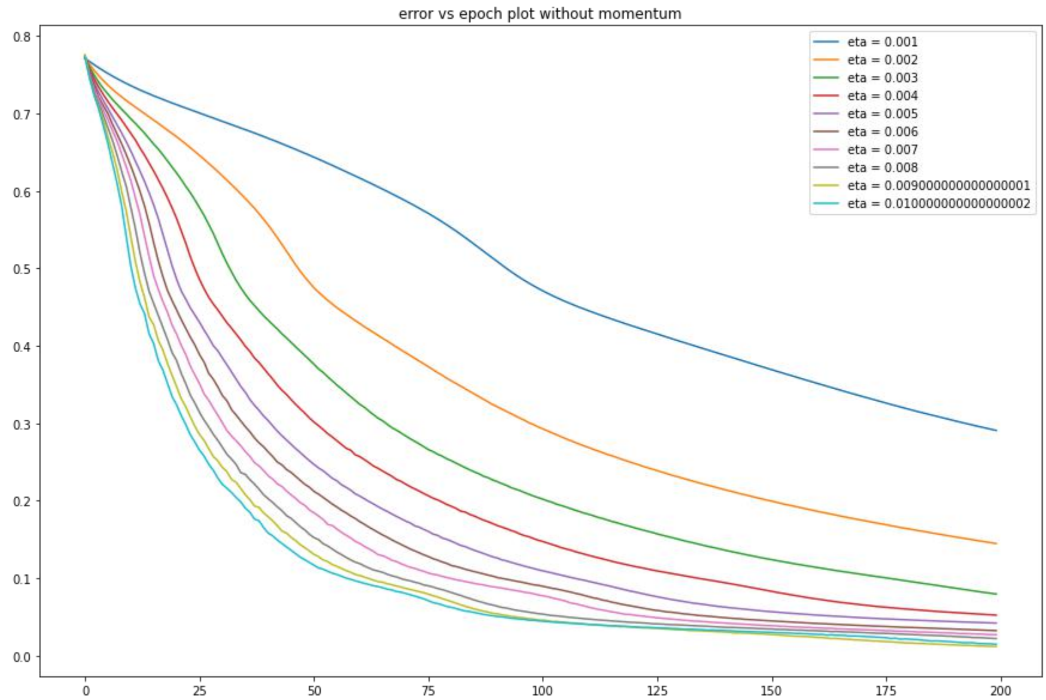
Convergence for eta =0.045 achieved at epoch 13
Convergence for eta =0.045 achieved at epoch 125

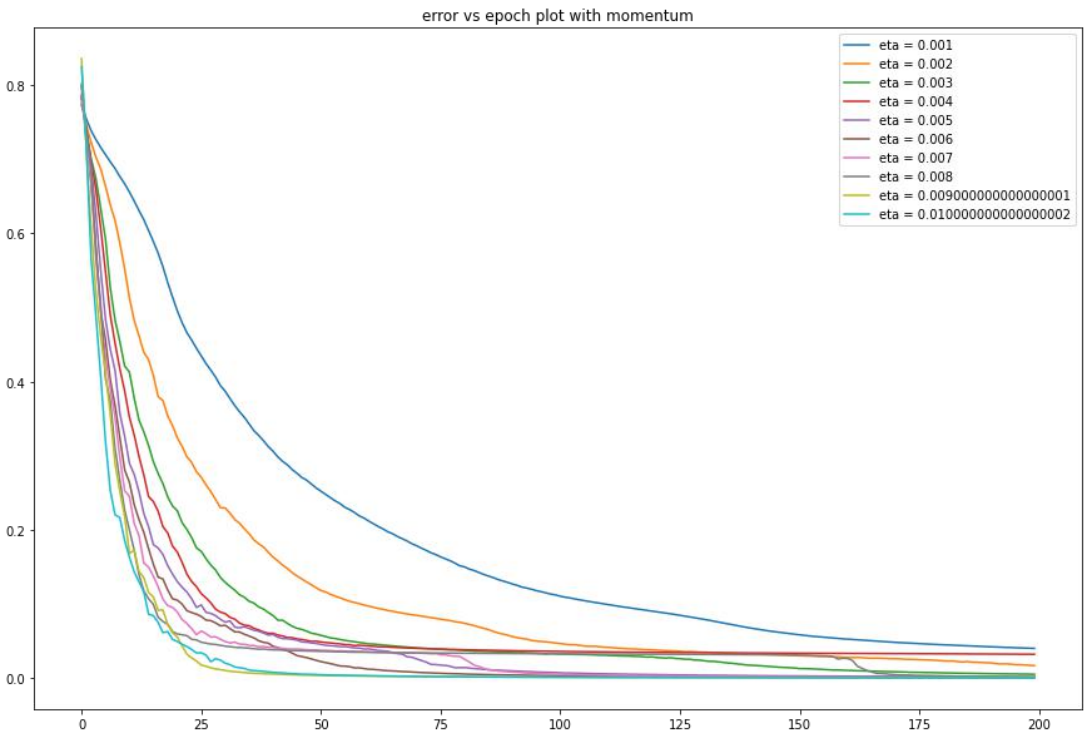


Here adding the momentum term has increased the convergence time.

As can be seen from the ouputs, eta = 0.045 has not yet converged after 200 epochs. This can be attributed to the fact that eta/(1-alpha) cannot be too large. When alpha = 0.8 , we have the new effective learning rate as 5*eta. So we need to have 5eta to be less than 0.05 atleast. Which means momentum addition can work for eta less than 0.01. When eta is greater than 0.01, the algorithm may end up in local minimas.

Effect of momentum for learning rates between 0.01 and 0.001

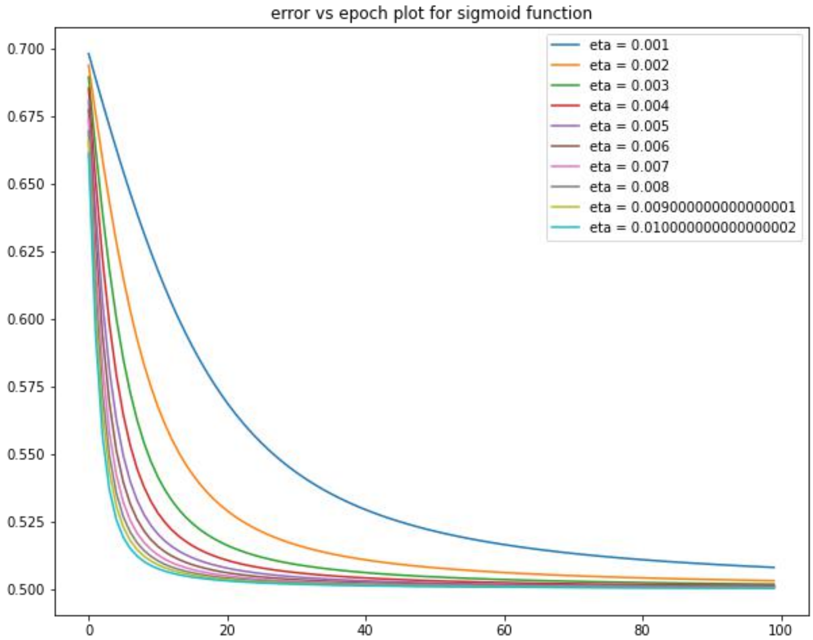


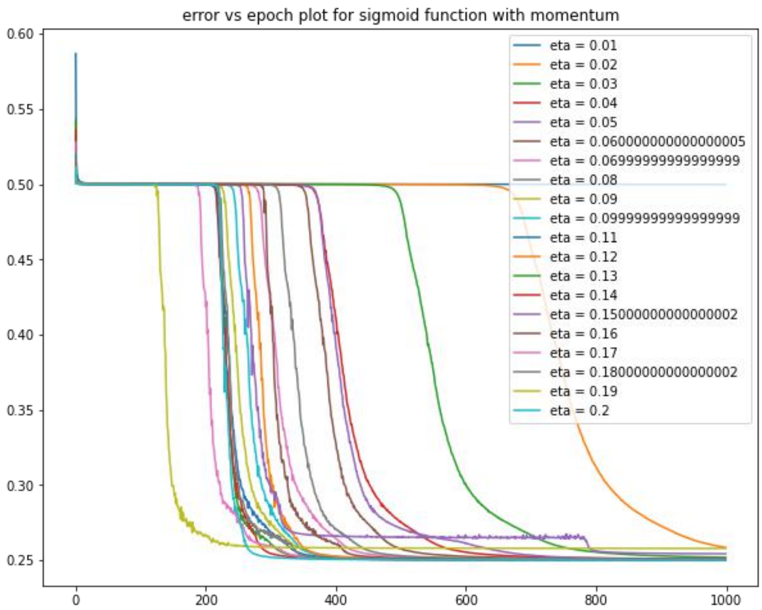


From the above observations it is clear that adding the momnetum term has helped speed up learning

Sigmoid Activation Function

For sigmoid activation function, as seen from the below graph the model converges to an error of 0.5 within 20 epochs. So it never reaches the error of 0.1 that we are looking at. Thus when we use sigmoid activation function, 50% of the data is classified incorrectly Also the max error for the sigmoid function is 0.7 which is lesser than the max error ofor tanh.





But adding the momentum just gets the error down to 0.25

Conclusion

The built neural netwok with 12 hidden layers can correctly classify the input data. For learning rates less than 0.05 it converges very fast in 15 epochs.
Adding momentum to the learning sometimes helps speed up the training process but in cases where the new effective eta crosses a certain threshold, the convergence is delayed.
Sigmoid activation function on the other hand fails to give an error of 0.1. Adding momentum here only helps bring down the min error.