**MLPR Lab 11**

This assignment builds upon the last week lab. This Network will be able to solve XOR problem which single perceptron will not be able to do because perceptron only classifies linearly separable data. Additionally, you will also learn a use of few hyperparameters in this assignment that you help you train a Network faster more efficiently.

Instructions:

* Due time: 1:30 PM
* Submit all the four graphs along with the complete code.
* Many of you are not appearing in the lab and just uploaded the results without showing me. Sometimes there had some twist in parameters to get the correct output that you might have not gotten and uploaded the output simply you produced. Please be in the lab.
* Using Google Colab is recommended for this lab as there might some issues occurs due to the tensorflow installation and their dependencies.
* There can be a difference in values in the plots as compared to the given output to what you produce. Show me the outputs for the verification of the results.

Step1: import libraries

* Numpy
* Keras from tensorflow
* Dense layers from keras
* Matplotlib

Step2: Take XOR input data and store in one variable (Input data), Store output data of XOR in another variable (Target data)

Step3: Create the model.

* Define sequential model.
* Add first layer in the model as given parameters
  + Input data=2
  + No of node =8
  + Activation = relu
* Add the second layer
  + No of node=1
  + Activation=sigmoid
* Keep learning rate = 0.1
* Use SGD as an optimizer with given learning rate.

Step4: Compile the model with defined optimizer in previous step with MSE as loss calculator.

Step5: Now, we need to record the learning rates so that we can capture the no of epochs at model converging.

* Create a callback to record learning rates using ***keras.callbacks.Callback***.

Step5: Train the model and monitor the convergence and learning rates.

* Converged=False
* Target\_loss=0.001 (Convergence criteria)
* Losses =[]
* Learning\_rates=[]

Step6: Initiate the while loop to check if the network is not converged. Check this after every 5 or 10 epochs. If network is not converging at this particular epoch, use this-

History =model.fit(X,y, epoch=10, verbose=1, callbacks =[LearningrateCallback()])

Loss=history.history[‘loss’][0]

Losses.append(loss)

Step6: Plot the SSE (Sum of squared error) vs. Number of epochs. In title, it should show the no of epoch at which the network converges. See the reference output.

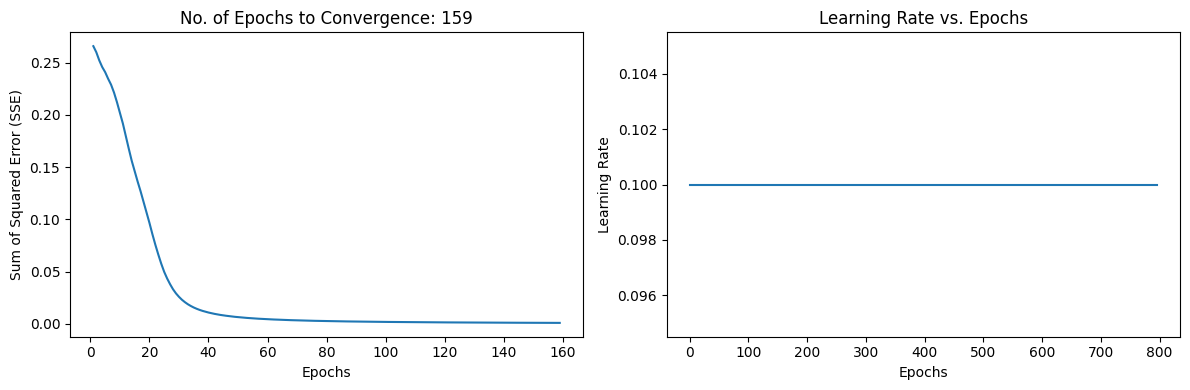
Step7: Plot the graph of learning rate vs Number of epochs. See the reference output.

A comparison of a graph

Description automatically generated

Step8: Now You need to do the same task by incorporating momentum based learning rate to accelerate the learning .

* Keep all the parameters same as above for all the steps.
* In step3, Add the momentum = 0.9 and use this momentum in optimizer along with the given learning rate.
* Keep the rest as same and generate both the graphs shown below.



Step9: Here, we use adaptive based learning rate. Adaption criteria of learning rate is given below (Taken from LEc-17, slide 14).

* If SSE at the current epoch exceeds the previous by more than 1.04, then decreased the learning rate by 0.7. If error is less than the previous one, increase the learning rate by 1.05.
* Train the model again with new learning rate . Do this after every 5 epochs.
* Do not use momentum here. Just replicate step1 to step 7 and produce the graphs.

A graph of a line

Description automatically generated with medium confidence

Step 10: Now use both Momentum and adaptive based learning rate.

* For this , keep step 9 as it is and just add momentum=0.9 in optimizer along with the learning rate

A graph of a number of people

Description automatically generated with medium confidence