# Lab 12

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

**Instructions:**

* Submit all the four sets of graphs along with the complete code ( and code+output pdf). (The images are shown for reference. You may get different convergence values)
* Using Google Colab is recommended (not necessary) for this lab as some issues might occur due to the tensorflow installation and their dependencies.

**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 the defined optimizer in the previous step with MSE as the loss term.

**Step5:** Now, we need to record the learning rates so that we can capture the number of epochs at model converging.

**TASK 1: Convergence speed for the default case**

**Step6:** Train the model and monitor the convergence and learning rates.

Target\_loss <= 0.002 (Convergence criteria)

**Step7:** Check every 10 epochs whether the model is converged.

**HINT:**

1. You can use a custom callback to monitor learning rates. You can use the following custom callback that appends learning rate every epoch into a list:

learning\_rates = []

class LearningRateCallback(keras.callbacks.Callback):

def on\_epoch\_begin(self, epoch, logs=None):

lr = self.model.optimizer.lr.numpy()

learning\_rates.append(lr)

1. You can check convergence every 10 epochs by running the following in a while loop until convergence criterion is met:

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

**Step7:** Plot the SSE (Sum of squared error) vs. Number of epochs. In title, it should show the no of epochs at which the network converges. See the reference output.

**Step8:** Plot the graph of learning rate vs Number of epochs. See the reference output.

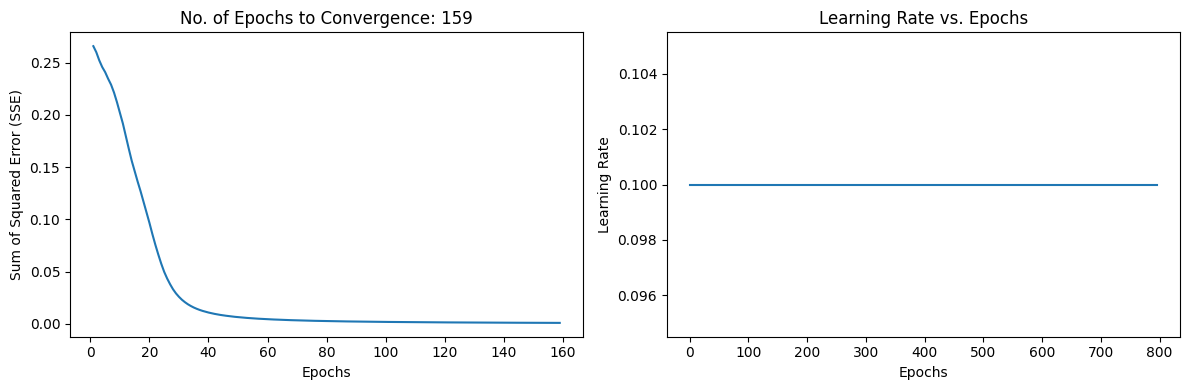
A comparison of a graph

Description automatically generated

**TASK 2: Understanding the effect of momentum on convergence speed**

**Step9:** 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.



**TASK 3: Understanding the effect of adaptive learning rate on convergence speed**

**Step10:** Here, we use adaptive learning rate. Adaption criteria of learning rate is given below

* If SSE at the current epoch exceeds the previous by more than 4 percent, then decrease the learning rate by 30 percent (multiply by 0.7). If error is less than the previous one, increase the learning rate by 5 percent.
* Train the model again with new learning rate . Do this after every 5 epochs.
* Do not use momentum here. Just replicate step1 to step 6 and produce the graphs.

A graph of a line

Description automatically generated with medium confidence

**TASK 4: Using both momentum and adaptive learning rates for faster convergence**

**Step 11:** Now use both Momentum and adaptive based learning rate.

* For this , keep step 10 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