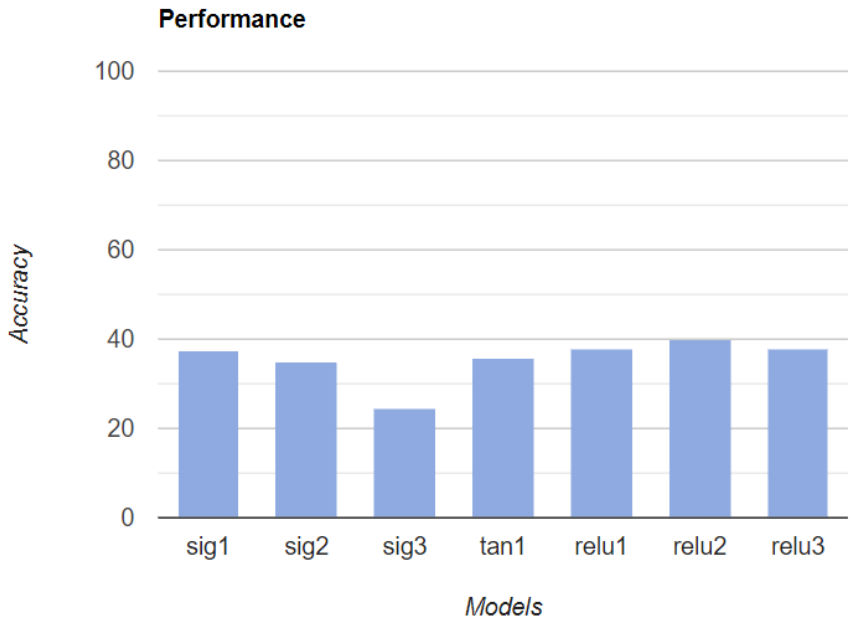
Report File
CSL4020
Assignment -1
Name: Ritam Sharma (B20BB0030)

Q1

▼ Utilize various activation functions like sigmoid, tanh and critique the performance in each case

The following MLP models were implemented :

1. One hidden layered MLP model with a sigmoid activation function `sig1`
2. Two hidden layered MLP model with a sigmoid activation function `sig2`
3. 3 hidden layered MLP model with a sigmoid activation function `sig3`
4. One hidden layered MLP model with a Tanh activation function `tan1`
5. One hidden layered MLP model with a ReLu activation function `relu1`
6. One hidden layered MLP model with a ReLu activation function `relu2`
7. One hidden layered MLP model with a ReLu activation function `relu3`



⚠️ It is observed that as the number of hidden layers increased for the sigmoid, the performance significantly reduces, but the opposite is true for ReLu.

The reason for this observation is the vanishing gradient problem. The vanishing gradient problem occurs in deep neural networks during the training process, where the gradients of the parameters become very small, making it difficult for the optimizer to update the weights. This results in slow convergence or even a complete halt in the training process.

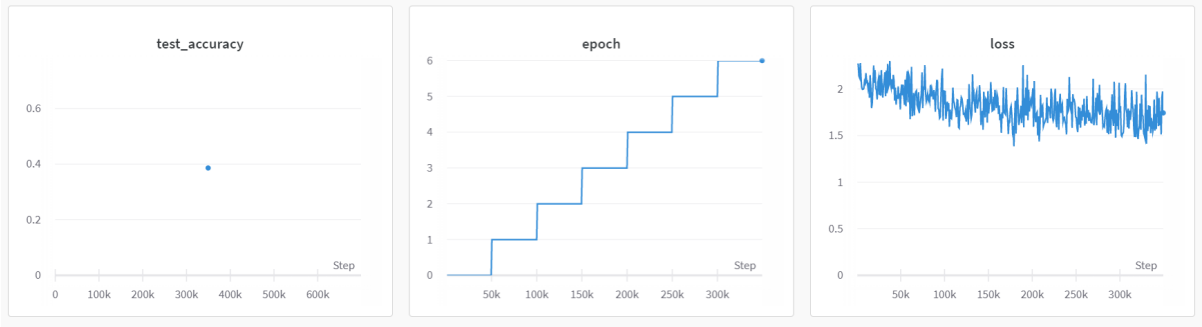
ReLU when used activation function helps address the vanishing gradient problem. The ReLU activation function returns the input if it's positive and 0 if it's negative. This allows the gradients to flow freely through the network, avoiding the saturation of activation functions like sigmoid or tanh, which can lead to the vanishing gradient problem

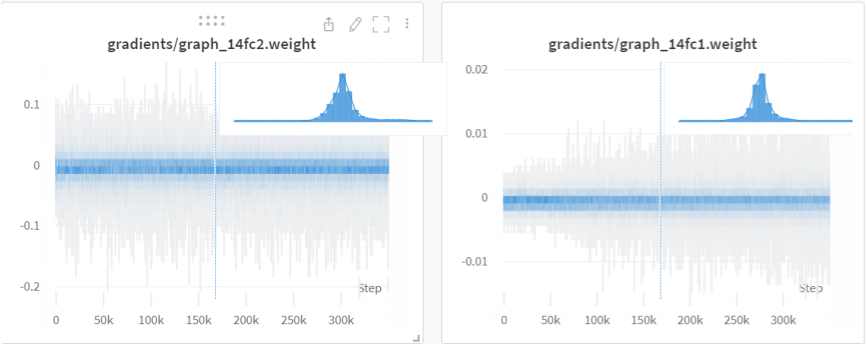
▼ Increase the depth of the given network by adding more Fully-Connected layers till the point you encounter the vanishing gradient problem. With the help of the results, mention how to identify it.

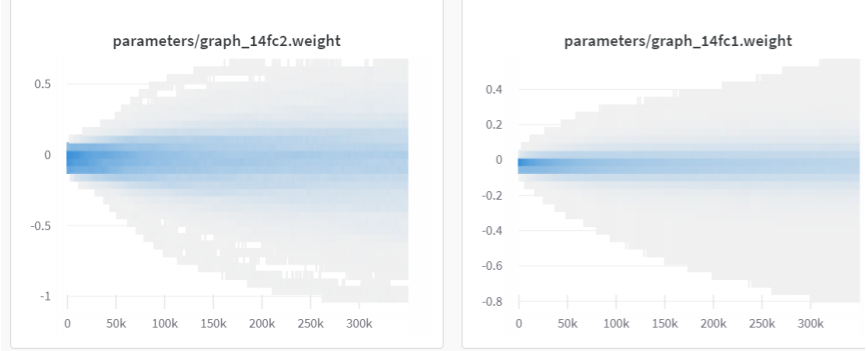
To observe the vanishing gradient problem, I plotted the

1. weight distribution throughout the training epochs
2. gradient distribution throughout the training epochs
3. accuracy

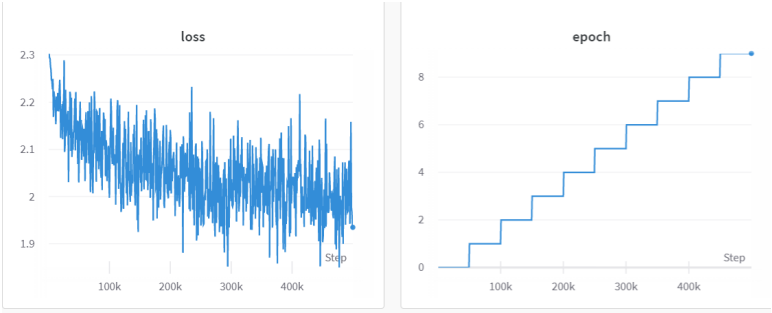
Plots for sig1 :

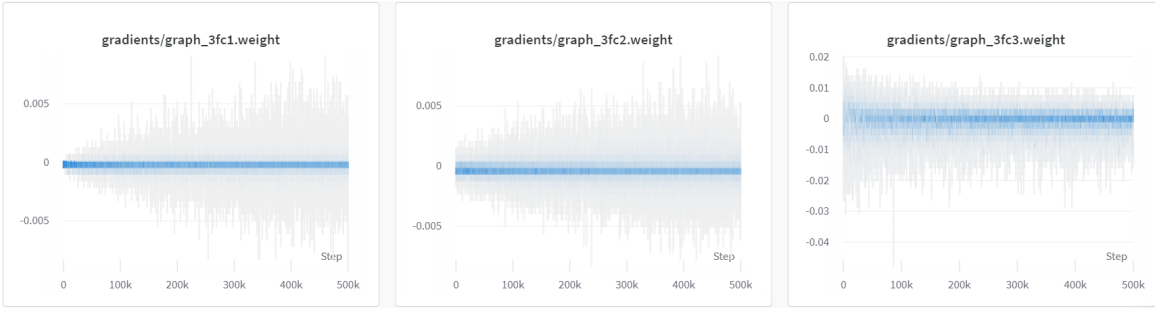




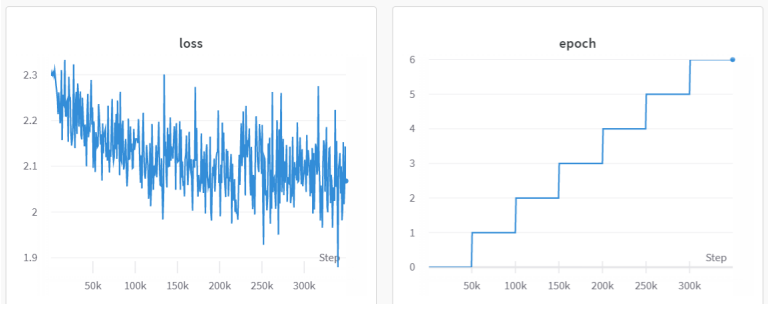


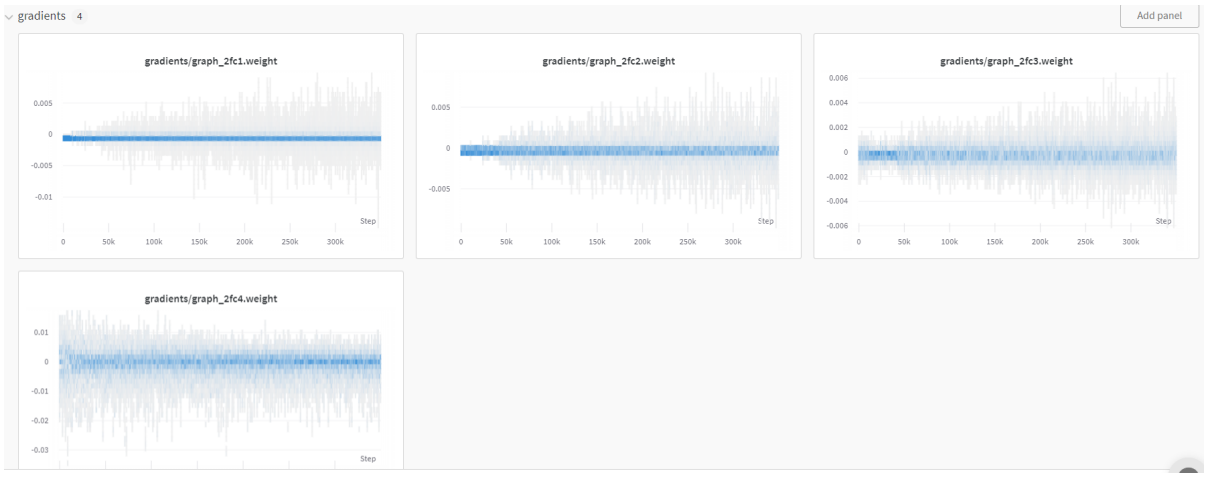
Plot for sig2:

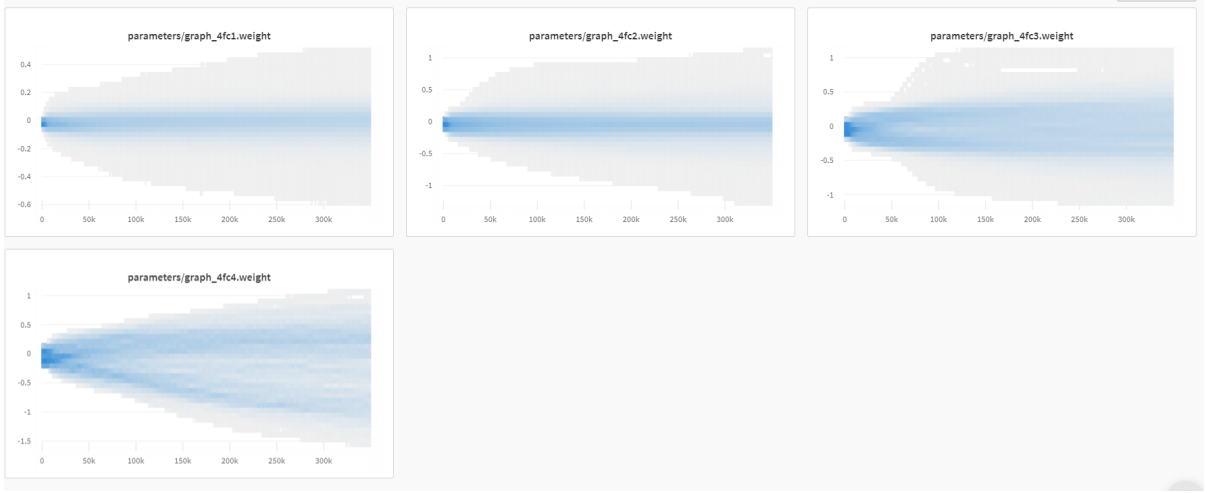




Plot for sig3 :







> vanishing gradient can be detected by the following observations :

⚠️ Observation 1: the fc1 layer for the sig1 model has gradients < |0.1| as observed from the graphs, however as we are increasing the hidden layers, for the model sig3, the gradients in fc1 are in the range <|0.005| as observed in the graphs
This is due to the vanishing gradient caused by sigmoid

⚠️ Observation 2: In model sig3, large changes are observed in the parameters of later layers(fc4), whereas parameters of earlier layers (fc1)change slightly or stay unchanged, this is an indicator of vanishing gradient

⚠️ Observation 3: the parameters of the model in layer fc1 in sig3 are a lot closer to zero than the parameters of the model in layer fc1 in sig1, this can be due to the vanishing gradient as well

⚠️ Observation 4: The declining accuracy on adding hidden layers and sigmoid activation function is also an indicator of vanishing gradient

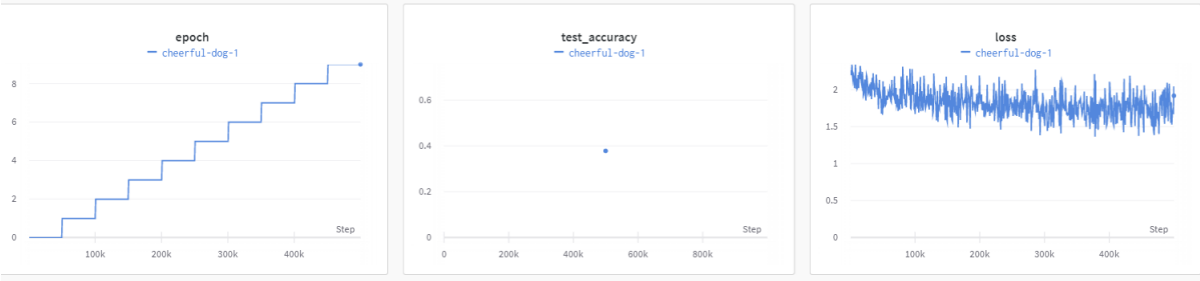
▼ Suggest and implement methods to overcome the above problem.

The vanishing gradient problem is caused by the derivative of the activation function used to create the neural network. The simplest solution to the problem is to replace the activation function of the network. Instead of sigmoid, use an activation function such as ReLU.

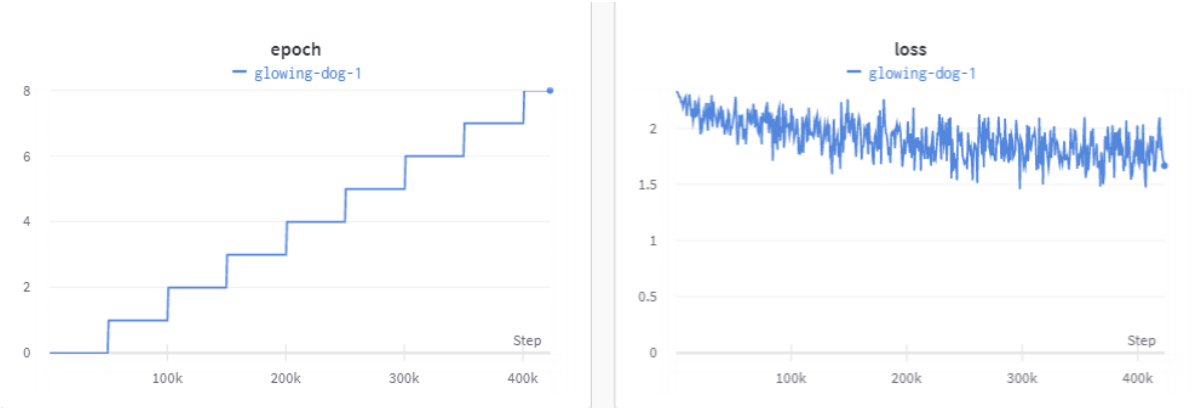
Rectified Linear Units (ReLU) are activation functions that generate a positive linear output when they are applied to positive input values. If the input is negative, the function will return zero.

result of ReLu implemented:

one hidden layer :



three hidden layer :



The 3 hidden layer MLP with Relu performs significantly better than sigmoid

Q2

L1 tends to shrink coefficients to zero whereas L2 tends to shrink coefficients evenly. L1 is therefore useful for feature selection, as we can drop any variables associated with coefficients that go to zero. L2, on the other hand, is useful when you have collinear/codependent features.

▼ L1 regularizer

▼ Run 1/experiment 1

Hyperparameters chosen:

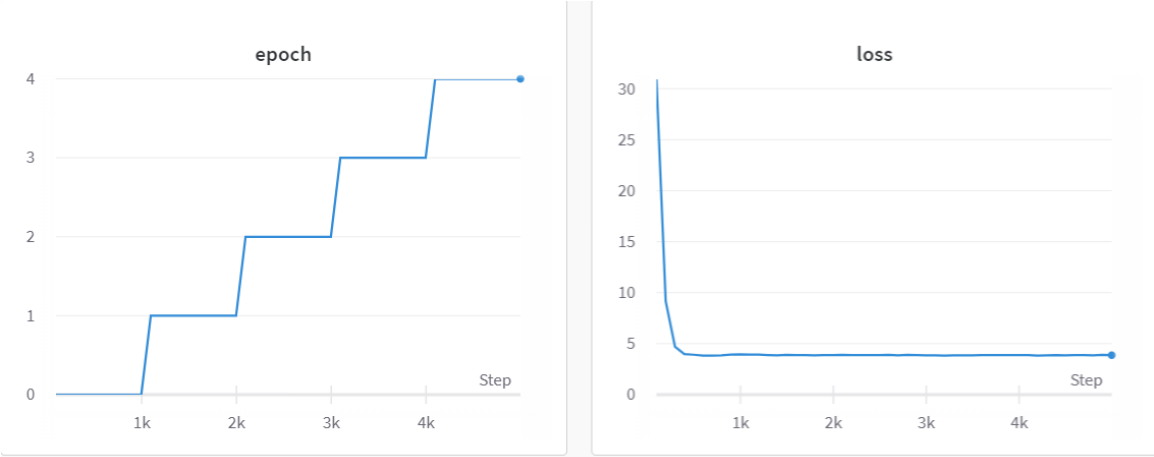
epoch: 5

lambda: 0.1

learning rate =0.01

Accuracy: 10.11%

Result:



Observations and reasons to update the hyperparameters :

⚠️ Accuracy is too low, the model is underfitting due to simple architecture and high lambda value, therefor the values of lambda is decreased to allow more complexity in the model and we can observe that the accuracy does improve upon decreasing lambda

⚠️ The loss curve has reached its minima much earlier, even when iterations are continuing, therefore it makes sense to reduce the epoch size

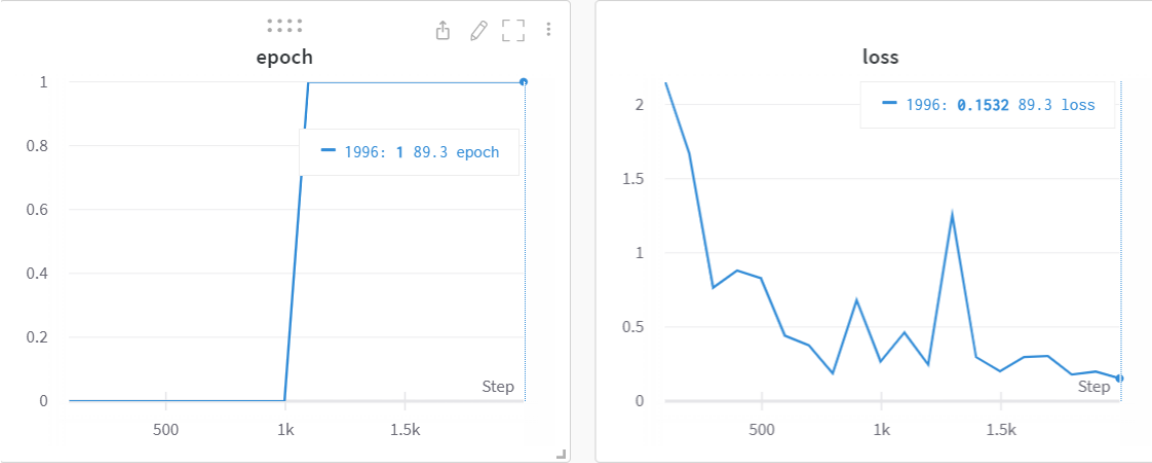
▼ Run 2/experiment 2 :

lambda=.001

epoch=2

Accuracy=89.3%

learning rate =0.01



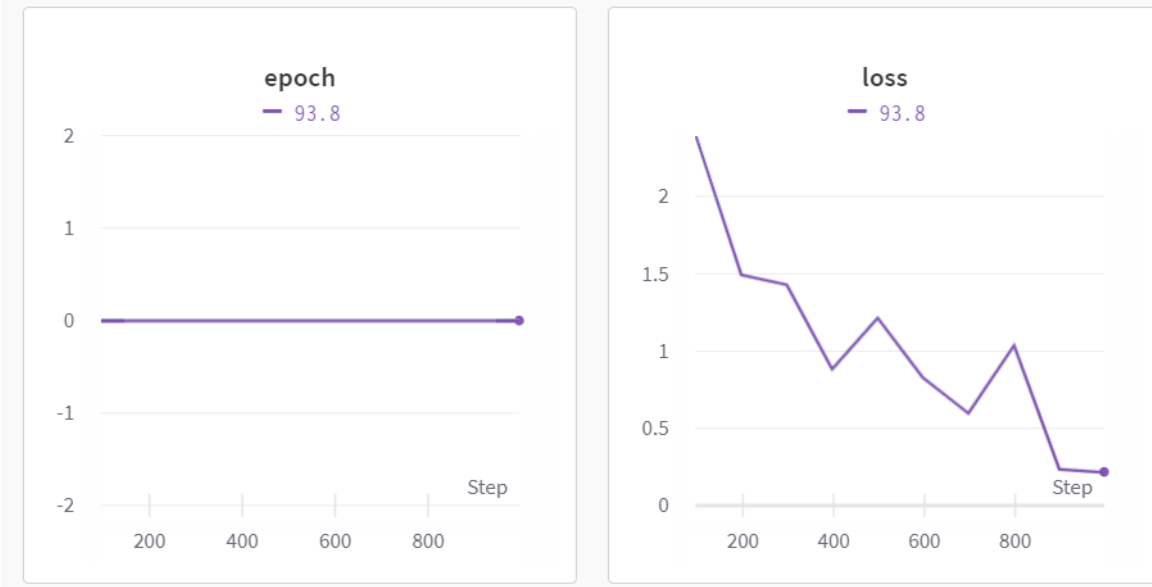
▼ Run 3/Experiment 3 :

Lambda=.0001

epoch=1

Accuracy: 93.8%

learning rate =0.01



These hyperparameters give the most fitting result therefore, these shall be used

▼ L2 regularizer

For the L2 regularizer, the model was performing significantly poorly as opposed to l1, this could imply that the data given is quite less codependent

therefore the choice of Hyperparameter lambda was equal = 0.00001

▼ Dropout

Dropout is a regularization method approximating the concurrent training of many neural networks with various designs. During training, some layer outputs are ignored or dropped at random This makes the layer appear and is regarded as having a different number of nodes and connectedness to the preceding layer

The Hyperparameters chosen were

1. 0.12
2. 0.5
3. 0.25

And the following result was received :

These values of the hyperparameter suggest that 0.25 is the best fit due to higher accuracy and better loss curve