MACHINE LEARNING
ASSIGNMENT 2 REPORT

AASTHA -2019224

1

Preprocessing

1. The No Column was dropped
2. Missing values were dropped from the dataset.
3. Year and cbwd columns were encoded using label encoder.

Implementing train_val_test_split from scratch

The lengths for train, test and validation sets were calculated by multiplying the size with the actual dataset length. The shuffled copy of the dataset was created and then the dataset was split into the 3 sets.

A

Training the decision tree model using Gini Index and Entropy the following accuracies were obtained:

```
criterion='gini'
```

```
Accuracy:  0.8227686412262494
```

```
 criterion='entropy'
```

```
Accuracy:  0.8305923678748204
```

In the following models the DecisionTree with `criterion='entropy'`

 is used as it gives better accuracy on the test set.

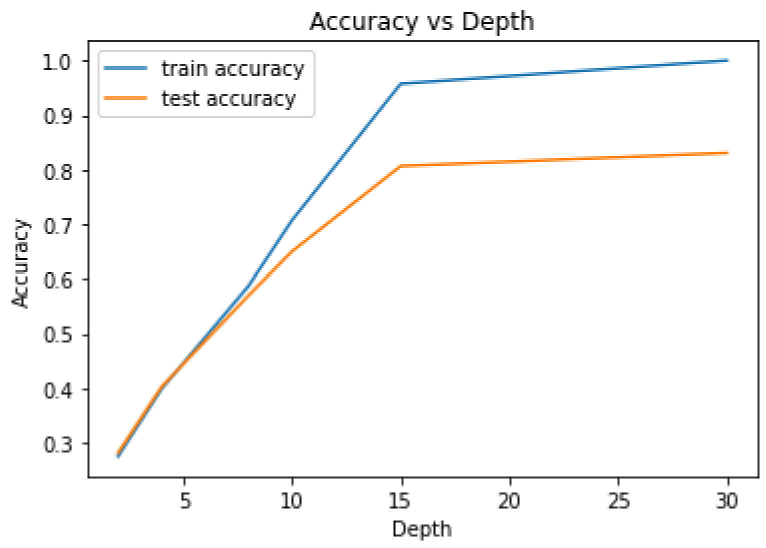
B

On training the decision tree on different depths = [2, 4, 8, 10, 15, 30] the following train and test accuracies were obtained:

```
Train Accuracy:  [0.27528993192159007, 0.39885737744175703,
0.5873558892956108, 0.7071602066299476, 0.9573398104751805, 1.0]
```

```
Test Accuracy:  [0.2818138272393422, 0.40268242056522435,
0.5700143701101709, 0.6501676512853265, 0.8064825163659588,
0.8305923678748204]
```

The best value of depth is 30 as it gives the highest rain and test accuracy.



The accuracy v/s depth graph also shows the highest accuracy on depth = 30.

C

Ensembling was done by creating 100 decision tree stumps by training on random 50% of the training data. The data was selected using a random_selection function created from scratch. The mode was taken over all the predictions to get the final prediction. The ensemble model gave an accuracy of `0.3557400606737985`. This shows that ensemble models don't always improve the accuracy as the performance has decreased compared to parts A and B where the best accuracy was `0.8305923678748204`. This indicates that our model has high bias and high variance. We can tune the model parameters to get a better accuracy.

D

Analysis

The decision tree was tuned by training it on depths = [4,8, 10, 15, 20,30] and n_estimators = [4,8,10,15,20,100]. The values obtained for train, validation and test accuracy for different depths and n_estimators are shown in the table below.

As we can see from the below table the accuracy is not being affected much by the n_estimators and is increasing with increasing depth. The test accuracy is best for depth = 30 and n_estimators = 100. The training accuracies are higher than validation and test accuracy.

	depth	n_estimator	train accuracy	validation accuracy	test accuracy
0	4	4	0.398926	0.385917	0.404119
1	4	8	0.407170	0.403640	0.410027
2	4	10	0.416510	0.408909	0.420885
3	4	15	0.416920	0.409708	0.419288
4	4	20	0.415347	0.404918	0.422162
5	4	100	0.412678	0.401565	0.419128
6	8	4	0.627040	0.595881	0.605301
7	8	8	0.634806	0.605301	0.609772
8	8	10	0.640450	0.612646	0.620789
9	8	15	0.647669	0.622385	0.624142
10	8	20	0.646369	0.622066	0.627335

11	8	100	0.656700	0.623663	0.634361
12	10	4	0.749102	0.687051	0.696950
13	10	8	0.788854	0.726489	0.731918
14	10	10	0.794294	0.735909	0.737027
15	10	15	0.806917	0.750758	0.745649
16	10	20	0.809038	0.755708	0.751557
17	10	100	0.827204	0.772633	0.767204
18	15	4	0.935445	0.825164	0.813189
19	15	8	0.969313	0.865879	0.853744
20	15	10	0.976190	0.875459	0.864602
21	15	15	0.984503	0.887913	0.880728
22	15	20	0.988129	0.896216	0.885837
23	15	100	0.994937	0.910586	0.905956
24	20	4	0.948993	0.826760	0.823567
25	20	8	0.981663	0.871467	0.861728
26	20	10	0.987753	0.880728	0.873703
27	20	15	0.993603	0.895577	0.888711
28	20	20	0.996032	0.903082	0.895737
29	20	100	0.999692	0.923359	0.914897
30	30	4	0.950908	0.825164	0.816701
31	30	8	0.982005	0.867955	0.859652
32	30	10	0.987616	0.876896	0.873064
33	30	15	0.994424	0.894140	0.887115
34	30	20	0.995997	0.901804	0.893821
35	30	100	0.999761	0.923040	0.916494

E

Adaboost classifier was used from sklearn with `DecisionTreeClassifier(criterion='entropy',random_state=0)` as the base estimator and n_estimators as [4,8,10,15,20]. The following test accuracies were obtained:

```
Accuracy Score for n_estimators = 4: 0.8278780137314387
Accuracy Score for n_estimators = 8: 0.8288360210761616
Accuracy Score for n_estimators = 10: 0.8329873862366278
Accuracy Score for n_estimators = 15: 0.8304326999840332
Accuracy Score for n_estimators = 20: 0.8305923678748204
```

The best test accuracy was obtained with n_estimators = 10.

We notice that the best test accuracy in the case of CustomRF(91.6%) is better than in the case of AdaBoost(83%). This is because of different sampling techniques. Random forest uses bagging whereas AdaBoost uses boosting. In adaboost all trees don't have equal weight in predicting the final values. Another reason for the difference in results is that in Random Forest all trees are independent so the order in which they are created does not matter whereas in case of Adaboost the order matters. Random forest decreases variance whereas adaboost aims to decrease bias.

2

Preprocessing for MNIST

1. y_train and y_val values were label binarized for the neural network from scratch.
2. Feature scaling was performed on X using standard scalar.

Implementing train_val_test_split from scratch

The lengths for train, test and validation sets were calculated by multiplying the size with the actual dataset length. The shuffled copy of the dataset was created and then the dataset was split into the 3 sets.

Implementing Neural Network from scratch

Neural network consists of parameters N_layers, activation function, weight_init-random, normal and zero, learning rate, epochs, batch size. It consists of activation functions - relu, leaky relu, sigmoid, linear, tanh and softmax and their gradients.

Fit-

In the fit method we iterate over the epochs and for each epoch we iterate over the batches, perform forward propagation, backward propagation and loss calculations.

Forward Propagation

The output of a layer is the input of the next layer. We perform a dot product of the input with the weights, add the bias term and pass it to the activation function to get the output of a layer.

Backward Propagation

Given the derivative of the error with respect to the output (dE/ dY) we find and return the derivative of the error with respect to the input(dE/ dX). This is done using the formula dE/dX = (dE/dY). W.T

The derivative of weights is calculated using the formula dE/ dW =(X.T) (dE/dY) and the weights are updated using the gradient descent algorithm using the formula W = W - learning_rate * dE/ dW.

In the activation layer dE/dX is calculated using the formula dE/dX = dE/dY * f'(x).

Cross entropy loss function

It is defined as H(p,q) = - $\sum_{x \in classes}$ p(x) log q(x) where p(x) is the true probability distribution and q(x) is the prediction distribution and its derivative is given by predicted value - true value.

Activation Functions-

ReLU- ReLU activation function is defined as f(x) = max(0,x) and its derivative is defined as f(x) = { 0 if x<0, 1 if x>0 }

Leaky ReLU - Leaky ReLU activation function is defined as f(x) = max(0.01*x, x) and its derivative is defined as

f(x) = { 1 if x>0, 0.01 if x<0}

Sigmoid - Sigmoid function is defined as f(x) = 1 / (1 + e^{-x}) and its derivative is defined as f(x) * (1 - f(x))

Linear - Linear activation function is defined as f(x) = x and its derivative is defined as f'(x) = 1.

tanh - Tanh activation function is defined as f(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x}) and its derivative is defined as 1 - $f(x)^2$

softmax - Softmax activation function is defined as f(x) = e^{xi} / $\sum_{j=1}^{k}$ e^{xj} and its derivative is defined as { f(xi) * (1- f(xi)) if i = j , - f(xi) * f(xj) if i$\neq j$.

Predict and Predict_proba Method -

We iterate over the testing samples and predict the output using forward propagation.

Saving model to files

The model is saved to files using pickel.

Using MNIST for training and testing of neural network model

1

The model is trained with 4 hidden layers, hidden layer sizes [256, 128, 64, 32] and learning rate = 0.08, epochs = 150, normal weights initialization, batch size = len(y_train)//20. The following test accuracies were obtained:

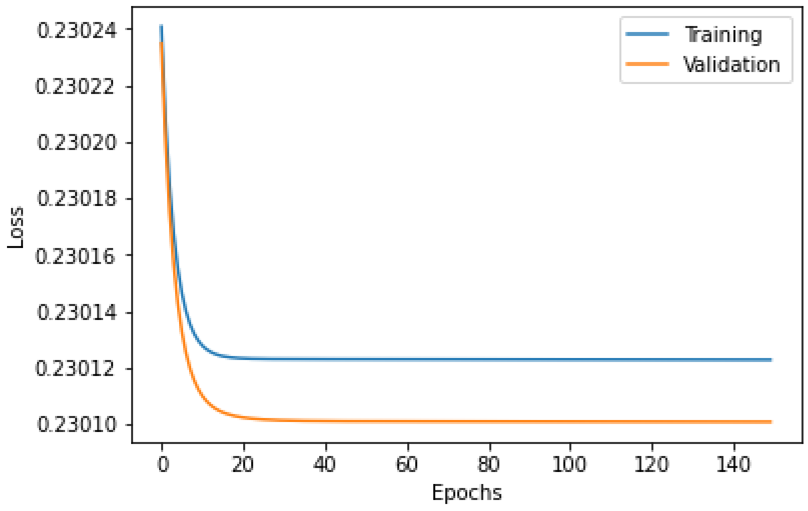
Activation Function	Accuracy (Normal init)	Accuracy(Random init)
ReLU	0.11814285714285715	0.495
Leaky ReLU	0.11814285714285715	0.42428571428571427
Sigmoid	0.11814285714285715	0.11814285714285715
Linear	0.11814285714285715	0.38971428571428574
tanh	0.11814285714285715	0.7761428571428571

As we can see the accuracy is not very good for normal weights for the given model parameters. It can be improved by reducing the batch size. There is a tradeoff between batch size and epochs. We can reduce the epochs and the batch size to get a better accuracy. Reducing the batch size increases the computation time for training hence the results were shown with batch size len(y_train)//20. Another way to improve the accuracy is to increase the learning rate as we can see in part E results- learning rate 1. gives an accuracy of 95.7%.

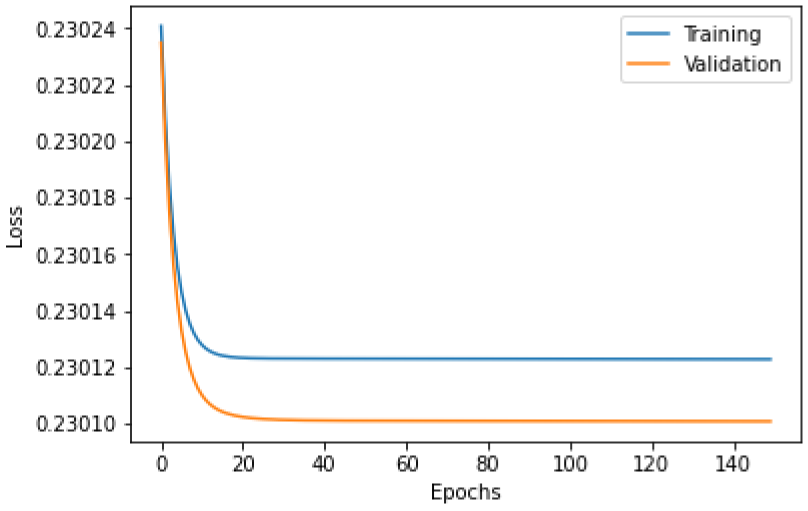
With normal weights the model gets stuck in a local minima during training and as we can also see from the graphs below the error also becomes constant very soon. In case of random weights we get better accuracy.

2

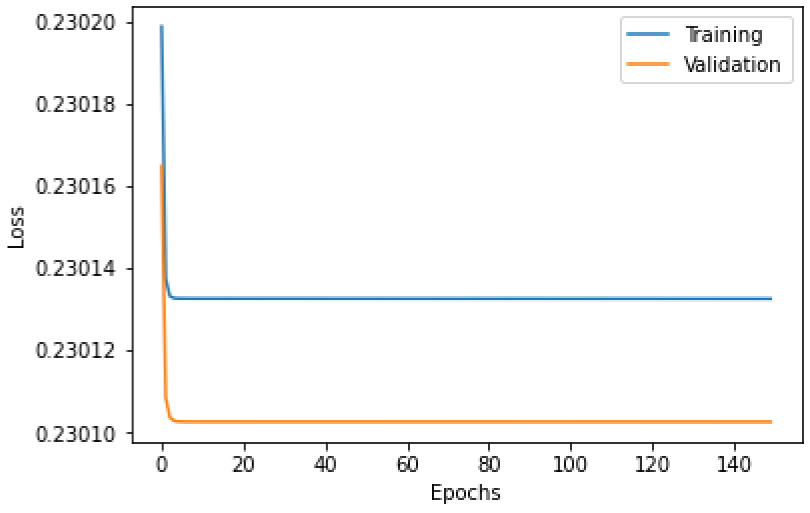
Loss v/s Epochs With Learning Rate = 0.08, Activation = ReLU , weight_init = normal



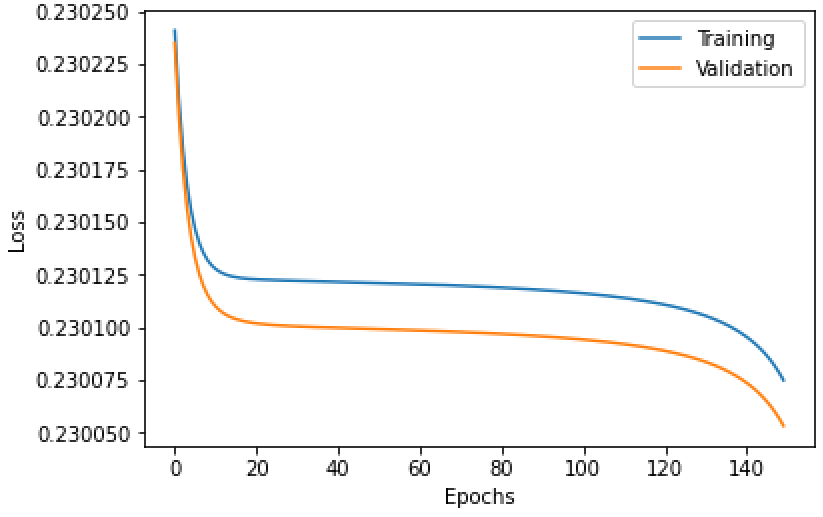
Loss v/s Epochs With Learning Rate = 0.08, Activation = Leaky ReLU , weight_init = normal

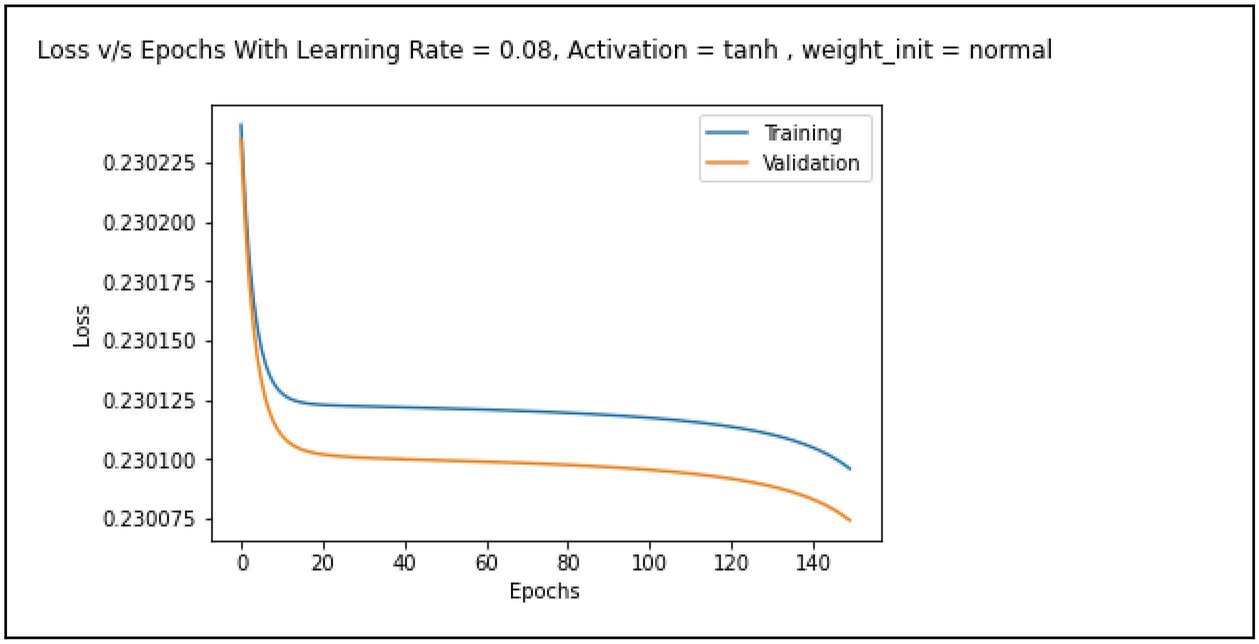


Loss v/s Epochs With Learning Rate = 0.08, Activation = sigmoid , weight_init = normal



Loss v/s Epochs With Learning Rate = 0.08, Activation = linear , weight_init = normal





Best Activation Function

For 150 epochs and learning rate 0.08, the performance is same on normal weights but on random weights it varies and tanh gives the best accuracy. Looking at the curves we see that tanh has lower loss around 150 epochs, hence it works the best in this case and is preferred over others here. It also converges fast as can be seen from the curve. Also in tanh the data is centered around zero so it performs better for mnist dataset and has lower loss.

Analysis and comparison

ReLU - The loss starts becoming constant around 20 epochs as can be seen from the train and validation v/s epochs curve.

Leaky ReLU - The loss starts becoming constant around 20 epochs similar to ReLU.

Sigmoid - The loss starts becoming constant around 2 epochs, much faster than ReLU and Leaky ReLU.

Linear- The loss starts becoming constant around 15 epochs and again starts decreasing around 130 epochs.

tanh - Starts converting around 15 epochs, has best accuracy on random weights for 150 epochs and 0.08 learning rate.

softmax - It is used as the activation function for the outer layer.

3

Softmax is used as the activation function for the outer layer. This is chosen as it is a multiclass classification problem. Softmax assigns decimal probabilities to each class in multi class classification problems hence it is used here. These probabilities are given by the output of predict_proba function.

4

With Normal Weights, Epochs = 150, Learning Rate = 0.08

Activation Function	Test Accuracy Using Sklearn	Test Accuracy Using Custom Implementation
ReLU	0.11814285714285715	0.11814285714285715
sigmoid	0.11814285714285715	0.11814285714285715
linear	0.9177142857142857	0.11814285714285715
tanh	0.9042857142857142	0.11814285714285715

Sklearn Model Parameters and analysis

The model parameters taken for sklearn MLP Classifier were

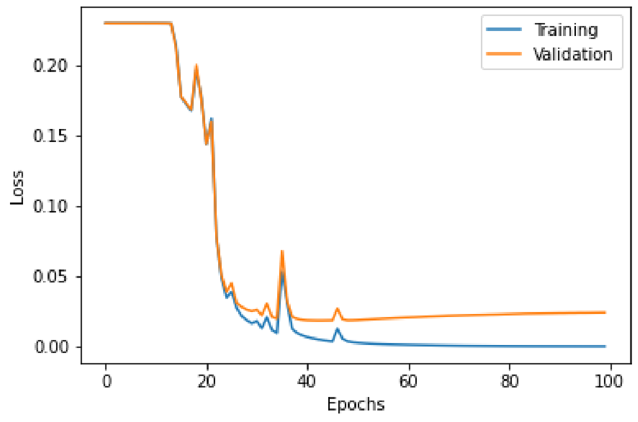
```
hidden_layer_sizes=(256,128,64,32),activation=func,learning_rate_init=0.08
,random_state=0,max_iter=150,batch_size=len(y_train)//20
```

The accuracy was the same as that obtained from sklearn for the activation functions ReLU and sigmoid and was different for linear and tanh. This difference in the values for linear and tanh activation functions is due to the difference in implementations of sklearn's MLPClassifier and our custom neural network. The inbuilt implementation uses adam solver which works well on large datasets like mnist whereas our custom implementation uses normal mini batch gradient descent. This leads to the difference in accuracy.

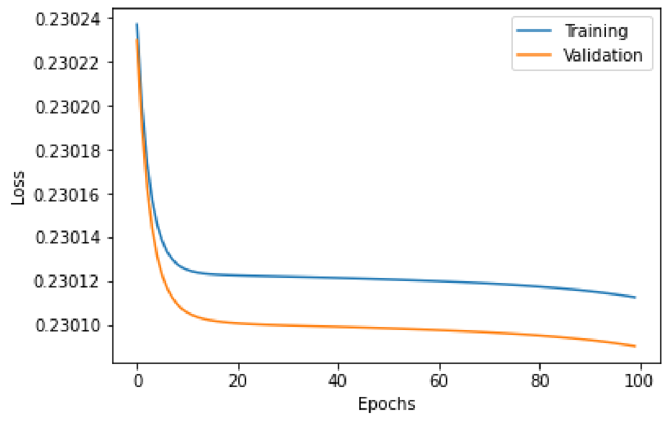
The difference in accuracy due to different activation functions is because in case of ReLU and sigmoid the model gets stuck during training. From the graphs in part 2 also we can see that in the case of ReLU and sigmoid the error becomes constant very soon hence the model gets stuck in a local minima. With activation functions tanh and linear the model is less likely to get stuck as the outputs for tanh are in the range (-1,1) whereas for sigmoid it is in the range (0,1). When the output value is near zero(sigmoid) it gets stuck.

5

The activation function chosen is 'tanh' and the number of epochs is 100. The model is trained with learning rates [0.001, 0.01, 0.1, 1.] and the following results were obtained

Learning Rate = 1. `Accuracy:  0.9574285714285714` Loss v/s Epochs With Learning Rate = 1.0, Activation = tanh , weight_init = normal 
Learning Rate = 0.1 `Accuracy:  0.11814285714285715`

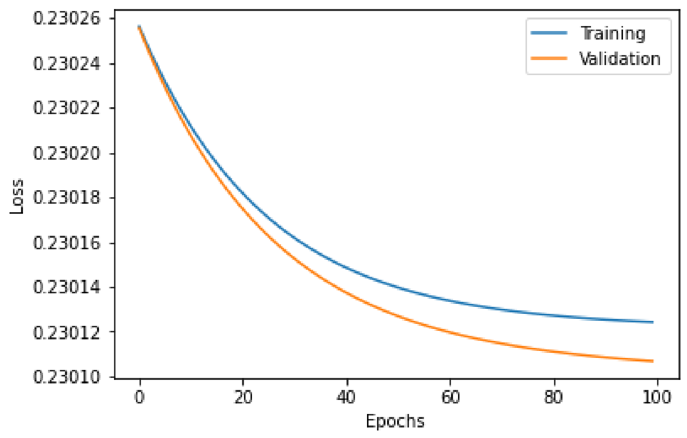
Loss v/s Epochs With Learning Rate = 0.1, Activation = tanh , weight_init = normal



Learning Rate = 0.01

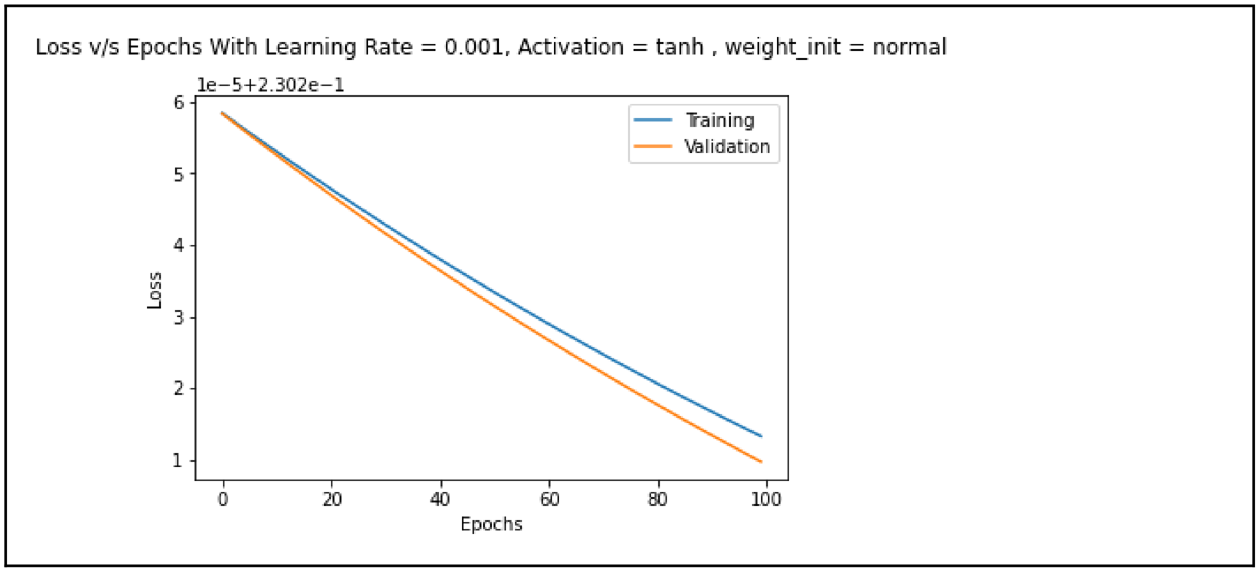
Accuracy: 0.11814285714285715

Loss v/s Epochs With Learning Rate = 0.01, Activation = tanh , weight_init = normal



Learning Rate = 0.001

Accuracy: 0.11814285714285715



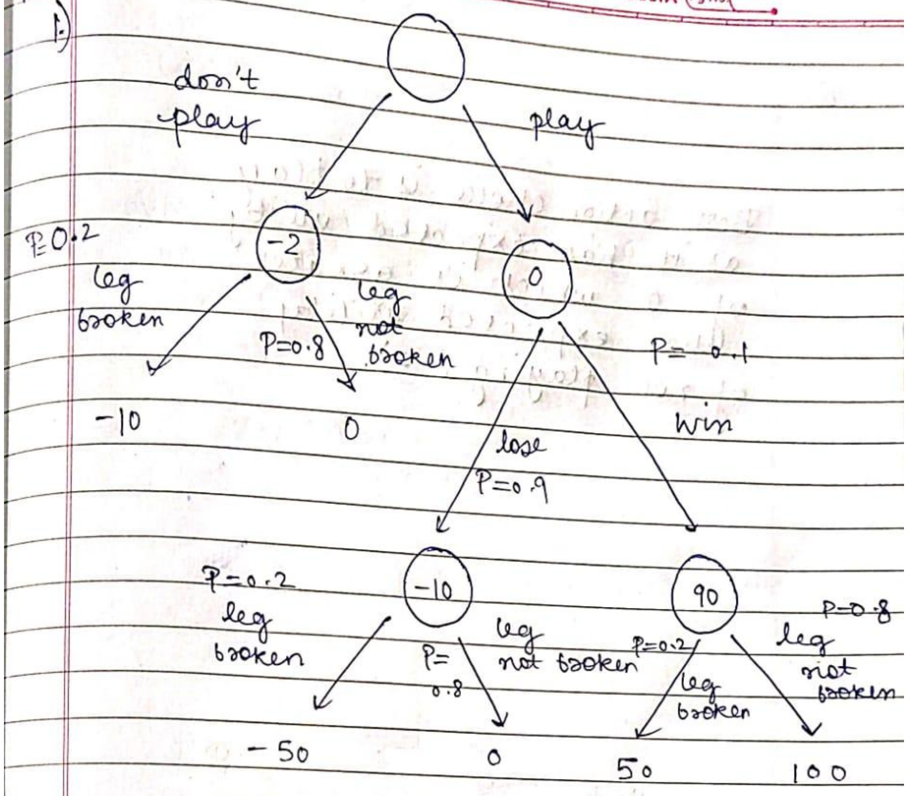
Analysis

The learning rate 0.1 performs the best as it has the highest accuracy of 95.7% and converges around 50 epochs. The accuracy for lower learning rates 0.1, 0.01 and 0.001 is low as the model is not able to converge properly in 150 epochs. Lower learning rates cause the gradient descent process to get stuck and the accuracy is only 11.8% for them as the model does not converge properly to a global minima.

3

1.

1.)



2.) EU (win) = 0.8 × 100 + 0.2 × 50

= 80 + 10

= 90

EU(lose) = 0.8 × 0 + 0.2 × (−50)

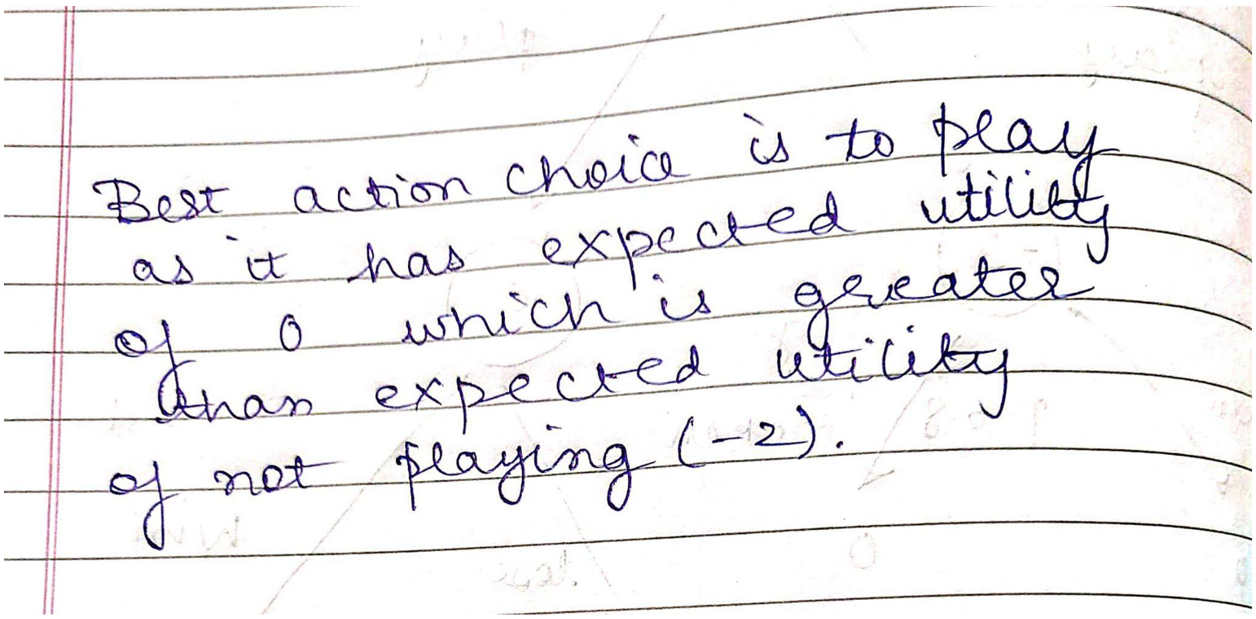
= −10

EU (don't play) = 0.2 × (−10) + 0.8 × 0

= −2

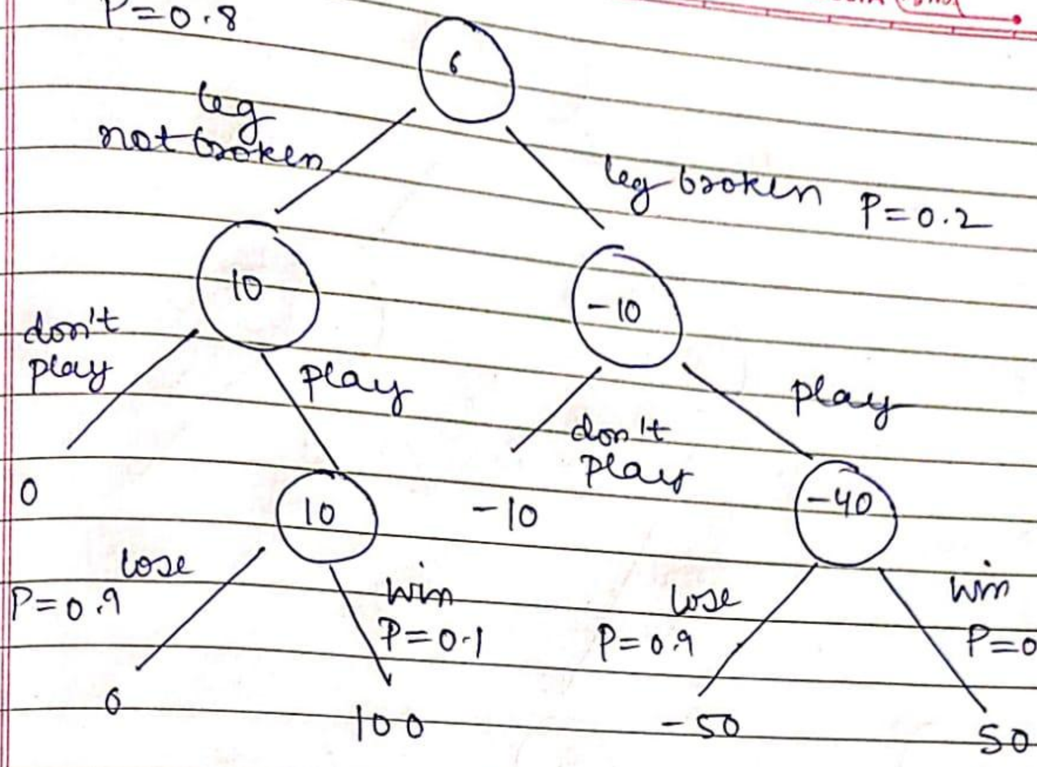
EU (play) = 0.1 × 90 + 0.9 (−10)

= 0



Best action choice is to play
as it has expected utility
of 0 which is greater
than expected utility
of not playing (-2).

3.

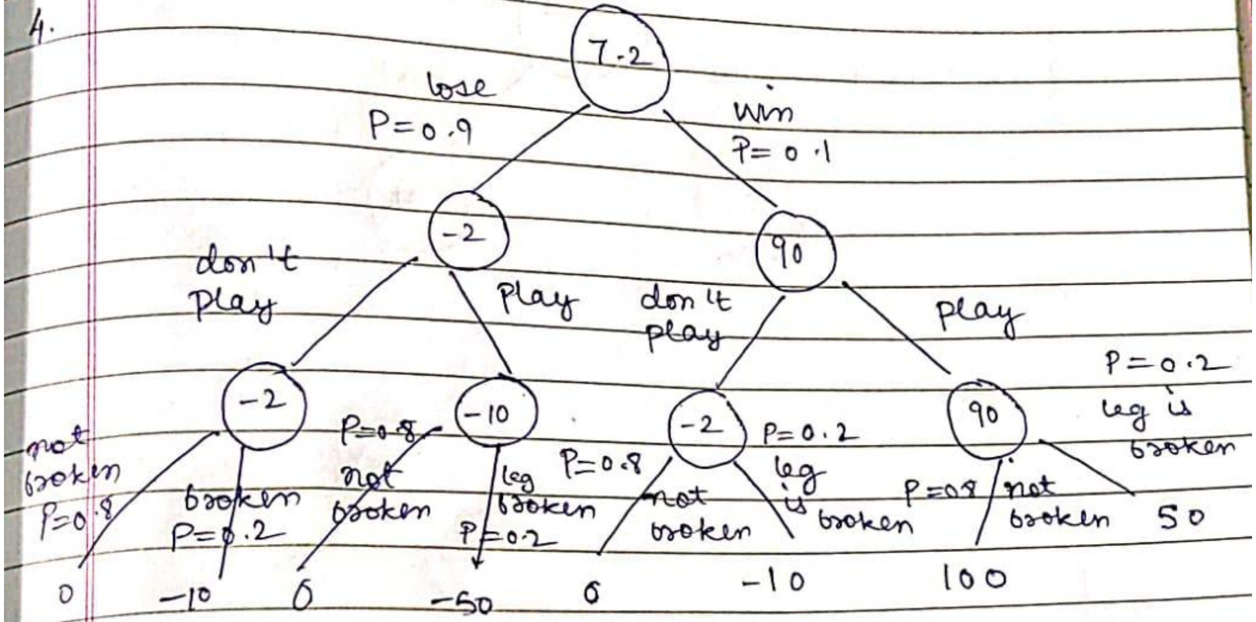


We get the above tree for perfect information about the state of leg.

The expected value of perfect information is given by $10 \times 0.8 + (-10) \times 0.2$
$= 6$

Date
DELTA Pg No

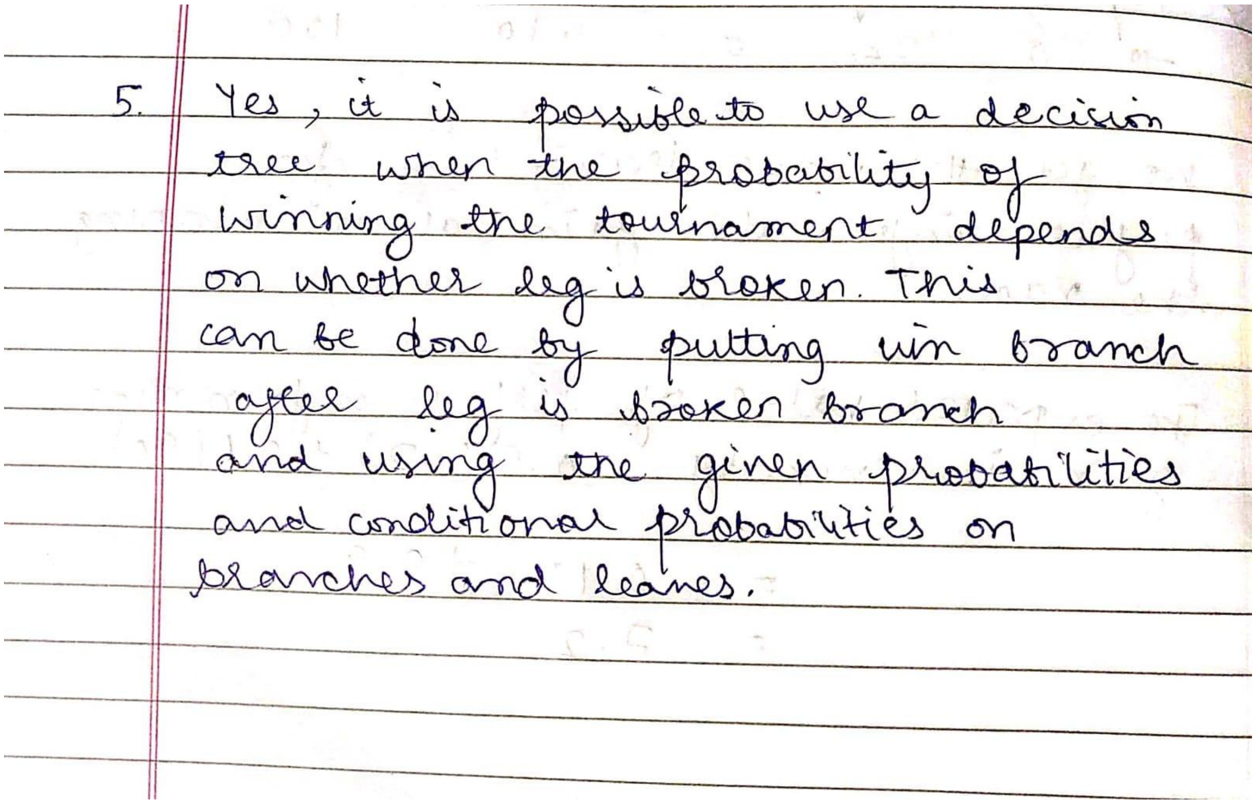
4.

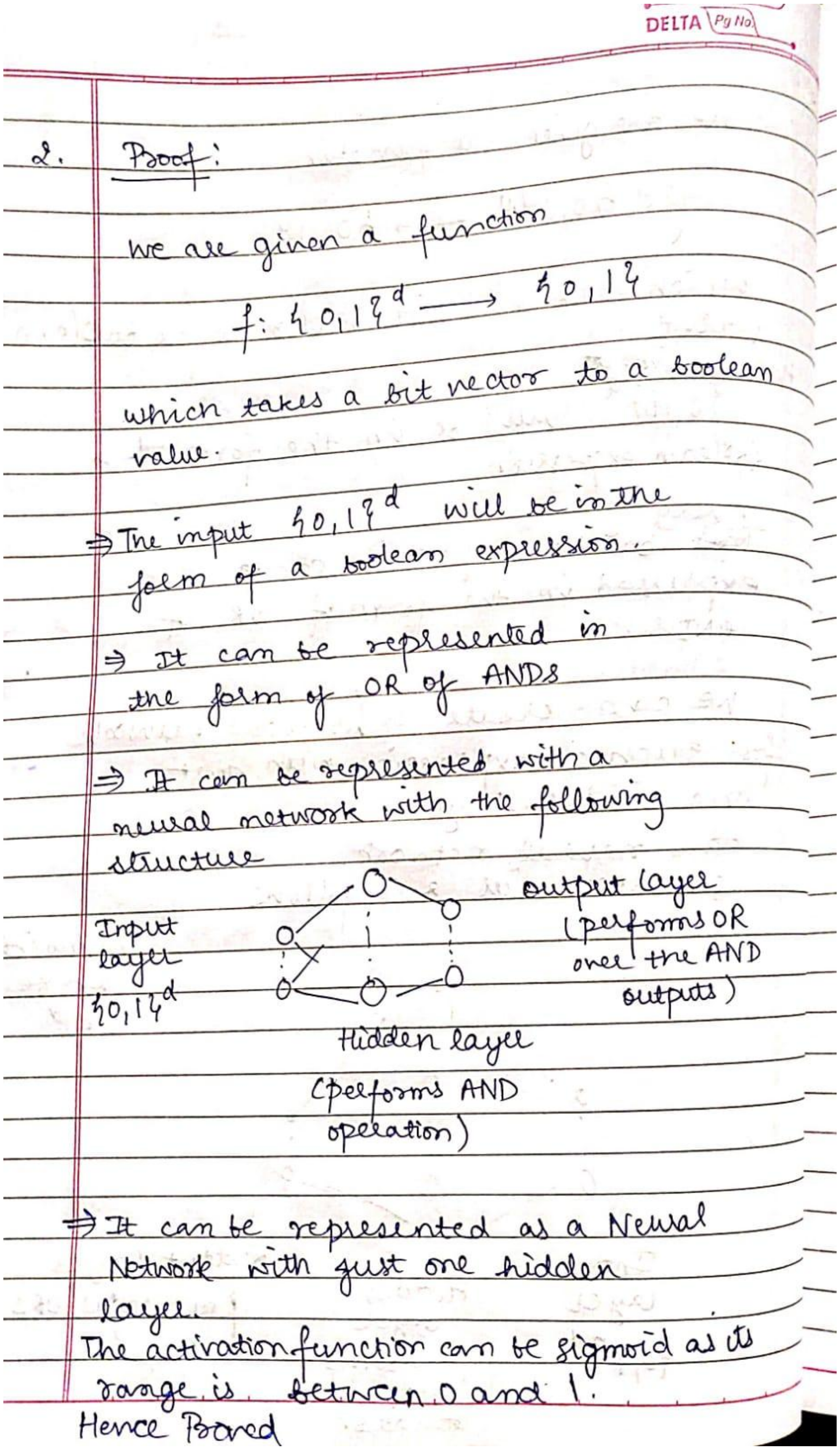


We get the above decision tree for perfect information gain about winning the race.

The expected value of perfect information gain is given by $0.9(-2) + 0.1(90)$

$$= -0.18 + 0.9$$
$$= 7.2$$


5. Yes, it is possible to use a decision tree when the probability of winning the tournament depends on whether leg is broken. This can be done by putting win branch after leg is broken branch and using the given probabilities and conditional probabilities on branches and leaves.



2. Proof:

We are given a function

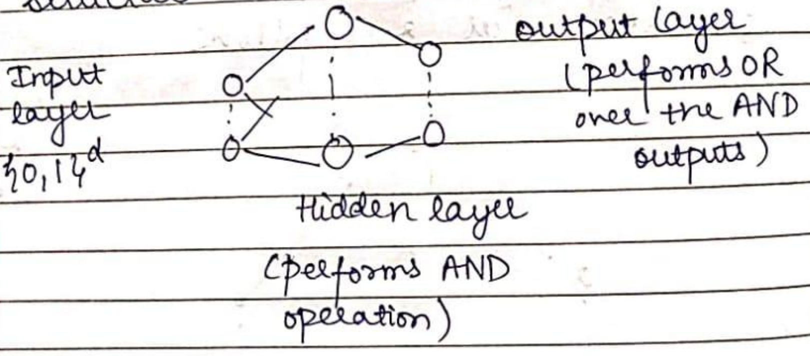
$$f : \{0,1\}^d \longrightarrow \{0,1\}$$

which takes a bit vector to a boolean value.

⇒ The input $\{0,1\}^d$ will be in the form of a boolean expression.

⇒ It can be represented in the form of OR of ANDs

⇒ It can be represented with a neural network with the following structure


Input layer $\{0,1\}^d$
output layer (performs OR over the AND outputs)
Hidden layer (performs AND operation)

⇒ It can be represented as a Neural Network with just one hidden layer.

The activation function can be sigmoid as its range is between 0 and 1.

Hence Proved

Date
DELTA Pg No.

3. The binary logistic regression classifier is given by:

$Y = \mathrm{argmax}_y \; P(Y = y \mid X)$

$$Y = P(Y=1 \mid X) = \frac{\exp(B_1X_1 + B_2X_2)}{1+\exp(B_1X_1+B_2X_2)}$$

$$Y = P(Y=-1 \mid X) = \frac{1}{1+\exp(B_1X_1+B_2X_2)}$$

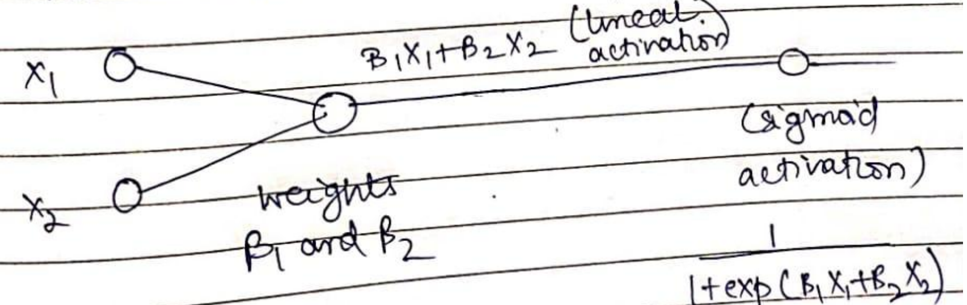
The neural network can be created as follows:

① we have X_1 and X_2 at the 2 input nodes.

② The weights of the hidden layer as B_1 and B_2.

③ The first hidden layer will have linear activation function. The output layer will have sigmoid activation function.

The neural network can be shown as.



X_1
X_2
$B_1X_1+B_2X_2$
(linear activation)
(sigmoid activation)
weights B_1 and B_2
$\frac{1}{1+\exp(B_1X_1+B_2X_2)}$