ASSIGNMENT 2

- Manvi Goel(2019472)

Question 1

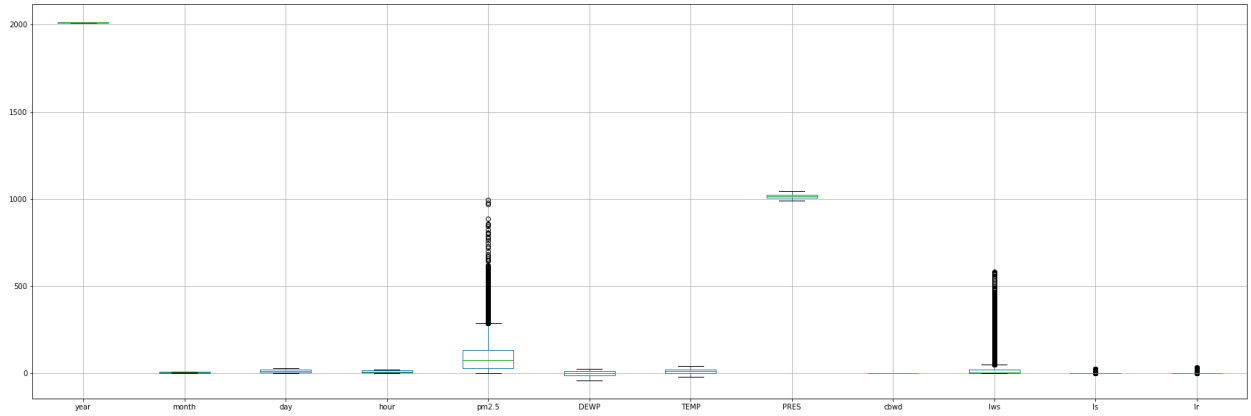
Preprocessing and Data Visualization

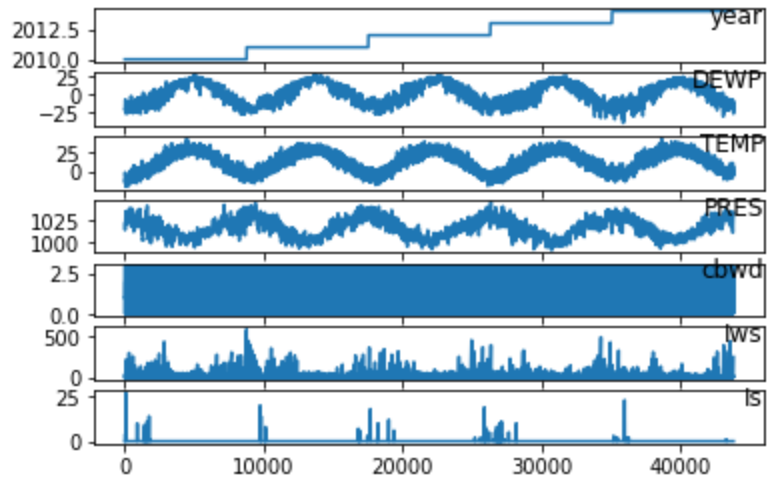
We first read the information of the data and data types and the description of mean, standard deviation.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43824 entries, 0 to 43823
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   No      43824 non-null  int64
 1   year    43824 non-null  int64
 2   month   43824 non-null  int64
 3   day     43824 non-null  int64
 4   hour    43824 non-null  int64
 5   pm2.5   41757 non-null  float64
 6   DEWP    43824 non-null  int64
 7   TEMP    43824 non-null  float64
 8   PRES    43824 non-null  float64
 9   cbwd    43824 non-null  object
 10  Iws     43824 non-null  float64
 11  Is      43824 non-null  int64
 12  Ir      43824 non-null  int64
dtypes: float64(4), int64(8), object(1)
memory usage: 4.3+ MB
```

Then we checked the missing values and changed the NaN values.

I also made box plots and others to visualise the data better.





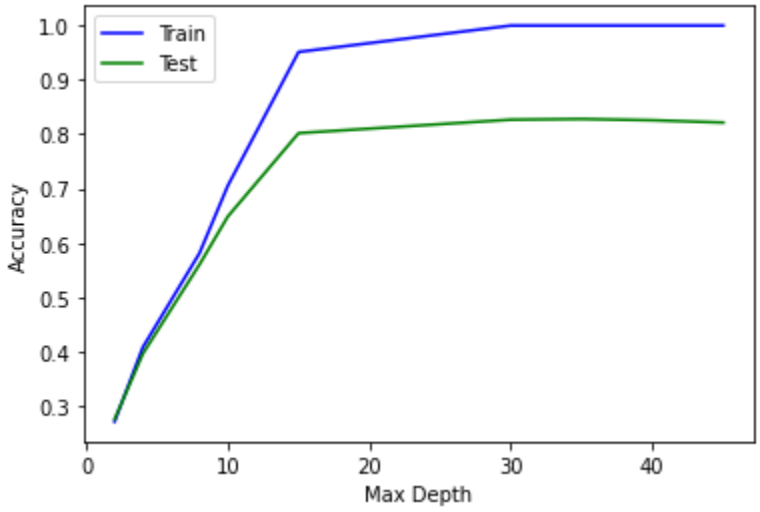
Encoded the values for cbwd.
And normalized the values.

Part a)
We see entropy gives better results.

```
The classifier giving better results is Entropy with mean accuracy =  0.8253726802555524
```

Thus, I use “Entropy” for the next parts.

Part b)
The best value comes out as 30. The accuracy stabilizes after this depth.

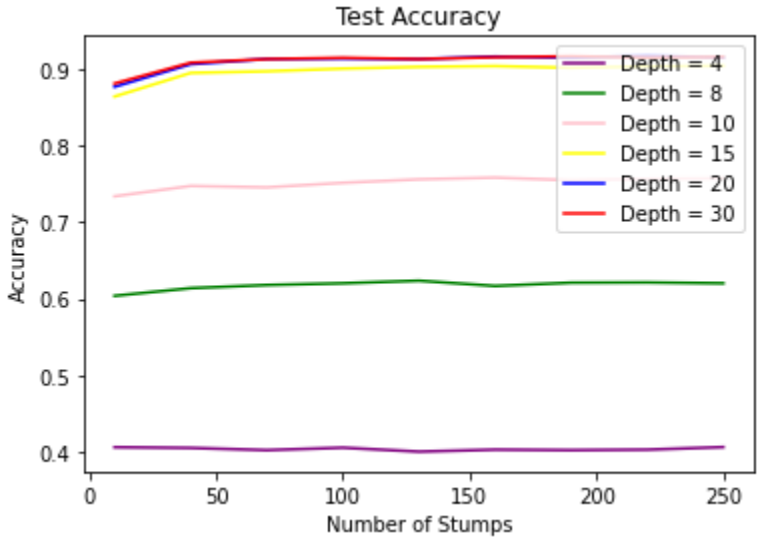


The actual depth of the classifier is: 26 (using the model history)

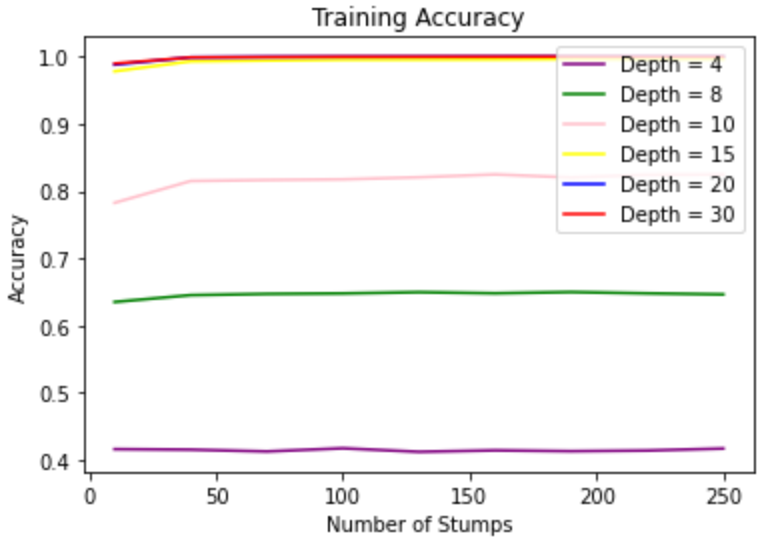
Part c)
The accuracy of the ensemble method is: 0.3576209309400669

The accuracy of the method cannot be compared to its counterpart models with better depths. But the ensembling raises the accuracy of one tree with the same depths. We can see the impact of ensembling the trees with this. But it cannot improve the accuracy to the level that we do not need complex models.

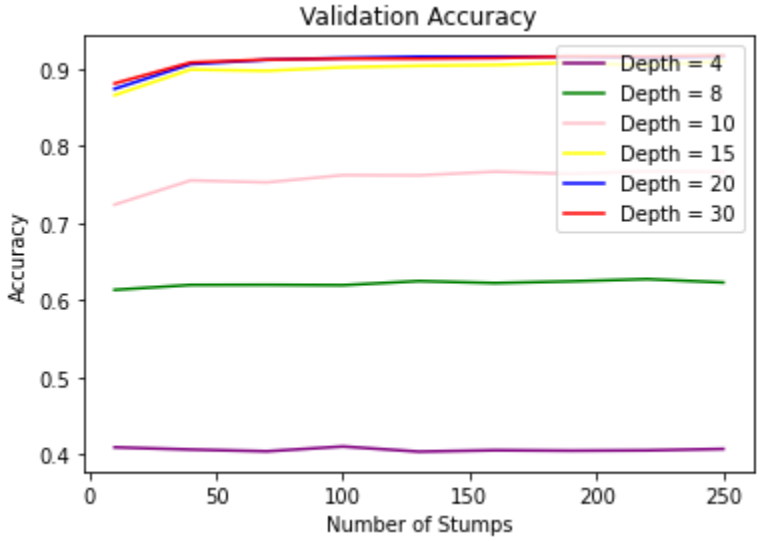
Part d)
Experimenting with the tree stumps and depth, I make plots for all the datasets.
The test data



The training data



And the validation set

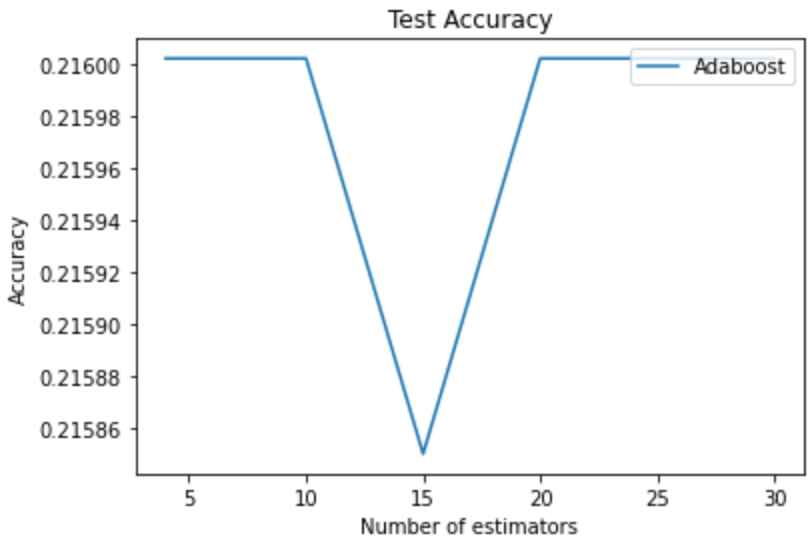


Combining the best results for the validation set.
depth = 30 and number of stumps = 70.

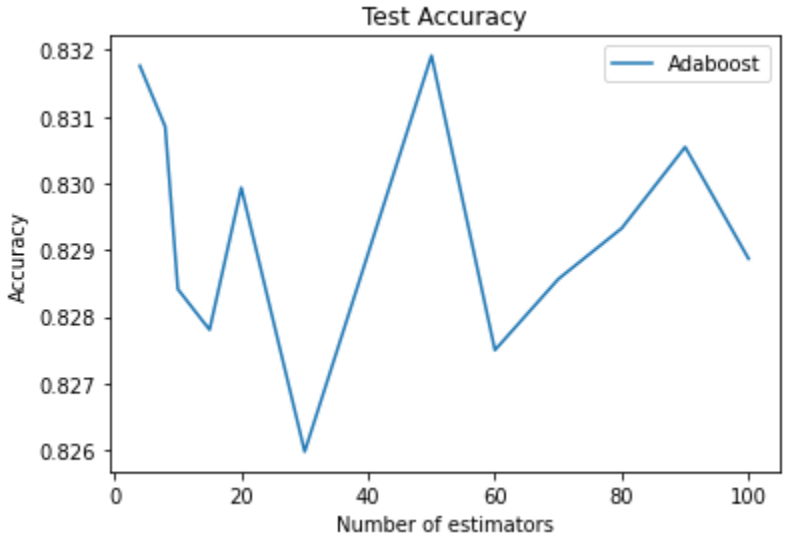
The accuracy of the ensemble method is: 0.9122999969577122

Part e)

The default max depth for the Adaboost classifier is 1. Plot for the same is



To get a better idea about the performance of Adaboost, I changed the max depth to 30. The plot for the same is:

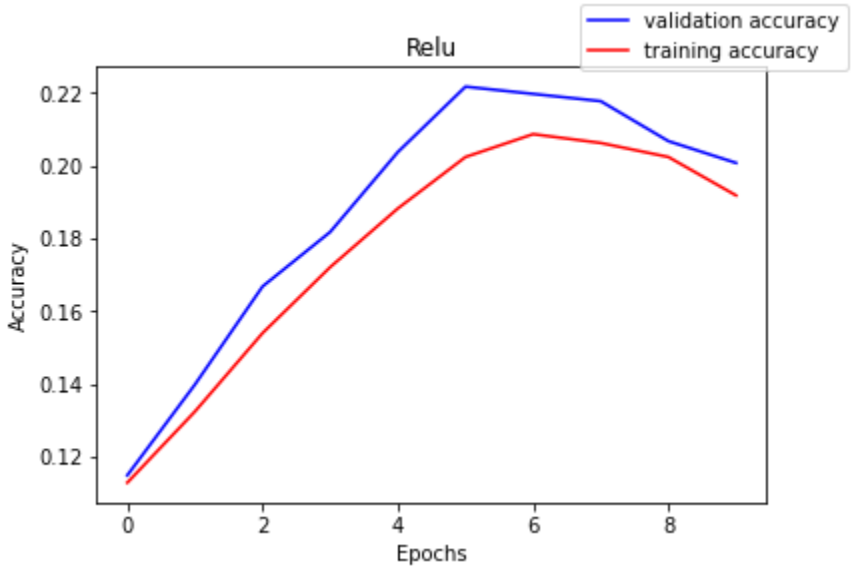


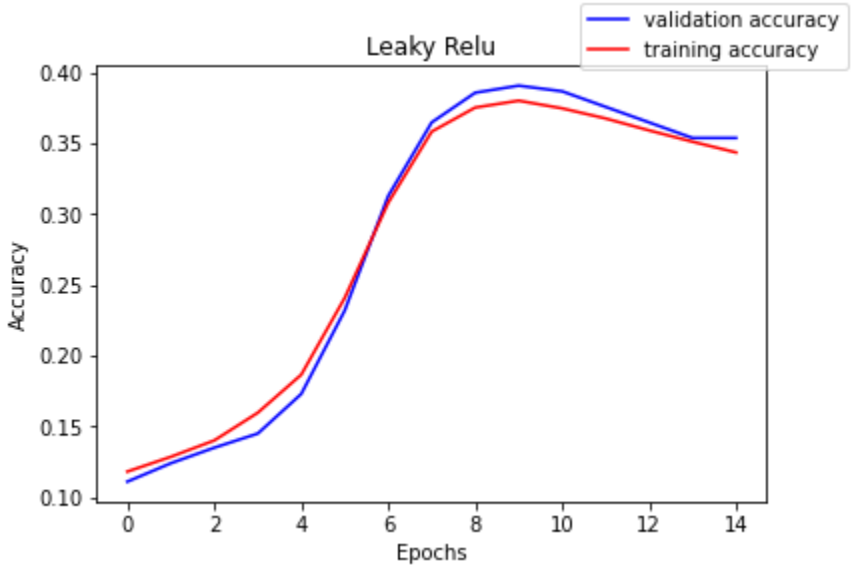
Comparison

We see that the accuracies are comparable in the range for the max depth. But even Adaboost can not resolve the problem of a weak model. The accuracy suggests the same. The sudden falls in the Adaboost model are not seen in Rf as the model is more robust.

//

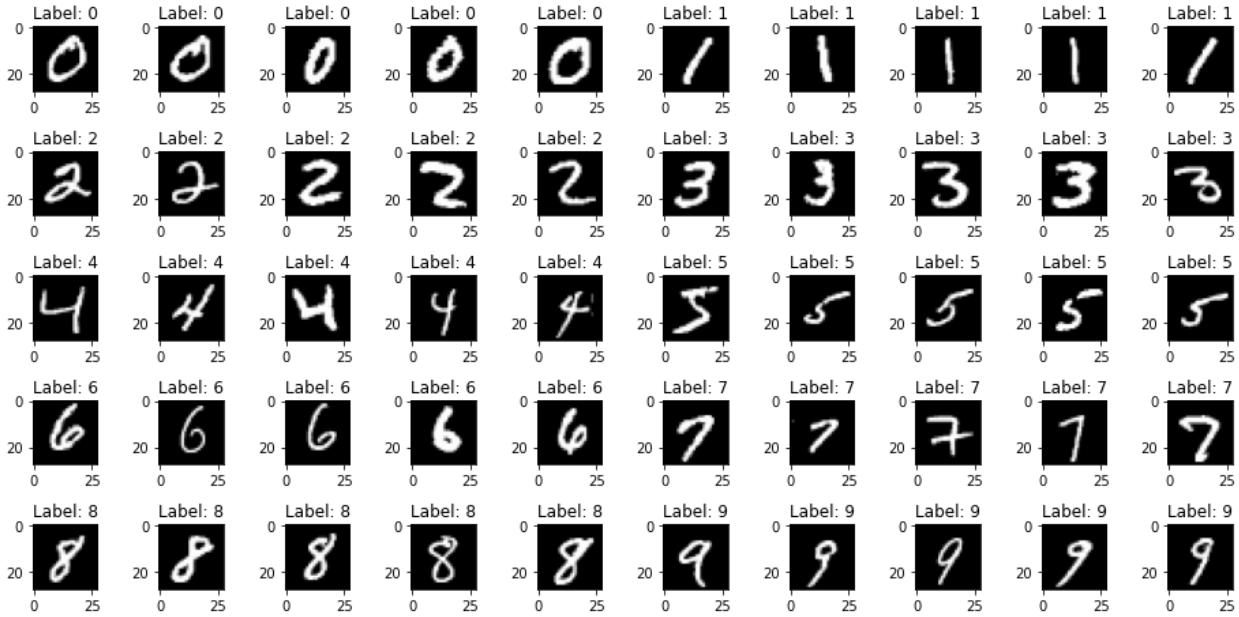
Question 2





Data Visualization and Preprocessing

Since the arrays represent numbers, I first visualized the arrays to see the digits for each class. Here we can also notice that the dataset has an equal number of samples for each class. Thus sampling and validation are not required.



We also notice that the images are in the format of 2-dimensional matrices with values ranging from 0 to 255. We normalized the images.
Since we are using a neural network, I flattened the array to a 1-dimensional array reshaping it with 784 values.

```
X = (X / 255) - 0.5
X = X.reshape((-1, 784))
X = np.array(X, dtype = np.float128)
y = np.array(y, dtype = np.float128)
```

Finally, we convert the label to one-hot encoding for training the model.

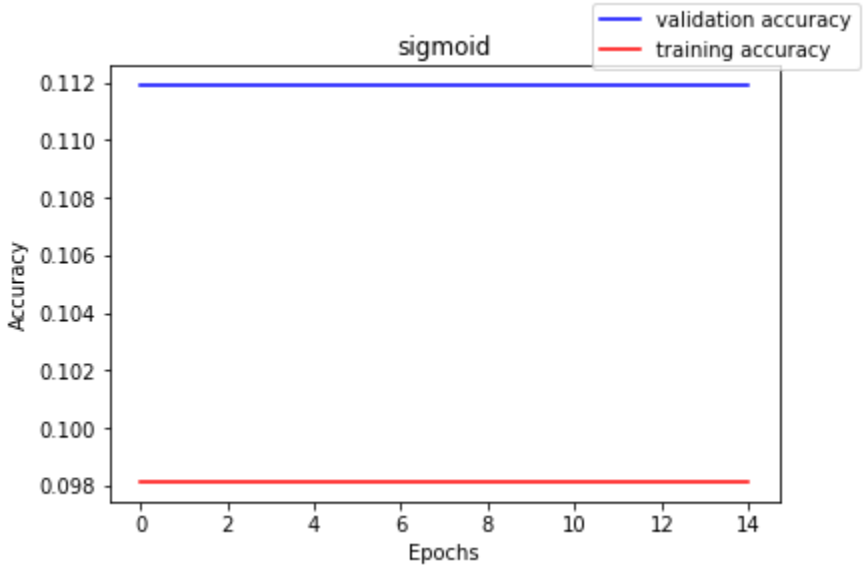
MyNeuralNetwork
The class implements all the required functions.

Part 1 and 2)

Sigmoid

Sigmoid suffers from a lack of complex computation and more computational complexity. Using exponent for data, even after smoothing results the data into giving constant NaN values. Due to an excess of NaN values, predictions suffer from constant accuracy.
This problem can be solved using expit from scipy library but is not used due to library constraints in the assignment.
The best accuracy is 19.86. It was constant across the number of epochs. Expectedly due to getting NaN values.

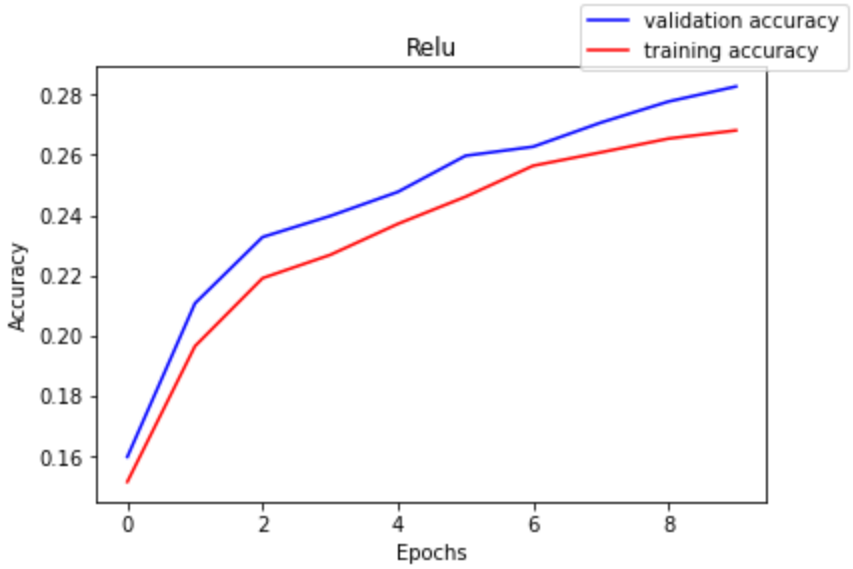
Plot.



Relu
Due to its simple implementation, my accuracy is highest for Relu. Reaching even its sklearn counterpart. It also shows steady upward performance.

The best accuracy for the model is 37%.

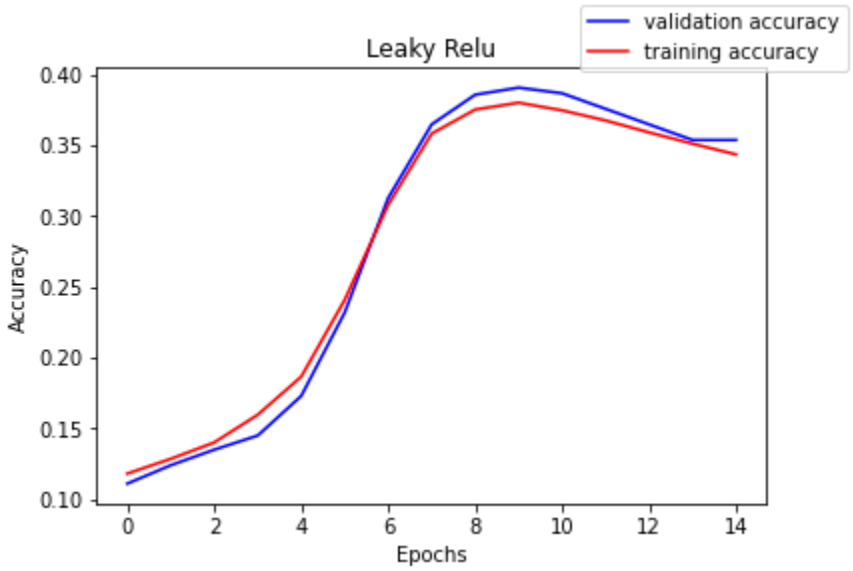
Plot



Leaky Relu

Leaky Relu also showed performance comparable to Relu. The overwhelming performance of both models can be attributed to the smoothing done.
The best accuracy for the test set is 35.36%

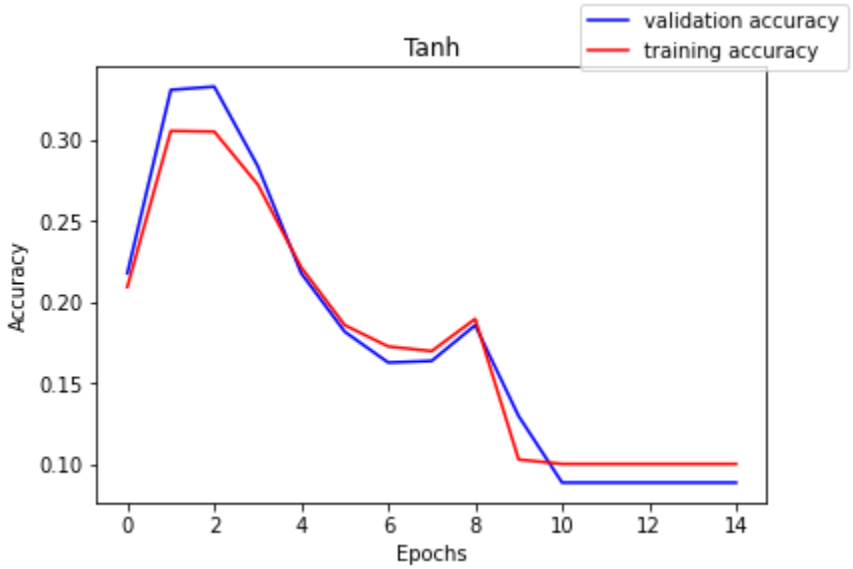
Plot



TanH

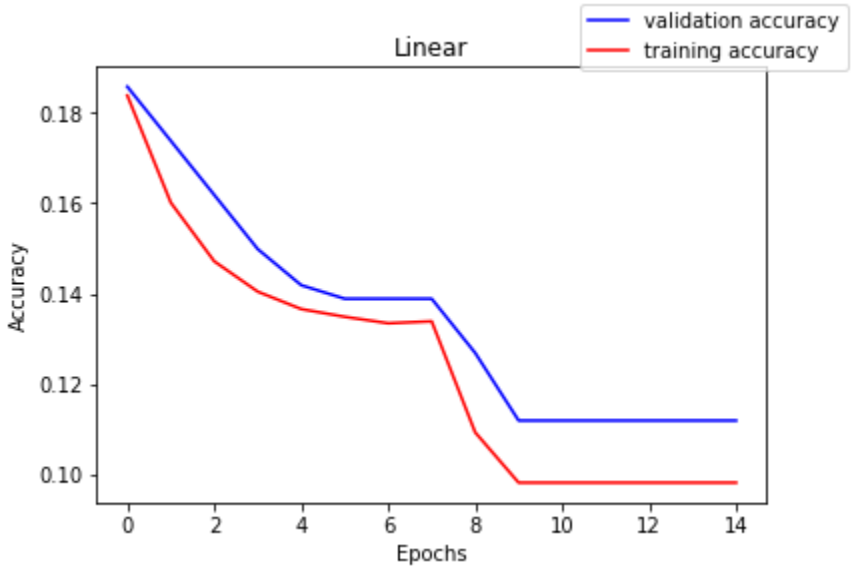
Tanh shows a drop in accuracy overall. This can be attributed to the same problem as sigmoid, as the data increasingly becomes null we lose the capacity to predict and improve. Even the sklearn model does not perform well.
The best accuracy for the model is 11.30%.

Plot



Linear

We see a small rise as the value starts to stabilize after oscillating but falls in the same problems as others. It shows a good start compared to the other and could have performed better.
The best performance is 17.05%.



3) The output layer can only use softmax activation functions for each case. Since we are classifying the array into 10 digits it is necessary to use an output layer that calculates the probability for each class, so we can assign the class with the maximum probability.

4) Using Sklearn Library.

The accuracy for Sigmoid is 0.8534895349667834.
Comparison: The definition of sigmoid function plays an important role in this case. We see a stagnance in our accuracy when compared to Sklearn, this is due to the matrices turning NaN and not getting the correct output.
This issue is resolved by using the *expit* function of scipy library that involves a much complex implementation without sacrificing the computational resources.

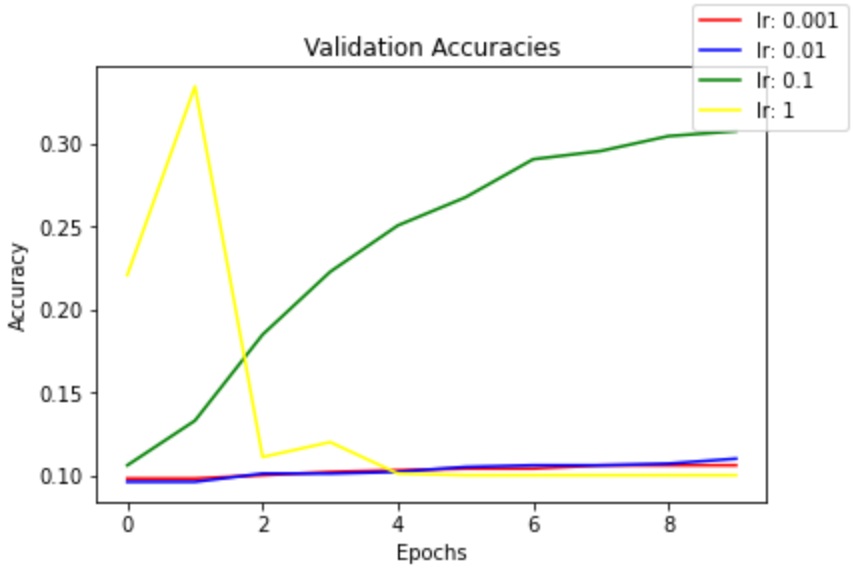
The accuracy for Relu is 0.6281877276948353.
Comparison: We notice a similarity in this accuracy. We can see an accuracy of around 50 percent in my implementation. This is due to the easy implementation of Relu. But the low accuracy can be attributed to the low computational resources and softmax reaching NaN values.

The accuracy for Linear is 0.867490535038217
Comparison: We notice an upward trend in the accuracy of the scratch model. But due to the low computational resources and softmax reaching NaN values, the model stops improving. Thus the sklearn model and scratch model show differences. If the model could have performed for longer, we should see comparable results.

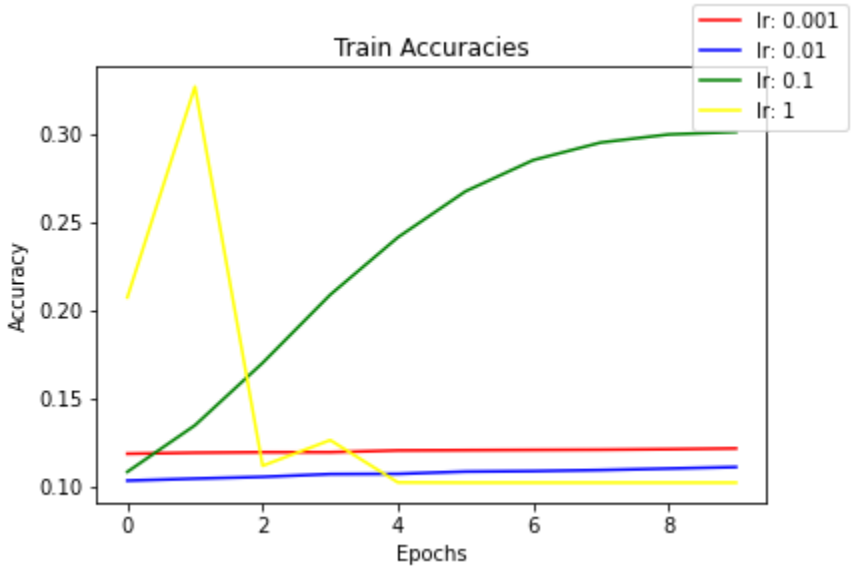
The accuracy for tanh is 0.5015358239874277.
Comparison: The accuracy for tanH is comparable. The model shows the lowest accuracy out of all. That is to say, that the model is suited for this dataset.

5) Using different learning rates.
I tried multiple learning rates to find the most optimal one.

The plot for validation accuracies



The plot for train accuracies across epochs.



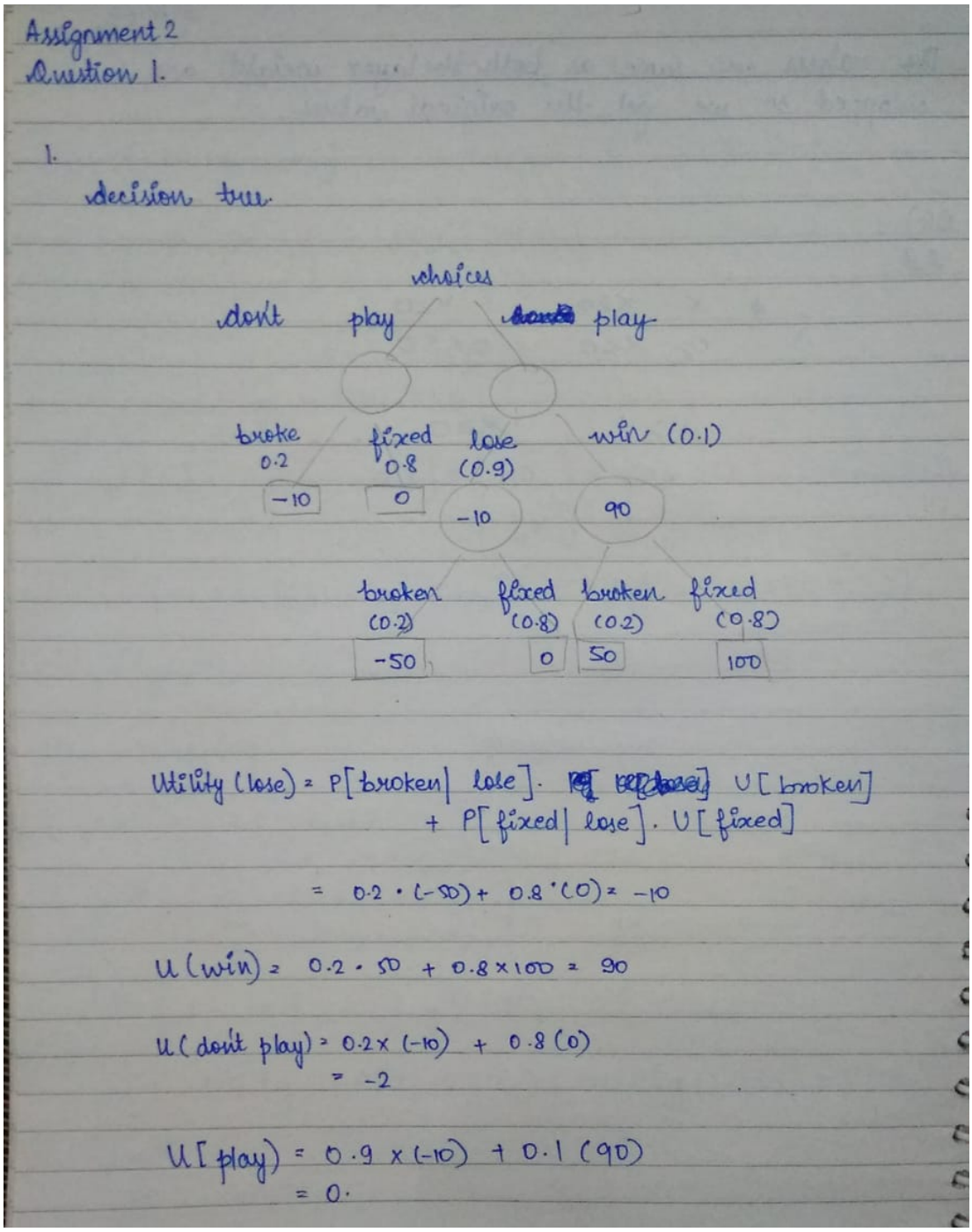
The test accuracies across different learning rates.

Analysis

We see that the best accuracies are given by 0.1 which shows a steady increase and the highest test set accuracy. 1 is too high and ends up losing the minima. For others, the values are increasing but are too slow.

Theory Questions

Question 1



Assignment 2
Question 1.

1.

decision tree.

choices

don't play | play

broke 0.2 | fixed 0.8 | lose (0.9) | win (0.1)

−10 | 0 | −10 | 90

broken (0.2) | fixed (0.8) | broken (0.2) | fixed (0.8)

−50 | 0 | 50 | 100

Utility (lose) = P[broken | lose] . U[broken] + P[fixed | lose] . U[fixed]

= 0.2 · (−50) + 0.8 · (0) = −10

U(win) = 0.2 · 50 + 0.8 × 100 = 90

U(don't play) = 0.2 × (−10) + 0.8 (0)
= −2

U[play) = 0.9 × (−10) + 0.1 (90)
= 0.

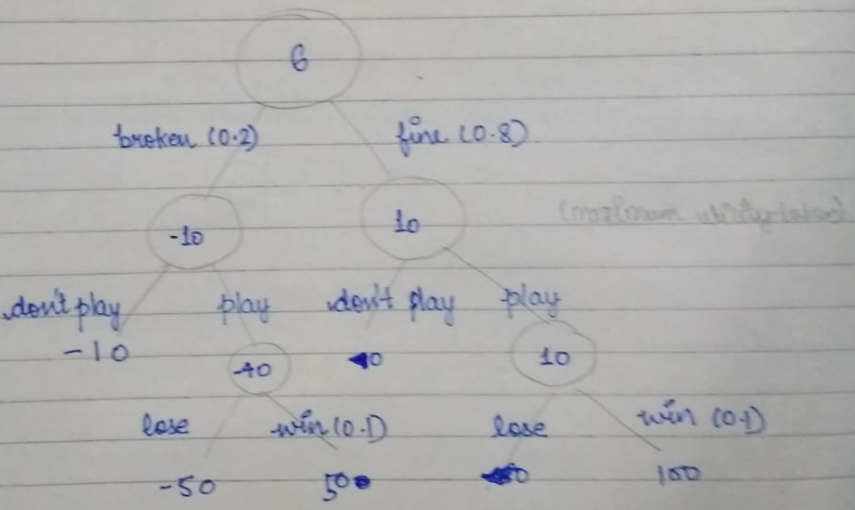
since the utility for play is greater than don't play, we suggest her to play.

2.

The utility for play is 0 is preffered over don't play with utility -2.

3.

To compute the value of perfect information about the state of leg we need to make a tree with the question.



Making the tree using the formula.

Expected value = (Perfect information) - U(No information)

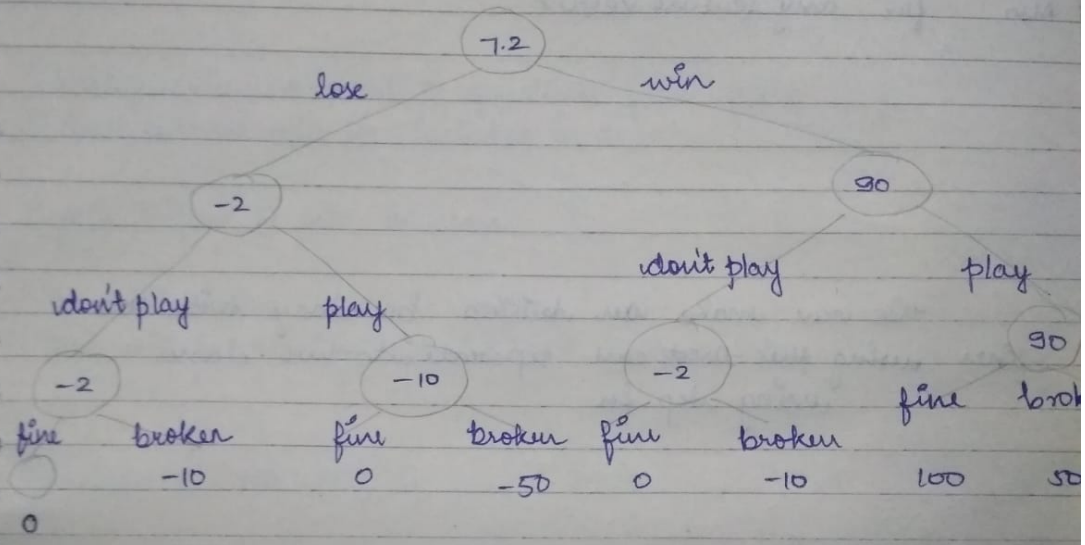
= 6-0

= 6.

The expected value about the perfect information about lg is 6.

4.

Finding EC (perfect information)



Calculating the

Expected value of perfect information = 7.2 - 0 = 7.2.

5.

Yes, we can use the conditional probability in the original tree. We can use the tree to classify.

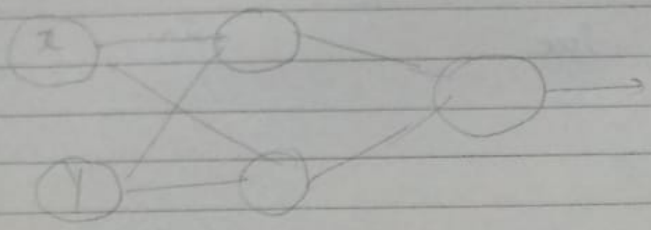
We need to change the values of P(win/lose) and we can make another tree.

Question 2

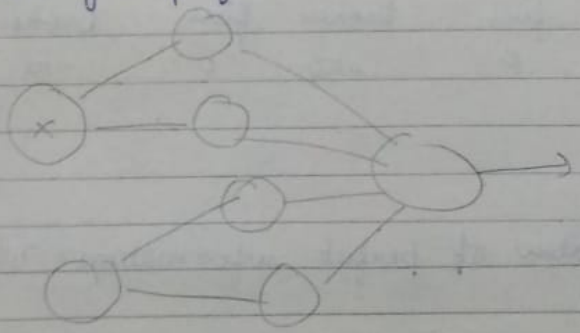
Ques 2

We can classify all functions using a combinations of logic gates that can be represented in one hidden layer.

Also for any feature vector

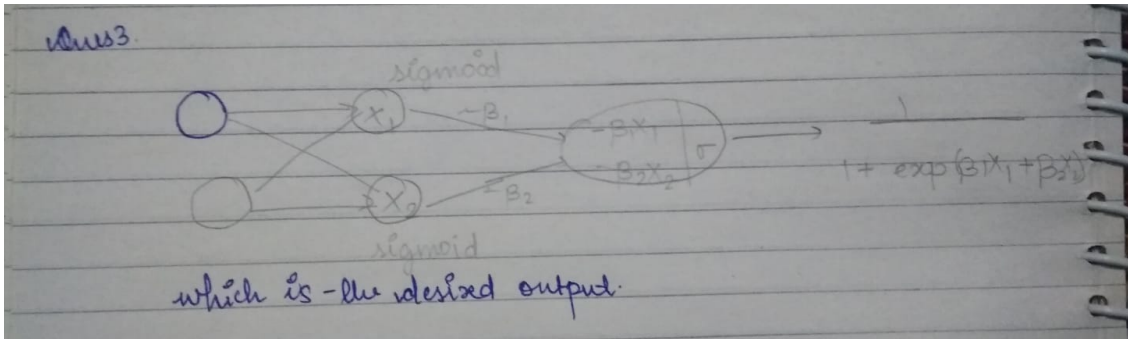


We can make any decision boundary with two lines using this that can separate into two classes using step fn



We can increase the length of hidden layer to represent any function

Question 3



Weights for the output layer are - (beta1) and - beta2
And the activation function is sigmoid.