Q1)

Preprocessing:

```
data=pd.read_csv('PRSA_data_2010.1.1-2014.12.31.csv')
data=data.sample(frac=1)   #random shuffling
```

```
In [38]: data.isna().sum().sum()

Out[38]: 2067
```

Removing nan rows and using one-hot encoding for cbwd labels.

```
In [39]: data = data.dropna()
         data.head()

         one_hot=pd.get_dummies(data['cbwd'])
         data=data.drop('cbwd',axis = 1)
         data=data.join(one_hot)
         data=data.drop(['No'],axis=1) #dropping number of rows
```

Train test split:

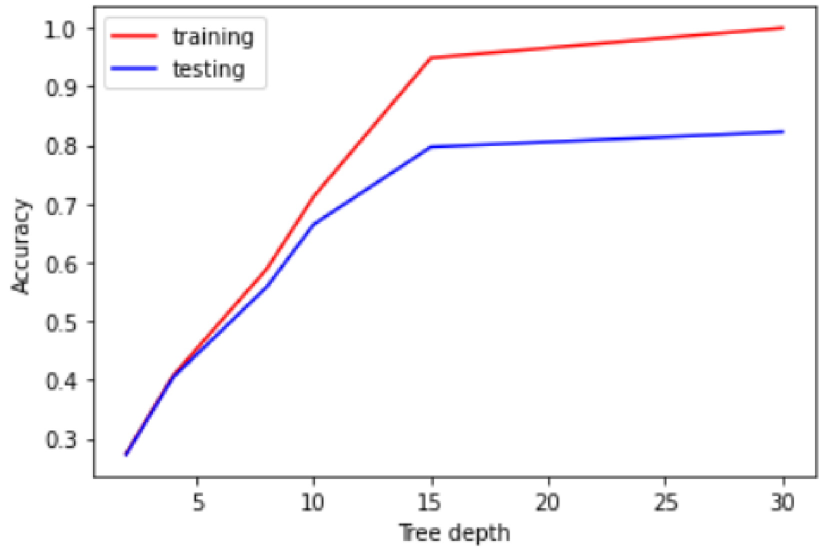
```
In [41]: def train_val_test_split(data,labels, train,val,test):
             """ a function that will get dataset and training dataset fraction as input and return x_train, x_test, y_train, y_test """

             print("Total length is "+str(len(data)))

             train_samples=len(data)*train//(train+test+val)
             val_samples=len(data)*val//(train+test+val)


             train_data=data[:train_samples]
             train_labels=labels[:train_samples]

             val_data=data[train_samples+1:train_samples+val_samples+1]
             val_labels=labels[train_samples+1:train_samples+val_samples+1]

             test_data=data[train_samples+val_samples:]
             test_labels=labels[train_samples+val_samples:]

             return train_data,train_labels,val_data,val_labels,test_data,test_labels
```

Part a) Entropy gives a better accuracy

```
Accuracy using gini index is 0.8215482841181165
```

```
Accuracy using entropy is 0.8233040702314446
```

Part b)



As depth increases the training accuracy increases and goes to 100 but the testing accuracy does not increase thus depicting over fitting after 15 depth of trees.

Part c)

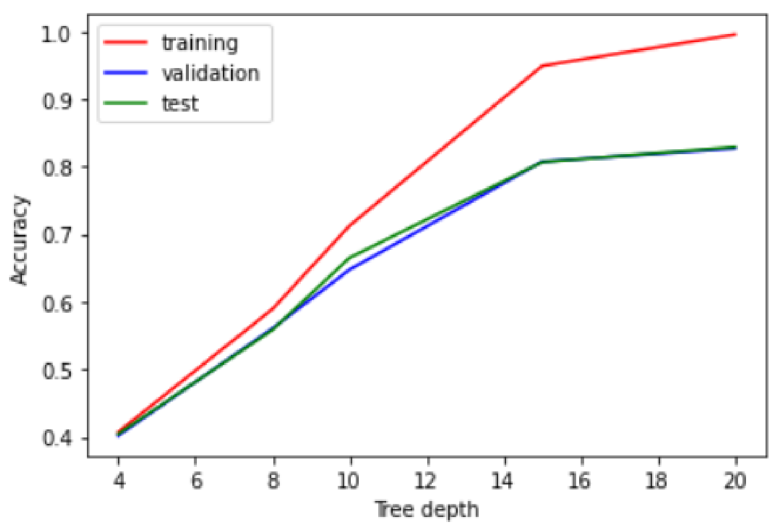
```
def max_vote(predictions):
    final_prediction=[]
    for j in range(len(predictions[0])):
        maxi={}
        for i in predictions:
            if(i[j] in maxi):
                maxi[i[j]]+=1
            else:
                maxi[i[j]]=1
        max_key=max(maxi, key= lambda x: maxi[x])
        final_prediction.append(max_key)

    return final_prediction
```

```
Accuracy obtained is 0.3599361532322426
```

The accuracy is very less because the stumps are weak learners and have depth only equal to 3 which is very less as compared to parts a and b.

Part d)

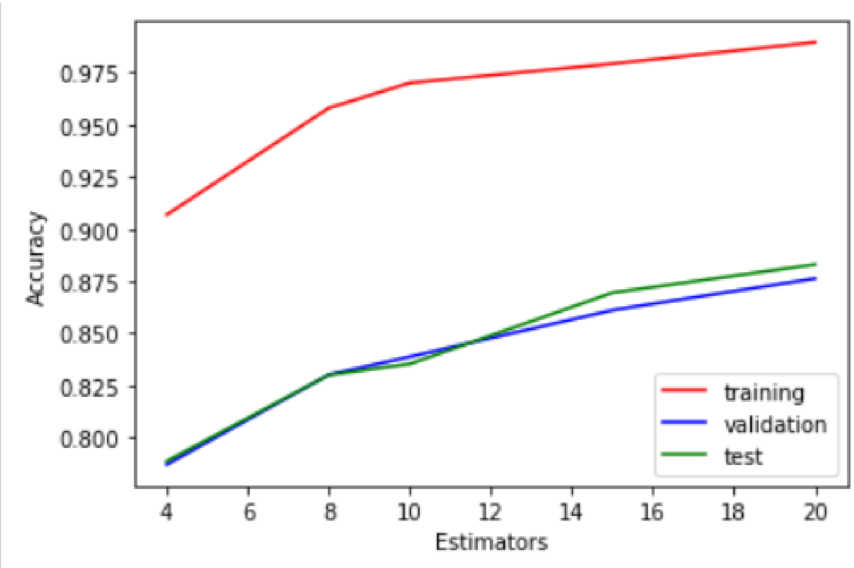
```
train_accuracies=[]
val_accuracies=[]
test_accuracies=[]

depth=[4,8,10,15,20]
for d in depth:
    stumps=[]
    predictions=[]
    val_predictions=[]
    test_predictions=[]
    for i in range(100):
        stumps.append(DecisionTreeClassifier(criterion="entropy",max_depth=d))
        x_train_frac=x_train.sample(frac=0.5)
        y_train_frac=y_train.loc[x_train_frac.index]
        stumps[i].fit(x_train,y_train)

        predictions.append(stumps[i].predict(x_train))
        val_predictions.append(stumps[i].predict(x_val))
        test_predictions.append(stumps[i].predict(x_test))

    final_prediction=max_vote(predictions)
    final_val_prediction=max_vote(val_predictions)
    final_test_prediction=max_vote(test_predictions)

    accuracy=np.sum(np.array(final_prediction)==np.array(y_train.to_list()))/len(y_train)
    val_accuracy=np.sum(np.array(final_val_prediction)==np.array(y_val.to_list()))/len(y_val)
    test_accuracy=np.sum(np.array(final_test_prediction)==np.array(y_test.to_list()))/len(y_test)
```



e)

```
estimators = [4, 8, 10, 15, 20]
train_accuracies=[]
val_accuracies=[]
test_accuracies=[]

for e in estimators:
    model=AdaBoostClassifier(n_estimators=e,base_estimator=DecisionTreeClassifier(criterion="entropy",max_depth=11))
    model.fit(x_train, y_train)
    train_accuracies.append(model.score(x_train, y_train))
    val_accuracies.append(model.score(x_val, y_val))
    test_accuracies.append(model.score(x_test , y_test))
```

Using max depth as 11 as after that overfitting starts and the training accuracy reaches 1 and testing/validation does not increase.



Q2)

Train test split:

```
In [5]: def train_val_test_split(data,labels, train,val,test):
            """ a function that will get dataset and training dataset fraction as input and return x_train, x_test, y_train, y_test """

            print("Total length is "+str(len(data)))

            train_samples=len(data)*train//(train+test+val)
            val_samples=len(data)*val//(train+test+val)

            train_data=data[:train_samples]
            train_labels=labels[:train_samples]

            val_data=data[train_samples+1:train_samples+val_samples+1]
            val_labels=labels[train_samples+1:train_samples+val_samples+1]

            test_data=data[train_samples+val_samples:]
            test_labels=labels[train_samples+val_samples:]

            return train_data,train_labels,val_data,val_labels,test_data,test_labels
```

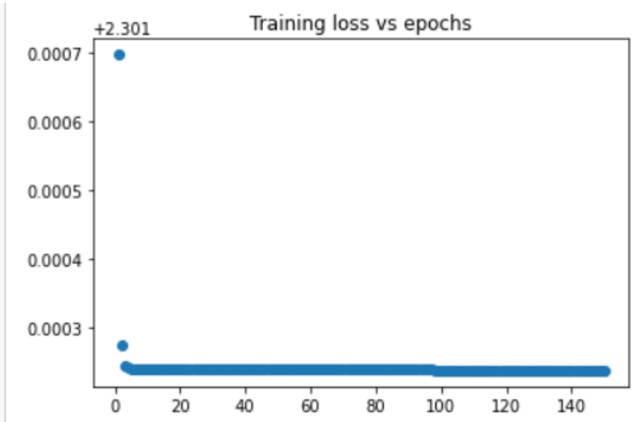
```
In [6]: x_train,y_train,x_val,y_val,x_test,y_test=train_val_test_split(data,labels,7,2,1)

        Total length is 70000
```

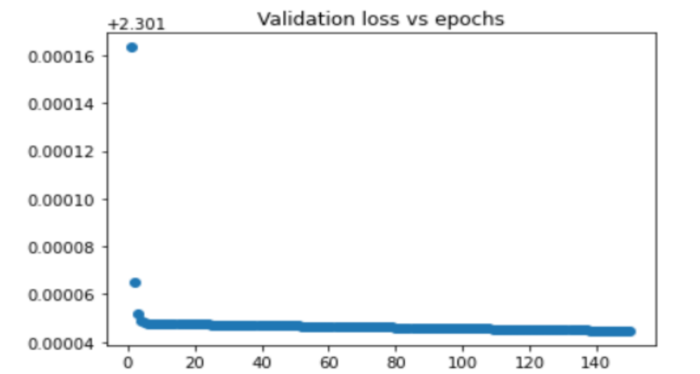
Preprocessing:

```
In [3]: 
        #pre processing
        standardscalar = StandardScaler()

        train_images = standardscalar.fit_transform(train_images)
        test_images = standardscalar.transform(test_images)

        data=[]
        labels=[]
        for i in range(len(train_images)):
            data.append(train_images[i])
            labels.append(train_labels[i])
        for i in range(len(test_images)):
            data.append(test_images[i])
            labels.append(test_labels[i])

        data=np.array(data)
        labels=np.array(labels)

        #random shuffling
        data,labels=shuffle(data,labels)

        #dividing by max value to convert every pixel to 0-1
        data=data/255
```

Scaling, division by max, random shuffling.

1) Training on model with the given parameters






```
Epoch 0
pred
Training Loss of epoch 0 is 2.3016974269042163
Validation Loss of epoch 0 is 2.3011633919801975
Epoch 1
pred
Training Loss of epoch 1 is 2.3012752655681212
Validation Loss of epoch 1 is 2.3010651037206378
Epoch 2
pred
Training Loss of epoch 2 is 2.3012457294966806
Validation Loss of epoch 2 is 2.301051576994632
Epoch 3
pred
Training Loss of epoch 3 is 2.3012421140364245
Validation Loss of epoch 3 is 2.3010487240603332
```

```
Training Loss of epoch 125 is 2.301238957322712
Validation Loss of epoch 125 is 2.3010451915143446
Epoch 126
pred
Training Loss of epoch 126 is 2.301238938842364
Validation Loss of epoch 126 is 2.3010451711310815
Epoch 127
pred
Training Loss of epoch 127 is 2.301238920363488
Validation Loss of epoch 127 is 2.3010451507650056
Epoch 128
```

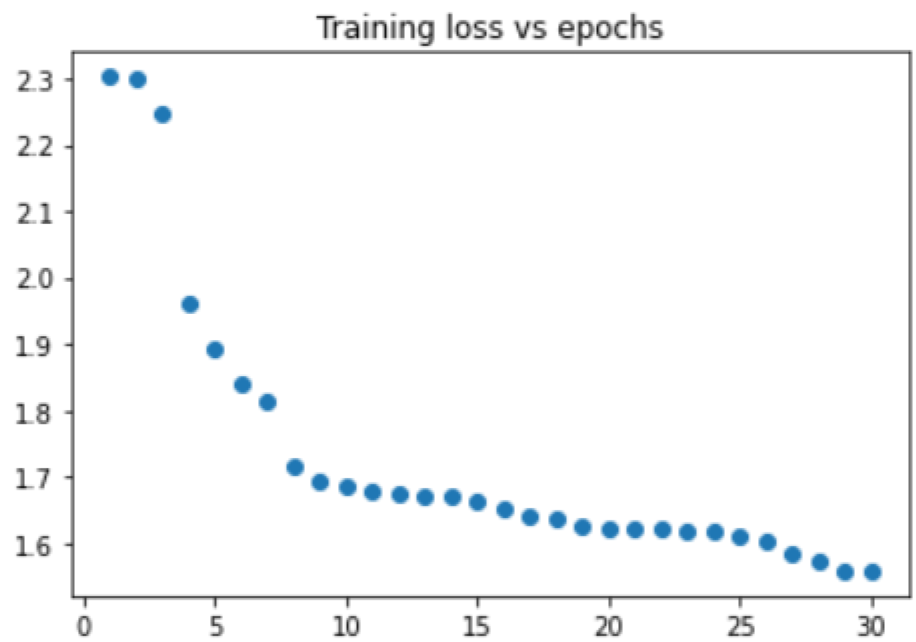
```
In [51]: net.score(test_images,test_labels)
         pred
Out[51]: 0.1135
```

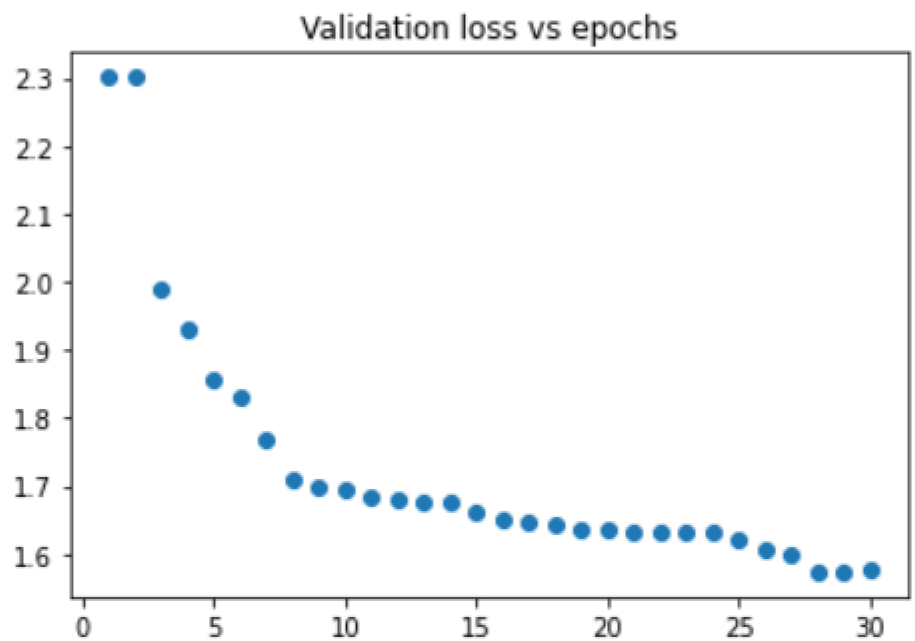
On training with the given parameters, the model is getting stuck on local minima and is unable to get out of it. On sklearn also the same is happening and even the loss is the same and so is the accuracy.

```
In [9]: clf = MLPClassifier(hidden_layer_sizes=(256,128,64,32),n_iter_no_change=5,random_state=1, max_iter=80, verbose=True,learning_rate
Iteration 1, loss = 2.50394541
Validation score: 0.111000
Iteration 2, loss = 2.31330875
Validation score: 0.097833
Iteration 3, loss = 2.31152429
Validation score: 0.111000
Iteration 4, loss = 2.31120904
Validation score: 0.099833
Iteration 5, loss = 2.30965051
Validation score: 0.103000
Iteration 6, loss = 2.30902260
Validation score: 0.108333
Iteration 7, loss = 2.30881308
Validation score: 0.108333
Validation score did not improve more than tol=0.000100 for 5 consecutive epochs. Stopping.

In [10]: clf.score(test_images,test_labels)
Out[10]: 0.1028
```

On training model on optimal parameters:

```
In [53]: net=MyNeuralNetwork(6,[784,256,128,64,32,10],'sigmoid',0.5,'normal',64,30)
```





Using early stopping as validation was increasing after 30 epochs

Accuracy:

```
In [40]: net.score(test_images,test_labels)

pred

<ipython-input-37-3f285724f306>:43: RuntimeWarning: overflow encountered in exp
  arr=1/(1+np.exp(-arr))

Out[40]: 0.9608
```

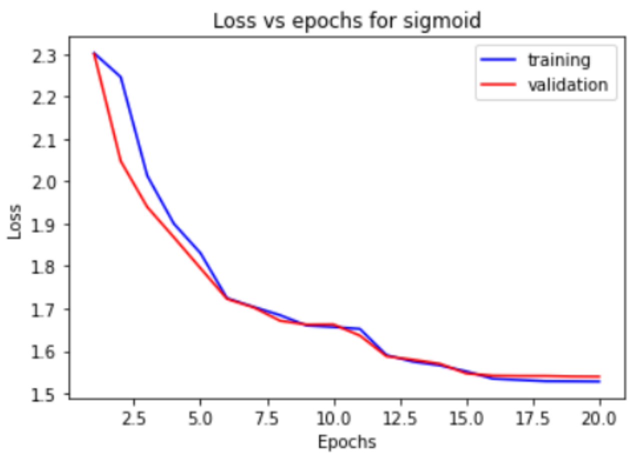
Similarly, on sklearn:

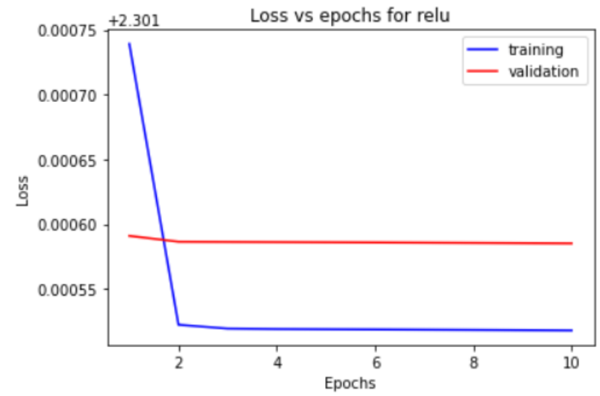
```
In [8]: clf = MLPClassifier(hidden_layer_sizes=(256,128,64,32),random_state=1, max_iter=20,batch_size=64,verbose=True)
```

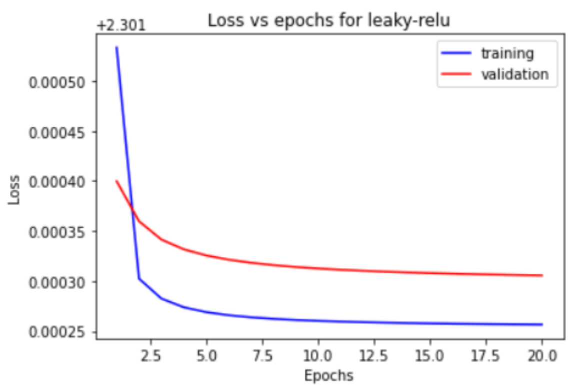
```
In [9]: clf.score(test_images,test_labels)

Out[9]: 0.9779
```

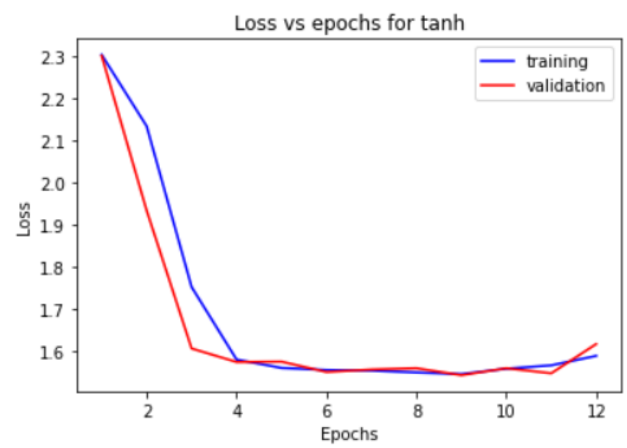
2)

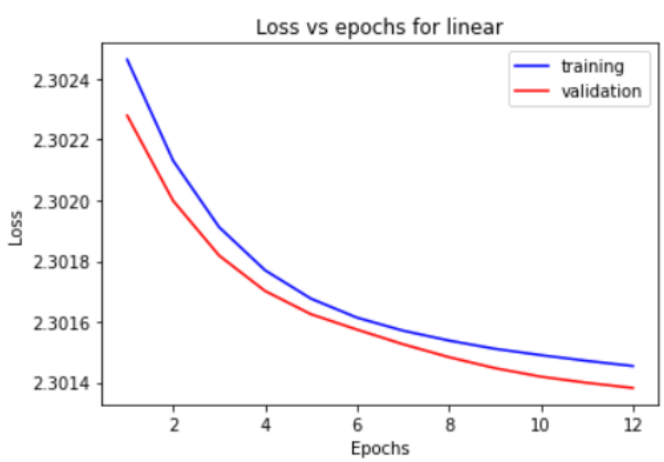
Plots for different activations:











Sigmoid and tanh perform well and converge fast.
Relu, leaky relu and linear get stuck in local minima as they are all subsets of linear activations which does not seem enough.

Hence I would prefer to use sigmoid/tanh for the problem,

Softmax is now used as an activation function for any layer but it is used as activation for last layer to convert the numbers to probabilities so that we can use them to calculate cross-entropy loss.
Also derivative of softmax is jacobian (2d).

3)
Softmax is used as the last layer of output activation function always because softmax is now used as an activation function for any layer but it is used as activation for last layer to convert the numbers to probabilities so that we can use them to calculate cross-entropy loss.

```
def forward_propagation(self):

    for l in range(1,self.n_layers):
        self.Z[l]=np.dot(self.A[l-1],self.W[l])
        if(l!=1):
            self.Z[l]+=self.B[l]

        self.A[l]=self.activate(self.Z[l],self.activation)

    #applying softmax on last layer as loss is cross entropy and we might end up taking log 0
    self.A[l]=self.activate(self.A[l],'softmax')
```

4)

Relu:

```
clf = MLPClassifier(learning_rate_init=0.2, activation='relu', max_iter=15,batch_size=64, verbose=T
```

Accuracy is 10%

```
Iteration 1, loss = 2.88967008
Validation score: 0.108333
Iteration 2, loss = 2.89126643
Validation score: 0.108333
Iteration 3, loss = 2.76392370
Validation score: 0.093667
Iteration 4, loss = 2.65377754
Validation score: 0.099833
Iteration 5, loss = 2.55203340
Validation score: 0.108333
Iteration 6, loss = 2.46759257
Validation score: 0.100000
Iteration 7, loss = 2.40689655
Validation score: 0.093667
Validation score did not improve more than tol=0.000100 for 5 consecutive epochs. Stopping.
```

Tanh:

```
In [11]: clf = MLPClassifier(hidden_layer_sizes=(256,128,64,32),activation='tanh', max_iter=20,batch_size=64,verbose=True).fit(train_image
```

```
Iteration 1, loss = 0.26495654
Iteration 2, loss = 0.11164164
Iteration 3, loss = 0.06664266
Iteration 4, loss = 0.04610161
Iteration 5, loss = 0.03416363
Iteration 6, loss = 0.03631448
Iteration 7, loss = 0.02803791
Iteration 8, loss = 0.02324141
Iteration 9, loss = 0.02364274
Iteration 10, loss = 0.01982537
Iteration 11, loss = 0.02040567
Iteration 12, loss = 0.01754388
Iteration 13, loss = 0.01626972
Iteration 14, loss = 0.01691518
Iteration 15, loss = 0.01873534
Iteration 16, loss = 0.01224891
Iteration 17, loss = 0.01642907
Iteration 18, loss = 0.01817790
Iteration 19, loss = 0.01625237
Iteration 20, loss = 0.01502028
```

```
C:\Users\Bhavya\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582: ConvergenceWarning: Stochasti
c Optimizer: Maximum iterations (20) reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
In [12]: clf.score(test_images,test_labels)
```

```
Out[12]: 0.9686
```

Identity (linear):

```
In [14]: clf = MLPClassifier(hidden_layer_sizes=(256,128,64,32),activation='identity', max_iter=20,batch_size=64,verbose=True).fit(train_i

Iteration 1, loss = 0.45975616
Iteration 2, loss = 0.34711756
Iteration 3, loss = 0.33014121
Iteration 4, loss = 0.32372233
Iteration 5, loss = 0.31605756
Iteration 6, loss = 0.30871312
Iteration 7, loss = 0.30492682
Iteration 8, loss = 0.29895897
Iteration 9, loss = 0.29270644
Iteration 10, loss = 0.29013802
Iteration 11, loss = 0.28783339
Iteration 12, loss = 0.28032053
Iteration 13, loss = 0.28101321
Iteration 14, loss = 0.27814602
Iteration 15, loss = 0.27819989
Iteration 16, loss = 0.27406415
Iteration 17, loss = 0.26971380
Iteration 18, loss = 0.26991537
Iteration 19, loss = 0.27061519
Iteration 20, loss = 0.26669716

C:\Users\Bhavya\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582: ConvergenceWarning: Stochasti
c Optimizer: Maximum iterations (20) reached and the optimization hasn't converged yet.
  warnings.warn(

In [15]: clf.score(test_images,test_labels)

Out[15]: 0.9188
```

Logistic(sigmoid):

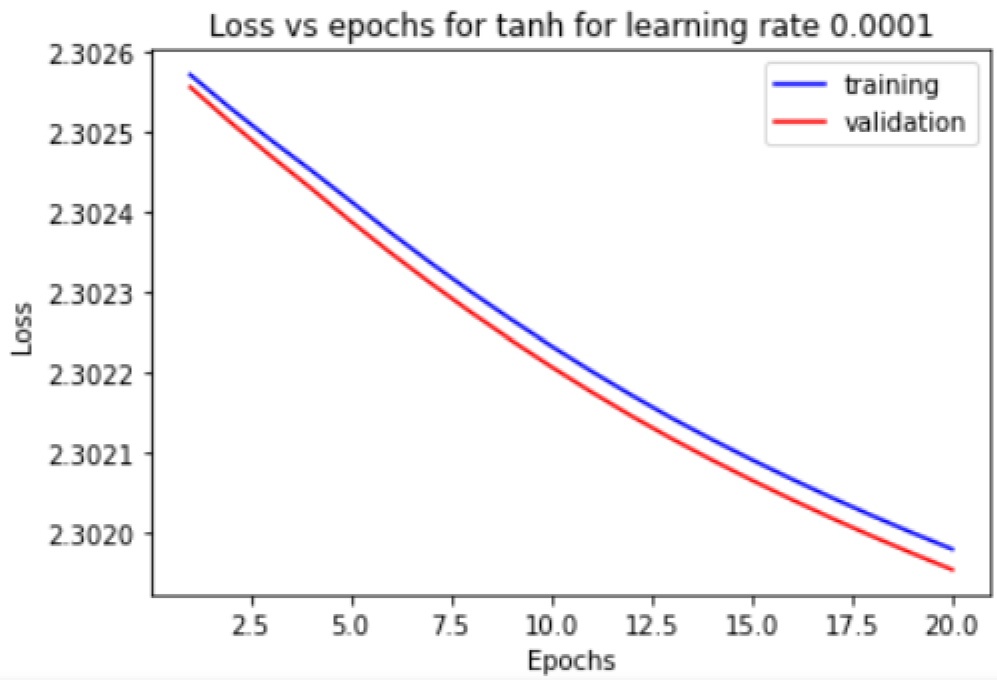
```
In [7]: clf = MLPClassifier(learning_rate_init=0.01, activation='logistic', max_iter=15,batch_size=256, verbose=True,early_stopping=True,

Iteration 1, loss = 0.71968778
Validation score: 0.932167
Iteration 2, loss = 0.19033473
Validation score: 0.948833
Iteration 3, loss = 0.13205673
Validation score: 0.951167
Iteration 4, loss = 0.10933347
Validation score: 0.962333
Iteration 5, loss = 0.09330412
Validation score: 0.959833
Iteration 6, loss = 0.07882485
Validation score: 0.959667
Iteration 7, loss = 0.07577204
Validation score: 0.961833
Iteration 8, loss = 0.06382179
Validation score: 0.964000
Iteration 9, loss = 0.06138147
Validation score: 0.964000
Iteration 10, loss = 0.06201700
Validation score: 0.960000
Iteration 11, loss = 0.05665255
Validation score: 0.958333
Iteration 12, loss = 0.05585646
Validation score: 0.962667
Iteration 13, loss = 0.04982381
Validation score: 0.963500
Iteration 14, loss = 0.05480232
```

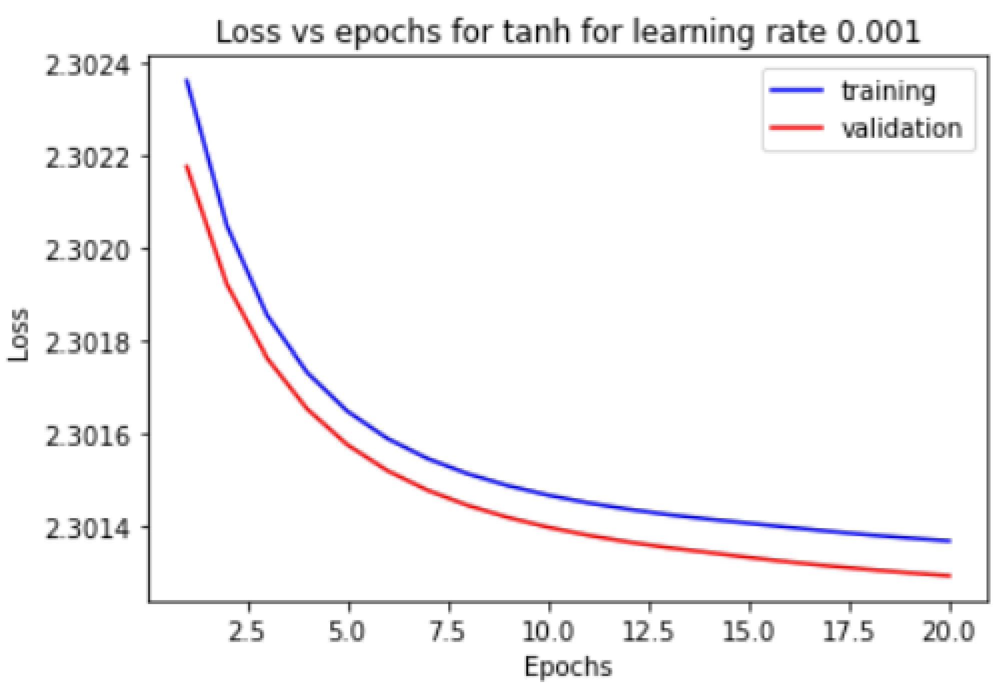
```
In [8]: clf.score(test_images,test_labels)

Out[8]: 0.9667
```

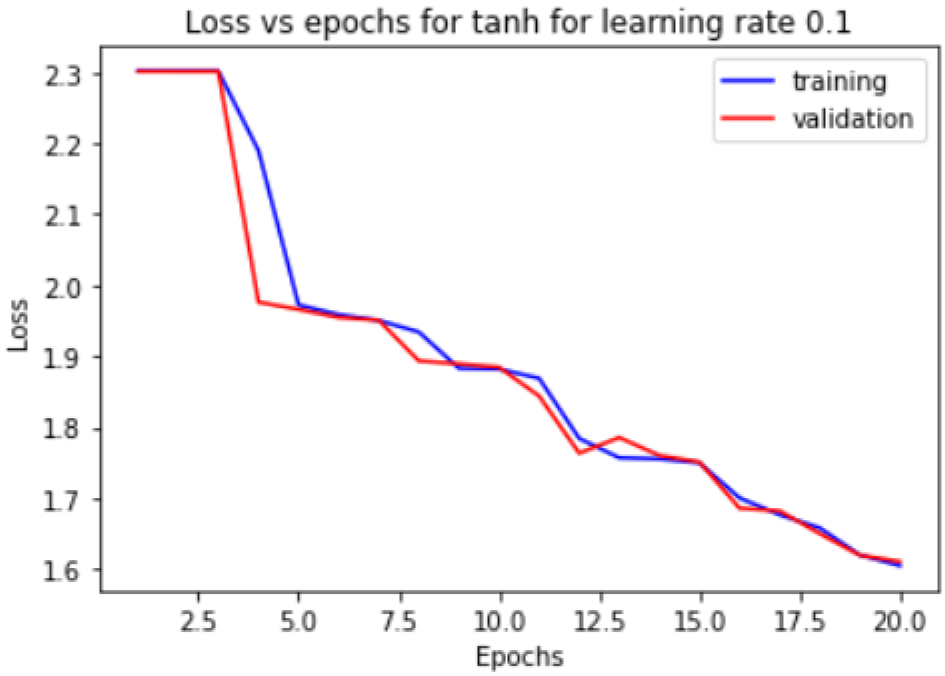
Relu and tanh,sigmoid are very similar.
Relu is 10% and sigmoid,tanh is very high 90+% same as my model.
Linear is different.

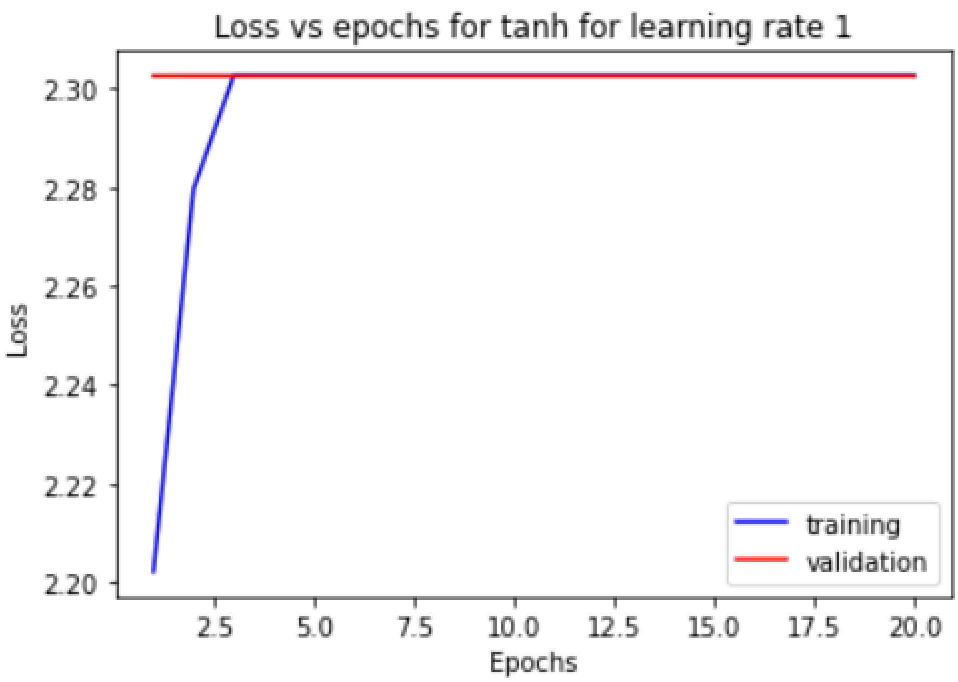
Part 5)

I have chosen tanh function as it was giving the best convergence and least error.









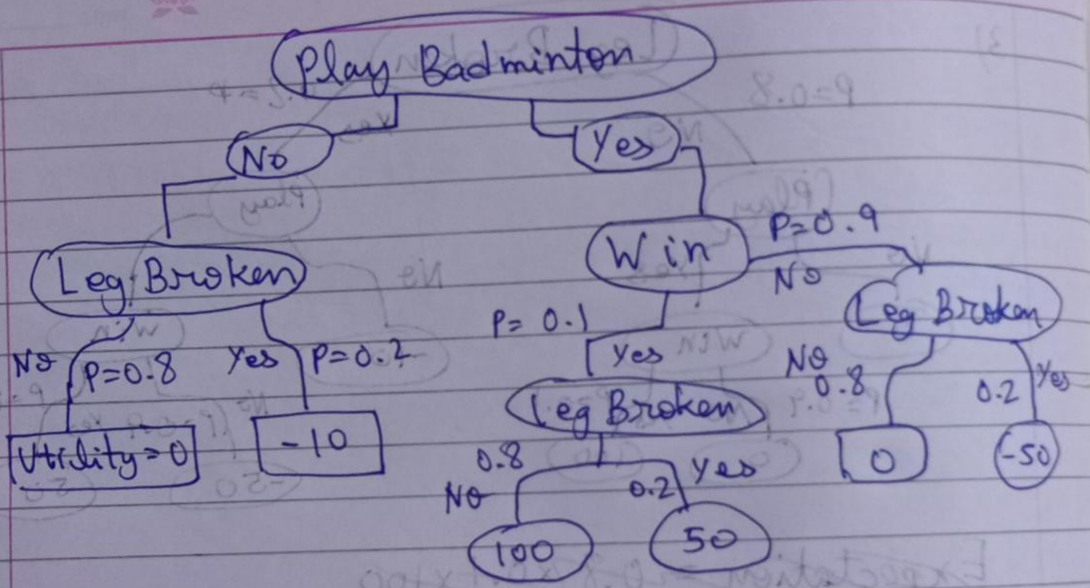
Clearly the loss diverges for lr=1, it is too slow for lr=0.0001, lr=0.001.
It is good on lr=0.1 hence I will choose function as tanh and lr=0.1

Theory Questions

Q1)

Q1]

1)

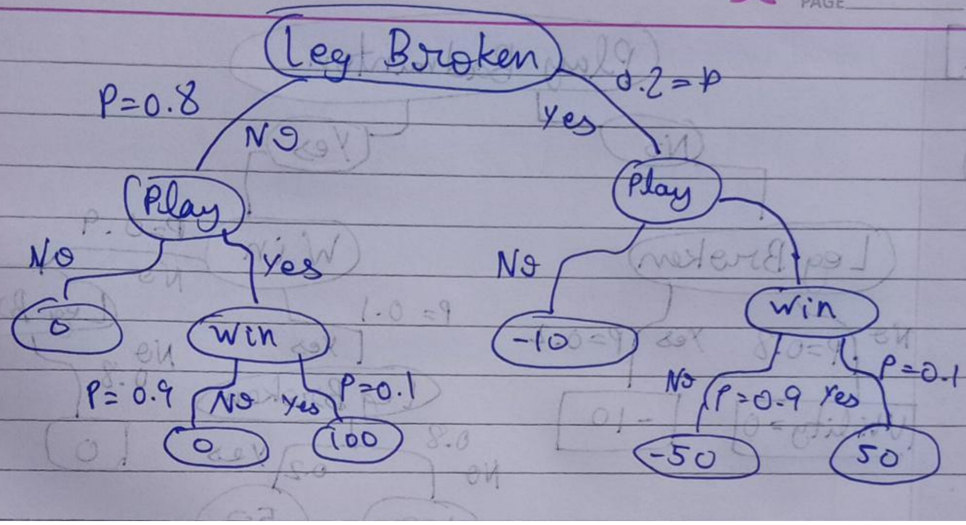


2) Utility of playing : $0.1 \times (0.8 \times 100 + 0.2 \times 50) + 0.9(0.8 \times 0 + 0.2 \times (-50))$
$= 0$

3) Utility of not playing $= 0.8 \times 0 + 0.2 \times (-10)$
$= -2$

According to utility player should play.

3)



$$\text{Expectation} = 0.2 \times \left(\frac{-10 + (0.1\times 50 - 0.9\times 50)}{2} \right)$$

$$+\,0.8 \times 10$$

Expectation = 6

4) $\text{Expectation} = 0.1 \times 90 + 0.9 \times (-2) = 7.2$

5) Yes its possible as we can use the broken branch ~~at~~ and then win branch and then use the probabilities.

Q2] Let $f : \{0,1\}^d \rightarrow \{0,1\}$

Therefore we know f is a boolean function of 'd' variables.
Also, we know every boolean function can be expressed in terms of 'AND' 'or' 'NOT'.

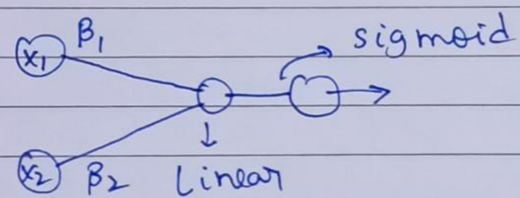
Hence we can say f can also be represented as 'AND', 'OR', 'NOT' of those d boolean layers

Now we know a neural network with 0 hidden layers can represent all 'AND' 'or' 'NOT'.
We use 1 hidden ~~lay~~ layer to combine all such functions and hence we can represent a boolean function with 1 hidden layer.

Q3] We can re write the function as

$$P(Y=y|X) = \frac{\cancel{e}\ 1}{1+e^{-(\beta_1 X_1+\beta_2 X_2)y}}$$

Now clearly this is ~~sigmoid~~ the given net:



That is, hidden layer activation is,

$a_1(\beta_1 X_1 + \beta_2 X_2) = +y \times (\beta_1 X_1 + \beta_2 X_2)$

Now the final layer is sigmoid, i.e.

$$a_2(x) = \frac{1}{1+e^{-x}}$$

$a_1 \to$ activation of first, $a_2 \to$ activation of second