q1

November 19, 2019

```
In [1]: import string
        from sklearn.model_selection import train_test_split
        import numpy as np
        from copy import deepcopy
        import matplotlib.pyplot as plt
        from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
        import math
        from statistics import mean

        %matplotlib inline
```

```
In [2]: # Activation function
        def sigmoid(t):
            return 1/(1+np.exp(-t))
```

```
In [3]: # Derivative of sigmoid
        def sigmoid_derivative(p):
            return p * (1 - p)
```

```
In [4]: class NeuralNetwork:
            def __init__(self, x, y, lr):
                self.input      = x
                self.hidden     = 48
                self.weights1   = np.random.rand(self.input.shape[1], self.hidden)
                self.weights2   = np.random.rand(self.hidden, 1)
                self.y          = y
                self.output     = np.zeros(self.y.shape)
                self.lr         = lr

            def feedforward(self):
                self.layer1 = sigmoid(np.dot(self.input, self.weights1))
                self.output = sigmoid(np.dot(self.layer1, self.weights2))
                return deepcopy(self.output)

            def backprop(self):
                # application of the chain rule to find derivative of the loss function with r
                d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) * sigmoid_derivati
                d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) * sigmoid_
                # update the weights with the derivative (slope) of the loss function
                self.weights1 += d_weights1 * self.lr
                self.weights2 += d_weights2 * self.lr

            def predict(self, x_input):
                l1 = sigmoid(np.dot(x_input, self.weights1))
                out_pred = sigmoid(np.dot(l1, self.weights2))
                return deepcopy(out_pred)
```

```
In [5]: pos_txt = open('./pos_sentiment.txt')
        neg_txt = open('./neg_sentiment.txt')

        X = []
        Y = []

        for line in pos_txt:
            line = line.translate(str.maketrans('', '', string.punctuation)).rstrip()
            X.append(line)
            Y.append(1)

        for line in neg_txt:
            line = line.translate(str.maketrans('', '', string.punctuation)).rstrip()
            X.append(line)
            Y.append(0)

        len(X)
```

```
Out[5]: 6662
```

```
In [6]: def makeVocab(data):
            vocab = []
            for d in data:
                for w in d.split(" "):
                    vocab.append(w)

            return sorted(set(vocab))
```

```
In [7]: def getIndexInVocab(vocab, word):
            if word not in vocab:
                return 0
            return vocab.index(word) + 2
```

```
In [8]: def padding(sentence, padLen):
            pad_sent = []
            i = 0
            for s in sentence:
                pad_sent.append(s)
                i += 1
                if i == padLen:
                    break

            rem_len = padLen - len(pad_sent)
            for i in range(rem_len):
                pad_sent.append(1)

            return pad_sent
```

```
In [9]: def makeVectorized(list_sent, vocab, maxlen=120):
            ans = []

            for l in list_sent:
                sent = []
                for w in l.split(" "):
                    sent.append(getIndexInVocab(vocab, w))

                sent = padding(sent, maxlen)
                ans.append(sent)

            return ans
```

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=
```

```
In [11]: print( len(X_train) , len(X_test))
```

```
5329 1333
```

```
In [12]: vocab = makeVocab(X_train)
```

```
In [13]: x_train = np.array(makeVectorized(X_train,vocab))
         x_test = np.array(makeVectorized(X_test, vocab))
```

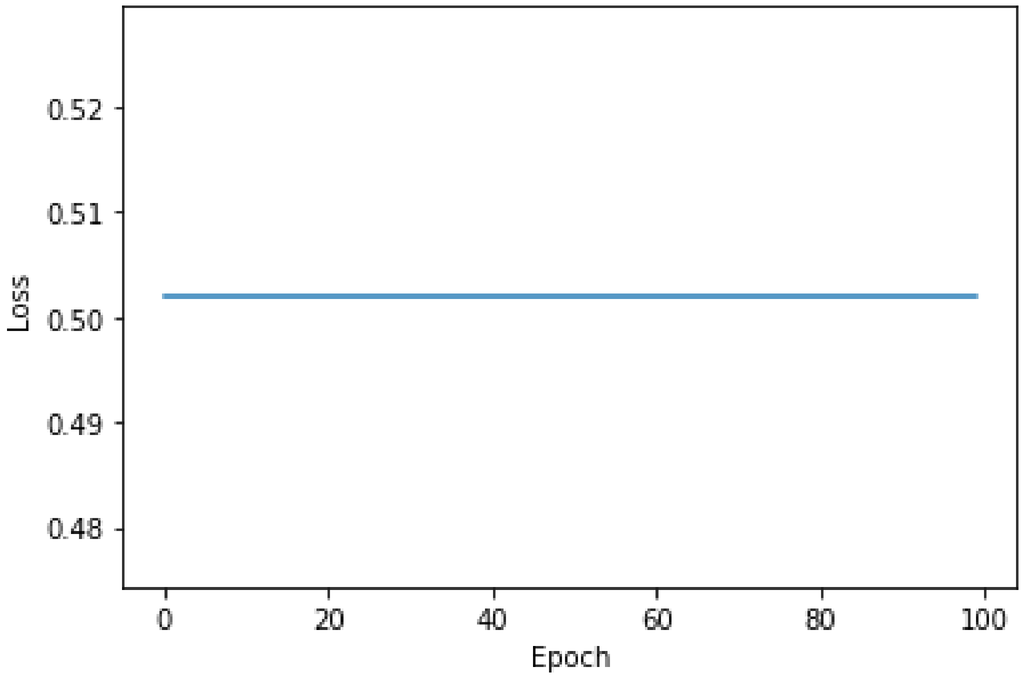
```
In [14]: y_train = np.array(y_train).reshape(-1, 1)
         y_train.shape
```

```
Out[14]: (5329, 1)
```

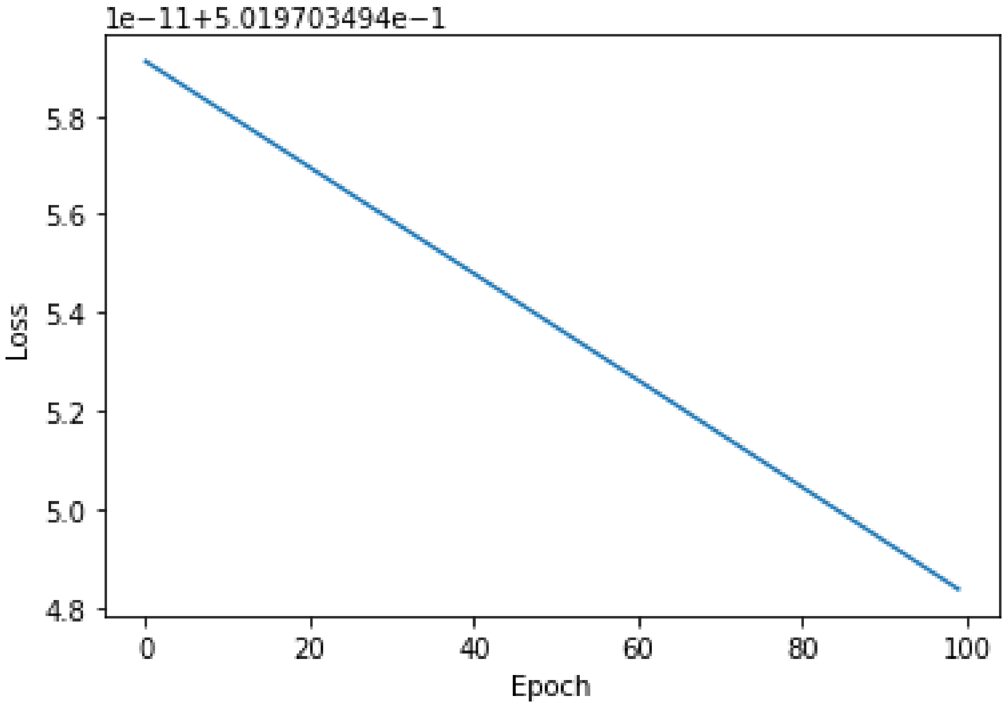
```
In [15]: epoch = 100
         bestModel = None
         lowestLoss = math.inf
         lr_val = []
         loss_lr_data = []

         for lr in range(1, 11):
             model = NeuralNetwork(x_train, y_train, lr/10)
             l = []
```

```
for i in range(epoch):
    pred_e = model.feedforward()
    model.backprop()
    l.append(np.mean(np.square(y_train - pred_e)))

if mean(l) < lowestLoss:
    bestModel = model
    lowestLoss = mean(l)

print("LR = " + str(lr/10))
plt.plot(range(len(l)), l)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()

lr_val.append(lr/10)
loss_lr_data.append(mean(l))
```

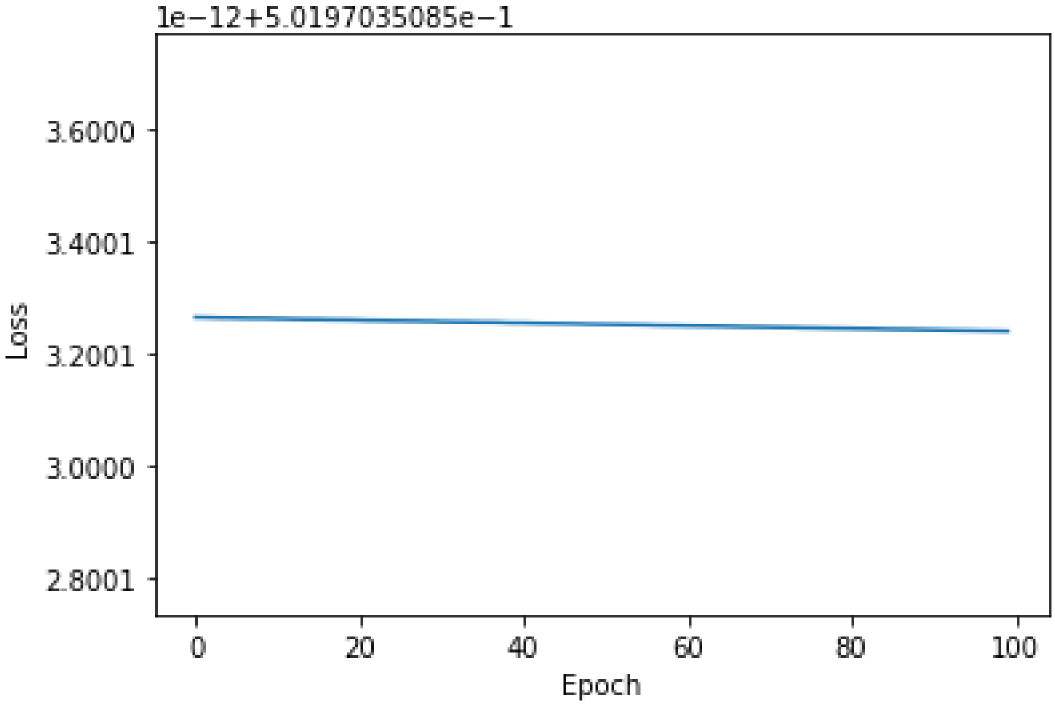
```
LR = 0.1
```



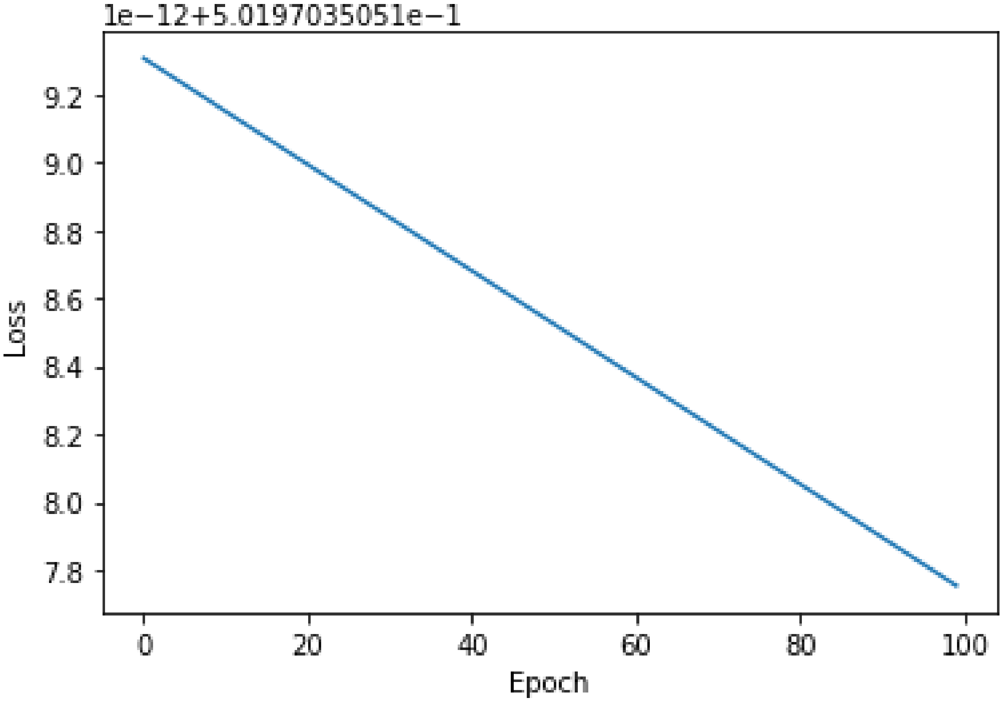
```
LR = 0.2
```


1e−11+5.019703494e−1
5.8
5.6
5.4
5.2
5.0
4.8
Loss
0
20
40
60
80
100
Epoch

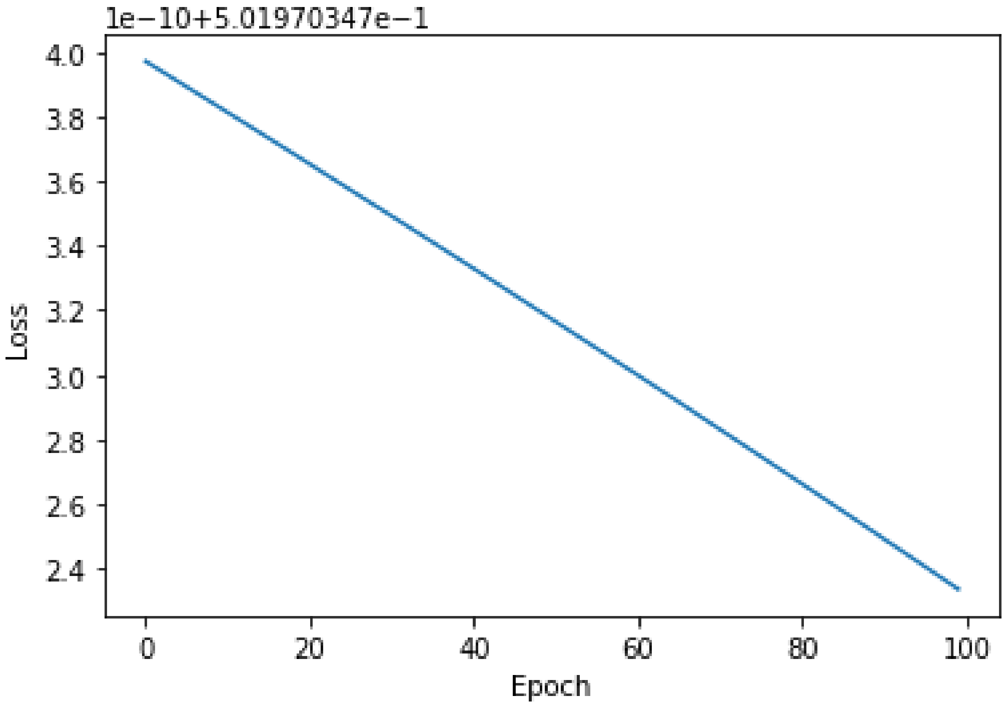
LR = 0.3


1e−12+5.0197035085e−1
3.6000
3.4001
3.2001
3.0000
2.8001
Loss
0
20
40
60
80
100
Epoch

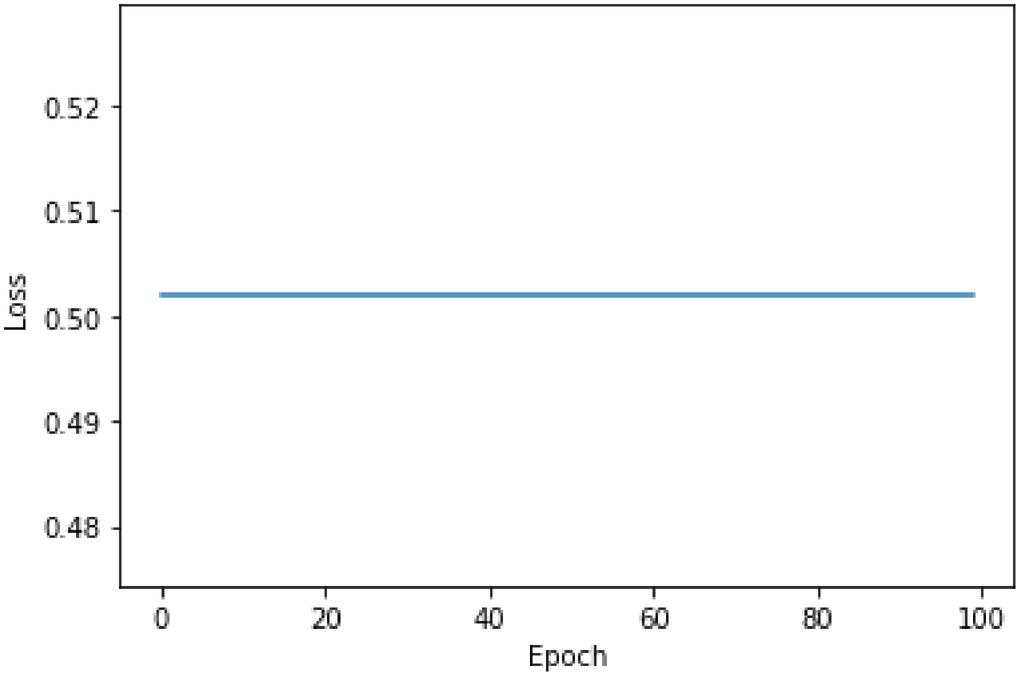
LR = 0.4


1e−12+5.0197035051e−1
9.2
9.0
8.8
8.6
8.4
8.2
8.0
7.8
Loss
0
20
40
60
80
100
Epoch

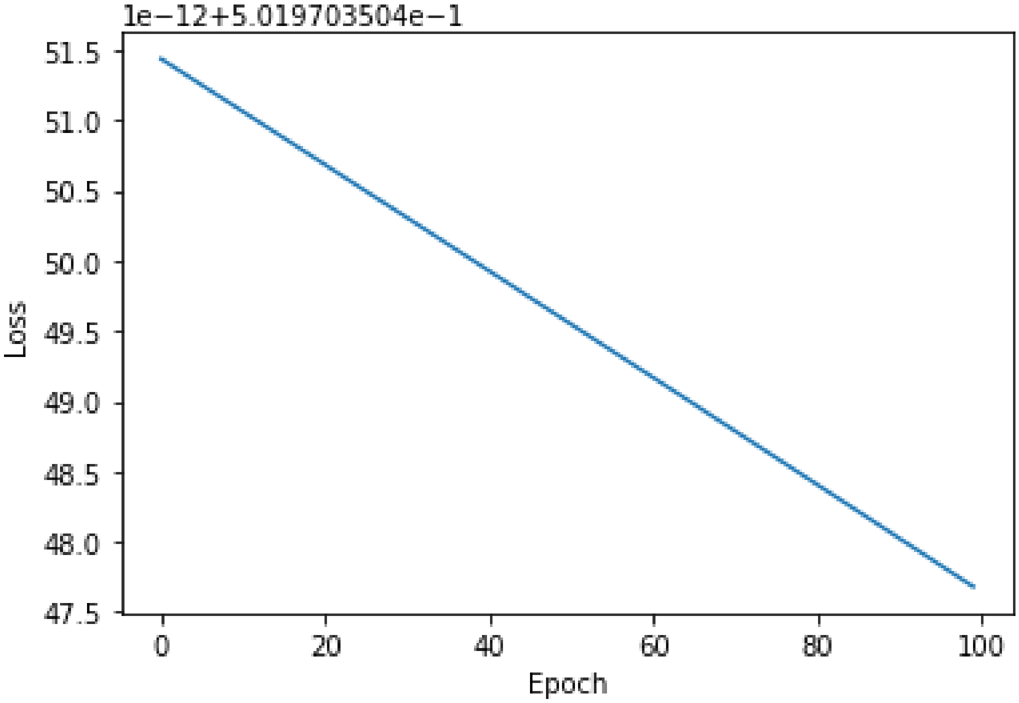
LR = 0.5


1e−10+5.01970347e−1
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
Loss
0
20
40
60
80
100
Epoch

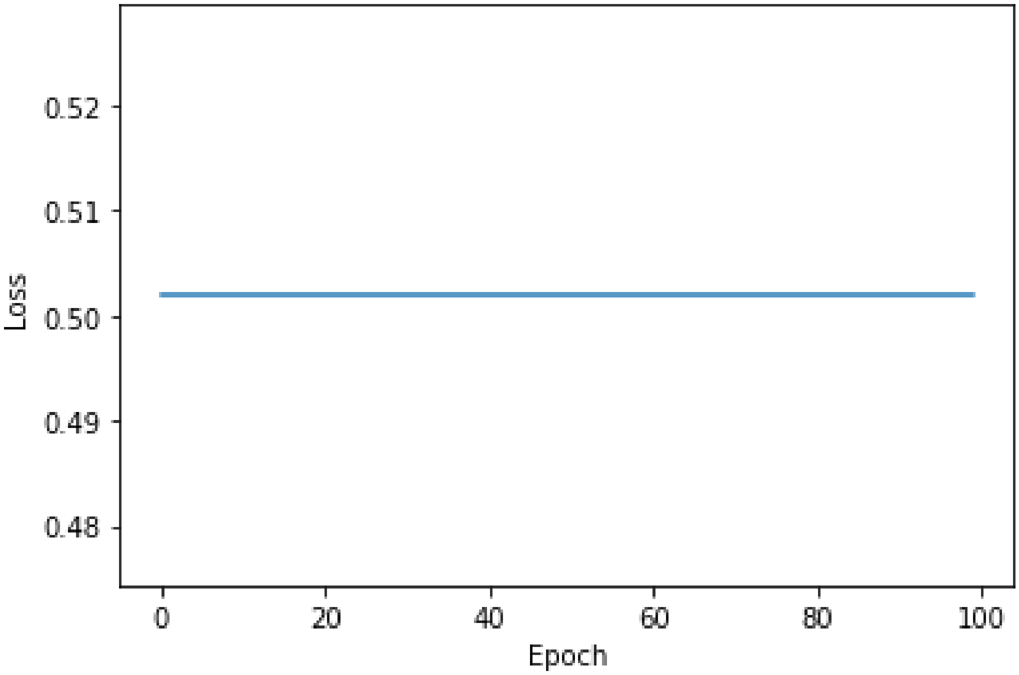
LR = 0.6


0.52
0.51
0.50
0.49
0.48
Loss
0
20
40
60
80
100
Epoch

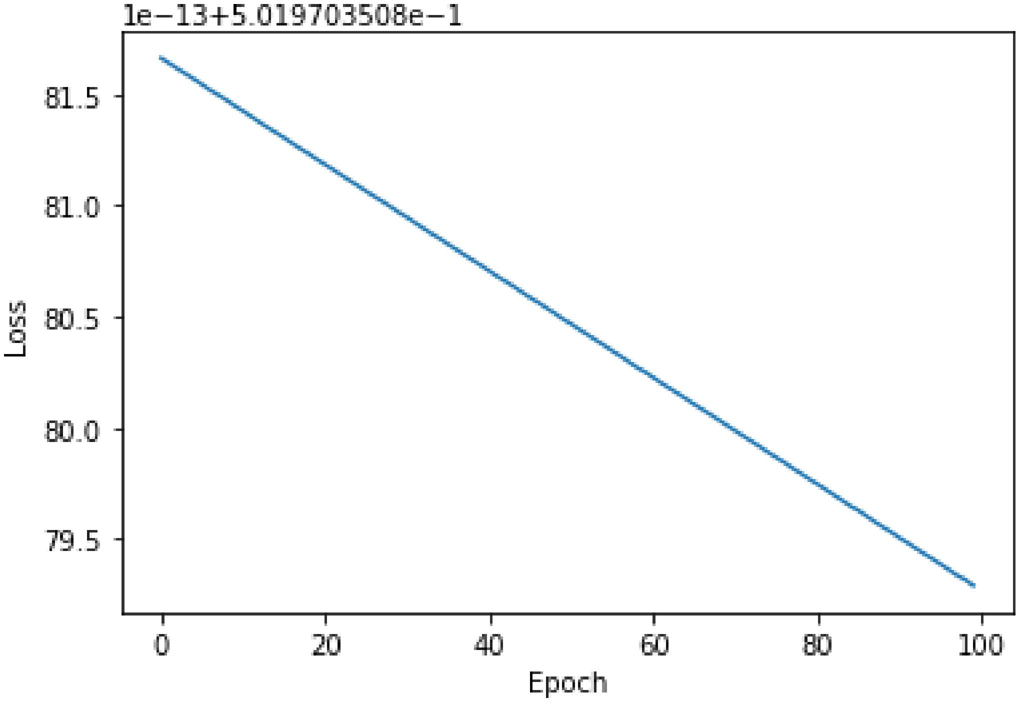
LR = 0.7


1e−12+5.019703504e−1
51.5
51.0
50.5
50.0
49.5
49.0
48.5
48.0
47.5
Loss
0
20
40
60
80
100
Epoch

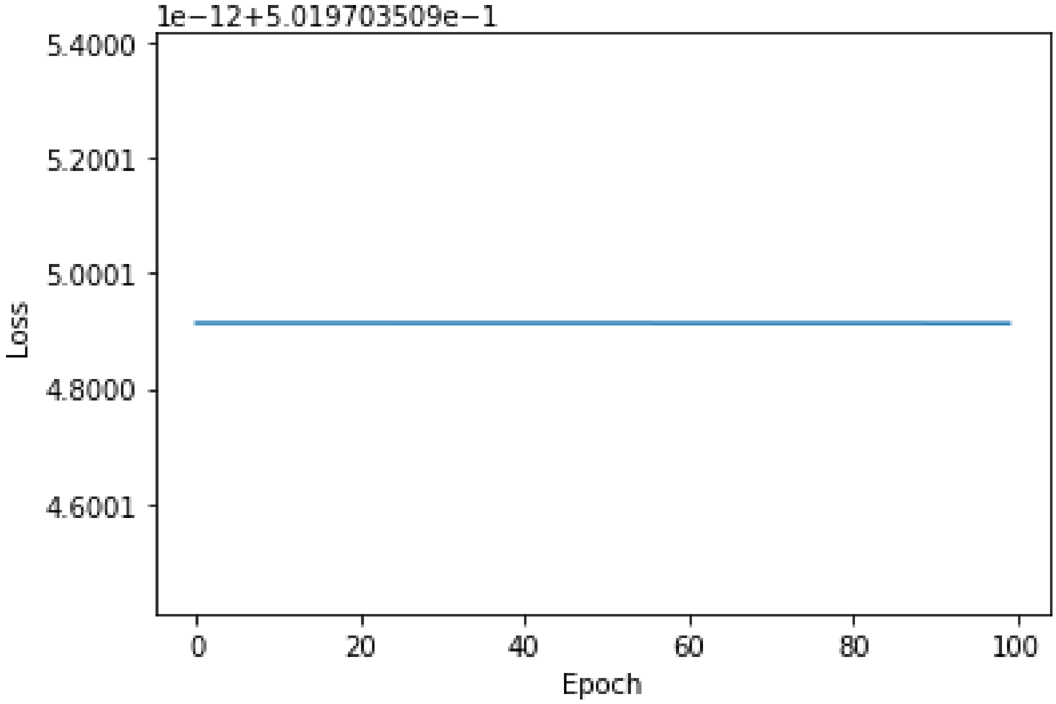
LR = 0.8


0.52
0.51
0.50
0.49
0.48
Loss
0
20
40
60
80
100
Epoch

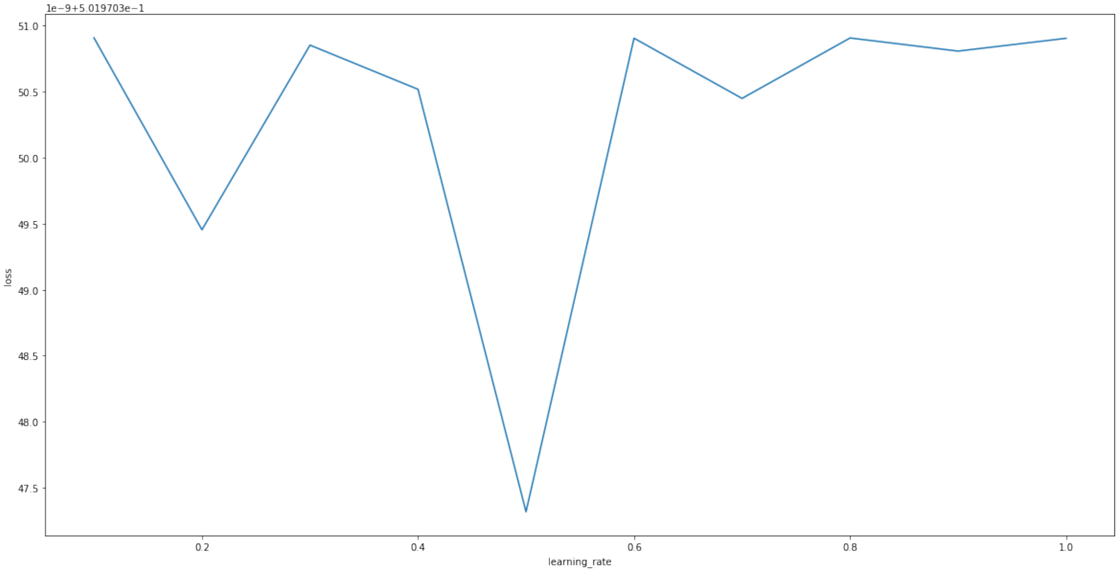
LR = 0.9


1e−13+5.019703508e−1
81.5
81.0
80.5
80.0
79.5
Loss
0
20
40
60
80
100
Epoch

LR = 1.0



```
In [16]: plt.rcParams['figure.figsize'] = [20,10]
         plt.xlabel('learning_rate')
         plt.ylabel('loss')
         plt.plot(lr_val, loss_lr_data)
         plt.show()
```



```
In [17]: # PREDICTION

         y_pred = bestModel.predict(x_test)
         y_pred = np.round(y_pred)

         print("Precision Score = " + str(precision_score(y_test, y_pred)))
         print("Recall Score = " + str(recall_score(y_test, y_pred)))
         print("F Score = " + str(f1_score(y_test, y_pred)))
         print("Accuracy = " + str(accuracy_score(y_test, y_pred)))
```

```
Precision Score = 0.5078769692423106
Recall Score = 1.0
F Score = 0.6736318407960199
Accuracy = 0.5078769692423106
```