19BCE248
2CS501
Practical 9

AIM: Implementing XOR from scratch and using sklearn library

Dataset:

It only consists of 4 rows and two columns which are a combination of 0 and 1 and output as 0 or 1.

Preprocessing:

It doesn't require any preprocessing.

Implementation:

First of all, we need to propose a forward propagation technique, and then for definite epochs, we will also backpropagate.

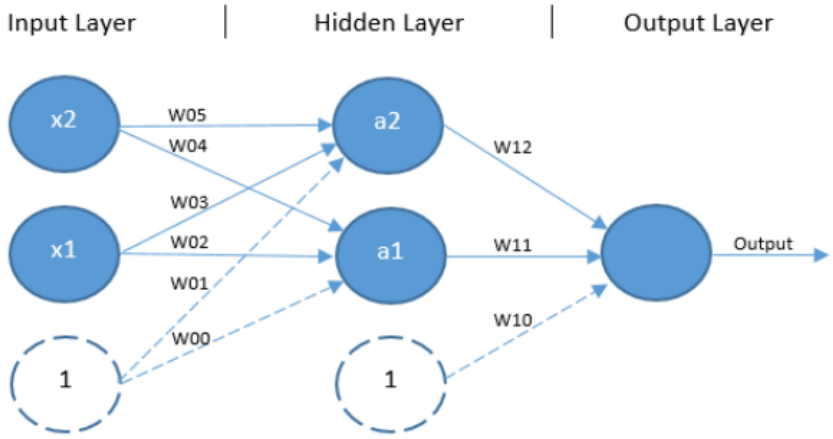
Forward Passing:

```
n1=np.dot(w[0],x)+b[0]
out_n1=sigmoid(n1)

n2=np.dot(w[1],x)+b[1]
out_n2=sigmoid(n2)
n3=w[2][0]*out_n1 + w[2][1]*out_n2+b[2]
out_n3=sigmoid(n3)

y_pred=out_n3
y_act=y[j]
```

Backward Passing:

```
        #backward propogation

        b[2]=b[2]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))

        w[2][0]=w[2][0]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*out_n1

        w[2][1]=w[2][1]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*out_n2

        w[0][0]=w[0][0]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*w[2][0]*sigmoid(n1)*(1-sigmoid(n1))*x[0]

        w[0][1]=w[0][1]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*w[2][0]*sigmoid(n1)*(1-sigmoid(n1))*x[0]

        w[1][0]=w[1][0]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*w[2][1]*sigmoid(n2)*(1-sigmoid(n2))*x[1]

        w[1][1]=w[1][1]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*w[2][1]*sigmoid(n2)*(1-sigmoid(n2))*x[1]

        b[0]=b[0]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*w[2][0]*sigmoid(n1)*(1-sigmoid(n1))

        b[1]=b[1]-rate*(-(y_act-y_pred))*sigmoid(n3)*(1-sigmoid(n3))*w[2][1]*sigmoid(n2)*(1-sigmoid(n2))
    cost.append((y_pred-y_act)**2)
          print(w,b)
```

Neural Network is as below:



Final Output:

```
y_predicted: 0 y_actual: 0
y_predicted: 0 y_actual: 0
y_predicted: 1 y_actual: 1
y_predicted: 1 y_actual: 1
y_predicted: 0 y_actual: 0
Time Taken 17.380882263183594
```

Using Sklearn module:

```
t1=time.time()
mlp = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(2),max_iter=10000)
mlp.fit(X, y)
y_pred=mlp.predict(X)
print(y_pred)
accuracy=metric.accuracy_score(np.array(y).flatten(), np.array(y_pred).flatten(), normalize=True)
print('acuracy=',accuracy)
t2=time.time()
print("Time Taken",t2-t1)
```

```
[0 1 1 0]
acuracy= 1.0
Time Taken 0.030000925064086914
```

Conclusion:

From these practicals, we learned how to implement a neural network from scratch and also we analyzed the time taken between both the process and how good is our implementation is with the sklearn module. By performing these practical all the theory concepts were much clear respective to earlier what it was.