MACHINE LEARNING – 2CS501

PRACTICAL 9

Name: Bhanderi Mihir

Roll No.: 19BCE023

Batch No.: A-1

1) Without sklearn

Code:

```
import numpy as np
from matplotlib import pyplot as plt

def initializeParameters(inputFeatures, neuronsInHiddenLayers, outputFeatures):
    W1 = np.random.randn(neuronsInHiddenLayers, inputFeatures)
    W2 = np.random.randn(outputFeatures, neuronsInHiddenLayers)
    b1 = np.zeros((neuronsInHiddenLayers, 1))
    b2 = np.zeros((outputFeatures, 1))

    parameters = {"W1" : W1, "b1": b1,
                  "W2" : W2, "b2": b2}
    return parameters

def forwardPropagation(X, Y, parameters):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]

    net1 = np.dot(W1, X) + b1
    A1 = 1 / (1 + np.exp(-net1))
    net2 = np.dot(W2, A1) + b2
    A2 = 1 / (1 + np.exp(-net2))

    cache = (net1, A1, W1, b1, net2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 - Y))
    cost = -np.sum(logprobs) / m
    return cost, cache, A2

def backwardPropagation(X, Y, cache):
    m = X.shape[1]
    (net1, A1, W1, b1, net2, A2, W2, b2) = cache

    dnet2 = A2 - Y
    dW2 = np.dot(dnet2, A1.T) / m
    db2 = np.sum(dnet2, axis = 1, keepdims = True)

    dA1 = np.dot(W2.T, dnet2)
    dnet1 = np.multiply(dA1, A1 * (1- A1))
    dW1 = np.dot(dnet1, X.T) / m
    db1 = np.sum(dnet1, axis = 1, keepdims = True) / m
```

```
    gradients = {"dZ2": dnet2, "dW2": dW2, "db2": db2,
                 "dZ1": dnet1, "dW1": dW1, "db1": db1}
    return gradients

def updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"] - learningRate * gradients["dW1"]
    parameters["W2"] = parameters["W2"] - learningRate * gradients["dW2"]
    parameters["b1"] = parameters["b1"] - learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"]
    return parameters

X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # XOR input
Y = np.array([[0, 1, 1, 0]]) # XOR output

# Define model parameters
neuronsInHiddenLayers = 2
inputFeatures = X.shape[0]
outputFeatures = Y.shape[0]
parameters = initializeParameters(inputFeatures, neuronsInHiddenLayers,
outputFeatures)
epoch = 100000
learningRate = 0.01
losses = np.zeros((epoch, 1))

for i in range(epoch):
    losses[i, 0], cache, A2 = forwardPropagation(X, Y, parameters)
    gradients = backwardPropagation(X, Y, cache)
    parameters = updateParameters(parameters, gradients, learningRate)


plt.figure()
plt.plot(losses)
plt.xlabel("EPOCHS")
plt.ylabel("Loss value")
plt.show()


cost, _, A2 = forwardPropagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0
print(A2)
print(prediction)

"""

Output:
[[0.01362852 0.98635087 0.98714781 0.01150988]]
[[0. 1. 1. 0.]]


"""
```

2) With sklearn

Code:

```
from sklearn.neural_network import MLPClassifier
import numpy as np

X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

y = np.array([0, 1, 1, 0]).reshape(4,)


clf = MLPClassifier(hidden_layer_sizes=(4,2), max_iter=30000)
clf.fit(X, y)
y_predict = clf.predict(X)
print("Predicted Output : ",y_predict)

"""

Output:
Predicted Output :  [0 1 1 0]

"""
```