EXPERIMENT NO:6

DATE: 15-02-2022

Aim :

Download the SEEDS dataset: Click Here. It has 210 data points each with 7 attributes, and 3 classes (1,2,3) denoting three different varieties of wheat. Consider the output as a one-hot vector. E.g., if a data point has a class '2', consider the output of that data point as [0 1 0]. Normalize each attribute by z-score normalization: for each feature, every value is to be normalized to (value - m)/s, where m is the mean value of the feature, and s is the standard deviation of the feature. Consider 80% of the dataset (randomly selected) as training data, and the rest 20% as test data.

Part 1: Building your own Neural Network

Build a neural network and perform the classification task with the specifications mentioned as follows. Your implementation should have the following modules:

1. Preprocess: Use this module to preprocess the data and divide into train and test.
2. Data loader (optional): Use this module to load all datasets and create mini batches, with each mini batch having 32 training examples.
3. Weight initializer: This module should initialize all weights randomly between -1 and 1.
4. Forward pass: Define the forward () function where you do a forward pass of the neural network.
5. Backpropagation: Define a backward () function where you compute the loss and do a backward pass (backpropagation) of the neural network and update all weights.
6. Training: Implement a simple mini batch SGD loop/ batch GD and train your neural network, using forward and backward passes.
7. Predict: To test the learned model weights to predict the classes of the test set.

NN Specification

1. No of hidden layers: 1

2. No. of neurons in hidden layer: 32

3. Activation function in the hidden layer: Sigmoid

4. 3 neurons in the output layer.

5. Activation function in the output layer: Softmax

6. Optimization algorithm : Mini Batch Stochastic Gradient Descent (SGD) / Batch Gradient Descent (GD)

7. Loss function: categorical cross entropy loss

8. Learning rate: 0.01

9. No. of epochs = 200

Your code must output the following for each of the two specifications:

1. Plot the training accuracy and test accuracy after every 10 epochs (in a single plot)

2. Print the final training accuracy

3. Print the final test accuracy

Part 2: Implementation with scikit learn

Use the MLP implementation of scikit learn: Click here for details. Use the same specifications and use the same training and test data. Your code must output

1. Final train accuracy

2. Final test accuracy

In [5]:
```
#Form a 3 layer neural network (one input, one hidden, and one output) to learn the XOR function.
#The input layer should contain 2 binary inputs.
#Second layer (first hidden layer) should contain 3 neurons.
#Output layer contains 1 neuron which will produce the output of the XOR function
```

In [6]:
```
import numpy as np

# Activation Functions
def tanh(x):
    return np.tanh(x)

def d_tanh(x):
    return 1 - np.square(np.tanh(x))

def sigmoid(x):
    return 1/(1 + np.exp(-x))

def d_sigmoid(x):
    return (1 - sigmoid(x)) * sigmoid(x)

# Loss Functions
def logloss(y, a):
    return -(y*np.log(a) + (1-y)*np.log(1-a))

def d_logloss(y, a):
    return (a - y)/(a*(1 - a))
```

In [7]:
```
# The layer class
class Layer:

    activationFunctions = {
        'tanh': (tanh, d_tanh),
        'sigmoid': (sigmoid, d_sigmoid)
    }
    learning_rate = 0.1
```

In [8]:
```
x_train = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # dim x m
y_train = np.array([[0, 1, 1, 0]]) # 1 x m

#dnn = DeepNeuralNetwork(sizes=[784, 128, 64, 10])

m = 4
epochs = 3000

layers = [Layer(2, 3, 'tanh'), Layer(3, 1, 'sigmoid')]
costs = [] # to plot graph

for epoch in range(epochs):
    A = x_train
    for layer in layers:
        A = layer.feedforward(A)

    cost = 1/m * np.sum(logloss(y_train, A))
    costs.append(cost)

    dA = d_logloss(y_train, A)
    for layer in reversed(layers):
        dA = layer.backprop(dA)
```

In [10]:
```
# predicting
A = np.array([[0], [0]]) # dim(=2) x m(=1)
for layer in layers:
    A = layer.feedforward(A)
print(A)
```

```
[[0.00222633]]
```

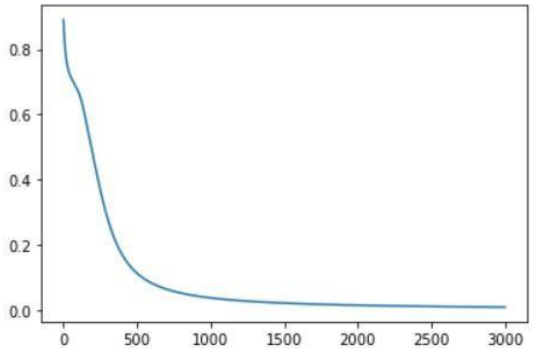
In [11]:
```
import matplotlib.pyplot as plt
plt.plot(range(epochs), costs)
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x2443c009340>]
```

In [7]:
```
# The layer class
class Layer:

    activationFunctions = {
        'tanh': (tanh, d_tanh),
        'sigmoid': (sigmoid, d_sigmoid)
    }
    learning_rate = 0.1

    def __init__(self, inputs, neurons, activation):# inputs to a layer=the number of neurons in the previous layer
                                                    # neurons- the number of neurons in a layer
                                                    # activation - the particular activation for a layer
        self.W = np.random.randn(neurons, inputs)
        self.b = np.zeros((neurons, 1))
        self.act, self.d_act = self.activationFunctions.get(activation)

    def feedforward(self, A_prev):
        self.A_prev = A_prev
        self.Z = np.dot(self.W, self.A_prev) + self.b
        self.A = self.act(self.Z)
        return self.A

    def backprop(self, dA):
        dZ = np.multiply(self.d_act(self.Z), dA)
        dA_prev = np.dot(self.W.T, dZ)

        dW = 1/dZ.shape[1] * np.dot(dZ, self.A_prev.T)
        db = 1/dZ.shape[1] * np.sum(dZ, axis=1, keepdims=True)

        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db

        return dA_prev
```

In [11]:
```
import matplotlib.pyplot as plt
plt.plot(range(epochs), costs)
```

Out[11]: [<matplotlib.lines.Line2D at 0x2443c009340>]



In [ ]:

EXPERIMENT NO:7

DATE: 16-02-2022

Aim :

Write a program to cluster a set of points using K-means. Consider K=3 as the number of clusters. Use Euclidean distance as the distance measure. Randomly initialize a cluster mean as one of the data points. Iterate for 10 iterations. After iterations are over, print the final cluster means for each of the clusters. Use the ground truth cluster label present in the data set to compute and print the Jacquard distance of the obtained clusters with the ground truth clusters for each of the three clusters.

Data Set Preparation:

Data Filename: data_iris.csv

The data set contains 150 data points, there are three clusters where each cluster refers to a type of iris plant. The first four columns represent the attributes listed below. Note that only the first four columns should be used as attributes. The last column is the ground truth cluster name and is to be used for evaluating the cluster quality.

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. Ground truth cluster name:

-- Iris Setosa

-- Iris Versicolour

-- Iris Virginica

Source Code:

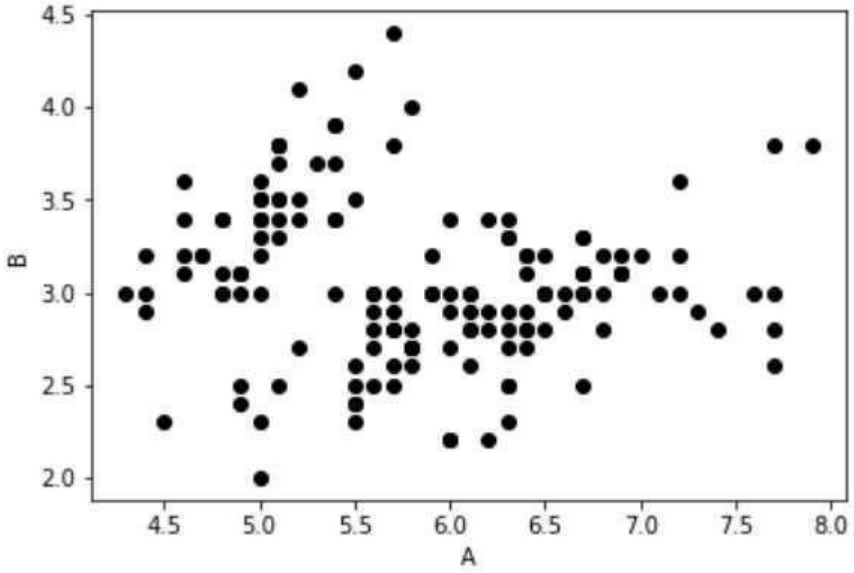
[2]:
```
#import libraries
import pandas as pd
import numpy as np
import random as rd
import matplotlib.pyplot as plt
```

[3]:
```
data = pd.read_csv('data_iris.csv')
data.head()
```

[3]:

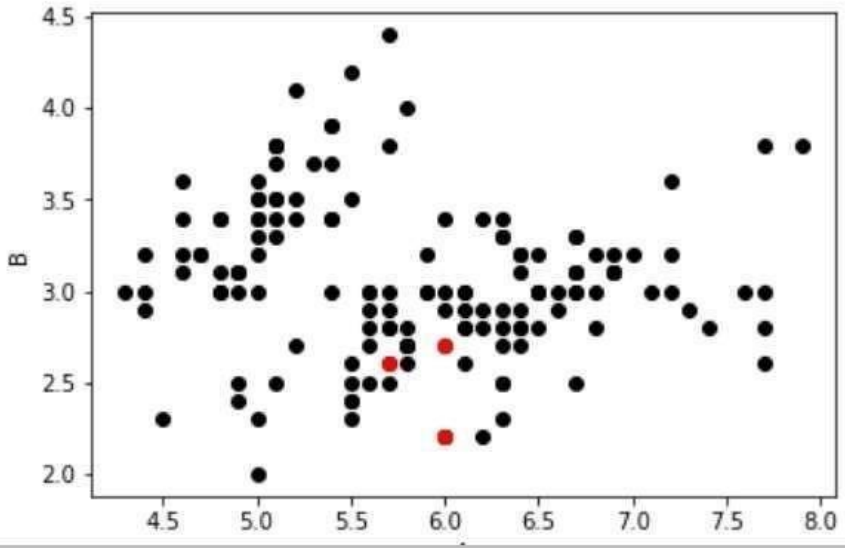
	A	B	C	D	E
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

[4]:
```
X = data[["A","B"]]
#Visualise data points
plt.scatter(X["A"],X["B"],c='black')
plt.xlabel('A')
plt.ylabel('B')
plt.show()
```



[5]:
```
#number of clusters
K=3
```

[6]:
```
# Select random observation as centroids
Centroids = (X.sample(n=K))
plt.scatter(X["A"],X["B"],c='black')
plt.scatter(Centroids["A"],Centroids["B"],c='red')
plt.xlabel('A')
plt.ylabel('B')
plt.show()
```

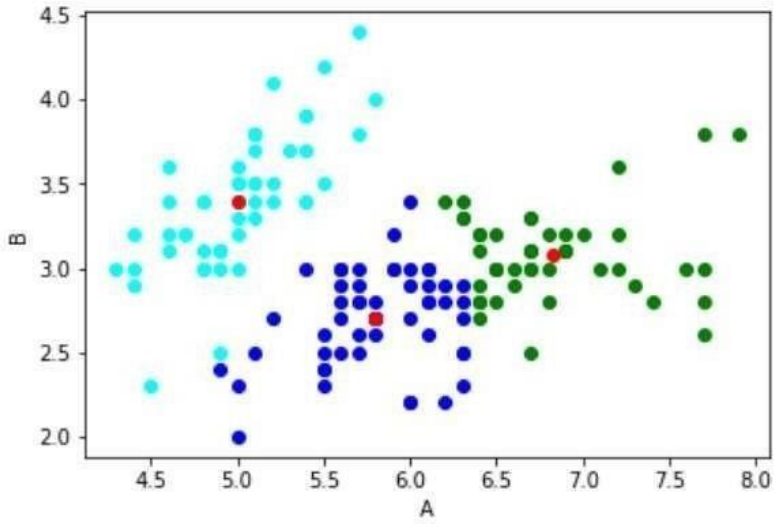


[7]:
```
diff = 1
j=0

while(diff!=0):
    XD=X
    i=1
    for index1,row_c in Centroids.iterrows():
        ED=[]
        for index2,row_d in XD.iterrows():
            d1=(row_c["A"]-row_d["A"])**2
            d2=(row_c["B"]-row_d["B"])**2
            d=np.sqrt(d1+d2)
            ED.append(d)
        X[i]=ED
        i=i+1
```

```
C=[]
for index,row in X.iterrows():
    min_dist=row[1]
    pos=1
    for i in range(K):
        if row[i+1] < min_dist:
            min_dist = row[i+1]
            pos=i+1
    C.append(pos)
X["Cluster"]=C
Centroids_new = X.groupby(["Cluster"]).mean()[["B","A"]]
if j == 0:
    diff=1
    j=j+1
else:
    diff = (Centroids_new['B'] - Centroids['B']).sum() + (Centroids_new['A'] - Centroids['A']).sum()
    print(diff.sum())
Centroids = X.groupby(["Cluster"]).mean()[["B","A"]]
```

```
0.1643683925182975
0.25578543952898736
0.13067050700186345
0.02384257119320843
0.0
```

[8]:
```
color=['blue','green','cyan']
for k in range(K):
    data=X[X["Cluster"]==k+1]
    plt.scatter(data["A"],data["B"],c=color[k])
plt.scatter(Centroids["A"],Centroids["B"],c='red')
plt.xlabel('A')
plt.ylabel('B')
plt.show()
```



[9]:
```
XD=X
i=1
m=0
for index1,row_c in Centroids.iterrows():
    ED=[]
    for index2,row_d in XD.iterrows():
        d1=(row_c["A"]-row_d["A"])**2
        d2=(row_c["B"]-row_d["B"])**2
        d=np.sqrt(d1+d2)
        ED.append(d)
        m=m+1
    X[i]=ED
    i=i+1

print(X)
```

	A	B	1	2	3	Cluster
0	5.1	3.5	1.063015	1.774751	0.138676	3
1	4.9	3.0	0.948683	1.925504	0.413279	3
2	4.7	3.2	1.208305	2.127399	0.363825	3
3	4.6	3.1	1.264911	2.224019	0.503143	3
4	5.0	3.6	1.204159	1.897069	0.200038	3
..	...	...	...	...	...	...
145	6.7	3.0	0.948683	0.146558	1.742608	2
146	6.3	2.5	0.538516	0.780302	1.577916	1
147	6.5	3.0	0.761577	0.333233	1.548629	2
148	6.2	3.4	0.806226	0.701985	1.196078	2
149	5.9	3.0	0.316228	0.927222	0.981303	1

[150 rows x 6 columns]