CSE 142 Assignment 5, Fall 2023

2 Questions, 100 pts, due: 23:59 pm, Dec 6th, 2023

`Your name: Airi Kokuryo  Student ID: 2086695`

Instruction

- Submit your assignments onto **Gradescope** by the due date. Upload a `zip` file containing:

  (1) The saved/latest `.ipynb` file, please **rename this file with your name included**.

  (2) Also save your file into a pdf version, if error appears, save an html version instead (easy to grade for written questions).

  For assignment related questions, please reach TA or grader through Slack/Piazza/Email.

- This is an **individual** assignment. All help from others (from the web, books other than text, or people other than the TA or instructor) must be clearly acknowledged.

Objective

- **Task 1:** EM algorithm (Coding)
- **Task 2:** Neural Networks (Coding + Theory)

Question 1. (EM algorithm, 30 pts)

Question 1.1. Implement EM Algorithm in Python from scratch.

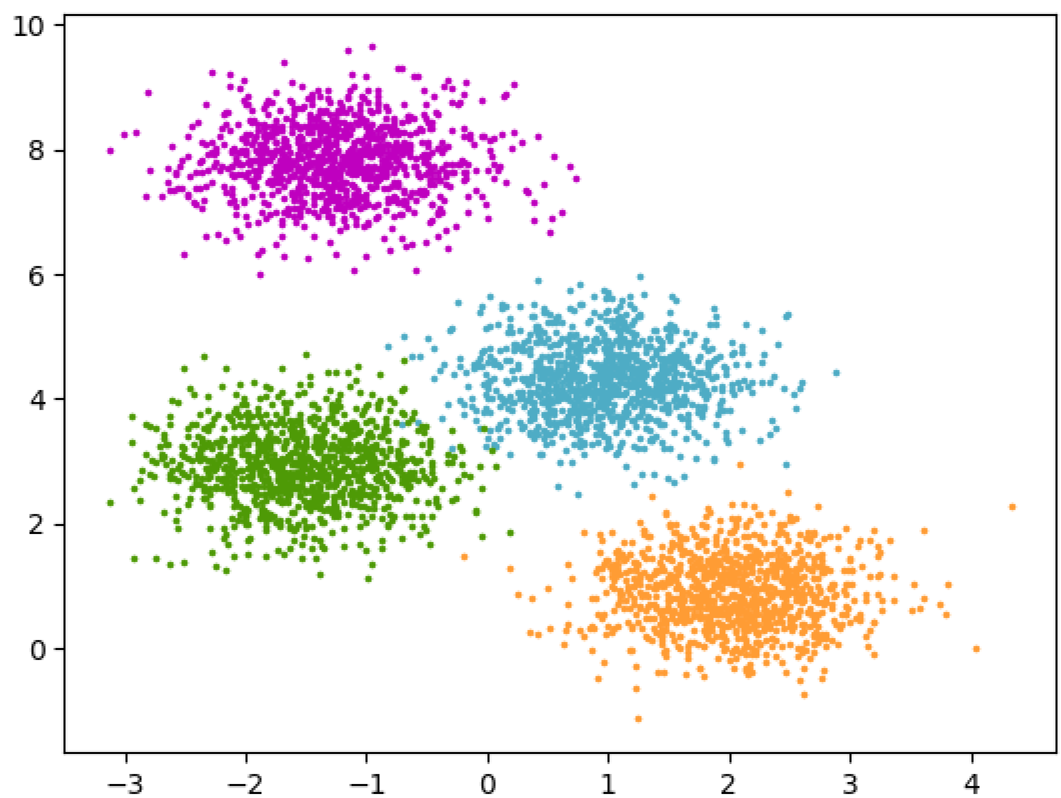
In [ ]:
```
from scipy.stats import multivariate_normal
import numpy as np

def e_step(data, mu, sigma, pi, k):

  ########################
  weights = np.zeros((len(data), k))

  for i in range(k):
        #find distribution given mu and sigma for each gaussian
        distribution = multivariate_normal(mean=mu[i], cov=sigma[i])
        #find likelihood for each datapoint
        weights[:, i] = pi[i] * distribution.pdf(data)
  #normalize weights
```

```
  weights = weights / np.sum(weights, axis=1, keepdims=True)
  ########################

  # HINTS: ##
  ## distribution = multivariate_normal() from scipy can be used to find distribution gi
  ## likelihood = distribution.pdf() can be then used to find likelihood for each datapo

  ## you will find z_ik here (let's call it as "weights")

  return weights

def m_step(data, mu, sigma, pi, weights, k):

  ##################################################
  for i in range(k):
        # update mean
        mu[i] = np.sum(weights[:, i].reshape(-1, 1) * data, axis=0) / np.sum(weights[:,

        # update covariance matrix
        sigma[i] = np.cov(data.T, aweights=weights[:, i])

        # update prior probability
        pi[i] = np.mean(weights[:, i])
  # HERE YOU WILL UPDATE VALUES OF mu, sigma and pi
  ##################################################

  ## numpy.cov() can be used to find sigma, i.e., covariance matrix


  return mu, sigma, pi      # updated mu, sigma, pi


def gmm(data, mu, sigma, pi, k, max_iterations=1000):

    for _ in range(max_iterations):

        # update cluter assignment weights
        weights = e_step(data, mu, sigma, pi, k)    # WRITE CODE FOR E-Step

        # update mu, sigma and prior proabilities locations
        mu, sigma, pi = m_step(data, mu, sigma, pi, weights, k)  # WRITE CODE FOR M-Step

    # final assignment update
    weights = e_step(data, mu, sigma, pi, k)
    assignments = np.argmax(weights, axis=1)      # pick cluster with maximum probabili

    return mu, sigma, pi, assignments
```

Question 1.2. Run your code on following toy dataset we provided in the Assignment 4. Run for different k-values, where k = {3, 4} and

1. visualize 2-D gaussian ellipses with $\mu \in \mathbb{R}^2$ and $\Sigma \in \mathbb{R}^{2\times 2}$ you obtained.
2. plot the cluster assignments for different k's (as done in assignment 4) for both GMM and K-Means side by side for comparison. There will total 4 plots, 2 plots for each 'k' value.
3. Write your observations (open question)

In [ ]:
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate sample data
n_samples = 4000
n_components = 4

X, y_true = make_blobs(
    n_samples=n_samples, centers=n_components, cluster_std=0.60, random_state=0
)

colors = ["#4EACC5", "#FF9C34", "#4E9A06", "m"]

for k, col in enumerate(colors):
    cluster_data = y_true == k
    plt.scatter(X[cluster_data, 0], X[cluster_data, 1], c=col, marker=".", s=10)
```

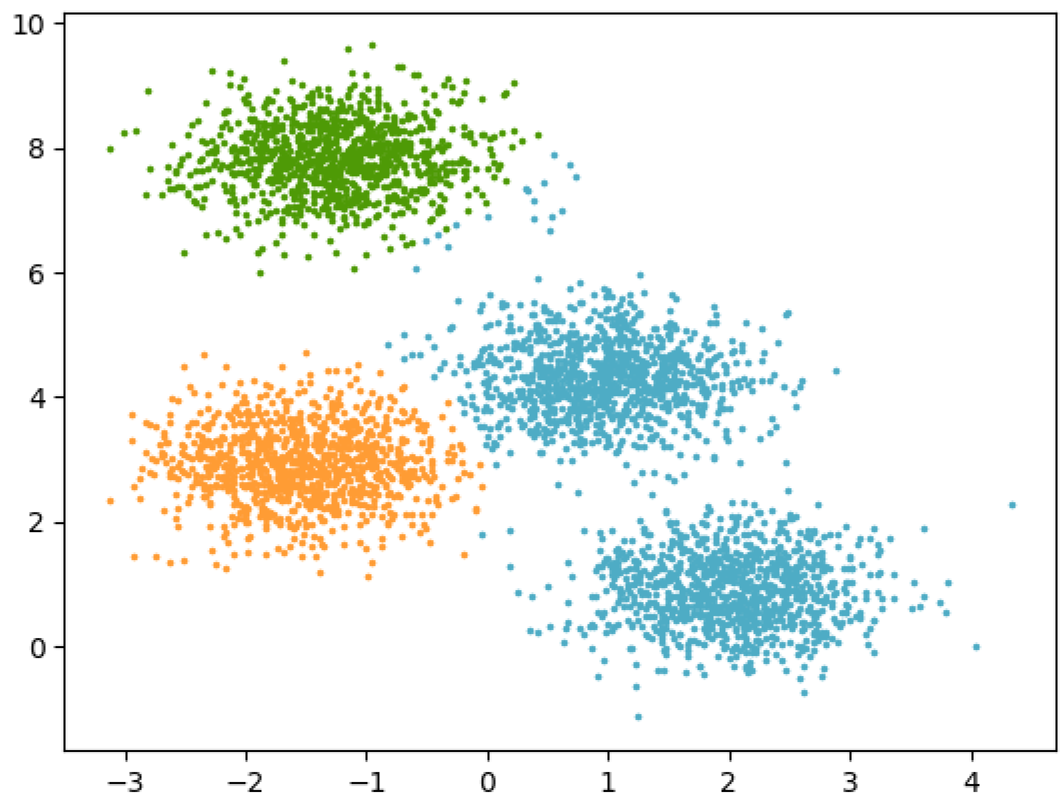


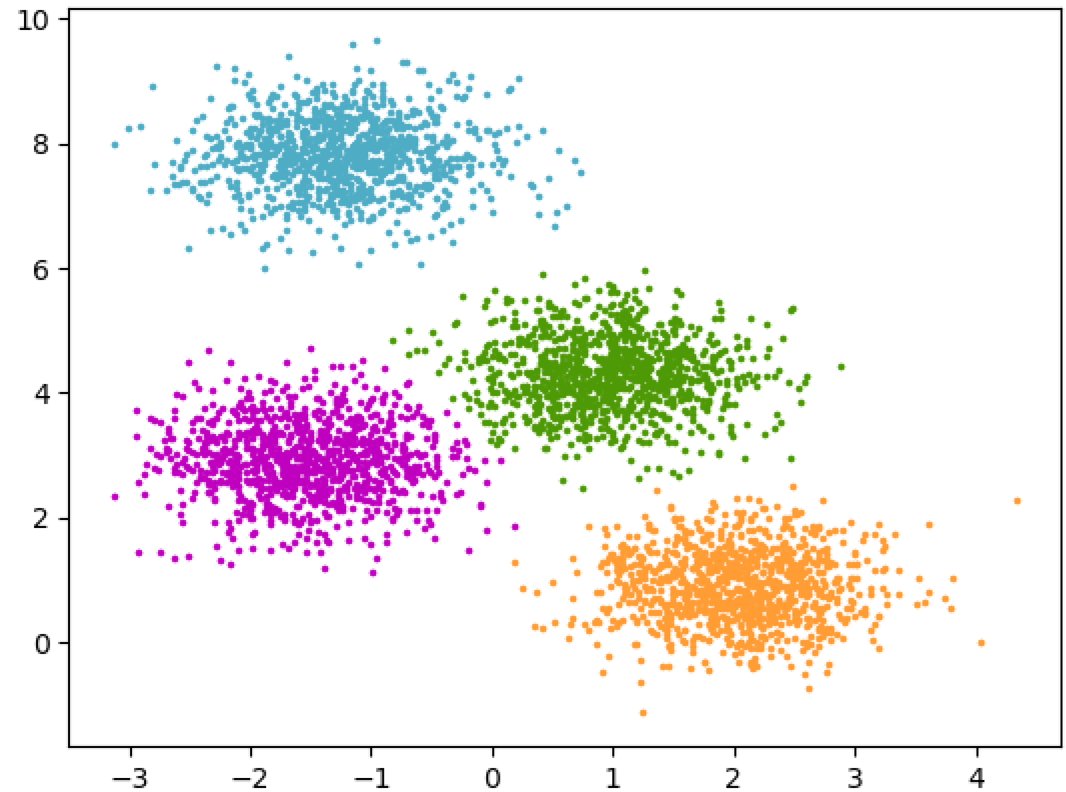
In [ ]:
```
import numpy as np

# function to get initial cluster parameters (mu, sigma and pi)
def get_initial_parameters(k, X):

  # initial weights given to each cluster
  pi = np.full(shape=k, fill_value=1/k)

  # dataset is divided randomly into k parts of unequal sizes
  random_row = np.random.randint(low=0, high=X.shape[0], size=k)

  # initial value of mean of k Gaussians
  mu = [  X[row_index,:] for row_index in random_row ]

  # initial value of covariance matrix of k Gaussians
  sigma = [ np.cov(X.T) for _ in range(k) ]
```

```
  return pi, mu, sigma

### KMEANS FUNCTIONS###
def update_assignments(data, centroids):
  k = np.shape(centroids)[0]
  assignment_list = []
  for i in range(k):
    #calculate norm of the vector
    norm = np.linalg.norm(data-centroids[i], axis=1)
    assignment_list.append(norm)
  #minimum element of the array in the particular axis
  assignments = np.argmin(assignment_list, axis=0)
  return assignments
def update_centroids(data,centroids,assignments):
  k = np.shape(centroids)[0]
  new_centroids = []
  for i in range(k):
    new_centroids.append(data[assignments == i].mean(axis=0))
  new_centroids = np.array(new_centroids)
  return new_centroids

def kmeans(data, centroids, max_iterations):
  for j in range(max_iterations):
    # update cluter assignments
    assignments = update_assignments(data,centroids)

    # update centroid locations
    centroids = update_centroids(data,centroids,assignments)

  # final assignment update
  assignments = update_assignments(data,centroids)
  return centroids, assignments

def get_initial_clusters(k, X):
  random_indices = np.random.randint(0, X.shape[0], k)
  initial_centroids = X[random_indices]
  return initial_centroids
#################


## Main code ##
print("GMM")
for k in [3, 4]:
  pi_initial, mu_initial, sigma_initial = get_initial_parameters(k, X)
  #### GMM #####
  mu_final, sigma_final, pi_final, gmm_cluster_assignments = gmm(X, mu_initial, sigma_i

  for k, col in enumerate(colors):
      cluster_data = gmm_cluster_assignments == k
      plt.scatter(X[cluster_data, 0], X[cluster_data, 1], c=col, marker=".", s=10)
  plt.show()


#### KMEANS ####
print(f"k_means")
for k in [3,4]:
  initial_centroids = get_initial_clusters(k, X)
  final_centroids, kmeans_cluster_assignments = kmeans(X, initial_centroids, max_iterat
```

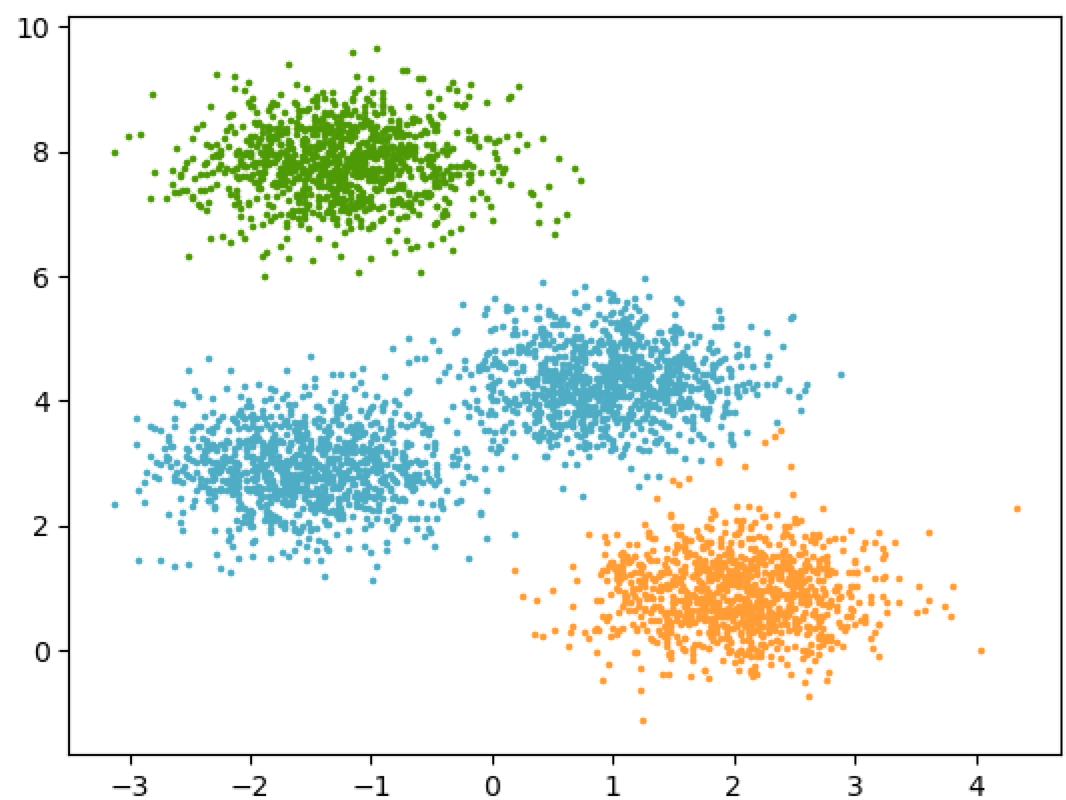
```
  for k, col in enumerate(colors):
    cluster_data = kmeans_cluster_assignments == k
    plt.scatter(X[cluster_data, 0], X[cluster_data, 1], c=col, marker=".", s=10)
  plt.show()

  ############################################################################
  #### Observations on Comparison of GMM vs KMEANS ###########################
print("The ways GMM and K-means start its cluster division is different, but they both p
print("For when k=3 while the cluster number is 4, k-means give a division according to
print("Also from how GMM considers both mean and covariance, it gives a softer assignmen
  ############################################################################
```

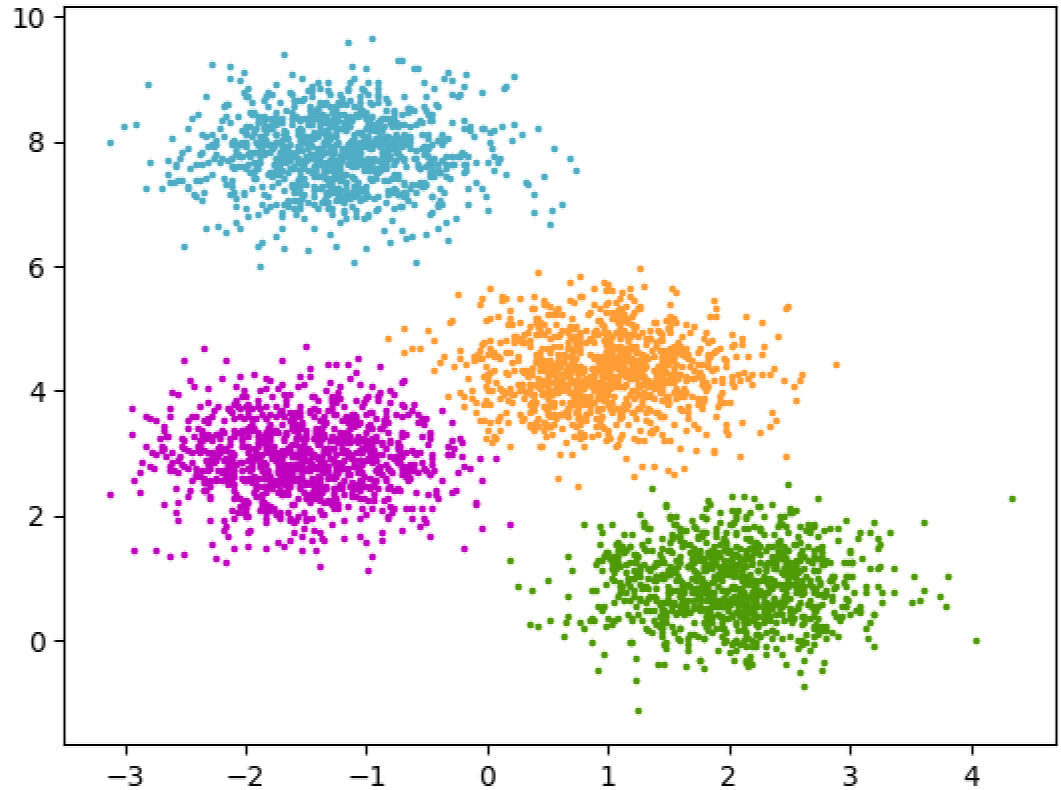
GMM





k_means






```
The ways GMM and K-means start its cluster division is different, but they both perform
well for the correct k.
For when k=3 while the cluster number is 4, k-means give a division according to its cl
oseness as a vector, while GMM gives a division according to its closeness vertically.
Also from how GMM considers both mean and covariance, it gives a softer assignments com
pared to K-means.
```

Question 2. Neural Networks (70 pts)

Consider the following neural network:

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
from IPython.display import Image

Image(filename='/content/drive/My Drive/CSE142/Assignment5/New_NN.png', width=400)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
c:¥ドキュメント¥github¥UCSC-CSE142-MachineLearning¥Assignment5¥Assignment5_CSE142.ipynb Cell 10 line 1
----> <a href='vscode-notebook-cell:/c%3A/%E3%83%89%E3%82%AD%E3%83%A5%E3%83%A1%E3%83%B3%E3%83%88/github/UCSC-CSE142-MachineLearning/Assignment5/Assignment5_CSE142.ipynb#X12sZmlsZQ%3D%3D?line=0'>1</a> from google.colab import drive
      <a href='vscode-notebook-cell:/c%3A/%E3%83%89%E3%82%AD%E3%83%A5%E3%83%A1%E3%83%B3%E3%83%88/github/UCSC-CSE142-MachineLearning/Assignment5/Assignment5_CSE142.ipynb#X12sZmlsZQ%3D%3D?line=1'>2</a> drive.mount('/content/drive')
      <a href='vscode-notebook-cell:/c%3A/%E3%83%89%E3%82%AD%E3%83%A5%E3%83%A1%E3%83%B3%E3%83%88/github/UCSC-CSE142-MachineLearning/Assignment5/Assignment5_CSE142.ipynb#X12sZmlsZQ%3D%3D?line=2'>3</a> from IPython.display import Image

ModuleNotFoundError: No module named 'google'
```

where $a_i = \Sigma_j w_j^i z_j$, $z_i = f_i(a_i)$ for $i = 1, 2, 3, 4, z_5 = a_5$ (an input neuron), and $f_1(x) = f_2(x) = f_3(x) = f_4(x) = \text{sigmoid}(x)$.

Question 2.1. Write a function to simulate the neural network (20 pts)

Answer:

$$a_2 = w_5^2 * z_5, a_3 = w_5^3 * z_5, a_4 = w_5^4 * z_5$$

$$z_2 = f(a_2), z_3 = f(a_3), z_4 = f(a_4)$$

$$a_1 = w_2^1 sigmoid(z_2) + w_3^1 sigmoid(z_3) + w_4^1 sigmoid(z_4)$$

$$z = \Sigma_i sigmoid(a_i)$$

Where

$$i = [2, 3, 4]$$

Therefore

$$z = sigmoid(w_2^1 sigmoid(f(w_5^2 * z_5)) + w_3^1 sigmoid(f(w_5^3 * z_5)) + w_4^1 sigmoid(f(w_5^4 * z_5)))$$

Question 2.2 Deduce the equation to calculate δ_i (the error value per neuron) for all the neurons. (20 pts)

Write a function that given a training sample and the weights of the network calculate δ_i for each neuron.

Hint:

#

SIGMOID

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

$$df(x)/dx = f(x)(1 - f(x))$$

#

LOSS FUNCTION

$$error = 0.5 * (z_1 - target)^2$$

#

DERIVATES

$\partial error/\partial w_3^1 = \partial error/\partial z_1 * \partial z_1/\partial a_1 * \partial a_1/\partial w_3^1$ (using chain rule)

$\delta_5 = \partial error/\partial w_5^3 + \partial error/\partial w_5^2 + \partial error/\partial w_5^4$ (total error from connected neurons)

Answer:

$\delta_1 = \partial error/\partial a_1 = \partial L/ * partial z_1/\partial z_1/\partial a_1 = (z_1 - t)f(a_1) = (z_1 - t) * f(a_1)(1 - f(a_1))$

$$\delta_2 = \partial error/\partial a_2 = \partial error/\partial z_1 * \partial z_1/\partial z_2 * \partial z_2/\partial a_2 = f'(a_2) * w_2^1 * \delta 1$$

$$\delta_3 = \partial error/\partial a_3 = \partial error/\partial z_1 * \partial z_1/\partial z_3 * \partial z_3/\partial a_3 = f'(a_3) * w_3^1 * \delta 1$$

$$\delta_4 = \partial error/\partial a_4 = \partial L/\partial z_1 * \partial z_1/\partial z_4 * \partial z_4/\partial a_4 = f'(a_4) * w_4^1 * \delta 1$$

$\delta_5 = \partial error/\partial w_5^2 + \partial error/\partial w_5^3 + \partial error/\partial w_5^4 = \partial L/\partial a_1 * \partial a_1/\partial w_5^2 + \partial L/\partial a_1 * \partial a_1/\partial w_5^3 +$

Therefore

$$\delta_1 = f(a_1)(1 - f(a_1)) * (z_1 - target)$$

$$\delta_2 = f(a_2)(1 - f(a_2) * w_2^1 * \delta 1$$

$$\delta_3 = f(a_3)(1 - f(a_3) * w_2^1 * \delta 1$$

$$\delta_4 = f(a_4)(1 - f(a_4) * w_2^1 * \delta 1$$

According to the derivative initialization in the question, δ_5 would be:

$$\delta_5 = \delta 1 * (z_2 + z_3 + z_4)$$

According to the derivative from the back propagation on the class slides, δ_5 would be:

$$\delta_5 = \delta_2 * w_5^2 + \delta_3 * w_5^3 + \delta_4 * w_5^4$$

Question 2.3 (15 pts)

Assuming that the weight matrix is:

	1	2	3	4
2	3			
3	-4			
4	-1			
5		-3	2	-10

use the functions from items (a) and (b) to calculate the output of each neuron, z_i, and the error, δ_i, for the following training samples:

x	y
0.0	0.5
1.0	0.1

Answer:

x	y	a_1	a_2	a_3	a_4	a_5	z_1	z_2	z_3	z_4	z_5
0.0	0.5	-1	0.0	0.0	0.0	0.0	0.269	0.5	0.5	0.5	0.0
1.0	0.1	-3.38	-3	2	-10	1.0	0.033	0.047	0.88	4.5×10^{-5}	1.0

x	y	δ_1	δ_2	δ_3	δ_4	δ_5
0.0	0.5	-0.045	-0.034	0.045	0.011	0.080
1.0	0.1	-0.0021	-0.00029	0.00090	9.7×10^{-8}	0.0027

Question 2.4 (15 pts)

Implement a function to train the neural network **from scratch** using the stochastic gradient descent **(mainly you need to implement forward_pass and backward_pass)**. Use this function to train the network with the following training samples:

x	y
-3.0	0.7312
-2.0	0.7339
-1.5	0.7438
-1.0	0.7832
-0.5	0.8903
0.0	0.9820
0.5	0.8114
1.0	0.5937
1.5	0.5219
2.0	0.5049
3.0	0.5002

Plot the evolution of the error and the final predictions of the trained network. Write down the weights of the trained nettwork.

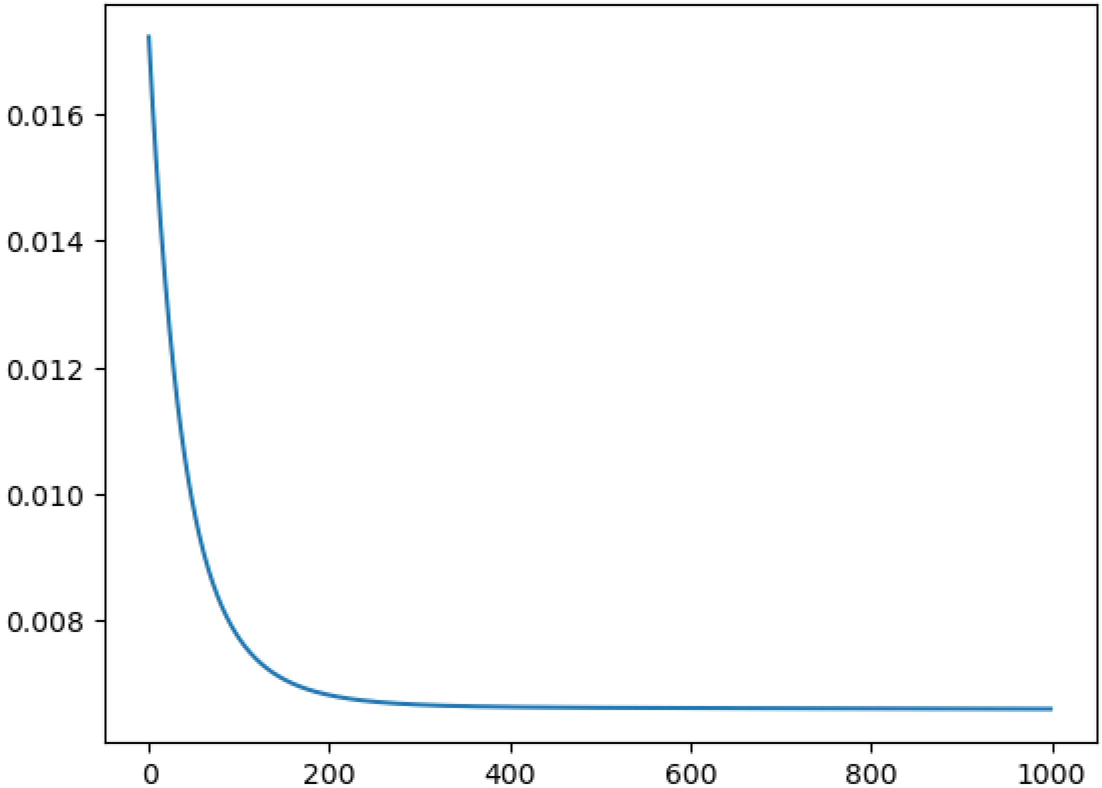
In [ ]:
```
## HELPER CODE##
```

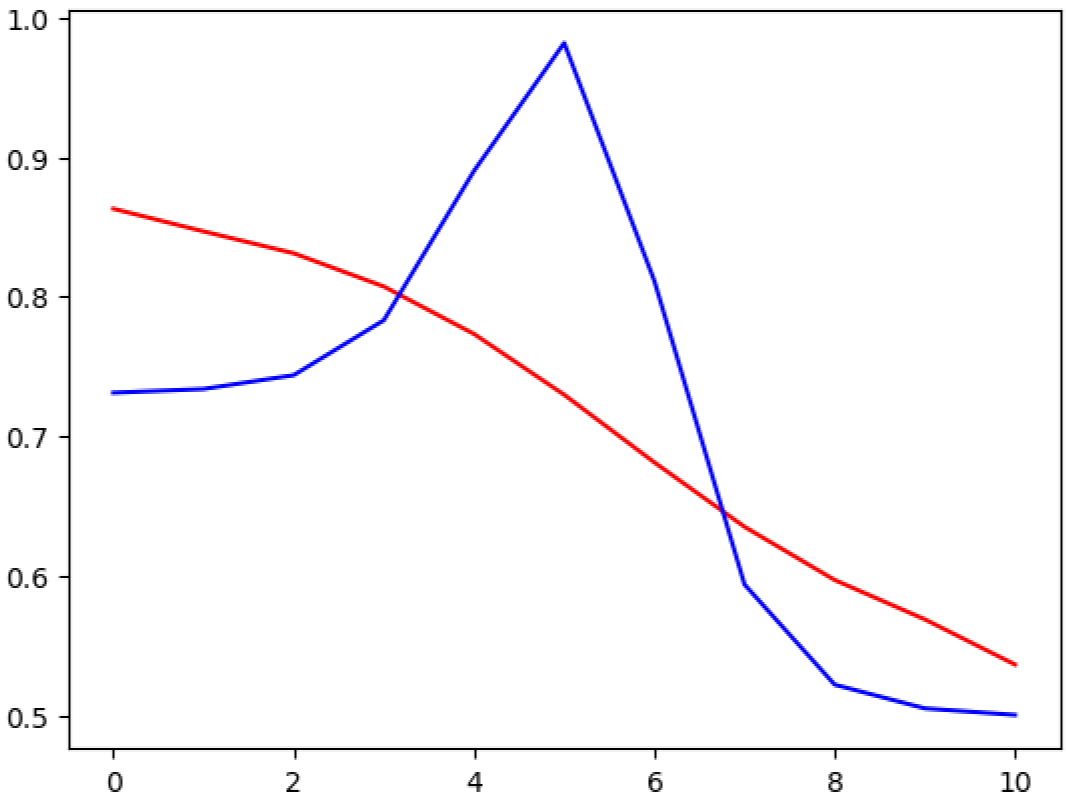
```
import numpy as np
import matplotlib.pyplot as plt

# defining the backprop for Sigmoid Function
def backprop_sigmoid(x):
  return x*(1-x)

# defining the Sigmoid Function
def sigmoid(x):
  return 1 / (1 + np.exp(-x))

def forward_pass(X, weights_input_hidden, weights_hidden_output):

    # calculating hidden layer activations
    # find "hiddenLayer_activations" here

    ########################
    hiddenLayer_activations = sigmoid(np.dot(X, weights_input_hidden))
    ########################


    # calculating the output
    # find "output" here

    ########################
    output = sigmoid(np.dot(hiddenLayer_activations, weights_hidden_output))
    ########################

    return output, hiddenLayer_activations


def backward_pass(X, y, output, weights_hidden_output, weights_input_hidden, hiddenLaye


  # calculating rate of change of error w.r.t weight between hidden and output layer
  # find gradients for w13, w12, w14 and let's store them in "error_wrt_weights_hidden_c

  # NOTE: "error_wrt_weights_hidden_output" will be same size as "weights_hidden_output"

  ########################
  error_output = output - y
  error_wrt_weights_hidden_output = np.dot(hiddenLayer_activations.T, error_output * ba
  ########################


  # calculating rate of change of error w.r.t weights between input and hidden layer
  # find gradients for w35, w45, w25 and let's store them in "error_wrt_weights_input_hi

  # NOTE: "error_wrt_weights_input_hidden" will be same size as "weights_input_hidden"

  ########################
  error_hidden = np.dot(error_output * backprop_sigmoid(output), weights_hidden_output.
  error_wrt_weights_input_hidden = np.dot(X.T, error_hidden * backprop_sigmoid(hiddenLa
  ########################

  return error_wrt_weights_hidden_output, error_wrt_weights_input_hidden


# Train function
def train(X_train, y_train):
```

```
    # defining the model architecture
    inputLayer_neurons = 1  # number of neurons at input
    hiddenLayer_neurons = 3  # number of hidden layers neurons
    outputLayer_neurons = 1  # number of neurons at output layer

    # initializing weight
    weights_input_hidden = np.random.uniform(size=(inputLayer_neurons, hiddenLayer_neur
    weights_hidden_output = np.random.uniform(
        size=(hiddenLayer_neurons, outputLayer_neurons)
    )


    # defining the parameters
    lr = 0.1        # CAN BE CHANGED IF REQUIRED
    epochs = 1000   # CAN BE CHANGED IF REQUIRED

    losses =  []

    for ep in range(epochs):

      output_, hiddenLayer_activations = forward_pass(X_train, weights_input_hidden, we

      ## Backward Propagation
      # calculating error
      error = np.square(output_ - y_train) / 2

      error_wrt_weights_hidden_output, error_wrt_weights_input_hidden = backward_pass(X_
                                                                                       ou
                                                                                       we
                                                                                       we
                                                                                       hi

      # updating the weights
      weights_hidden_output = weights_hidden_output - lr * error_wrt_weights_hidden_out
      weights_input_hidden = weights_input_hidden - lr * error_wrt_weights_input_hidden

      # print error at every 100th epoch
      epoch_loss = np.average(error)
      if ep % 100 == 0:
          print(f"Error at epoch {ep} is {epoch_loss:.5f}")

      # appending the error of each epoch
      losses.append(epoch_loss)

    plt.plot(losses)
    plt.show()

    final_pred, _ = forward_pass(X_train, weights_input_hidden, weights_hidden_output)

    plt.plot(final_pred, c='r')
    plt.plot(y_train, c='b')
    plt.show()

    print("Weights of the network are: ")
    print("w13, w12, w14", weights_hidden_output.T)
    print("w35, w25, w45", weights_input_hidden)


## HELPER CODE ##
```

```python
# defining training data
X_train = np.zeros((11, 1)).astype(np.float32)
y_train = np.zeros((11, 1)).astype(np.float32)

X_train[0] = -3
X_train[1] = -2
X_train[2] = -1.5
X_train[3] = -1.0
X_train[4] = -0.5
X_train[5] = 0.0
X_train[6] = 0.5
X_train[7] = 1.0
X_train[8] = 1.5
X_train[9] = 2.0
X_train[10] = 3.0


y_train[0] = 0.7312
y_train[1] = 0.7339
y_train[2] = 0.7438
y_train[3] = 0.7832
y_train[4] = 0.8903
y_train[5] = 0.9820
y_train[6] = 0.8114
y_train[7] = 0.5937
y_train[8] = 0.5219
y_train[9] = 0.5049
y_train[10] = 0.5002


train(X_train, y_train)
```

```
Error at epoch 0 is 0.01722
Error at epoch 100 is 0.00772
Error at epoch 200 is 0.00683
Error at epoch 300 is 0.00667
Error at epoch 400 is 0.00664
Error at epoch 500 is 0.00663
Error at epoch 600 is 0.00662
Error at epoch 700 is 0.00662
Error at epoch 800 is 0.00661
Error at epoch 900 is 0.00661
```






```
Weights of the network are:
w13, w12, w14 [[ 0.40971061 -0.11156769  1.68888114]]
w35, w25, w45 [[-0.08029881  0.78963782 -1.0612097 ]]
```

In [ ]: