CSE 242 Assignment 5, Fall 2022

2 Questions, 100 pts, due: 23:59 pm, Dec 5th, 2022

`Your name: Nistha Kumar Student ID: 2005437`

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 [1]:
```
from scipy.stats import multivariate_normal
import numpy as np
```

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

  #######################
  #### YOUR CODE HERE ####
  #######################

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

  ## you will find z_ik here (let's call it as "weights")
  likelihood = np.zeros((data.shape[0],k))
  for i in range(k):
    dis=multivariate_normal(mean=mu[i],cov=sigma[i])
    likelihood[:,i] = dis.pdf(data)

  prob = likelihood * pi
  z = prob.sum(axis=1)[:,np.newaxis]
  weights = prob/z

  return weights

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

  ##############################################
  #### YOUR CODE HERE ##############################
  # HERE YOU WILL UPDATE VALUES OF mu, sigma and pi
  ##############################################

  ## numpy.cov() can be used to find sigma, i.e., covariance matrix
  for i in range(k):
    weight = weights[:, [i]]
    t_weight=weight.sum()
    mu[i] = (data * weight).sum(axis=0)/t_weight
    sigma[i]= np.cov(data.T,aweights=(weight/t_weight).flatten(),bias=True)

  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 COD

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

```
    return mu, sigma, pi, assignments

## Reference: https://www.oranlooney.com/post/ml-from-scratch-part-5-gmm/
```

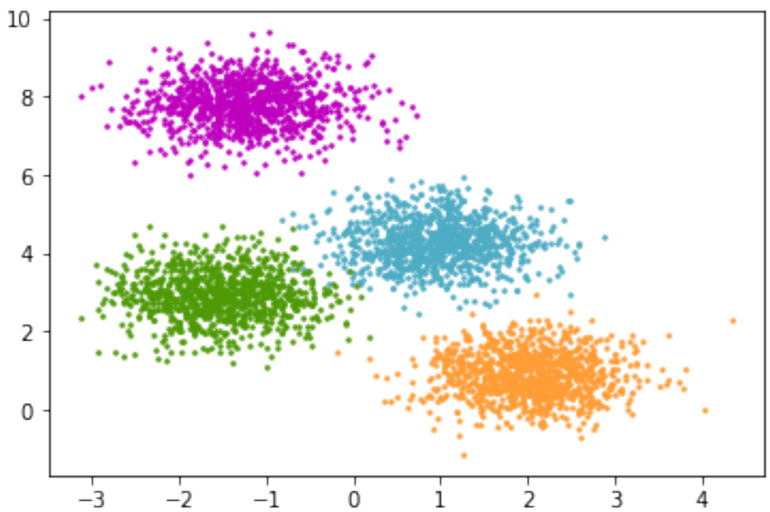
Question 1.2. Run your code on following toy dataset we provided in the Assignment 4. Run for different k-values, where k = {3, 4} and

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 [2]:
```
import numpy as np
from scipy.spatial.distance import cdist
def update_assignments(data, centroids):

  ########################
  #### YOUR CODE HERE ####
  row, col = data.shape
  assignments = np.empty([row])
  distances = cdist(data, centroids)
  assignments=np.argmin(distances,axis=1)


  ## you will get cluster#
  ##assignments here #####
  ########################

  return assignments

def update_centroids(data,centroids,assignments):

  ########################
  #### YOUR CODE HERE ####
  ########################
  K = centroids.shape[0]
  centroids=np.empty(centroids.shape)
  for i in range(K):
    centroids[i]=np.mean(data[assignments ==i], axis=0)

  return centroids


def kmeans(data, centroids, max_iterations):

    for j in range(max_iterations):
        # update cluter assignments
        assignments = update_assignments(data,centroids)    # WRITE CODE FOR

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

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

In [3]:
```
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_stat
)
print(X)
print(y_true)

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
```

```
[[-1.50824765e+00  2.52510964e+00]
 [-2.28628272e+00  7.46823809e+00]
 [-4.68814302e-01  2.10725384e+00]
 ...
 [ 2.12055668e+00  4.17134782e-04]
 [-1.12993421e+00  2.38265914e+00]
 [ 3.08948529e+00  1.32547340e+00]]
[2 3 2 ... 1 2 1]
```

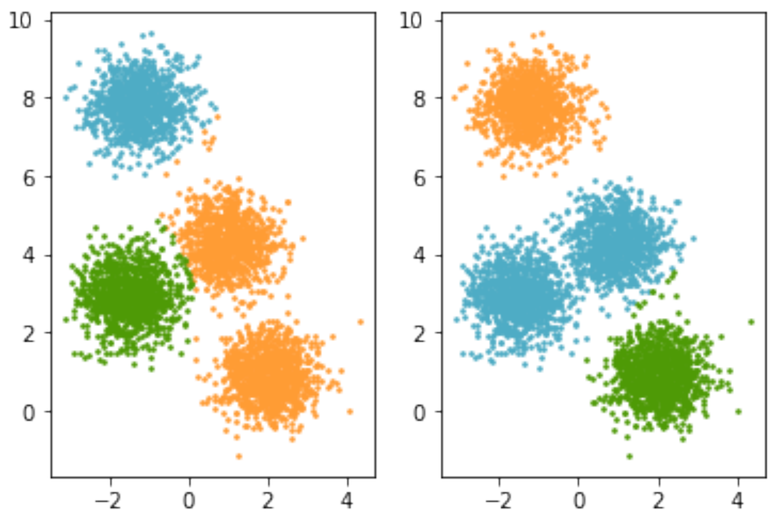


In [4]:
```
import numpy as np

# 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
```

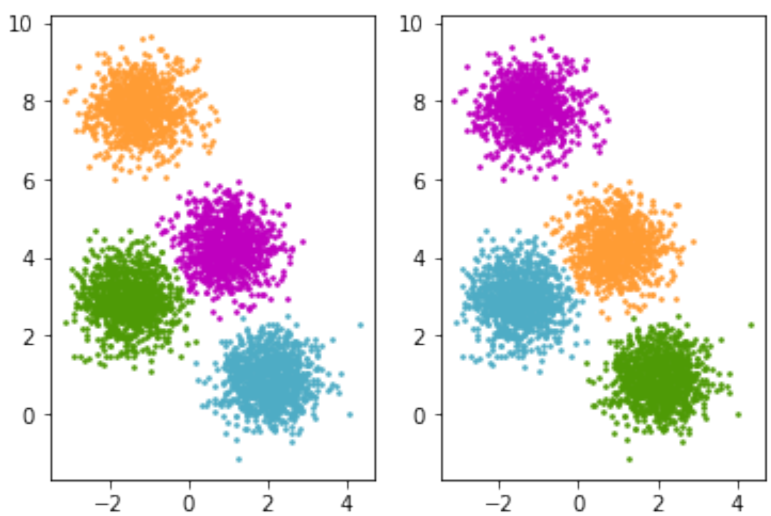
```
  random_row = np.random.randint(low=0, high=X.shape[0], size=k)

  # 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


## Main code
for k in [3, 4]:

  f, (plt1, plt2) = plt.subplots(1, 2)

  pi_initial, mu_initial, sigma_initial = get_initial_parameters(k, X)

  #### GMM #####

  mu_final, sigma_final, pi_final, gmm_cluster_assignments = gmm(X, mu_initi

  for j, col in enumerate(colors):
      cluster_data = gmm_cluster_assignments == j
      plt1.scatter(X[cluster_data, 0], X[cluster_data, 1], c=col, marker="."
  #plt.show()


  ##############################################################
  #### KMEANS #####
  #### Use your implementation of KMEANS from assignment 4 ####
  ##############################################################

  # kmeans_cluster_assignments = your_kmeans_implementation()

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

    return initial_centroids

# your code here.
#clusters=[3,4]
#colors = ["#4EACC5", "#FF9C34", "#4E9A06", "m"]
#fig, axs = plt.subplots(len(clusters), figsize=[8,30])
#for idx, i in enumerate(clusters):
  centroids=get_initial_clusters(k, X)
  centroids,assignments=kmeans(X,centroids,100)
    #print(assignments.shape)
  for j,col in enumerate(colors):
      cluster_data = assignments == j
      plt2.scatter(X[cluster_data, 0], X[cluster_data, 1], c=col, marker="."
        #axs[idx].scatter(X[cluster_data, 0], X[cluster_data, 1], color=colo
```

```
    print("Plotting for k : "+str(k))
    plt.show()

    # 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=".
    # plt.show()

    ##########################################################################
    #### Observations on Comparison of GMM vs KMEANS ########################
    ##########################################################################
```

```
Plotting for k : 3
```



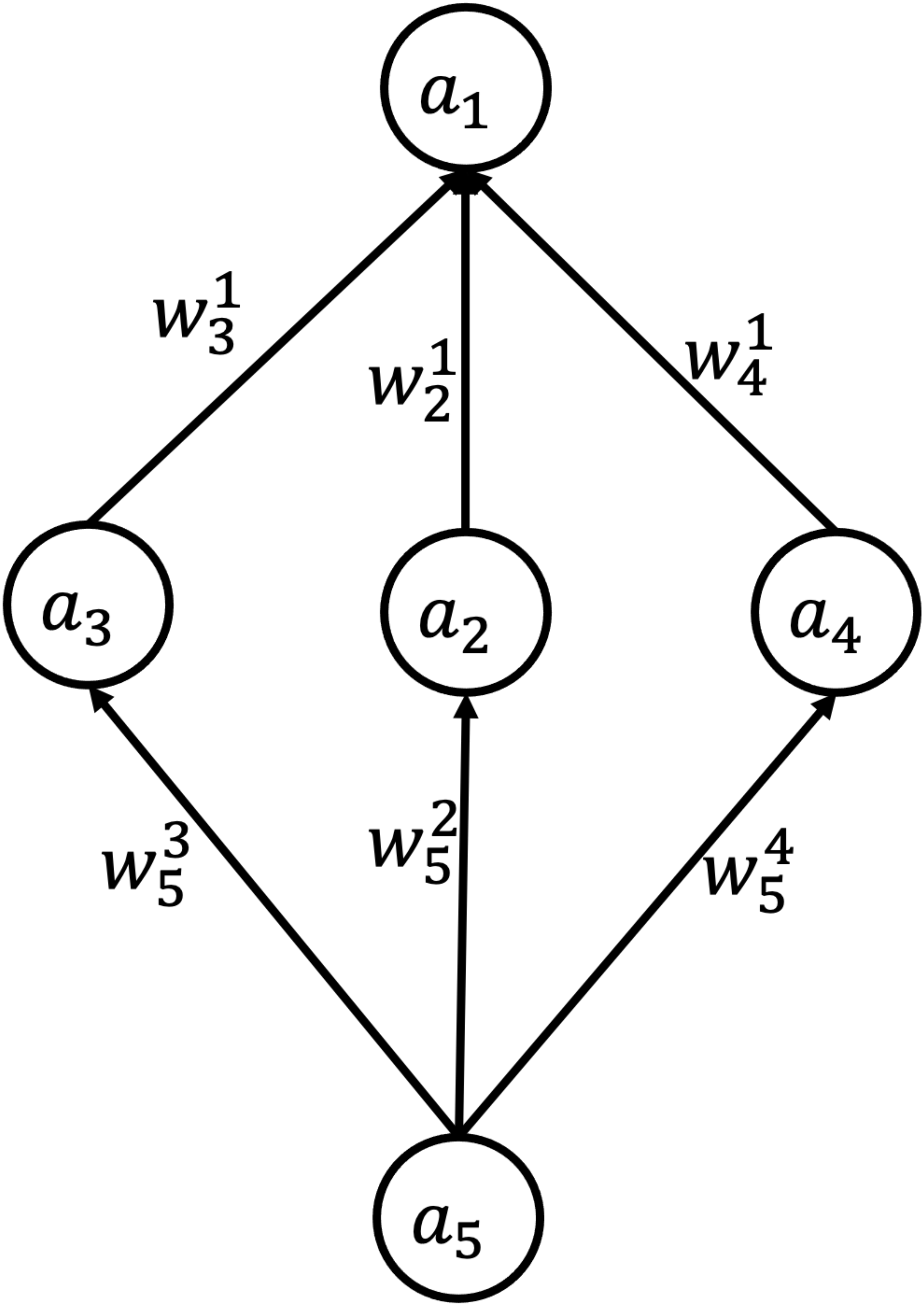
```
Plotting for k : 4
```



Question 2. Neural Networks (70 pts)

Consider the following neural network:

In [5]:
```
#from google.colab import drive
#drive.mount('/content/drive')
from IPython.display import Image
a = Image(filename='New_NN.png')
display(a)

#Image(filename='/content/drive/My Drive/CSE242/Assignment5/New_NN.png', wic
```


a_1
w_3^1
w_2^1
w_4^1
a_3
a_2
a_4
w_5^3
w_5^2
w_5^4
a_5

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:

In [6]:
```
Image(filename='2.1.jpg')
```

Out[6]:

2.1 $a_i = \Sigma_j w_j^i z_j$, $z_i = f_i(a_i)$ for $i = 1,2,3,4$, $z_5 = a_5$ (input neuron)

$f_1(x) = f_2(x) = f_3(x) = f_4(x) = \text{sigmoid } \$(x)\$$.

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

$$a_3 = \sum_j w_j^3 z_j = W_5^3 a_5 \quad [\text{As } z_5 = a_5]$$

$$\boxed{z_3 = \frac{1}{1+e^{a_3}} = \frac{1}{1+e^{-W_5^3 a_5}}}$$

$$a_4 = \sum_j W_j^4 z_j = W_5^4 a_5$$

$$\boxed{z_4 = \frac{1}{1+e^{a_4}} = \frac{1}{1+e^{-W_5^3 a_5}}}$$

$$a_2 = \sum_j w_j^2 z_j = w_5^2 a_5 \qquad \boxed{z_2 = \frac{1}{1+e^{a_2}} = \frac{1}{1+e^{-W_5^2 a_5}}}$$

$$a_1 = \sum_j w_j z_j = w_3^1 z_3 + w_2^1 z_2 + w_4^1 z_4$$

$$\boxed{z_1 = \frac{1}{1+e^{a_1}} = \frac{1}{1+e^{-(w_3^1 z_3 + w_2^1 z_2 + w_4^1 z_4)}}}$$

function which simulates the neural network is

$$\text{func}(a_5) = z_i =$$

$$\boxed{z_1 = \frac{1}{1+e^{-\left[\frac{w_3^1}{1+e^{W_5^3 a_5}} + \frac{w_2^1}{1+e^{W_5^2 a_5}} + \frac{w_4^1}{1+e^{W_5^4 a_5}}\right]}}}$$

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:

In [7]:
```
Image(filename='2.2_1.jpg')
```

Out[7]:

2.2

$$error = 0.5 \times (z_1 - target)^2$$

$$\delta_1 = \frac{\partial error}{\partial a_1} = \frac{\partial error}{\partial z_1} \times \frac{\partial z_1}{\partial a_1} = (z_1 - target) \times \left(\frac{\partial f(a_1)}{\partial a_1} \right)$$

$$= (z_1 - target) \times f(a_1)(1 - f(a_1))$$

Thus $\boxed{\delta_1 = (z_1 - target) \times [z_1(1 - z_1)]}$

$$\frac{\delta error}{\delta w_2'} = \frac{\partial error}{\partial a_1} \frac{\partial a_1}{\partial w_2'} = \delta_1 z_2$$

$$\frac{\delta error}{\delta w_3'} = \frac{\partial error}{\partial a_3} \frac{\partial a_3}{\partial w_3'} = \delta_1 z_3$$

$$\frac{\delta error}{\delta w_4'} = \frac{\partial error}{\partial a_4} \frac{\partial a_4}{\partial w_4'} = \delta_1 z_4$$

$$\delta_2 = \frac{\partial error}{\partial a_2} = \frac{\partial error}{\partial z_2} \frac{\partial z_2}{\partial a_2} = \left(\frac{\partial error}{\partial a_1} \frac{\partial a_1}{\partial z_2} \right) \frac{\partial z_2}{\partial a_2}$$

Thus $\boxed{\delta_2 = \delta_1 w_2' (z_2(1 - z_2))}$

$$\delta_3 = \frac{\partial error}{\partial a_3} = \frac{\partial error}{\partial z_3} \frac{\partial z_3}{\partial a_3} = \left(\frac{\partial error}{\partial a_1} \frac{\partial a_1}{\partial z_3} \right) \frac{\partial z_3}{\partial a_3}$$

$\Rightarrow \boxed{\delta_3 = \delta_1 w_3' (z_3(1 - z_3))}$

In [8]:
```
Image(filename='2.2_2.jpg')
```

Out[8]:

$$\delta_4 = \frac{\partial error}{\partial a_4} = \frac{\partial error}{\partial z_4}\frac{\partial z_4}{\partial a_4} = \left(\frac{\partial error}{\partial a_1}\frac{\partial a_1}{\partial z_4}\right)\frac{\partial z_4}{\partial a_4}$$

$$\Rightarrow \quad \delta_4 = \delta_1 w_3^1 (z_4(1-z_4))$$

Now, $\dfrac{\partial error}{\partial w_5^3} = \dfrac{\partial error}{\partial a_3}\dfrac{\partial a_3}{\partial w_5^3} = \delta_3 z_5$

$$\frac{\partial error}{\partial w_5^4} = \frac{\partial error}{\partial a_4}\frac{\partial a_4}{\partial w_5^4} = \delta_4 z_5$$

$$\frac{\partial error}{\partial w_5^2} = \frac{\partial error}{\partial a_2}\frac{\partial a_2}{\partial w_5^2} = \delta_2 z_5$$

Since $\delta_5 = \dfrac{\partial error}{\partial w_5^3} + \dfrac{\partial error}{\partial w_5^4} + \dfrac{\partial error}{\partial w_5^2}$

$$\Rightarrow \quad \delta_5 = \delta_3 z_5 + \delta_4 z_5 + \delta_2 z_5$$

Thus $\delta_5 = z_5(\delta_3 + \delta_4 + \delta_2)$

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	0.1										

x	y	δ_1	δ_2	δ_3	δ_4	δ_5
0.0	0.5					
1.0	0.1					

In [9]:
```
Image(filename='2.3_1.jpg')
```

Out[9]:

2.3

$w_5^2 = -3 \qquad w_5^3 = 2 \qquad w_5^4 = -10$

$w_4^1 = -1 \qquad w_3^1 = -4 \qquad w_2^1 = 3$

for $x = 0.0$ & $y = 0.5$: $z_5 = a_5 = x = 0.0$

$$a_3 = w_5^3 a_5 = 2 \times 0 = 0 \Rightarrow \boxed{a_3 = 0}$$

$$z_3 = \frac{1}{1+e^{-a_3}} = \frac{1}{1+e^{0}} = 0.5 \Rightarrow \boxed{z_3 = 0.5}$$

$$a_4 = w_5^4 a_5 = 0 \Rightarrow \boxed{a_4 = 0}$$

$$z_4 = \frac{1}{1+e^{-a_4}} = \frac{1}{1+1} = 0.5 \Rightarrow \boxed{z_4 = 0.5}$$

$$a_2 = w_5^2 a_5 = 0 \Rightarrow \boxed{a_2 = 0}$$

$$z_2 = \frac{1}{1+e^{-a_2}} = \frac{1}{1+1} = 0.5 \Rightarrow \boxed{z_2 = 0.5}$$

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

$$= -4(0.5) + 3(0.5) + (-1)(0.5) = -2(0.5) = -1$$

$$\Rightarrow \boxed{a_1 = -1}$$

$$z_1 = \frac{1}{1+e^{-a_1}} = \frac{1}{1+e} \approx 0.268 \Rightarrow \boxed{z_1 = 0.268}$$

In [10]:
```
Image(filename='2.3_2.jpg')
```

Out[10]:

For $x = 1, \; y = 0.1$

$$z_5 = a_5 = x = 1$$

$$a_3 = w_5^3 a_5 = 2 \times 1 = 2 \quad \Rightarrow \boxed{a_3 = 2}$$

$$z_3 = \frac{1}{1+e^{-a_3}} = \frac{1}{1+e^{-2}} = 0.8807 \quad \boxed{z_3 = 0.8807}$$

$$a_4 = w_5^4 a_5 = -10(1) = -10 \quad \Rightarrow \boxed{a_4 = -10}$$

$$z_4 = \frac{1}{1+e^{-a_4}} = \frac{1}{1+e^{10}} \quad \Rightarrow \boxed{z_4 = 0.0000453}$$

$$a_2 = w_5^2 a_5 = -3(1) \quad \Rightarrow \boxed{a_2 = -3}$$

$$z_2 = \frac{1}{1+e^{-a_2}} = \frac{1}{1+e^{-(-1)}} \quad \Rightarrow \boxed{z_2 = 0.0474}$$

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

$$= -4(0.8807) + 3(0.0474) + (-1)(0.0000453)$$

$$\Rightarrow \boxed{a_1 = -3.3806}$$

$$z_1 = \frac{1}{1+e^{-a_1}} = 0.032907$$

$$\Rightarrow \boxed{z_1 = 0.032907}$$

In [11]:
```
Image(filename='2.3_3.jpg')
```

Out[11]:

Final tables are:

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	0.268	0.5	0.5	0.5	0
1.0	0.1	-3.3806	-3	2	-10	1	0.032907	0.9474	0.8807	0.0000453	1

Calculating δ values

$w_4^1 = -1$, $w_3^1 = -4$, $w_2^1 = 3$, $w_5^2 = -3$, $w_5^3 = 2$, $w_5^4 = -10$

for $x = 0$, $y = 0.5$

$$\delta_1 = (z_1 - \text{target})(z_1(1-z_1)) = (0.268 - 0.5)(0.268)(1-0.268)$$

$$\Rightarrow \boxed{\delta_1 = -0.0455}$$

$$\delta_2 = \delta_1 w_2^1 z_2(1-z_2) = (-0.0455)(3)(0.5)(1-0.5)$$

$$\Rightarrow \boxed{\delta_2 = -0.03412}$$

$$\delta_3 = \delta_1 w_3^1 z_3(1-z_3) = (-0.0455)(-4)(0.5)(1-0.5)$$

$$\boxed{\delta_3 = 0.0455}$$

$$\delta_4 = \delta_1 w_4^1 z_4(1-z_4) = (-0.0455)(-1)(0.5)(1-0.5)$$

$$\boxed{\delta_4 = 0.011375}$$

In [12]:

```
Image(filename='2.3_4.jpg')
```

Out[12]:

$$\delta_5 = z_5(\delta_3 + \delta_4 + \delta_2) = 0(\delta_3 + \delta_4 + \delta_2)$$

$$\Rightarrow \boxed{\delta_5 = 0}$$

For $x = 1.0,\ y = 0.1$:

$$\delta_1 = (z_1 - \text{target})(z_1(1 - z_1))$$
$$= (0.032907 - 0.1)(0.0329(1 - 0.0329))$$

$$\boxed{\delta_1 = -0.002134}$$

$$\delta_2 = \delta_1 w_2' z_2(1 - z_2) = (-0.002134)(3)(0.0474)(1 - 0.0474)$$

$$\boxed{\delta_2 = -2.8907 \times 10^{-4}}$$

$$\delta_4 = \delta_1 w_4' z_4(1 - z_4) w_4'$$

$$= (-0.002134)(-1)(0.0000453)(1 - 0.0000453)$$

$$\Rightarrow \boxed{\delta_4 = 0.966658 \times 10^{-8}}$$

$$\delta_3 = \delta_1 w_3' z_3(1 - z_3) = (-0.002134)(-4)(0.8807)(1 - 0.8807)$$

$$\boxed{\delta_3 = 0.00089685}$$

$$\delta_5 = z_5(\delta_3 + \delta_4 + \delta_2) = 1(0.00089685 + 0.96665 \times 10^{-8} + -2.8907 \times 10^{-4})$$

$$\Rightarrow \boxed{\delta_5 = 6.077 \times 10^{-4}}$$

In [13]:

```
Image(filename='2.3_5.jpg')
```

Out[13]:

final table for δ :

x	y	δ_1	δ_2	δ_3	δ_4	δ_5
0	0.5	-0.0455	-0.03412	0.0455	0.011375	0
1.0	0.1	-0.002134	-2.807×10^{-4}	0.00089685	0.96×10^{-8}	6.077×10^{-4}

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 [14]:
```
## 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

    ########################
    #### YOUR CODE HERE ####
```

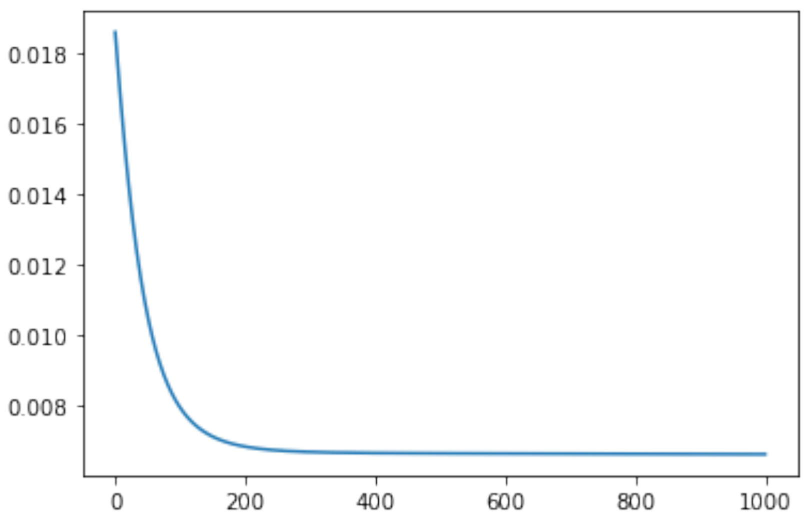
```
    hiddenLayer_linearTransformation = np.dot(weights_input_hidden.T, X.T)
    hiddenLayer_activations = sigmoid(hiddenLayer_linearTransformation)

    ########################


    # calculating the output
    # find "output" here

    ########################
    #### YOUR CODE HERE ####

    outputLayer_linearTransformation = np.dot(
        weights_hidden_output.T, hiddenLayer_activations
    )
    output = sigmoid(outputLayer_linearTransformation)

    ########################

    return output, hiddenLayer_activations


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


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

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

  ########################
  #### YOUR CODE HERE ####

  error_wrt_output = -(y.T - output)

  output_wrt_outputLayer_LinearTransform = np.multiply(output, (1 - output))
  outputLayer_LinearTransform_wrt_weights_hidden_output = hiddenLayer_activa

  error_wrt_weights_hidden_output = np.dot(
    outputLayer_LinearTransform_wrt_weights_hidden_output,
    (error_wrt_output * output_wrt_outputLayer_LinearTransform).T,
  )
  ########################


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

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

  ########################
  #### YOUR CODE HERE ####

```

```
    outputLayer_LinearTransform_wrt_hiddenLayer_activations = weights_hidden_c
    hiddenLayer_activations_wrt_hiddenLayer_linearTransform = np.multiply(
      hiddenLayer_activations, (1 - hiddenLayer_activations)
    )
    hiddenLayer_linearTransform_wrt_weights_input_hidden = X.T
    error_wrt_weights_input_hidden = np.dot(
      hiddenLayer_linearTransform_wrt_weights_input_hidden,
      (
          hiddenLayer_activations_wrt_hiddenLayer_linearTransform
          * np.dot(
              outputLayer_LinearTransform_wrt_hiddenLayer_activations,
              (output_wrt_outputLayer_LinearTransform * error_wrt_output),
          )
      ).T,
    )

    ########################

    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, hidde
    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

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

      error_wrt_weights_hidden_output, error_wrt_weights_input_hidden = back
```

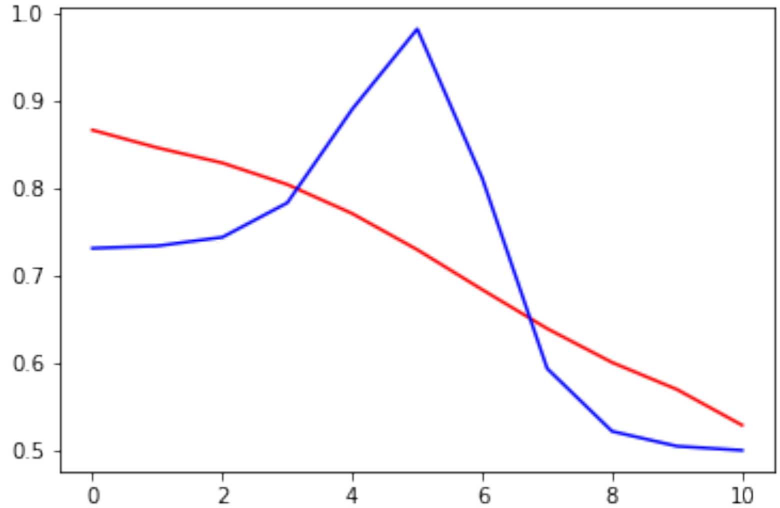
```
        # updating the weights
        weights_hidden_output = weights_hidden_output - lr * error_wrt_weights
        weights_input_hidden = weights_input_hidden - lr * error_wrt_weights_i

        # 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_hidd

    print("jhh")
    print(final_pred)

    plt.plot(final_pred.T, 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 ##

# 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 0.01858
Error at epoch 100 is 0.00795
Error at epoch 200 is 0.00685
Error at epoch 300 is 0.00670
Error at epoch 400 is 0.00667
Error at epoch 500 is 0.00666
Error at epoch 600 is 0.00666
Error at epoch 700 is 0.00665
Error at epoch 800 is 0.00665
Error at epoch 900 is 0.00664
```




```
jhh
[[0.86637688 0.84611671 0.82878111 0.80415039 0.77087621 0.72960145
  0.68393988 0.63940064 0.60065372 0.56972766 0.52894417]]
```




```
Weights of the network are:
w13, w12, w14 [[-0.15638164  0.69934992  1.44223465]]
w35, w25, w45 [[ 0.9382966  -0.3224773  -0.98765751]]
```