ROLLNO:225229105

Lab1. Python Functions and Numpy

Part-I: Write a method for sigmoid function

In [1]:
```
import math

def mysigmoid(x):
    return 1 / (1 + math.exp(-x))
```

In [3]:
```
mysigmoid(4)
```

Out[3]:
```
0.9820137900379085
```

In [6]:
```

x = [1, 2, 3]
sigmoid_values = [mysigmoid(i) for i in x]
print("The sigmoid values of", x, "are", sigmoid_values)
```

```
The sigmoid values of [1, 2, 3] are [0.7310585786300049, 0.8807970779778823, 0.
9525741268224334]
```

In [16]:
```
import numpy as np

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

x =np.array([1, 2, 3])
sigmoid_values = mysigmoid(x)
print("The sigmoid values of", x, "are", sigmoid_values)
```

```
The sigmoid values of [1 2 3] are [0.73105858 0.88079708 0.95257413]
```

Part-II: Gradient or derivative of sigmoid function

In [18]:
```
import numpy as np

def mysigmoid(X):
    return 1 / (1 + np.exp(-X))

def sig_derivative(X):
    s = mysigmoid(X)
    return s * (1 - s)
```

In [19]:
```
gradient = sig_derivative(2)
print("The gradient of s for X=2 is", gradient)
```

```
The gradient of s for X=2 is 0.10499358540350662
```

Part-III: Write a method image_to_vector()

In [25]:
```
# GRADED FUNCTION: image2vector
def image2vector(image):
    ### START CODE HERE ### (≈ 1 line of code)
    v = image.reshape(image.shape[0]*image.shape[1]*image.shape[2],1)
    ### END CODE HERE ###

    return v
```

In [26]:
```
image = np.array([[[ 0.67826139,  0.29380381],
        [ 0.90714982,  0.52835647],
        [ 0.4215251 ,  0.45017551]],

       [[ 0.92814219,  0.96677647],
        [ 0.85304703,  0.52351845],
        [ 0.19981397,  0.27417313]],

       [[ 0.60659855,  0.00533165],
        [ 0.10820313,  0.49978937],
        [ 0.34144279,  0.94630077]]])

print ("image2vector(image) = " + str(image2vector(image)))
```

```
image2vector(image) = [[0.67826139]
 [0.29380381]
 [0.90714982]
 [0.52835647]
 [0.4215251 ]
 [0.45017551]
 [0.92814219]
 [0.96677647]
 [0.85304703]
 [0.52351845]
 [0.19981397]
 [0.27417313]
 [0.60659855]
 [0.00533165]
 [0.10820313]
 [0.49978937]
 [0.34144279]
 [0.94630077]]
```

Part-IV: Write a method normalizeRows()

In [27]:
```
# GRADED FUNCTION: normalizeRows

def normalizeRows(x):


    ### START CODE HERE ### (≈ 2 lines of code)
    # Compute x_norm as the norm 2 of x. Use np.linalg.norm(..., ord = 2, axis =
    x_norm = np.linalg.norm(x, ord = 2, axis = 1, keepdims = True)

    # Divide x by its norm.
    x = x / x_norm
    ### END CODE HERE ###

    return x
```

In [28]:
```
x = np.array([
    [0, 3, 4],
    [1, 6, 4]])
print("normalizeRows(x) = " + str(normalizeRows(x)))
```

```
normalizeRows(x) = [[0.         0.6        0.8       ]
 [0.13736056 0.82416338 0.54944226]]
```

Part-V: Multiplication and Vectorization Operations

In [30]:
```
import numpy as np

# Input vectors
x1 = np.array([9, 2, 5])
x2 = np.array([7, 2, 2])

# Compute multiplication and dot product using NumPy
mul = np.multiply(x1, x2) # equivalent to x1 * x2
dot = np.dot(x1, x2)

print("Multiplication:", mul)
print("Dot product:", dot)
import numpy as np
# Input vectors
x1 = np.array([9, 2, 5, 0, 0, 7, 5, 0, 0, 0, 9, 2, 5, 0, 0, 4, 5, 7])
x2 = np.array([7, 2, 2, 9, 0, 9, 2, 5, 0, 0, 9, 2, 5, 0, 0, 8, 5, 3])
```

```
Multiplication: [63  4 10]
Dot product: 77
```

In [32]:
```
import timeit
N = 1000000

a = np.random.rand(N)
b = np.random.rand(N)

start_time = timeit.default_timer()

result = np.zeros(N)
for i in range(N):
    result[i] = a[i] * b[i]

for_loop_time = timeit.default_timer() - start_time

start_time = timeit.default_timer()

result = np.multiply(a, b)

vectorization_time = timeit.default_timer() - start_time

print('For loop time: {:.6f} seconds'.format(for_loop_time))
print('Vectorization time: {:.6f} seconds'.format(vectorization_time))
```

```
For loop time: 0.451486 seconds
Vectorization time: 0.003417 seconds
```

Part-VI: Implement L1 and L2 loss functions

In [37]:
```
import numpy as np

def loss_l1(y, ypred):
    return np.sum(np.abs(y - ypred))

# Test the function with given vectors
y = np.array([1, 0, 0, 1, 1])
ypred = np.array([.9, 0.2, 0.1, .4, .9])
print("L1 loss:", loss_l1(y, ypred))
```

```
L1 loss: 1.1
```

In [38]:
```
import numpy as np

def loss_l2(y, ypred):
    return np.sum(np.square(y - ypred))

# Test the function with given vectors
y = np.array([1, 0, 0, 1, 1])
ypred = np.array([.9, 0.2, 0.1, .4, .9])
print("L2 loss:", loss_l2(y, ypred))
```

```
L2 loss: 0.43
```

In [ ]: