Python Functions and Numpy

Part-I: Write a method for sigmoid function

```
1. Write a function, mysigmoid(x), that takes the real number x and returns
the sigmoid value using math.exp().
```

In [3]:
```
import math

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

```
2. Call mysigmoid() with x=4 and print the sigmoid value of 4.
```

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

Out[4]:
```
0.9820137900379085
```

```
3. Now, find the sigmoid values for x=[1, 2, 3]. Observe the results.
```

In [7]:
```
import numpy as np
x = np.array([1, 2, 3])
print(np.exp(x))
```

```
[ 2.71828183  7.3890561  20.08553692]
```

```
4. Rewrite mysigmoid() using np.exp() function.
```

In [12]:
```
import numpy as np
def mysigmoid(x):
    s = 1./(1+np.exp(-x))
    return s
```

```
5. Now call your function with x=[1, 2, 3] and observe the results
```

In [13]:
```
x = np.array([1, 2, 3])
mysigmoid(x)
```

Out[13]:
```
array([0.73105858, 0.88079708, 0.95257413])
```

Part-II: Gradient or derivative of sigmoid function

In [ ]:

In [19]:
```
def sig_derivative(s):
    t = mysigmoid(s)
    ds = t*(1-t)
    return ds
```

In [21]:
```
s = np.array([1, 2, 3])
print ("sig_derivative(s) = " + str(sig_derivative(s)))
```

```
sig_derivative(s) = [0.19661193 0.10499359 0.04517666]
```

Part-III: Write a method image_to_vector()

In [24]:
```
def image2vector(image):
    v = image.reshape(image.shape[0]*image.shape[1]*image.shape[2],1)
    return v
```

In [25]:
```
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 [26]:
```
def normalizeRows(x):
    x_norm = np.linalg.norm(x, ord = 2, axis = 1, keepdims = True)
    x = x / x_norm
    return x
```

In [27]:
```
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

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

mul_result = np.multiply(x1, x2)
print("Multiplication:", mul_result)

dot_result = np.dot(x1, x2)
print("Dot product:", dot_result)
```

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

In [31]:
```
import numpy as np

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])

mul_result = np.multiply(x1, x2)
print("Multiplication:", mul_result)

dot_result = np.dot(x1, x2)
print("Dot product:", dot_result)
```

```
Multiplication: [63  4 10  0  0 63 10  0  0  0 81  4 25  0  0 32 25 21]
Dot product: 338
```

In [32]:
```
import numpy as np
import time

N = 1000000
x1 = np.random.random(N)
x2 = np.random.random(N)

start_time = time.time()
mul_result = np.multiply(x1, x2)
end_time = time.time()
mul_time = end_time - start_time

start_time = time.time()
dot_result = np.dot(x1, x2)
end_time = time.time()
dot_time = end_time - start_time

print("Multiplication time:", mul_time)
print("Vectorization (dot product) time:", dot_time)
```

```
Multiplication time: 0.0039899349212646484
Vectorization (dot product) time: 2.3989293575286865
```

Part-VI: Implement L1 and L2 loss functions

In [38]:
```
import numpy as np
actual=np.array([1, 0, 0, 1, 1])
prediction=np.array([.9, 0.2, 0.1, .4, .9])
```

In [39]:
```
l1_loss=np.sum(abs(actual - prediction))
print(l1_loss)
```

```
1.1
```

In [42]:
```
def loss_l2(y, ypred):
    return np.sum((y - ypred) ** 2)
y = np.array([1, 0, 0, 1, 1])
ypred = np.array([.9, 0.2, 0.1, .4, .9])
print(loss_l2(y, ypred))
```

```
0.43
```

In [ ]: