PDL Lab1: Python Functions and Numpy

In [21]:
```
# DINESH KUMAR_225229108
```

In [1]:
```
# PART-I Write a method for sigmoid function
```

In [2]:
```
#1. Write a function, mysigmoid(x), that takes the real number x and returns the sigmoid value using math.exp
import math
def mysigmoid(x):
    return 1 / (1 + math.exp(-x))
```

In [3]:
```
#2. Call mysigmoid() with x=4 and print the sigmoid value of 4
x = 4
sig= mysigmoid(x)
print("Sigmoid value of 4:", sig)
```

```
Sigmoid value of 4: 0.9820137900379085
```

In [4]:
```
#3. Now, find the sigmoid values for x=[1, 2, 3]. Observe the results.
x = [1, 2, 3]
sig = [mysigmoid(val) for val in x]
print("Sigmoid values for x=[1, 2, 3]:", sig)
```

```
Sigmoid values for x=[1, 2, 3]: [0.7310585786300049, 0.8807970779778823, 0.9525741268224334]
```

In [5]:
```
#4. Rewrite mysigmoid() using np.exp() function
import numpy as np
def mysigmoid(x):
    return 1 / (1 + np.exp(-x))
```

In [6]:
```
#5. Now call your function with x=[1, 2, 3] and observe the results
x = [1, 2, 3]
sig = [mysigmoid(val) for val in x]
print("Sigmoid values for x=[1, 2, 3] ", sig)
```

```
Sigmoid values for x=[1, 2, 3]  [0.7310585786300049, 0.8807970779778823, 0.9525741268224334]
```

In [7]:
```
#PART-II: Gradient or derivative of sigmoid function
```

In [8]:
```
def sig_derivative(s):
    return s * (1 - s)
x = 5
sigm=mysigmoid(x)
gradient = sig_derivative(sigm)
print("Gradient of sigmoid(4):", gradient)
```

```
Gradient of sigmoid(4): 0.006648056670790033
```

In [9]:
```
#Part-III: Write a method image_to_vector()
```

```
In [10]: def image_to_vector(image):
             length, height, channels = image.shape
             vector = image.reshape((length * height * channels, 1))
             return vector
         image = np.array([
             [[1, 2, 3], [4, 5, 6]],
             [[7, 8, 9], [10, 11, 12]],
             [[13, 14, 15], [16, 17, 18]]])
         vector = image_to_vector(image)
         print("Vector shape:", vector.shape)
         print("Vector:")
         print(vector)
```

```
Vector shape: (18, 1)
Vector:
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]
 [11]
 [12]
 [13]
 [14]
 [15]
 [16]
 [17]
 [18]]
```

In [11]:
```python
def image_to_vector(image):
    length, height, channels = image.shape
    vector = image.reshape((length * height * channels, 1))
    return vector
image = np.array([
    [[255, 0, 0], [0, 255, 0], [0, 0, 255]],
    [[255, 255, 0], [255, 0, 255], [0, 255, 255]],
    [[128, 128, 128], [64, 64, 64], [192, 192, 192]]
])
vector = image_to_vector(image)
print("Vector shape:", vector.shape)
print("Vector:")
print(vector)
```

```
 [255]
 [255]
 [255]
 [  0]
 [255]
 [  0]
 [255]
 [  0]
 [255]
 [255]
 [128]
 [128]
 [128]
 [ 64]
 [ 64]
 [ 64]
 [192]
 [192]
 [192]]
```

In [12]:
```python
#Part-IV: Write a method normalizeRows()
```

In [13]:
```python
def normalizeRows(x):
    norms = np.linalg.norm(x, axis=1, keepdims=True)
    return x / norms
x = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
normalized_x = normalizeRows(x)
print("Normalized matrix:")
print(normalized_x)
```

```
Normalized matrix:
[[0.26726124 0.53452248 0.80178373]
 [0.45584231 0.56980288 0.68376346]
 [0.50257071 0.57436653 0.64616234]]
```

In [14]:
```python
#Part-V: Multiplication and Vectorization Operations
```

In [15]:
```python
#1)A)
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 [16]:
```
#B)
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_fun = np.multiply(x1, x2)
print("Multiplication:", mul_fun)
dot_fun = np.dot(x1, x2)
print("Dot product:", dot_fun)
```

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

In [17]:
```
#2)
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: 0.0
Vectorization (dot product) time: 0.20307421684265137
```

In [18]:
```
#Part-VI: Implement L1 and L2 loss functions
```

In [19]:
```
#1)
def loss_l1(y, ypred):
    return np.sum(np.abs(y - ypred))
y = np.array([1, 0, 0, 1, 1])
ypred = np.array([0.9, 0.2, 0.1, 0.4, 0.9])
l1_loss = loss_l1(y, ypred)
print("L1 Loss:", l1_loss)
```

```
L1 Loss: 1.1
```

In [20]:
```
#2)
def loss_l2(y, ypred):
    return np.sum(np.square(y - ypred))
y = np.array([1, 0, 0, 1, 1])
ypred = np.array([0.9, 0.2, 0.1, 0.4, 0.9])
l2_loss = loss_l2(y, ypred)
print("L2 Loss:", l2_loss)
```

```
L2 Loss: 0.43
```