Lab1. Python Functions and Numpy

Name:P.Asha Belcilda

Rollno:225229104

In [12]:
```
import math
import numpy as np
```

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 [16]:
```
def mysigmoid(x):
    return 1 / (1 + math.exp(-x))
```

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

In [18]:
```
x=4
sig=mysigmoid(x)
print("The sigmoid value of 4 is:",sig)
```

```
The sigmoid value of 4 is: 0.9820137900379085
```

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

In [22]:
```
x=[1,2,3]
sig=[mysigmoid(val) for val in x]
print("Sigmoid values for x=[1,2,3] are:",sig)
```

```
Sigmoid values for x=[1,2,3] are: [0.7310585786300049, 0.8807970779778823, 0.
9525741268224334]
```

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

In [21]:
```
def sigmoid(x):
    return 1/(1+np.exp(x))
```

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

In [23]:
```python
x=[1,2,3]
sig=[mysigmoid(val) for val in x]
print("Sigmoid values for x=[1,2,3] are:",sig)
```

```
Sigmoid values for x=[1,2,3] are: [0.7310585786300049, 0.8807970779778823, 0.
9525741268224334]
```

6. Understand the difference between scalar and vector.

```
The quantity, which has only magnitude and no direction, is termed as a
scalar quantity.
For example length, mass, speed, etc are some of the examples of scalar.
```

In [ ]:
```python
The physical quantity, which comprises of both magnitude and direction, is ter
For example, velocity, momentum, force, etc are some of the examples of scalar
```

Part-II: Gradient or derivative of sigmoid function

We compute gradients to optimize loss functions using backpropagation. s_derivative = s * (1 – s), where s = sigmoid(X) Write a function, sig_derivative(s) that returns the gradient of s. You should call your earlier function, mysigmoid() to compute the sigmoid value.

In [26]:
```python
def sig_derivative(s):
    return s * (1 - s)
x = 7
sigm=mysigmoid(x)
gradient = sig_derivative(sigm)
print("Gradient of sigmoid(4):", gradient)
```

```
Gradient of sigmoid(4): 0.000910221180121784
```

Part-III: Write a method image_to_vector()

1). Now, implement image_to_vector() that takes an input of shape (length, height, 3) and returns a vector of shape (length*height*3, 1). For example, if you would like to reshape an array v of shape (a, b, c) into a vector of shape (a*b,c) you would do: v = v.reshape((v.shape[0]*v.shape[1], v.shape[2])) #v.shape[0] = a ; v.shape[1] = b ; v.shape[2] = c

In [27]:
```python
def image_to_vector(image):
    length, height, channels = image.shape
    vector = image.reshape((length * height * channels, 1))
    return vector
```

Example:

In [28]:
```
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]]
```

2). Test with a 3x3x2 array of values of RGB color values.

In [29]:
```
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)
```

```
Vector shape: (27, 1)
Vector:
[[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]]
```

Part-IV: Write a method normalizeRows()

Implement normalizeRows() to normalize the rows of a matrix. After applying this function to an input matrix x, each row of x should be a vector of unit length (meaning length 1). You can use np.linalg.norm() method.

In [30]:
```python
def normalizeRows(x):
    norms = np.linalg.norm(x, axis=1, keepdims=True)
    return x / norms

# Example usage:
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]]
```

Part-V: Multiplication and Vectorization Operations

1). For the following two vectors, find the multiplication value and the dot product. First compute multiplication and dot product manually. Then, you can use np.multiply() and np.dot() functions.

a). x1 = [9, 2, 5] x2 = [7, 2, 2]

In [32]:
```python
x1 = np.array([9, 2, 5])
x2 = np.array([7, 2, 2])

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

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

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

1)b). x1 = [9, 2, 5, 0, 0, 7, 5, 0, 0, 0, 9, 2, 5, 0, 0, 4, 5, 7] x2 = [7, 2, 2, 9, 0, 9, 2, 5, 0, 0, 9, 2, 5, 0, 0, 8, 5, 3]

In [33]:
```
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
multiplication = np.multiply(x1, x2)
print("Multiplication:", multiplication)

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

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

2). Create two random vectors of N elements, perform multiplication and vectorization operations and print the respective running times. Is running time of vectorization is less?.

In [35]:
```
import time

N = 1000000  # No.of elements
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: 0.0029921531677246094
Vectorization (dot product) time: 1.1173417568206787
```

Part-VI: Implement L1 and L2 loss functions

1). Write a method loss_l1(y, ypred) that takes the actual value y and predicted value ypred and returns l1 loss value. Test your function with the following vectors. y = np.array([1, 0, 0, 1, 1]) ypred = np.array([.9, 0.2, 0.1, .4, .9])

In [4]:
```
import numpy as np
y = np.array([1, 0, 0, 1, 1])
ypred = np.array([0.9, 0.2, 0.1, 0.4, 0.9])
l1_loss=np.sum(abs(y-ypred))

print("L1 Loss:", l1_loss)
```

```
L1 Loss: 1.1
```

2). Write a method loss_l2(y, ypred) that takes the actual value y and predicted value ypred and returns l2 loss value. Test your function with the following vectors. y = np.array([1, 0, 0, 1, 1]) ypred = np.array([.9, 0.2, 0.1, .4, .9])

In [9]:
```
y = np.array([1, 0, 0, 1, 1])
ypred = np.array([0.9, 0.2, 0.1, 0.4, 0.9])

l2_loss = np.sum((y - ypred) ** 2)
print("L2 Loss:", l2_loss)
```

```
L2 Loss: 0.43
```