Name -> ROHAN NYATI

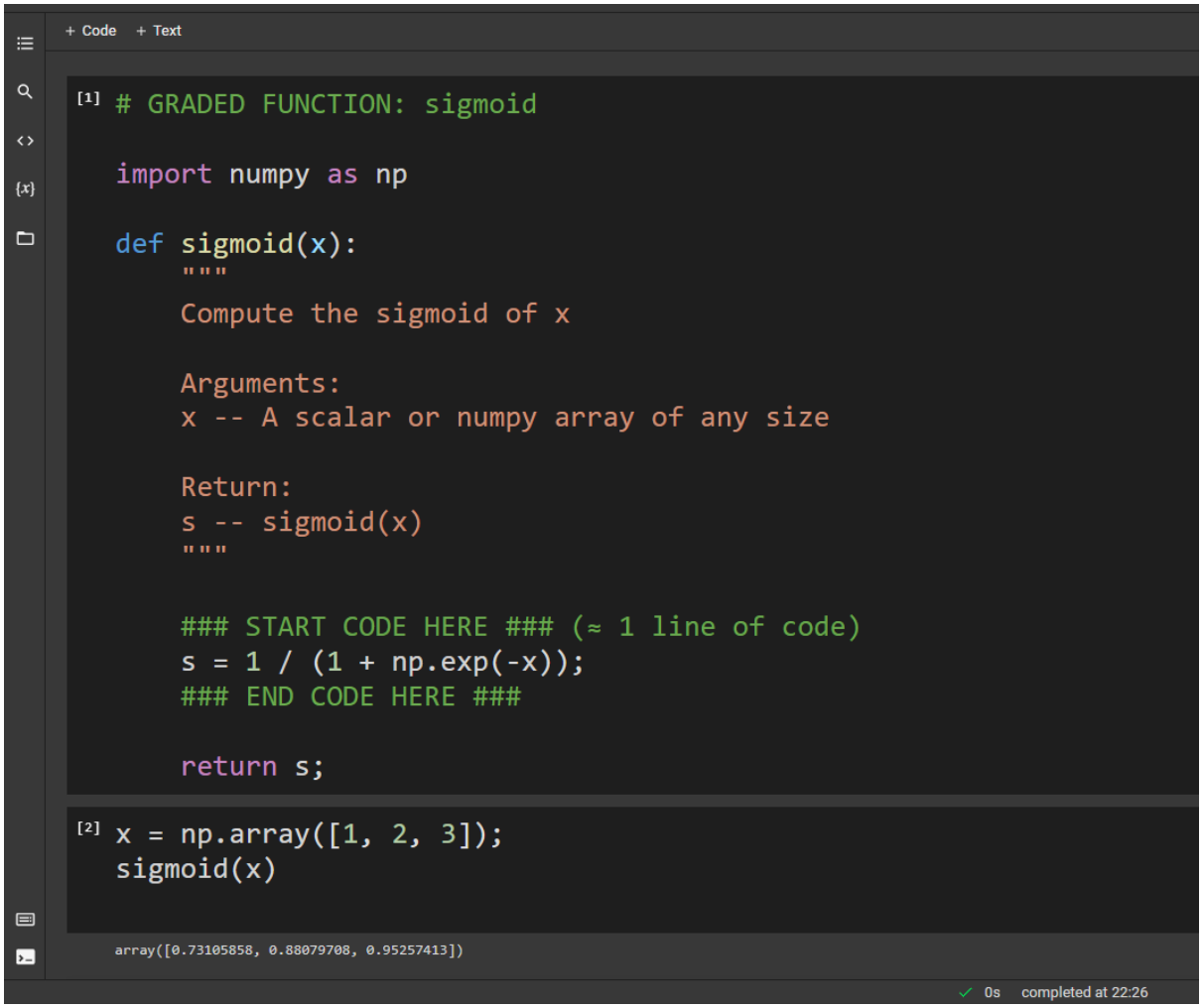
Roll No -> R177219148

Sap Id -> 500075940

Course -> B-Tech CSE AI&ML B5

Experiment -> 3

1) Exercise: Implement the sigmoid function using numpy.
Instructions: x could now be either a real number, a vector, or a matrix.



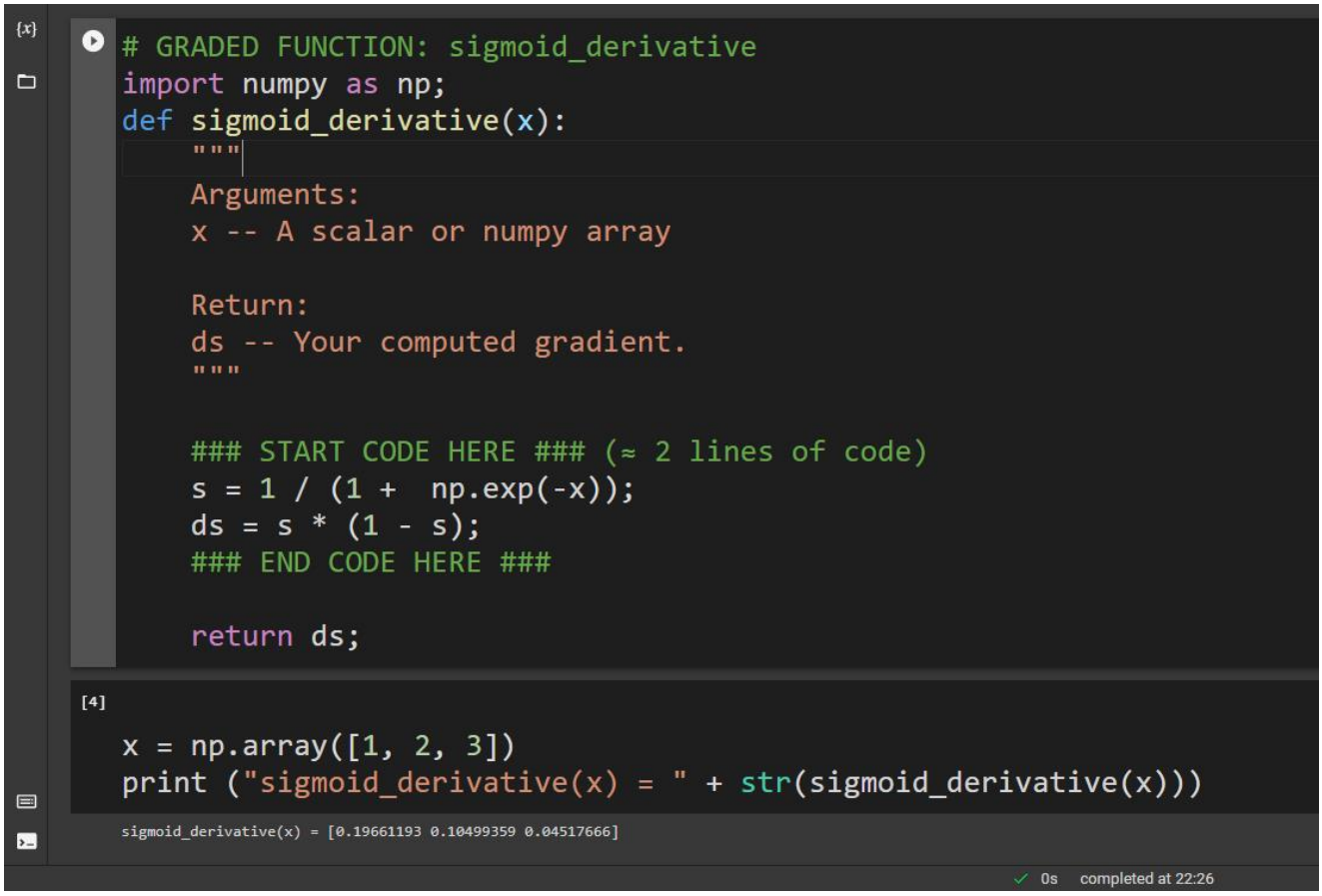
2) Exercise: Implement the function sigmoid_grad() to compute the gradient of the sigmoid function with respect to its input x.

The formula is: sigmoid_derivative(x) = σ ′ (x) = σ(x)(1 − σ(x)) (1.1.2)

Code this function in two steps:

1. **Set s to be the sigmoid of x. You might find your sigmoid(x) function useful.**
2. **Compute σ ′ (x) = s(1 − s)**

⇨



```
# GRADED FUNCTION: sigmoid_derivative
import numpy as np;
def sigmoid_derivative(x):
    """
    Arguments:
    x -- A scalar or numpy array

    Return:
    ds -- Your computed gradient.
    """

    ### START CODE HERE ### (≈ 2 lines of code)
    s = 1 / (1 +  np.exp(-x));
    ds = s * (1 - s);
    ### END CODE HERE ###

    return ds;
```

```
[4]
x = np.array([1, 2, 3])
print ("sigmoid_derivative(x) = " + str(sigmoid_derivative(x)))
```

```
sigmoid_derivative(x) = [0.19661193 0.10499359 0.04517666]
```

0s completed at 22:26

3) 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).

⇨

```
+ Code  + Text

[5] # GRADED FUNCTION: normalizeRows

def normalizeRows(x):
    """
    Implement a function that normalizes each row of the matrix x (to have unit length).

    Argument:
    x -- A numpy matrix of shape (n, m)

    Returns:
    x -- The normalized (by row) numpy matrix. You are allowed to modify x.
    """

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

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

    return x;
```

4) Implement a softmax function using numpy. You can think of softmax as a normalizing function used when your algorithm needs to classify two or more classes.

⇨

```
+ Code  + Text

[7] # GRADED FUNCTION: softmax
import numpy as np;

def softmax(x):
    """Calculates the softmax for each row of the input x.

    Your code should work for a row vector and also for matrices of shape (n, m).

    Argument:
    x -- A numpy matrix of shape (n,m)

    Returns:
    s -- A numpy matrix equal to the softmax of x, of shape (n,m)
    """

    ### START CODE HERE ### (≈ 3 lines of code)
    # Apply exp() element-wise to x. Use np.exp(...).
    x_exp = np.exp(x);

    # Create a vector x_sum that sums each row of x_exp. Use np.sum(..., axis = 1, keepdims = True).
    x_norm = np.linalg.norm(x_exp, axis = 1, keepdims = True);

    # Compute softmax(x) by dividing x_exp by x_sum. It should automatically use numpy broadcasting.
    s = x_exp / x_norm;

    return s;

✓ 0s  completed at 22:23
```

```
[8]
x = np.array([
    [9, 2, 5, 0, 0],
    [7, 5, 0, 0 ,0]])
print(softmax(x));
```

```
[[9.99831880e-01 9.11728660e-04 1.83125597e-02 1.23389056e-04
  1.23389056e-04]
 [9.90964875e-01 1.34112512e-01 9.03642998e-04 9.03642998e-04
  9.03642998e-04]]
```

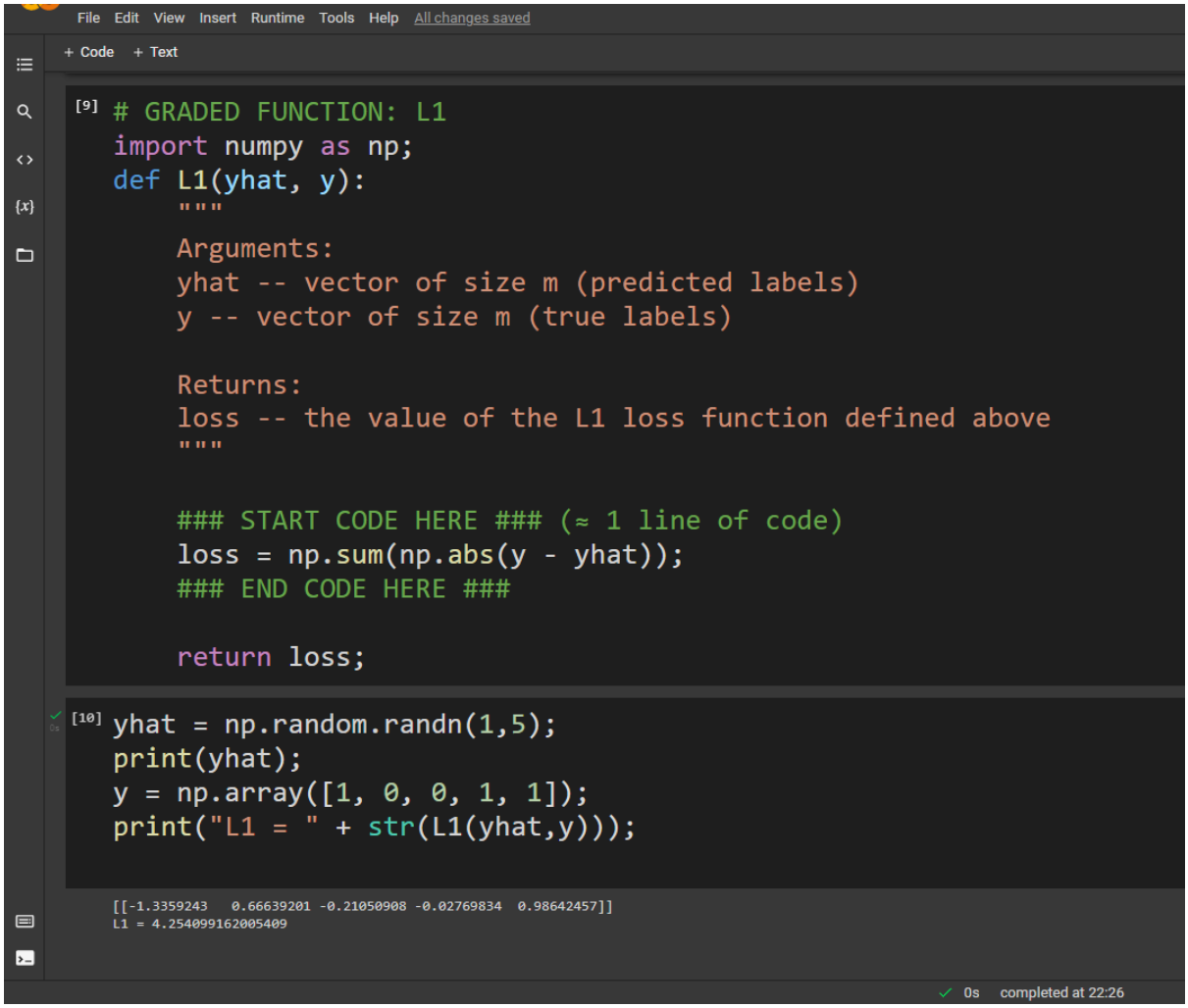
5) Implement the numpy vectorized version of the L1 loss. You may find the function abs(x) (absolute value of x) useful.

Reminder:

- **The loss is used to evaluate the performance of your model. The bigger your loss is, the more different your predictions ($\hat{y}$) are from the true values (y).**

  In deep learning, you use optimization algorithms like Gradient Descent to train your model and to minimize the cost.
- **L1 loss is defined as: $L_1(\hat{y}, y) = \sum_{i=0}^{m} |y^{(i)} - \hat{y}^{(i)}|$**

⇨


File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[9] # GRADED FUNCTION: L1
import numpy as np;
def L1(yhat, y):
"""
Arguments:
yhat -- vector of size m (predicted labels)
y -- vector of size m (true labels)
Returns:
loss -- the value of the L1 loss function defined above
"""
START CODE HERE ### (≈ 1 line of code)
loss = np.sum(np.abs(y - yhat));
END CODE HERE
return loss;
[10] yhat = np.random.randn(1,5);
print(yhat);
y = np.array([1, 0, 0, 1, 1]);
print("L1 = " + str(L1(yhat,y)));
[[-1.3359243 0.66639201 -0.21050908 -0.02769834 0.98642457]]
L1 = 4.254099162005409
0s completed at 22:26