Python+Basics+With+Numpy+v3

January 27, 2018

1 Python Basics with Numpy (optional assignment)

Welcome to your first assignment. This exercise gives you a brief introduction to Python. Even if you've used Python before, this will help familiarize you with functions we'll need.

Instructions: - You will be using Python 3. - Avoid using for-loops and while-loops, unless you are explicitly told to do so. - Do not modify the (# GRADED FUNCTION [function name]) comment in some cells. Your work would not be graded if you change this. Each cell containing that comment should only contain one function. - After coding your function, run the cell right below it to check if your result is correct.

After this assignment you will: - Be able to use iPython Notebooks - Be able to use numpy functions and numpy matrix/vector operations - Understand the concept of "broadcasting" - Be able to vectorize code

Let's get started!

1.1 About iPython Notebooks

iPython Notebooks are interactive coding environments embedded in a webpage. You will be using iPython notebooks in this class. You only need to write code between the ### START CODE HERE ### and ### END CODE HERE ### comments. After writing your code, you can run the cell by either pressing "SHIFT"+"ENTER" or by clicking on "Run Cell" (denoted by a play symbol) in the upper bar of the notebook.

We will often specify "(X lines of code)" in the comments to tell you about how much code you need to write. It is just a rough estimate, so don't feel bad if your code is longer or shorter.

Exercise: Set test to `"Hello World"` in the cell below to print "Hello World" and run the two cells below.

```
In [1]: ### START CODE HERE ### ( 1 line of code)
        test = "Hello World"
        ### END CODE HERE ###

In [2]: print ("test: " + test)

test: Hello World
```

Expected output: test: Hello World

What you need to remember: - Run your cells using SHIFT+ENTER (or "Run cell") - Write code in the designated areas using Python 3 only - Do not modify the code outside of the designated areas

1.2 1 - Building basic functions with numpy

Numpy is the main package for scientific computing in Python. It is maintained by a large community (www.numpy.org). In this exercise you will learn several key numpy functions such as np.exp, np.log, and np.reshape. You will need to know how to use these functions for future assignments.

1.2.1 1.1 - sigmoid function, np.exp()

Before using np.exp(), you will use math.exp() to implement the sigmoid function. You will then see why np.exp() is preferable to math.exp().

Exercise: Build a function that returns the sigmoid of a real number x. Use math.exp(x) for the exponential function.

Reminder: $sigmoid(x) = \frac{1}{1+e^{-x}}$ is sometimes also known as the logistic function. It is a nonlinear function used not only in Machine Learning (Logistic Regression), but also in Deep Learning.

To refer to a function belonging to a specific package you could call it using package_name.function(). Run the code below to see an example with math.exp().

In [3]:
```
# GRADED FUNCTION: basic_sigmoid

import math

def basic_sigmoid(x):
    """
    Compute sigmoid of x.

    Arguments:
    x -- A scalar

    Return:
    s -- sigmoid(x)
    """

    ### START CODE HERE ### ( 1 line of code)
    s = 1/(1+math.exp(-x))
    ### END CODE HERE ###

    return s
```

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

Out[4]:
```
0.9525741268224334
```

Expected Output:

```
<tr>
<td>** basic_sigmoid(3) **</td>
    <td>0.9525741268224334 </td>
</tr>
```

Actually, we rarely use the "math" library in deep learning because the inputs of the functions are real numbers. In deep learning we mostly use matrices and vectors. This is why numpy is more useful.

```
In [5]: ### One reason why we use "numpy" instead of "math" in Deep Learning ###
        x = [1, 2, 3]
        basic_sigmoid(x) # you will see this give an error when you run it, because x is a vecto


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-5-2e11097d6860> in <module>()
          1 ### One reason why we use "numpy" instead of "math" in Deep Learning ###
          2 x = [1, 2, 3]
    ----> 3 basic_sigmoid(x) # you will see this give an error when you run it, because x is a v


        <ipython-input-3-951c5721dbfa> in basic_sigmoid(x)
         15
         16     ### START CODE HERE ### ( 1 line of code)
    ---> 17     s = 1/(1+math.exp(-x))
         18     ### END CODE HERE ###
         19


        TypeError: bad operand type for unary -: 'list'
```

In fact, if $ x = (x_1, x_2, ..., x_n)$ is a row vector then $np.exp(x)$ will apply the exponential function to every element of x. The output will thus be: $np.exp(x) = (e^{x_1}, e^{x_2}, ..., e^{x_n})$

```
In [6]: import numpy as np

        # example of np.exp
        x = np.array([1, 2, 3])
        print(np.exp(x)) # result is (exp(1), exp(2), exp(3))
```

```
[  2.71828183   7.3890561   20.08553692]
```

Furthermore, if x is a vector, then a Python operation such as $s = x + 3$ or $s = \frac{1}{x}$ will output s as a vector of the same size as x.

```
In [7]: # example of vector operation
        x = np.array([1, 2, 3])
        print (x + 3)
```

```
[4 5 6]
```

```
In [8]: np.exp?
```

Any time you need more info on a numpy function, we encourage you to look at the official documentation.

You can also create a new cell in the notebook and write `np.exp?` (for example) to get quick access to the documentation.

Exercise: Implement the sigmoid function using numpy.

Instructions: x could now be either a real number, a vector, or a matrix. The data structures we use in numpy to represent these shapes (vectors, matrices...) are called numpy arrays. You don't need to know more for now.

$$\text{For } x \in \mathbb{R}^n, sigmoid(x) = sigmoid \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-x_1}} \\ \frac{1}{1+e^{-x_2}} \\ ... \\ \frac{1}{1+e^{-x_n}} \end{pmatrix} \tag{1}$$

```
In [9]: # GRADED FUNCTION: sigmoid

        import numpy as np # this means you can access numpy functions by writing np.function()

        def sigmoid(x):
            """
            Compute the sigmoid of x

            Arguments:
            x -- A scalar or numpy array of any size

            Return:
            s -- sigmoid(x)
            """

            ### START CODE HERE ### ( 1 line of code)
            s = 1/(1+np.exp(-x))
            ### END CODE HERE ###

            return s
```

```
In [10]: x = np.array([1, 2, 3])
         sigmoid(x)
```

```
Out[10]: array([ 0.73105858,  0.88079708,  0.95257413])
```

Expected Output:

```
<tr>
    <td> **sigmoid([1,2,3])**</td>
    <td> array([ 0.73105858,  0.88079708,  0.95257413]) </td>
</tr>
```

1.2.2 1.2 - Sigmoid gradient

As you've seen in lecture, you will need to compute gradients to optimize loss functions using backpropagation. Let's code your first gradient function.

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

$$sigmoid\_derivative(x) = \sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2}$$

You often 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 $\sigma'(x) = s(1 - s)$

```
In [13]: # GRADED FUNCTION: sigmoid_derivative

         def sigmoid_derivative(x):
             """
             Compute the gradient (also called the slope or derivative) of the sigmoid function
             You can store the output of the sigmoid function into variables and then use it to

             Arguments:
             x -- A scalar or numpy array

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

             ### START CODE HERE ### ( 2 lines of code)
             s = sigmoid(x)
             ds = s*(1-s)
             ### END CODE HERE ###

             return ds
```

```
In [14]: x = np.array([1, 2, 3])
         print ("sigmoid_derivative(x) = " + str(sigmoid_derivative(x)))
```

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

Expected Output:

```
<tr>
    <td> **sigmoid_derivative([1,2,3])**</td>
    <td> [ 0.19661193  0.10499359  0.04517666] </td>
</tr>
```

1.2.3 1.3 - Reshaping arrays

Two common numpy functions used in deep learning are np.shape and np.reshape(). - X.shape is used to get the shape (dimension) of a matrix/vector X. - X.reshape(...) is used to reshape X into some other dimension.

For example, in computer science, an image is represented by a 3D array of shape $(length, height, depth = 3)$. However, when you read an image as the input of an algorithm you convert it to a vector of shape $(length * height * 3, 1)$. In other words, you "unroll", or reshape, the 3D array into a 1D vector.

Exercise: Implement `image2vector()` 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
```

- Please don't hardcode the dimensions of image as a constant. Instead look up the quantities you need with `image.shape[0]`, etc.

```
In [15]: # GRADED FUNCTION: image2vector
         def image2vector(image):
             """
             Argument:
             image -- a numpy array of shape (length, height, depth)

             Returns:
             v -- a vector of shape (length*height*depth, 1)
             """

             ### START CODE HERE ### ( 1 line of code)
             a,b,c = image.shape
             v = image.reshape(a*b*c,1)
             ### END CODE HERE ###

             return v
```

```
In [16]: # This is a 3 by 3 by 2 array, typically images will be (num_px_x, num_px_y,3) where 3
         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]
```