NumPy-Assignment

May 3, 2020

0.1 NumPy Assignment

In this assignment, you will use Numpy to implement simple mathematical functions, which are very frequently used in Deep Learning.

These functions are the building blocks of a Neural Network.

0.2 Marking Scheme

Maximum Points: 30

```
<table>
    <tr><td><h3>Sr. no.</h3></td> <td><h3>Problem</h3></td> <td><h3>Points</h3></td> </tr>
    <tr><td><h3>1</h3></td> <td><h3>ReLU Implementation</h3></td> <td><h3>5</h3></td> </tr>
    <tr><td><h3>2</h3></td> <td><h3>Softmax Implementation</h3></td> <td><h3>10</h3></td> </tr
    <tr><td><h3>3</h3></td> <td><h3>Neuron Implementation</h3></td> <td><h3>15</h3></td> </tr>
</table>
```

```
In [3]: import numpy as np
```

0.3 1. ReLU Implementation (5 Points)

ReLU stands for Rectified Linear Unit. It is defined as follows:

```
z = max(0, x)
```

In this part, you have to implement this ReLU function definition using NumPy.

So if the input is:

```
array([[ 1. ,  2. , -3. ],
       [ 2.5, -0.2,  6. ]])
```

You should return the following output:

```
array([[1. , 2. , 0. ],
       [2.5, 0. , 6. ]])
```

```
<b>1. ReLU Implementation: 5 Points</b>
```

In [4]:
```
def ReLU(array):

    ###
    return np.maximum(0, array)
    ###
```

In [5]:
```
# test your result
a = np.array([[1, 2, -3], [2.5, -0.2, 6]])
ReLU(a)
```

Out[5]:
```
array([[1. , 2. , 0. ],
       [2.5, 0. , 6. ]])
```

In [ ]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

0.4 2. Softmax Implementation (10 Points)

Softmax is defined as follows:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

For example, we have following an array as an input:

```
array([0.6, 5.2, 9.2])
```

Then, the function should return the following as output:

```
array([1.80761747e-04, 1.79829587e-02, 9.81836280e-01])
```

2. Softmax Implementation: 10 Points

In [6]:
```
def softmax(array):

    ###
    expo = np.exp(array)
    expo_sum = np.sum(np.exp(array))
    return expo / expo_sum
    ###
```

In [7]:
```
# Test your result
a = np.array([0.6, 5.2, 9.2])
softmax(a)
```

Out[7]:
```
array([1.80761747e-04, 1.79829587e-02, 9.81836280e-01])
```

In [ ]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

0.5 3. Neural Network Neuron Implementation (15 Points)

We all are familiar with the `1-dimensional` linear equation:

$$y = mx + c$$

We can re-write the equation as:

$$y = w_1 x + b$$

We can write the `n-dimensional` linear equation as follows:

$$y = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b$$

Following is a pictorial representation `n-dimensional` linear equations. This linear function is called a neuron in neural networks.

let's define W as,

$$W = \begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_n \end{bmatrix}$$

and X as,

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Using above W and X, we can re-write the n-dimensional linear equation as follows:

$$y = WX + b$$

In a neural network, X, W, b, and y are called input, weight, bias, and output of the neuron, respectively.

In a neural network, usually, we use to have much more than one neuron. The same input X passes through all neurons of the layer and gets output y for each neuron. We don't have to calculate linear function for each neuron at a time; we can calculate all in one go using Numpy.

Let's assume we have `m` neurons. So we will have `m` output. Let's stack all weight horizontally and do matrix multiplication by input and add stacked b. It will give `m` output for all `m` neurons.

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & w_{m3} & \dots & w_{mn} \end{bmatrix}$$

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

You have to implement the following function:

$$Y = WX + B$$

The function will take weight W, bias B, and input X as arguments. You have to return outputs Y.

For example, arguments W, B and X are as follows:

```
W = np.array([[1.2, 0.3, 0.1], [.01, 2.1, 0.7]])
B = np.array([2.1, 0.89])
X = np.array([0.3, 6.8, 0.59])
```

Function should return:

```
array([ 4.559, 15.586])
```

3. Neuron Implementation: 15 Points

```
In [10]: def neural_network_neurons(W, B, X):

             ###
             Y = np.matmul(W, X) + B
             print('Result:\n{}'.format(Y))
             return Y
             ###
```

```
In [11]: W = np.array([[1.2, 0.3, 0.1], [.01, 2.1, 0.7]])
         B = np.array([2.1, 0.89])
         X = np.array([0.3, 6.8, 0.59])

         neural_network_neurons(W, B, X)
```

```
Result:
[ 4.559 15.586]
```

```
In [ ]: ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###
```

```
In [ ]:
```