3. Activation Functions

Project due Apr 5, 2023 08:59 -03 Completed

The first step is to design the activation function for each neuron. In this problem, we w
weights to 1, use **ReLU** for the activation function of the hidden layers, and use an iden
output neuron. The hidden layer has a bias but the output layer does not. Complete the
`neural_networks.py`, including `rectified_linear_unit` and `rectified_linear_`
you to use in the `NeuralNetwork` class, and implement them below.

You will be working in the file `part2-nn/neural_nets.py` in this problem

Correction note (Nov 1): In the `part2-nn/neural_nets.py`, in the definition of `Class`
the initialization of weights has now been changed to an initialization as `float` rather t
either re-download the updated project release mnist.tar.gz, or change the correspond
`nn/neural_nets.py` to the following, where we have added decimal points to all numb

```
class NeuralNetwork():

    def __init__(self):

        # DO NOT CHANGE PARAMETERS (Initialized to floats instead of
        self.input_to_hidden_weights = np.matrix('1. 1.; 1. 1.; 1. 1
        self.hidden_to_output_weights = np.matrix('1. 1. 1.')
        self.biases = np.matrix('0.; 0.; 0.')
```

Rectified Linear Unit

2.0/2.0 points (graded)

First implement the ReLu activation function, which computes the ReLu of a scalar.

Note: Your function does not need to handle a vectorized input

Available Functions: You have access to the NumPy python library as np

```
1 def rectified_linear_unit(x):
2     """ Returns the ReLU of x, or the maximum between 0 and x."""
3     return np.maximum(0, x)
4
```

Note: Your function does not need to handle a vectorized input

Available Functions: You have access to the NumPy python library as np

```
def rectified_linear_unit_derivative(x):
    """ Returns the derivative of ReLU."""
    return np.greater(x, 0).astype(int)

```

Previous Next

Press ESC then TAB or click outside of the code editor to exit

Correct