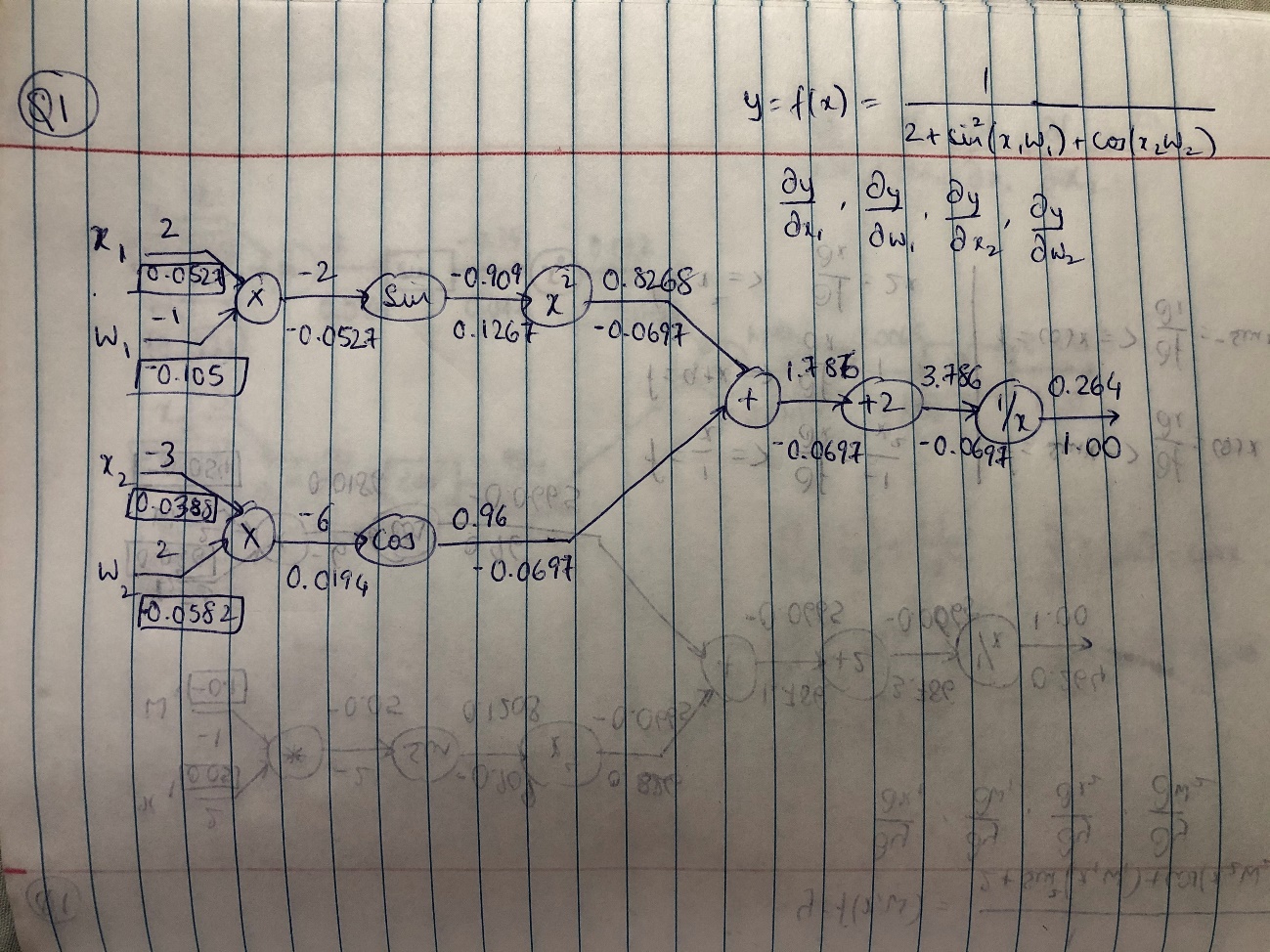
**Intro to Deep Learning**

**Assignment 2**

1. **Practice of scalar-based backpropagation**
   1. **Manual Calculation**

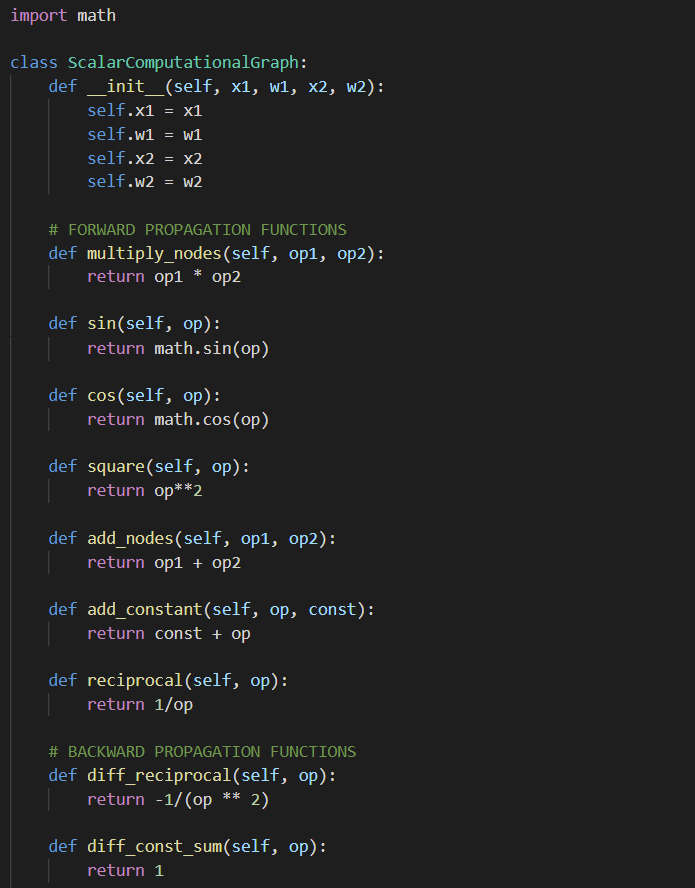
The computation graph along with the intermediate calculations are shown in the figure below:

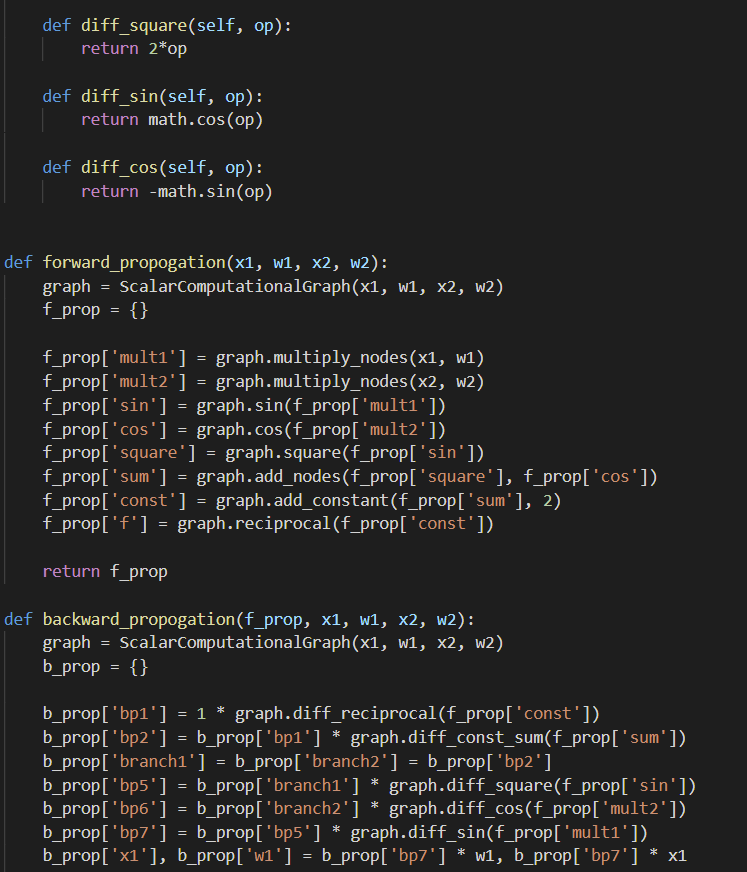


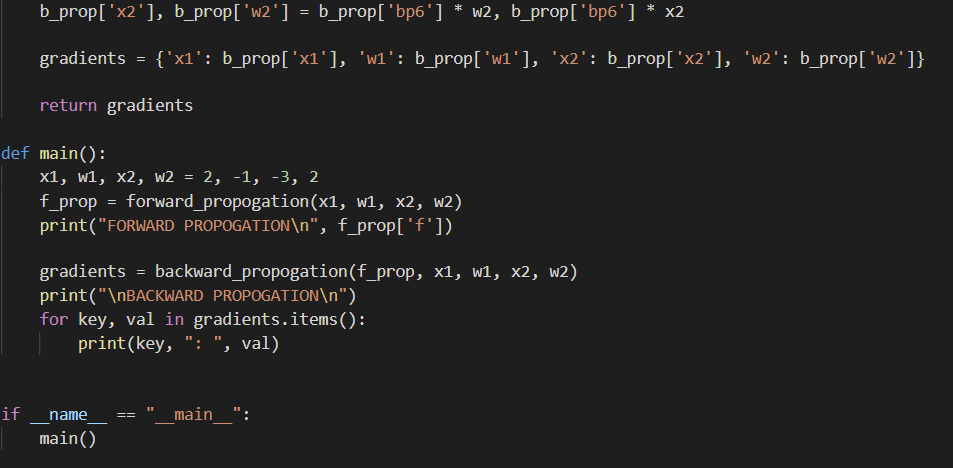
The values shown above the arrows between each node in the graph represent the values obtained by forward propagation and those below represent the values obtained by back propagation. The values in the boxes below x1, x2, w1 and w2 represent the final gradient values () obtained by back propagation.

* 1. **Program to verify the calculation.**

The code used to verify the above result is shown in the screenshots below:

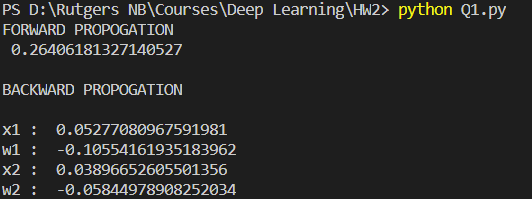






To solve this problem, I used a class to initialize the computation graph with the initial values for x1, x2, w1 and w2. I also used the class to define the functions that would calculate the forward as well as back propagation gradients. After defining these, I created two functions, one for forward propagation and for back propagation, that would compute the gradients using the class functions for all nodes in the computation graph. I used dictionaries to store the intermediate gradient values so that it was convenient for me to verify the calculations at each stage.

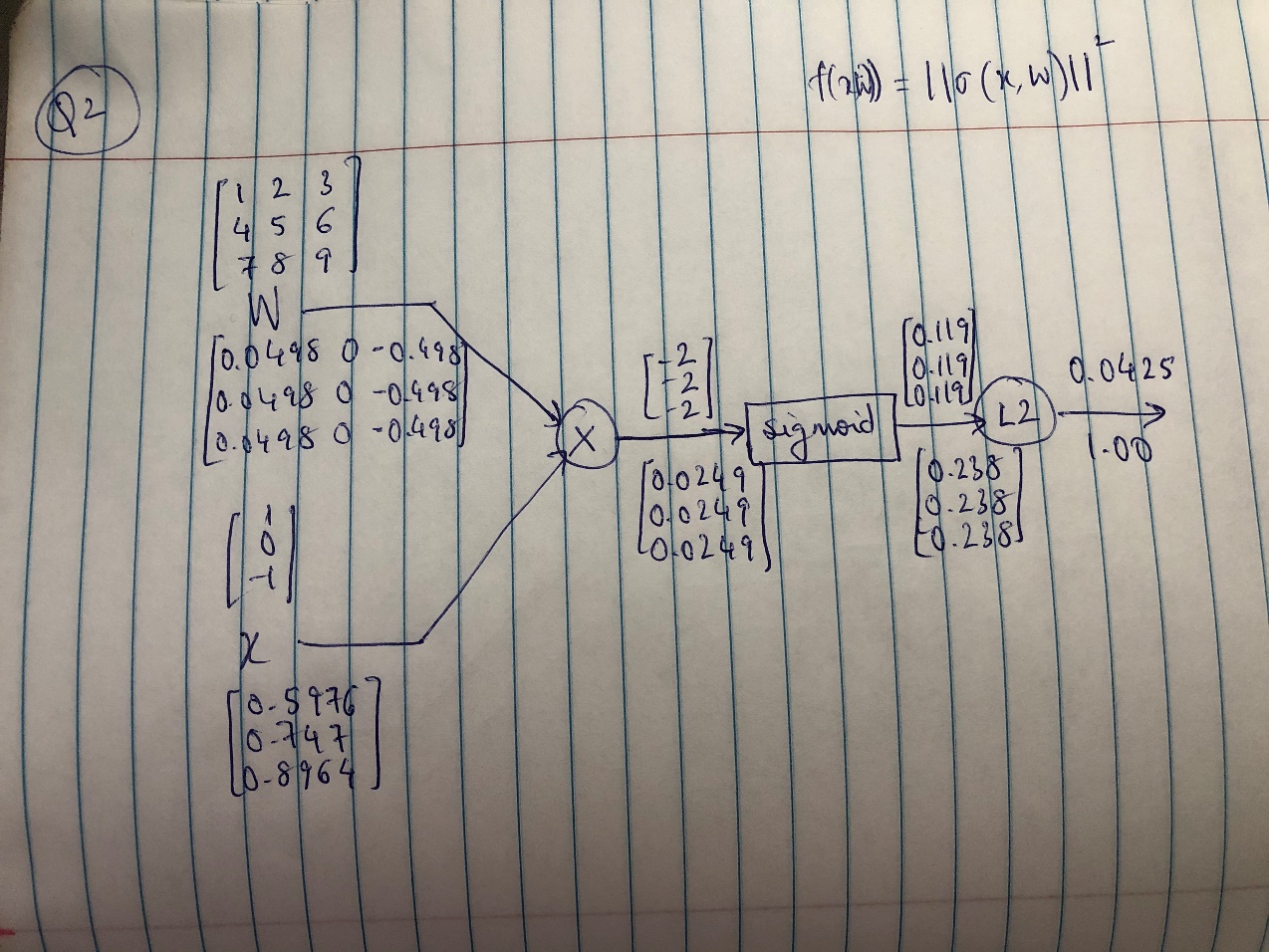
The output obtained on running the code is shown below:



These values match what I had obtained by manual calculation, so the result is verified.

1. **Practice of vector-based backpropagation**
   1. **Manual Calculation**

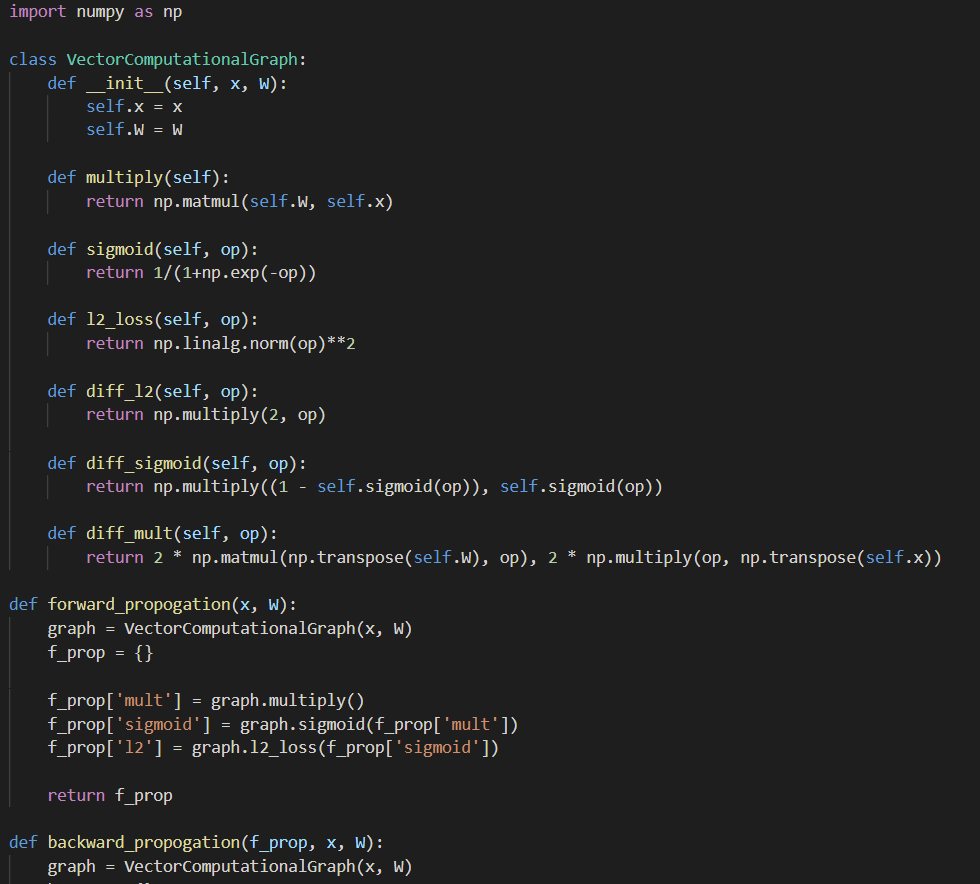
The computation graph along with the intermediate calculations are shown in the figure below:

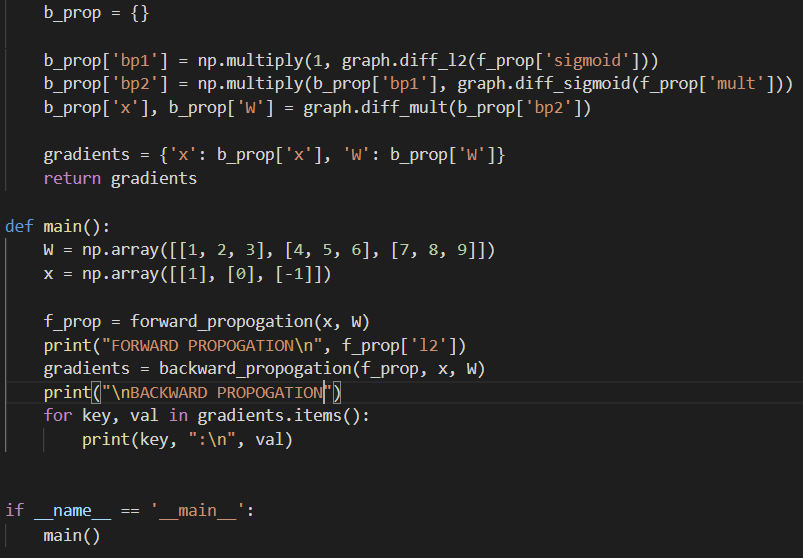


The values shown above the arrows between each node in the graph represent the values obtained by forward propagation and those below represent the values obtained by back propagation. The values in the boxes below x and W represent the final gradient values () obtained by back propagation.

* 1. **Program to Verify the Calculation**

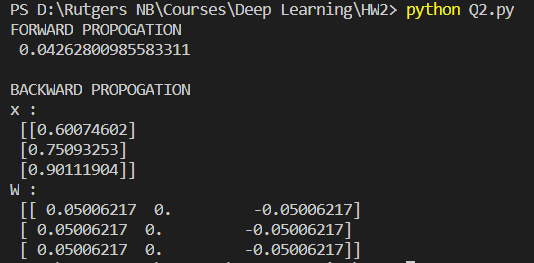
The code used to verify the above result is shown in the screenshots below:





Like the first question, I created a class to initialize the computation graph with the input and weight vectors as well as defined functions to compute the intermediate values. I also used dictionaries to store the intermediate calculations for the gradients and to simplify the matrix operations involved in the code, I used numpy functions to compute the gradients.

The output obtained on running the code is shown below:



The results obtained match with the manually computed results, so the result is verified.