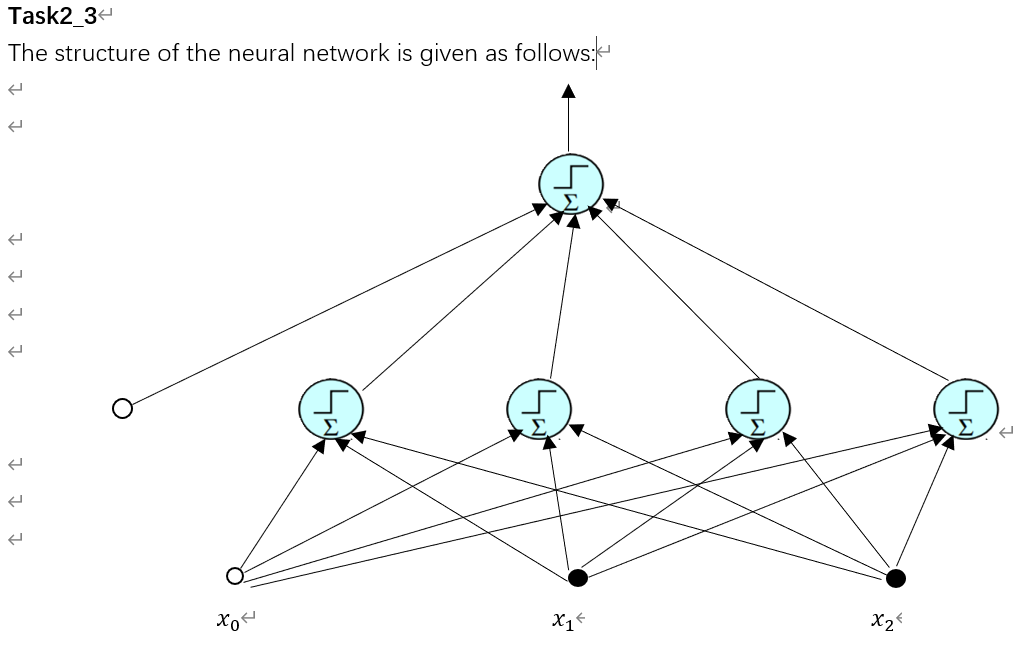
**Name: Liu Rongxing**

**UUN: s1810054**

**Informatics 2b——Learning Coursework 1 Task2**



There are four perceptrons on the first layer, which gives the decision boundaries for the four sides of the polygon.

Take the points (1.37204, 3.13575) and (1.24725,2.63709) first and calculate the equation of the line passing through these two points. We will get the equation as . Hence we have the weight vector, where 2.34691 is the bias. After normalizing, we get . Hence we have .

We can use the same method to find the weights for the other three perceptrons.

When it comes to the perceptron on the second layer, we first find out what input vector is required to output 1 as the result i.e. (y(**x**)=1).

For the line joining (1.37204, 3.13575) and (1.24725,2.63709), we need the point to be below the line i.e. (y(**x**)=0).

For the line joining (2.04748,2,45496) and (1.24725,2.63709), we need the point to be above the line i.e. (y(**x**)=1).

For the line joining (2.04748,2,45496) and (2.34399,2.7955), we need the point to be above the line i.e. (y(**x**)=1).

For the line joining (2.34399,2.7955) and (1.37204, 3.13575), we need the point to be below the line i.e. (y(**x**)=0).

Hence, the only input **x** that can output 1 i.e. (y(**x**)=1) is , which is after adding the bias.

In this way, we can find such a vector that outputs 1 only when the input is . This is because when calculating the dot product between **w** and x, we already have -1 as the product of the first elements. For the rest four elements of **x**, should any of them not being the desired value, the value of the dot product will be less than or equal to 0, which will output 0 as a result.

**Task2\_5, Task2\_7 and Task2\_9:**

Of the graphs of all three tasks, the yellow portion represents the Class 1, while the blue portion represents the Class 0.

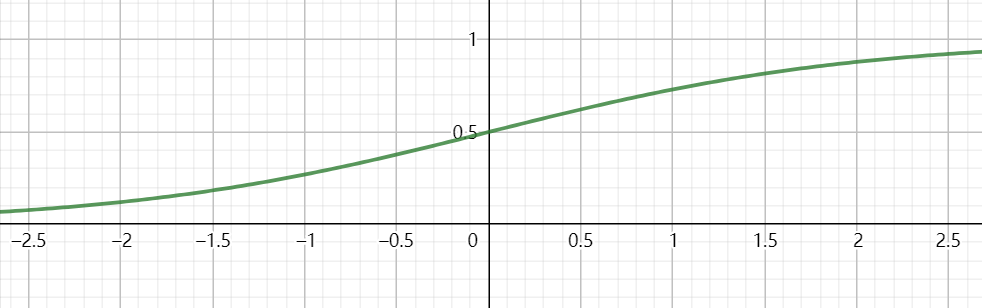
**Task2\_10**

After comparing the graphs obtained in **Task2\_7 and Task2\_9,** we will notice that the graph in Task2\_7 is more clear-cut, whereas the graph in Task2\_9 has some mis-classifications near the edges and the vertices.

This is because in task2\_7, we are using the step function to classify points. Hence, all the points, after passing through the perceptrons, will end up with only two results: either 0 or 1. There will only be two possible classes on the graph and the edges and vertices will have distinct boundaries.

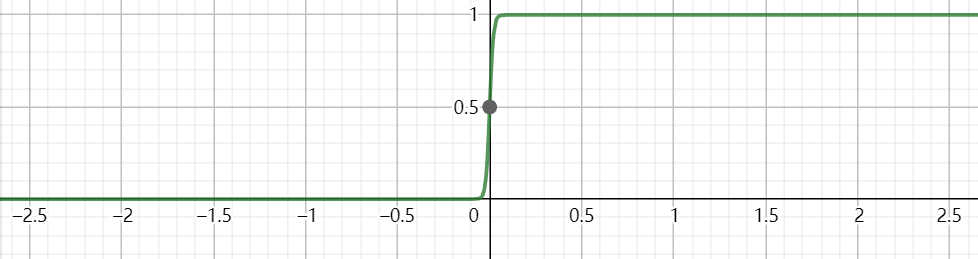
In contrast, in task2\_9, we are using the sigmoid function. This time all the points will end up with rather unique values after passing through the perceptrons. There will be much more classes this time. Hence, the decision boundaries will not be as clear as in task2\_7. In order to make the decision boundaries clearer, we can multiply the weights with some big numbers to make the sigmoid function as close as the step function. The graphs below illustrates how this method works:

This is the graph of the original sigmoid function :



As we can see from the graph, the gradient is very smooth at the point x=0. Change of values of **x** from 0 will not lead to significant change in values of **y**.

Here is the graph of the new sigmoid function , which is obtained by scaling the graph along the x-axis with a factor of .



As we can see from the graph, the graph is squeezed towards the middle. This time a small change of **x** from 0 will lead to a drastic change in the value of **y**. Hence the points now can be classified into two major classes, namely **0** and **1.** As a result, the decision boundaries for the graph will become clearer.