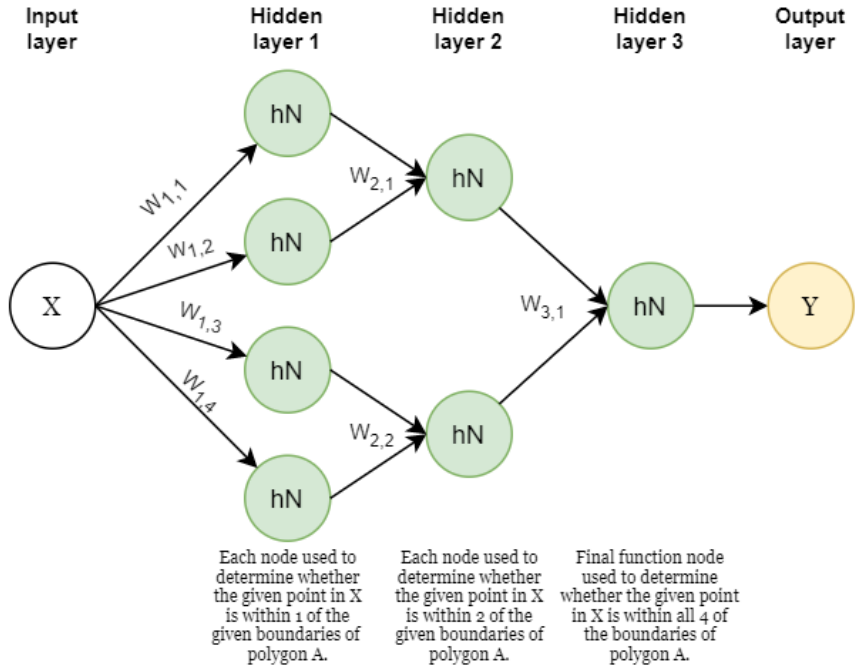
Inf2B Coursework 1 Report

Task 2 – Neural networks

2.3) Structure of the neural network



How weights were determined:
For this given neural network weights are created in order to classify data correctly using the given activation function (hNeuron).

WEIGHTS: Input -> Hidden 1
(All calculations within *task2_find_hNN_A_weights.m*)

For the first layer of weights, the activation function classification is dependent on the coordinate of a point being within the boundaries of polygon A or not. Thereby we must calculate the functions of these boundaries and find how they can be converted to appropriate weights.

Given we are calculating weights we must ensure to account for every variable:

$$W_0(bias) = y - intercept \qquad W_1(X\ co-ord.\ coeff.) = gradient \qquad W_2(Y\ co-ord.\ coeff.) = -1$$

The sign of Y is negative after isolating all terms to one side of the equation, and the default coefficient of Y has a constant magnitude of 1 so we only need to calculate the gradient and y-intercept.

So firstly, I created a function *task2_calcGrads(x)* which calculates the constants for all boundary functions of a given polygon x. It takes an input of a polygon coordinate matrix and produces an output of a boundary function matrix (where each row represents a boundary function and each column represents the associated gradient and y-intercept respectively). Implementing this function promotes much higher accuracy for boundary calculations than mere hardcoding.

Now, given we have the respective boundary function constants we must identify which boundaries are the maxima and minima of this polygon. This is very relevant as it denotes which side of the boundaries represents

the polygon. I identified these maxima and minima by capitalizing on the order of polygon vertex input, in which the maximum Y vertex is always first and the other vertices follow in a clockwise/anti-clockwise fashion. In order to represent these maxima and minima boundaries we can just multiply the minima boundary functions by -1.

Given that we only account for points inside the polygon (not including the periphery) I deducted a tiny value ($-1*10^{-14}$) from the maxima boundaries and added a tiny weight ($+1*10^{-14}$) to the minima boundaries.

At this point, the only thing left to do is normalize the weights. So, I divided each weight vector by its associated maximum magnitude element.

WEIGHTS: Hidden 1 -> Output

Given that the first layer of neurons each output a 1 or a 0 to represent if the given point is within the given boundary, the rest of the neurons act like AND logic gates. This is because the point must be within <u>all</u> borders to be within the polygon.

So, in order to create a suitable weighting, I created a large negative bias which could only be overcome if both W_1 and W_2 inputs were 1.

$$W_0 = -1 \qquad W_1 = 0.51 \qquad W_2 = 0.51$$

2.10) Difference in decision boundary calculations for *task2_hNN_AB()* & *task2_sNN_AB()*

The only difference between the layout of these neural networks was the activation functions used. The first using a step function (hNeuron) and the second a sigmoid function (sNeuron).

The predominant difference I found in calculating these decision boundaries was the weights I used. This is because in contrast to the step function which produces an output of 0 or 1 the output of a sigmoid is in the range from 0 to 1 which makes it far more difficult to simulate logical gates without any threshold to classify the output of a given neuron. In order to cater for this, I decided it would be useful to use non-normalized weights for the sigmoid function, I did this by multiplying each weight vector by 10^8, this in effect:

creates a very small number

$$if \quad a > 0 \quad then \quad e^{-(a)} \to 0 \quad \therefore g(a) = \frac{1}{1+0} \cong 1$$

and a very large number

$$if \quad a \leq 0 \quad then \quad e^{-(-a)} \to \infty \quad \therefore g(a) = \frac{1}{1+\infty} \cong 0$$

This method worked perfectly in order to simulate logical gates using the sigmoid function, and no threshold was even required to classify the final output(s) of the sigmoidal neural network upon estimation of the decision boundaries.

As to be expected, the visualisations of decision boundaries produced by task2_plot_regions_hNN_AB.mat and task2_plot_regions_sNN_AB.mat were identical.