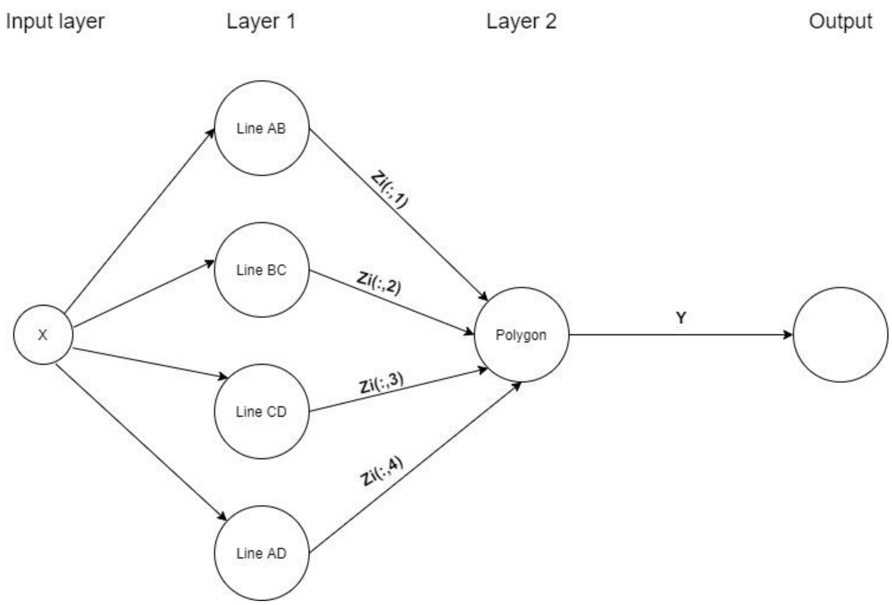
Report for task 2

Task 2.3

Structure of the network:



Explanation of chosen weights:

I firstly found the equations of the line:

d(AB): 16.16192*x - 38.61822 - y = 0
d(BC): -1.27549*x + 6.67024 - y = 0
d(CD): 0.72552*x + 0.63949 - y = 0
d(AD): -0.47359*x + 4.96765 - y = 0

For the points to be inside the polygon, d(AB) and d(AD) need to be greater than 0, while d(BC) and d(CD) need to be less than 0. For the hNeuron to work on my weights, I multiplied d(BC) and d(CD) with (-1):

d(AB): 16.16192*x - 38.61822 - y = 0
d(BC): 1.27549*x - 6.67024 + y = 0
d(CD): -0.72552*x - 0.63949 + y = 0
d(AD): -0.47359*x + 4.96765 - y = 0

We may rewrite the equations to have the bias in front:

d(AB): - 38.61822 + 16.16192*x - y = 0
d(BC): - 6.67024 + 1.27549*x + y = 0
d(CD): - 0.63949 - 0.72552*x + y = 0
d(AD): 4.96765 - 0.47359*x - y = 0

Because we need the normalized form of the vectors, I divided d(AB) by 100, d(BC) by 10 and d(AD) by 10:

d(AB): - 0.3861822 + 0.1616192*x - 0.01*y = 0
d(BC): - 0.667024 + 0.127549*x + 0.1*y = 0
d(CD): - 0.63949 - 0.72552*x + y = 0
d(AD): 0.496765 - 0.047359*x - 0.1*y = 0

What followed was just to assign w(1,1,:) = d(AB), w(1,2,:) = d(BC), w(1,3,:) = d(CD) and w(1,4,:) = d(AD):

w(1,1,0) : -0.386122, w(1,1,1) : 0.1616192, w(1,1,2) : -0.01;
w(1,2,0) : -0.667024, w(1,2,1) : 0.127549, w(1,2,2) : 0.1;
w(1,3,0) : 0.63949, w(1,3,1) : -0.72552, w(1,3,2) : 1;
w(1,4,0) : 0.496765, w(1,4,1) : -0.047359, w(1,4,2) : -0.1.

For layer 2, because I had only 0's and 1's in the label matrix I got from the first layer, I chose to initialize the weight for each line of the polygon uniformly : 0.25 each, so that summed up gives 1. Because I need all 4 labels on a line to be 1 in order to have the ith point inside the polygon, I needed the bias to be smaller or equal than -¾ = -0.75. Hence:

w(2,1,0) : -0.75, w(2,1,1) : 0.25, w(2,1,2) : 0.25, w(2,1,3) : 0.25, w(2,1,4) : 0.25.

Task 2.10

I obtained approximately the same boundaries, hence I will explain how I got to have this result.

Looking at the sigmoid function graph, I observed it tends to 0 when x goes to (- infinity) and to 1 when x goes to (+ infinity). Testing in Matlab, I saw that it approximates to 1 for large values of x, and to 0 for very small values. So, I took the lines' equations and multiplied them by either 10^2 or 10^3 so as to have large weights and very small bias.

The resulting graph is similar to 2.4. It really helped and made an important difference that the weight matrix needn't be normalized for this exercise as well. It was interesting to note that for small weights, the boundaries were not well defined.