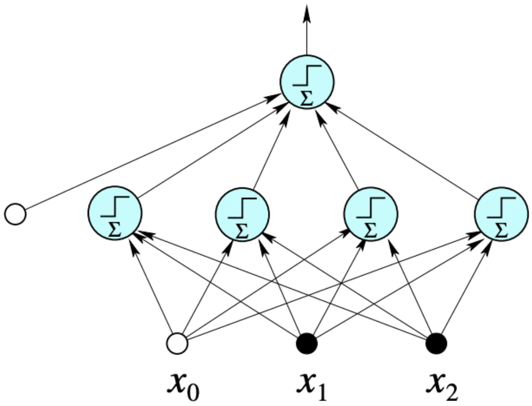
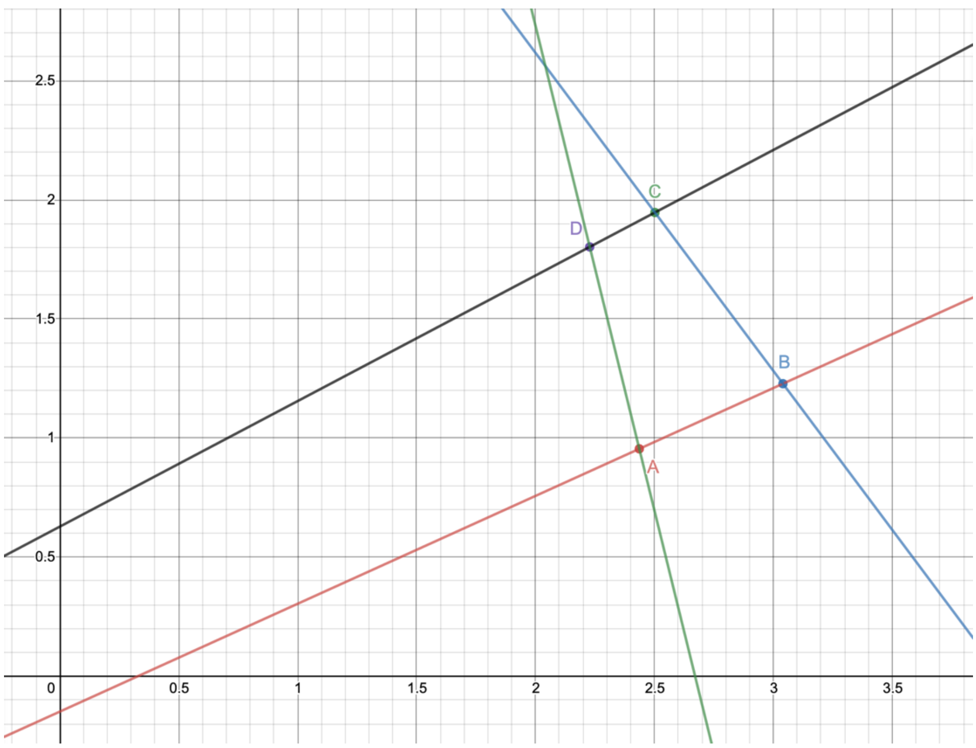
INF2B Report – TASK 2

TASK 2.3



- The structure of the network is shown on the left. (Taken from lecture 12, slide 7)
- It's a two-layers neural network. The neurons in the hidden layer is denoted by (from left to right) $\mathbf{z_0, z_1, z_2, z_3, z_4}$ and the output is denoted by $\mathbf{y}$.

- First, we find the input-to-hidden weights:
  - Polygon_A has 4 vertices:
    - **A** (2.43627, 0.954496),
    - **B** (3.04028, 1.22806),
    - **C** (2.50204, 1.94681),
    - **D** (2.22797, 1.80233).
  - Plot the points into a x_2 against x_1 plane and find the equation of the lines with the points of A and B, B and C, C and D, A and D using y = mx + c.



- We now form inequalities such that the inside of Polygon_A is classified as Class 1 and the outside and periphery as Class 0.
- Note that $b(x) = w_0 + w_1x_1 + w_2x_2$
- With A and B, we obtain the weight of neuron 1 in layer 1 from neuron i in layer 0, where $i \in \{0,1,2\}$:
  - $x_2 > 0.4529130312\ x_1 - 0.1489224305$
  - $b(x) = 0.1489224305 - 0.4529130312\ x_1 + x_2$
  - Hence $w_{10}^1 = 0.1489224305$, $w_{11}^1 = -\ 0.4529130312$, $w_{12}^1 = 1$
- With B and C, we obtain the weight of neuron 2 in layer 1 from neuron i in layer 0, where $i \in \{0,1,2\}$:
  - $x_2 < -1.335370838\ x_1 + 5.287961252$
  - $b(x) = 5.287961252 - 1.335370838\ x_1 - x_2$
  - Hence $w_{20}^1 = 5.287961252$, $w_{21}^1 = -1.335370838$, $w_{22}^1 = -1$
- With C and D, we obtain the weight of neuron 3 in layer 1 from neuron i in layer 0, where $i \in \{0,1,2\}$:
  - $x_2 < 0.527164593\ x_1 + 0.6278231018$
  - $b(x) = 0.6278231018 + 0.527164593\ x_1 - x_2$
  - Hence $w_{30}^1 = 0.6278231018$, $w_{31}^1 = 0.527164593$, $w_{32}^1 = -1$
- With A and D, we obtain the weight of neuron 4 in layer 1 from neuron i in layer 0, where $i \in \{0,1,2\}$:
  - $x_2 > -4.070254441\ x_1 + 10.87073479$
  - $b(x) = -10.87073479 + 4.070254441\ x_1 + x_2$
  - Hence $w_{40}^1 = -10.87073479$, $w_{41}^1 = 4.070254441$, $w_{42}^1 = 1$

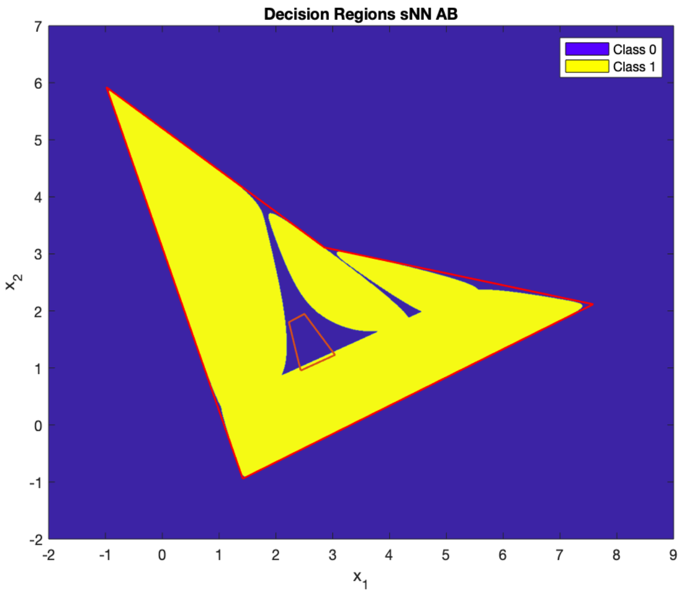
- Next, we find the hidden-to-output weights:
  - Observe that the inputs to the neuron in the second layer, particularly $\mathbf{z_1}$, $\mathbf{z_2}$, $\mathbf{z_3}$, $\mathbf{z_4}$, will all need to be 1 in order for an input feature (x_1, x_2) to be classified as class 1. (that is, it is inside Polygon_A)
  - Note that $\mathbf{z_1}$, $\mathbf{z_2}$, $\mathbf{z_3}$, $\mathbf{z_4}$ will always be 0 or 1.
  - Appropriate weights for neuron in layer 2 from neuron i in layer 1, $i \in \{1,2,3,4\}$, would be 1, that is, $w_{11}^2\ w_{12}^2 = w_{13}^2 = w_{14}^2 = 1$
  - We now need to set w_{10}^2 so that when an input feature should be classified as class 1 (hence $\mathbf{z_1}$, $\mathbf{z_2}$, $\mathbf{z_3}$, $\mathbf{z_4}$ is all 1), you want w_{10}^2+1+1+1+1 > 0. This is, in

fact, should be the only case where it is greater than 0. So that when one or more z_i is 0, $w_{10}^2 + z_1 + z_2 + z_3 + z_4 <= 0$.

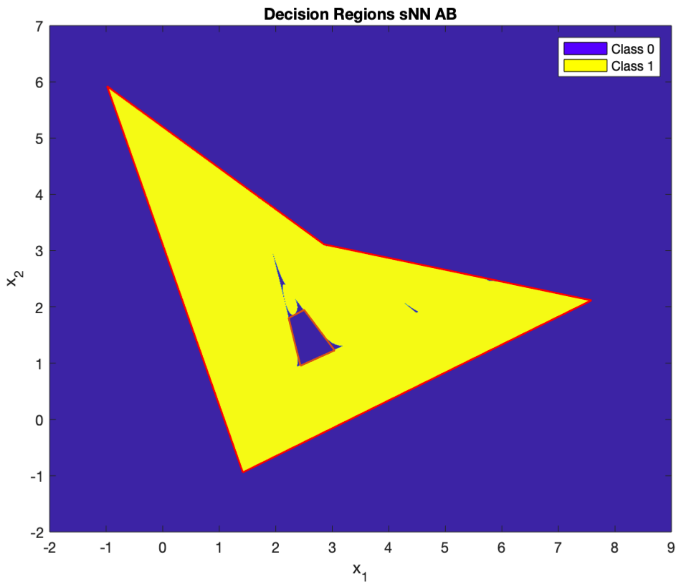
  - Hence we find that the weight w_{10}^2 that satisfies above conditions is any number between -3 and -4. (I used -3.5 in the codes)

- Finally, we normalize the weights by dividing each weight by the weight with largest magnitude, which is w_{40}^1 = -10.87073479.

<u>TASK 2.10</u>



n = 20



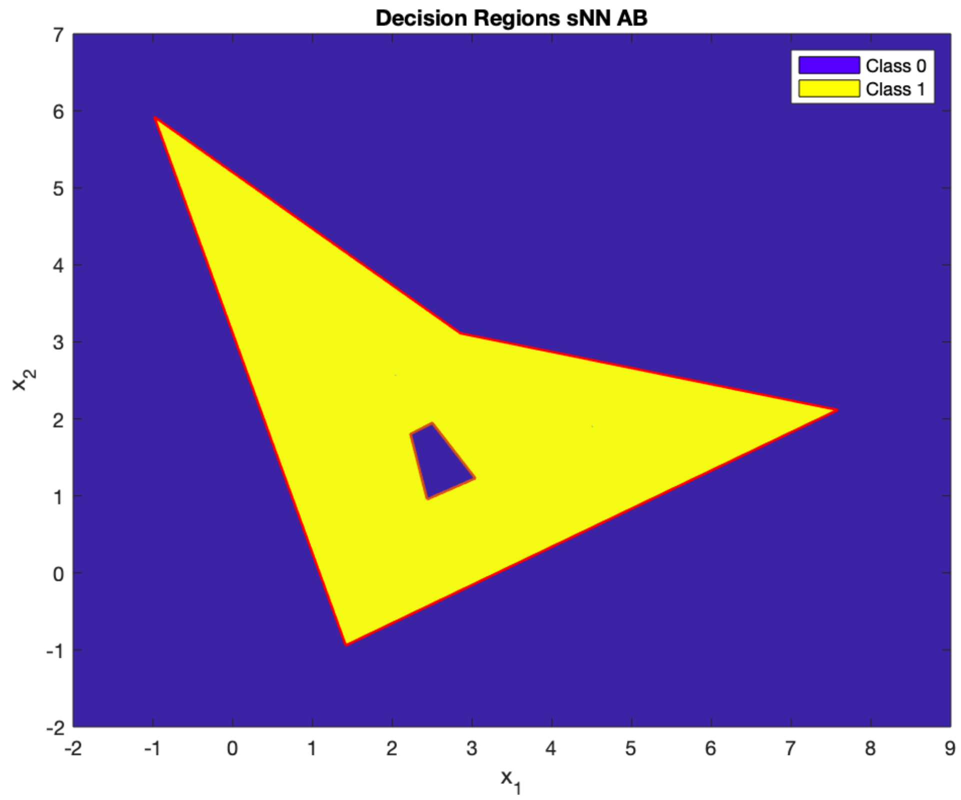
n = 100

<u>How the decision regions differ from those for task2_hNN_AB</u>

The decision regions when we used the sigmoid function instead of step function differ when we scale up the weight vectors for Polygon_A and Polygon_B by a small number. The pictures above shows the decision regions when sigmoid function is used, labeled with the corresponding factor that the weight vectors for Polygon_A and Polygon_B are scaled up. It can be observed that there are misclassifications in both cases, but there are less misclassifications when we scale up the weight vectors by a higher factor of n. (The two red lines in the pictures indicate Polygon_A and Polygon_B)

Why the decision regions differ from those for task2_hNN_AB

When using sigmoid function, we scale up the weights because if we have a point within any decision boundary (used to create final decision regions) but is close to the edges, the dot product $\boldsymbol{w}^T\boldsymbol{x}$ will be a relatively small positive number; hence, after adding with other dot products from neurons in the same layer, the final output may result in a negative number and the point might be wrongly classified, resulting in a false negative. This is why scaling up the weights by a larger factor reduce the likelihood for a point to be misclassified.



n = 1000

The picture above shows decision regions for task2_sNN_AB when weight vectors of Polygon_A and Polygon_B are scaled up by a factor of 1000, giving the expected regions for class 0 and class 1.