INF2B Learning CW Report Task 2

Linda Mazánová
S1801828

2.3 Structure of the network

The structure of the network is an input layer of 3 neurons, a hidden layer of 5 neurons and an output layer of 1 neuron.

The network has 3 input neurons because the data comes in the form [1,x,y] for each point on the 2D plane that we want to classify. 1 is multiplied by w0 (bias), x is multiplied by w1 and y is multiplied by w2. Each of these input neurons is connected to 4 neurons in the hidden (second) layer hence we need 4 weight vectors. Each weight vector corresponds to a line that is an edge of the polygon A. After dot product of the input vector with the weights matrix we apply the step function that changes the value to 0 or 1. Each of the neurons in the hidden layer is connected to the output layer + there is z0 in the hidden layer that is the bias.

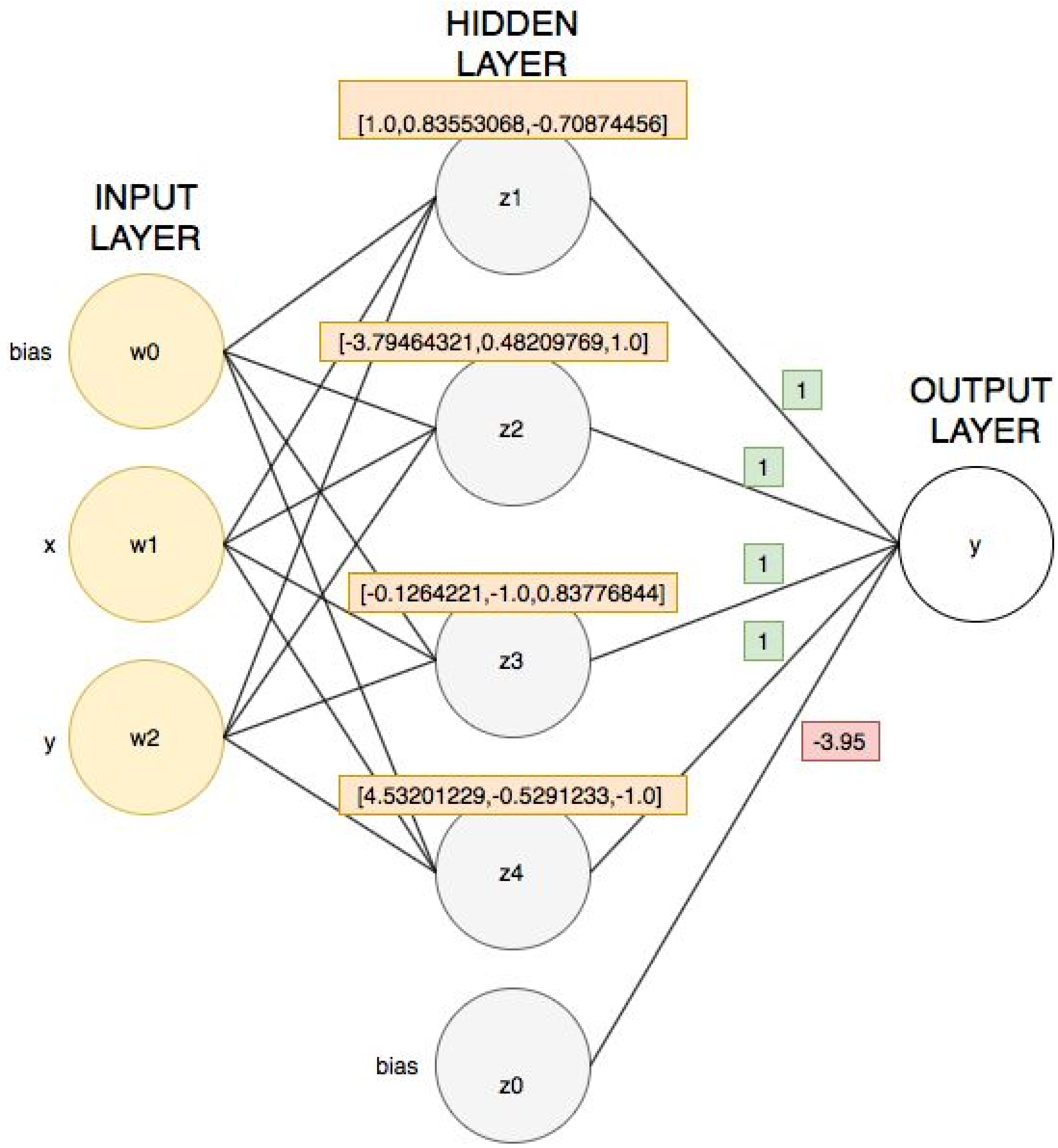
How to find weights:

First layer: Using basic geometry I found the equations of lines that form the edges of the polygons. Each line forms an equation of the form y=mx+c. I determined the weights in the following way:

1. Set w1=1
2. Set w2=-1/m
3. Set w0=-c*w2

After that I normalized the weights and obtained a 4x3 matrix of weights.

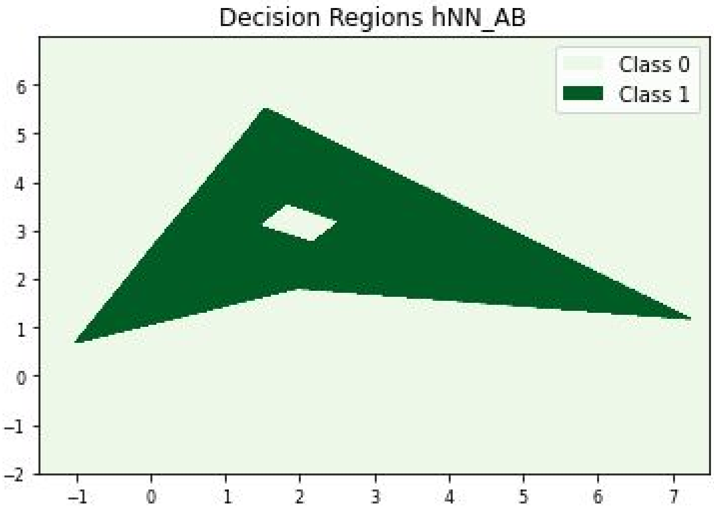
Second layer: After applying the step function, the nodes in the hidden layer can only be 0 or 1. Therefore it makes sense that all of the weights in the second layer (except for bias) have an absolute value of 1. We need to multiply 2 weights by -1 (to satisfy all 4 inequalities) to make sure we only classify the point inside the polygon A as 1. We want to make sure to classify a point to be inside polygon A iff all restrictions for all 4 lines are satisfied so iff all of the neurons z1,z2,z3,z4 in the hidden layer are 1. That is why we apply the bias of -3.5 because only 4 ones will give us a positive output after summation. Otherwise the point will be classified as 0 if any of the line inequalities are not satisfied.



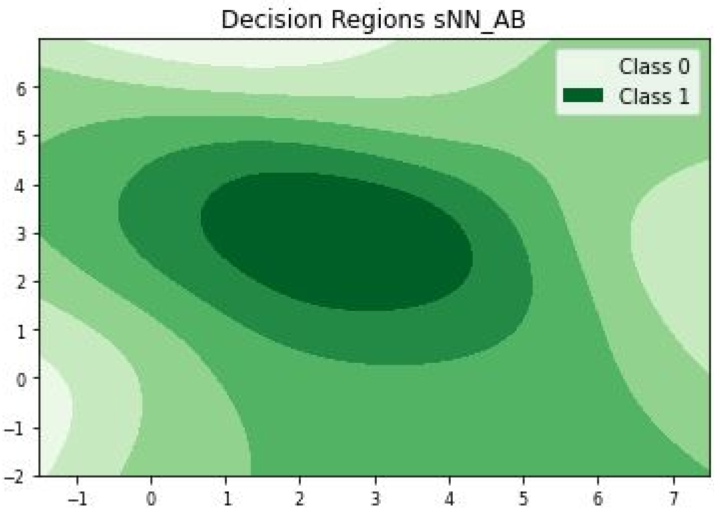
2.10 Comparing the decision regions for task_2_sNN_AB(), explaining how and why they differ from task2_hNN_AB().

The main difference between task2_hNN_AB() and task2_sNN_AB() is that the neurons are using different function after multiplying the input vectors with the weights matrix. task2_sNN_AB() uses the sigmoid function which unlike the step function does not output just values 0 (for negative values of x or 0) or 1 (for x>0). The sigmoid function is in the form 1/(1+np.exp(-x)) so for very large values of x, the output of the function approaches 1 and for very small values of x, the output of the function approaches 0. As a result, if we do not change the weights at all, the decision boundary will be distorted and not clearly defined. This will make binary classification impossible so we must implement some changes to the weights.

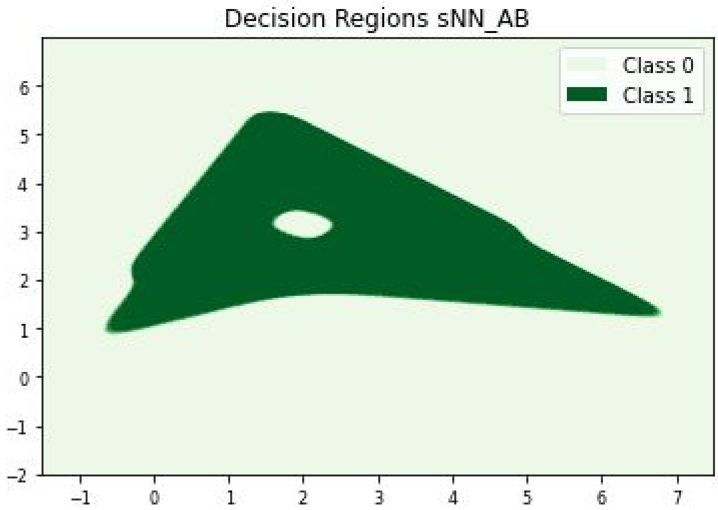
For comparison purposes, this is the decision boundary for task2_hNN_AB():



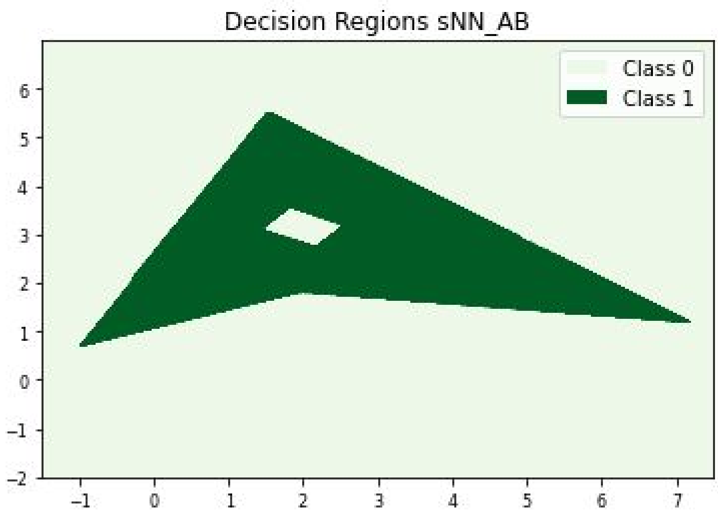
In order to obtain the decision regions for task2_sNN_AB() we need to multiply the weights by a large scalar number so that it is able to perform binary classification like the step function does. We can experiment by multiplying the weights in the 1st and 2nd layer by various scalars and see how the decision boundary changes:



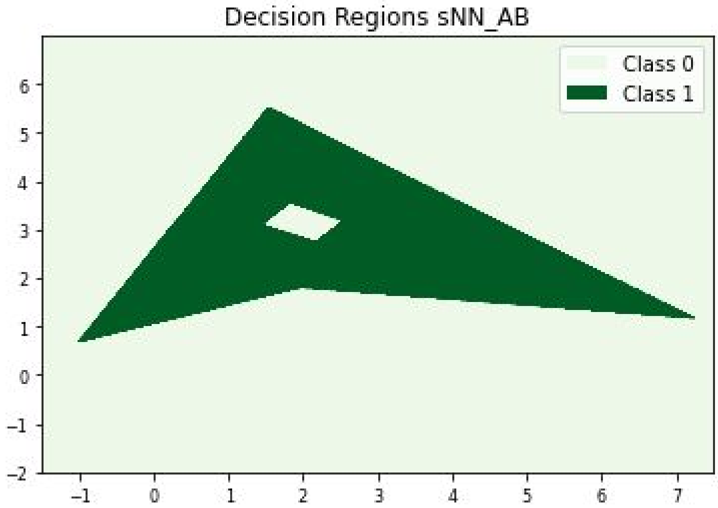
1. No change in weights (same weights applied as in task2_hNN_AB()) -> We see that the decision boundary is not discrete and binary classification would not be possible. There are more than 2 colors and the decision regions do not look like polygons.



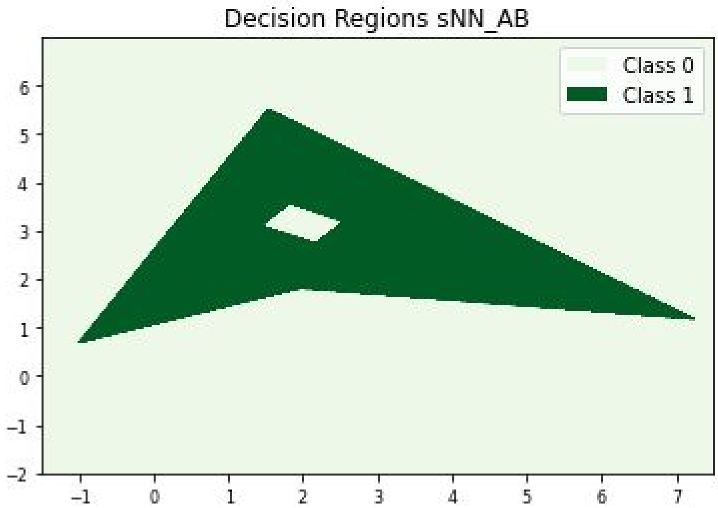
2. Weights in the 1st and 2nd layer multiplied by 10 -> This change improves the decision boundary, however, we see that the edges of the polygons are still blurred. The accuracy of classification, especially close to the boundary, would be quite poor.



3. Weights in the 1st and 2nd layer multiplied by 100 -> The decision regions become more clear and we see a significant improvement from figure 2.



4. Weights in the 1st and 2nd layer multiplied by 1000 -> We still observe a slight improvement in terms of greater accuracy of classification close to the decision boundary.



5. Weights in the 1st layer multiplied by 1000 and in 2nd layer multiplied by 3000 + threshold function applied -> This setup results in a very accurate classification of the points just like we have seen on the plot for task2_hNN_AB().

We can conclude that scaling the weights used for task2_hNN_AB() by a large enough scalar will result in a good approximation of the decision regions for task task2_sNN_AB(). Multiplication by a large number will ensure that the output of the sigmoid function stays close to 1 or close to 0.