INF2B Coursework

April 16, 2020

Task 2

2.3

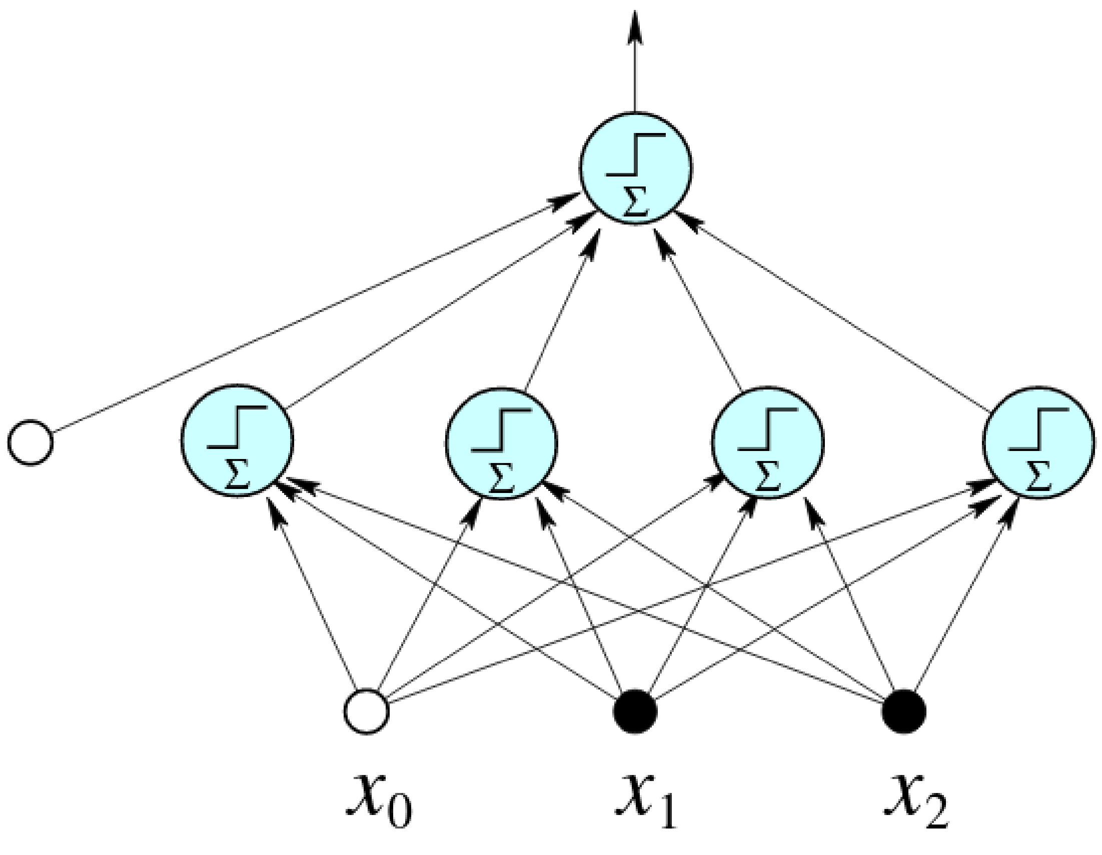

Figure 1: Structure of my neural network (image taken from Lecture 11 slides)

In my code I have implemented the neural network for decision boundary classification of a $4-$ *sided* discussed in the lectures. In our neural network we have two inputs x_1 and x_2 and bias value in x_0. These become inputs for the 4 nodes representing the 4 sides of the polygon in our hidden layer. Each of these nodes classify the points as 1 or 0 corresponding to whether it lies inside or outside with respect to line.

We then take there outputs and combine them using an AND function, to classify only those points which are inside the polygon with respect to all 4 lines as 1. All the other points are given the value 0.

To calculate the weights for the network with step activation function I first calculated the inequalities for each of the 4 lines. I then converted them to the form,

$$w_0 + w_1x_1 + w_2x_2 \geq 0$$

Using this form we extract $w = [w_0, w_1, w2]$ and the normalise them. The AND function is simply modelled using $w = [-3.5, 1, 1, 1, 1]$.

To verify these, I also wrote a python script to validate these result. I used numpy functions to calculate these results, for eg. `numpy.lstsq`.

2.10

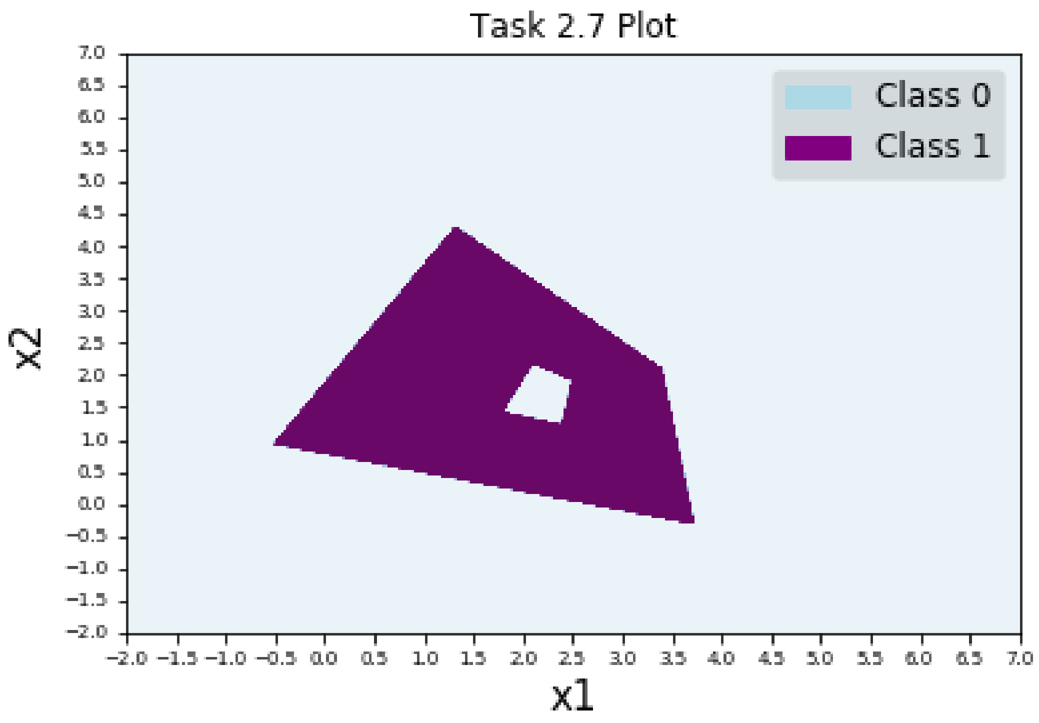

Figure 2: hNN function sharp edges

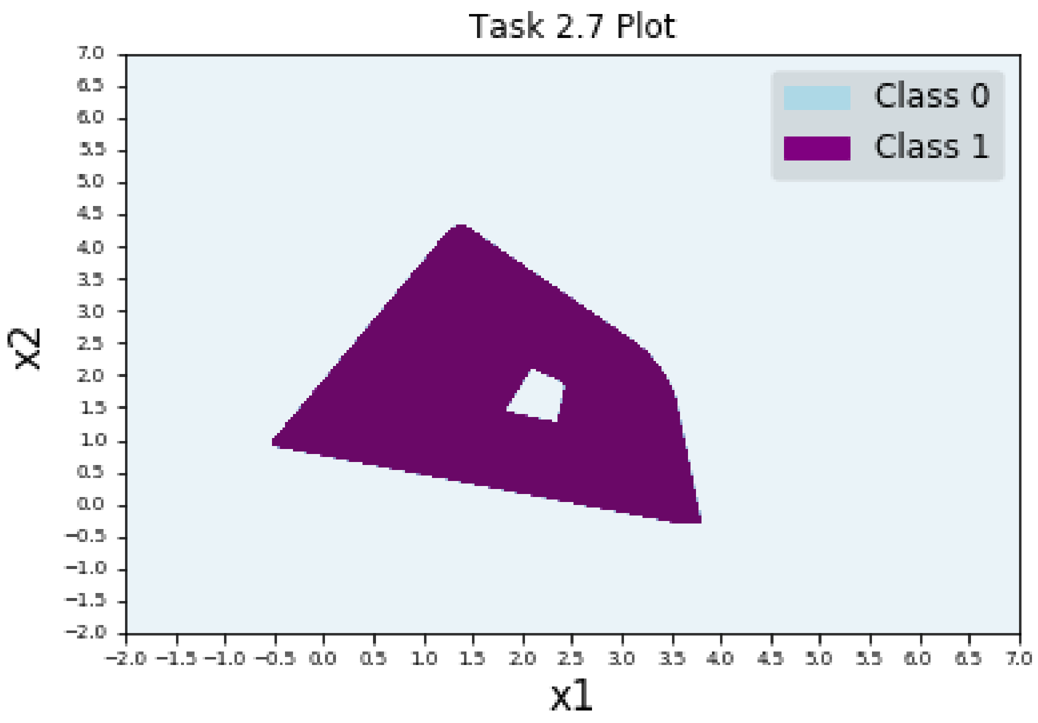

Figure 3: sNN function round edges

The sigmoid function due to nature around zero and one leads to classification problems and round edges. It has two horizontal asymptotes at $y = 0$ and $y = 1$ and looks like a smooth step function. The sigmoid function converts linear inputs to non-linear outputs and is used in more complex neural networks where we see the use of back-propagation, which would be impossible to do with a logical perceptron as used in

The smoothness around the edges comes due to point on the edges being classified around 0.5. Now, to classify them we use 0.5 as the point to separate class 1 and class 2 which leads to inconsistent decisions. To fix this we multiply the weights by a multiplier value. By the increasing the multiplier value we can eliminate these inconsistencies. At around 350 we see most of the points and edges being classified correctly. For perfect classification we need a very large value.

Therefore for this task `hNN_AB` **performs better than** `sNN_AB` .

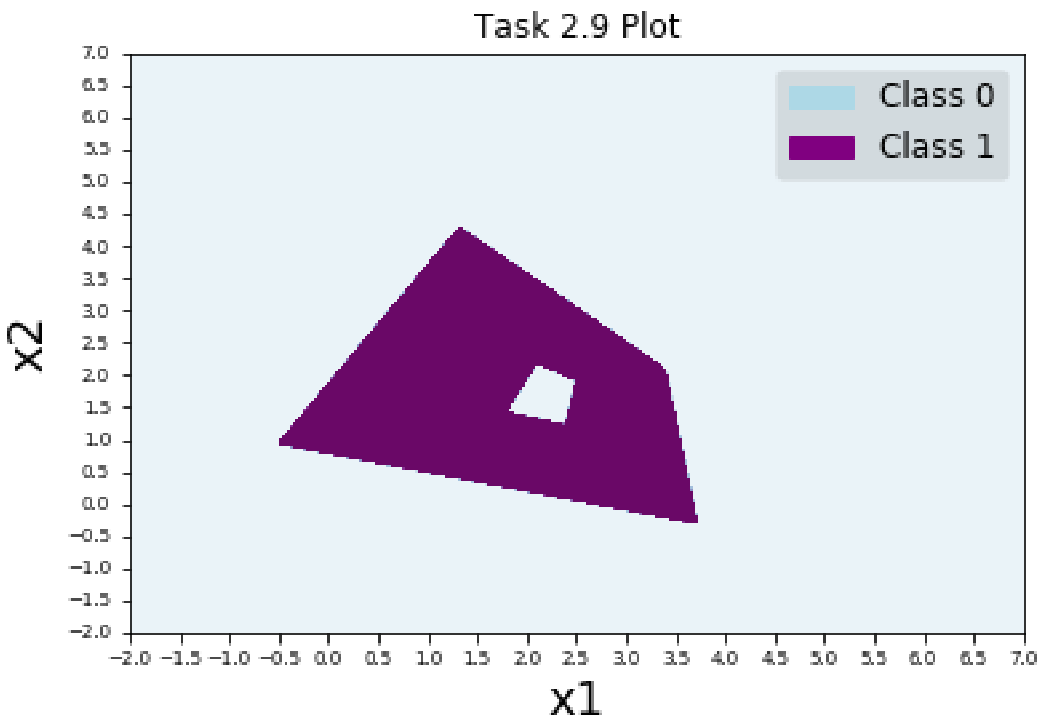

Figure 4: sNN function output when weights multiplied by 300.