Learning Coursework: Task 2 report

April 10, 2020

2.3 - Polygon A classification

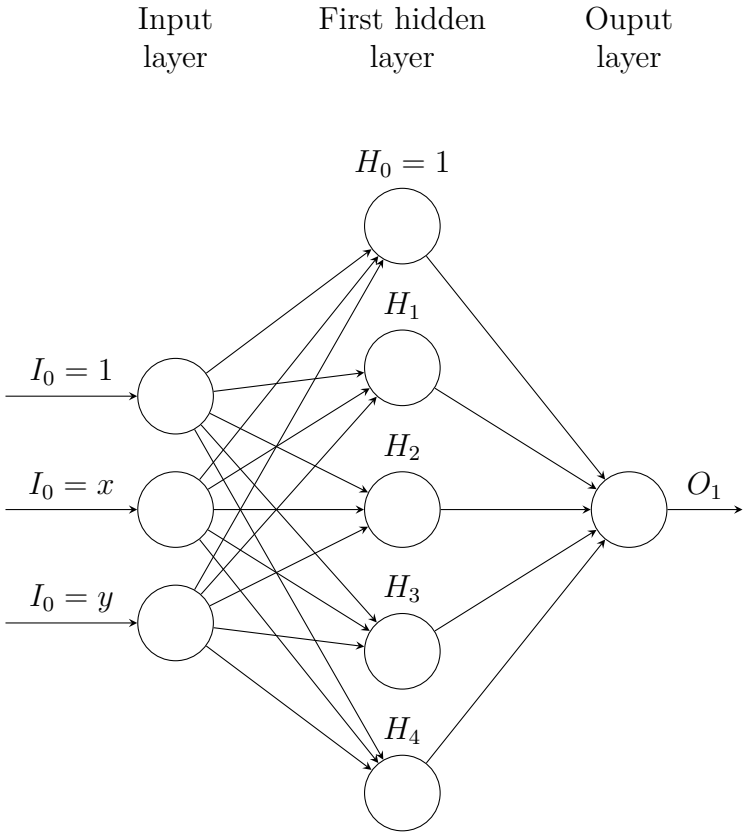

Figure 1: **Polygon A classification - neural network layout**

Weights

1. The coordinates of the point we are classifying are in the form: (x, y). Therefore, we need to put them in the input layer along with a one: $I_0 = 1, I_1 = x, I_2 = y$.

2. As for the first hidden layer, we need a neuron for each edge of the polygon. Starting from the upper-left edge (as seen on Figure 2) and going along the polygon in counter-clockwise order, each neuron decides if the input point is below/above the line that passes through the corresponding edge.

   (a) $\underline{H_1}$: **1** if below the line that passes through the first edge; otherwise **0**

   (b) $\underline{H_2}$: **1** if above the line that passes through the second edge; otherwise **0**

   (c) $\underline{H_3}$: **1** if above the line that passes through the third edge; otherwise **0**

   (d) $\underline{H_4}$: **1** if below the line that passes through the fourth edge; otherwise **0**

   For example, for H_1 we need to make sure that the value of the neuron corresponds to the fact that the following inequality holds (approximately): $y < 1.116 * x + 1.170$ as $y = 1.115 * x + 1.170$ is the equation of the line that passes through the upper-left edge. Now, we can transform this inequality into vector form and normalise it by dividing by the biggest absolute value (1.170 in this case):

$$\begin{bmatrix}1\\x\\y\end{bmatrix} \cdot \begin{bmatrix}1.170\\1.116\\-1\end{bmatrix} > 0 \Leftrightarrow \begin{bmatrix}1\\x\\y\end{bmatrix} \cdot \begin{bmatrix}1\\0.954\\-0.855\end{bmatrix} > 0 \tag{1}$$

   Now, we can take the vector on the right as the weight vector for H_1. Once the step function is applied, this will ensure that the requirements for the output of H_1 set above are met. For H_2, H_3, H_4, we can proceed in an equivalent fashion.

3. Finally, the output layer weight are constructed so that the output is 1 if and only if $H_1 = H_2 = H_3 = H_4 = 1$. This can be achieved by taking $[-\frac{4}{5}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$ as the weight vector. If at least one of H_1, H_2, H_3, H_4 is equal to 0, then the following will hold (as the rest can only sum up to $\frac{3}{4}$ at maximum, which is smaller than $\frac{4}{5}$):

$$\begin{bmatrix}1\\H_1\\H_2\\H_3\\H_4\end{bmatrix} \cdot \begin{bmatrix}-4/5\\1/4\\1/4\\1/4\\1/4\end{bmatrix} < 0 \tag{2}$$

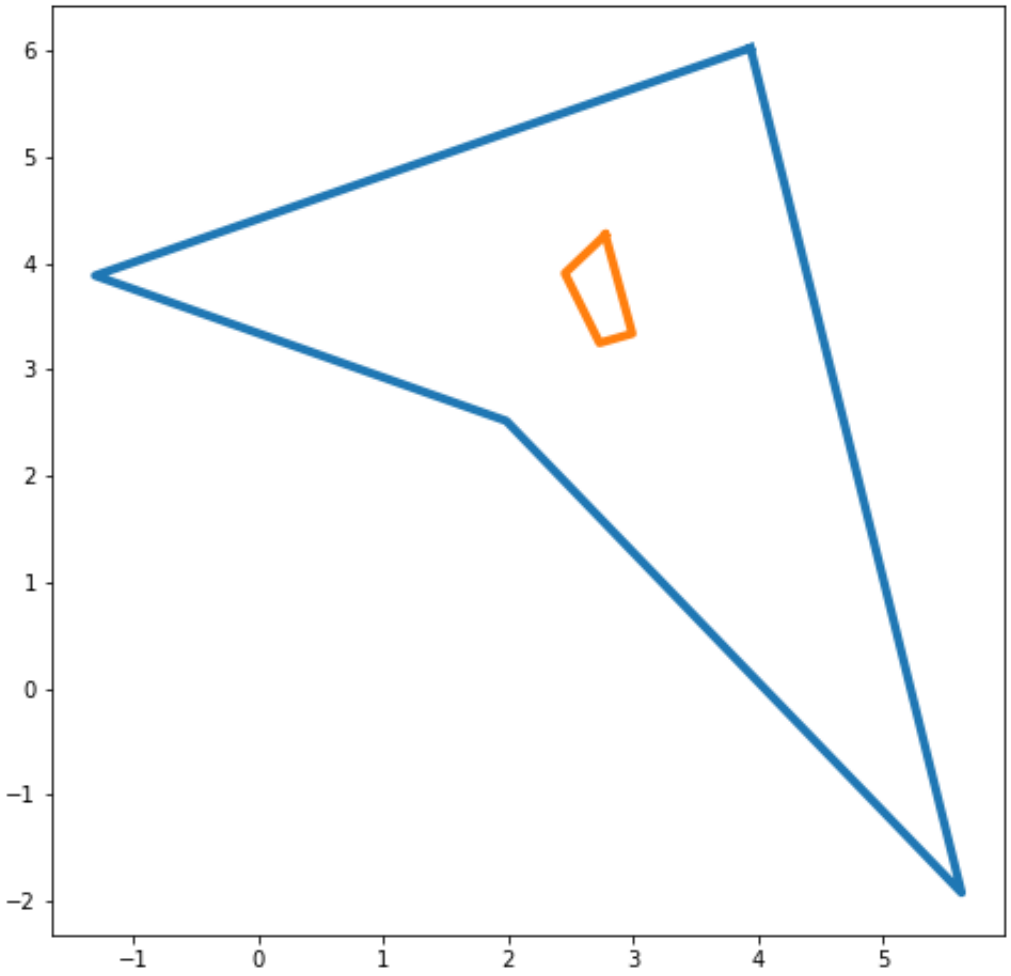

Figure 2: **Polygons A, B**; The A polygon is in orange, the B polygon is in blue.

2.10 - Different decision regions for step/standard logistic functions

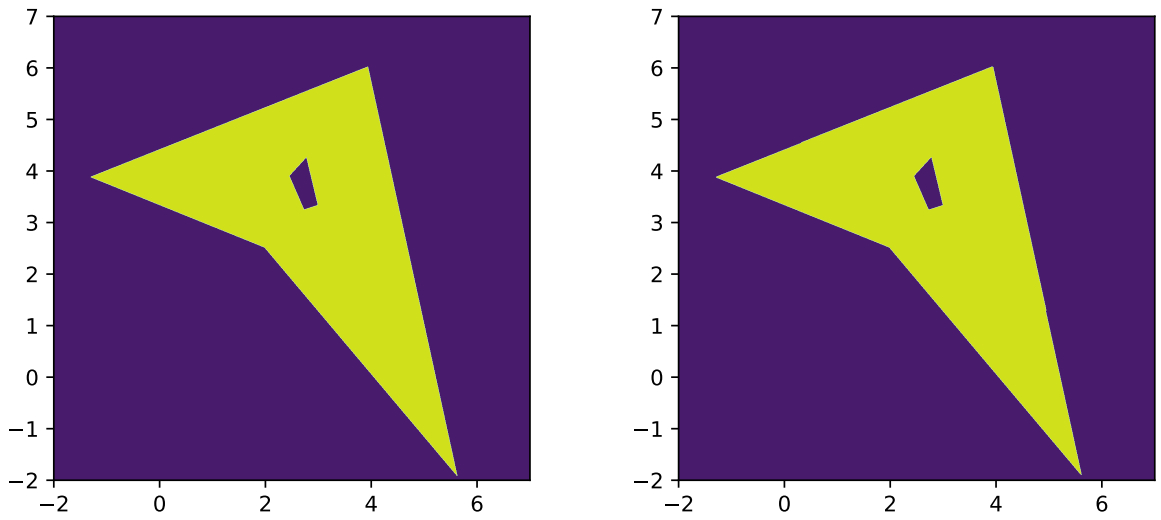

Figure 3: Decision regions when using the step and the standard logistic function, respectively; graphs of regions of size 9x9.

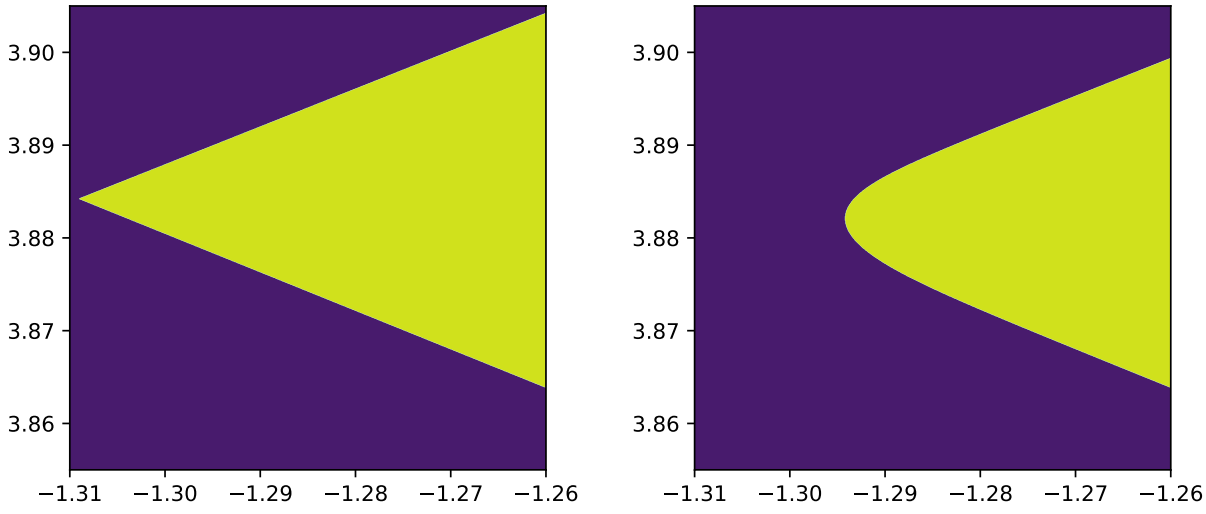

Figure 4: Decision regions when using the step and the standard logistic function, respectively; graphs of regions of size 0.05x0.05.

In order to approximate the decision regions as closely as posible, I have multiplied the weights used with the step function by a scalar (1000). Such a transformation does not make a difference when using the step function, but it makes a significant difference when using the standard logistic function. Figure 5 shows what effect this has on a standard logisitc function.

In order to make it approximate the step function even closer, we could, in theory, make the scaling factor even bigger but this does not prove to be practical - it causes an overflow when using standard floating-point arithmetic.

Figures 3, 4 show that approximating the step function in this way works well, but it does not solve the problem entirely. However big the scaling factor is, there will always be misclassified points. By making it bigger, we restrict their number.

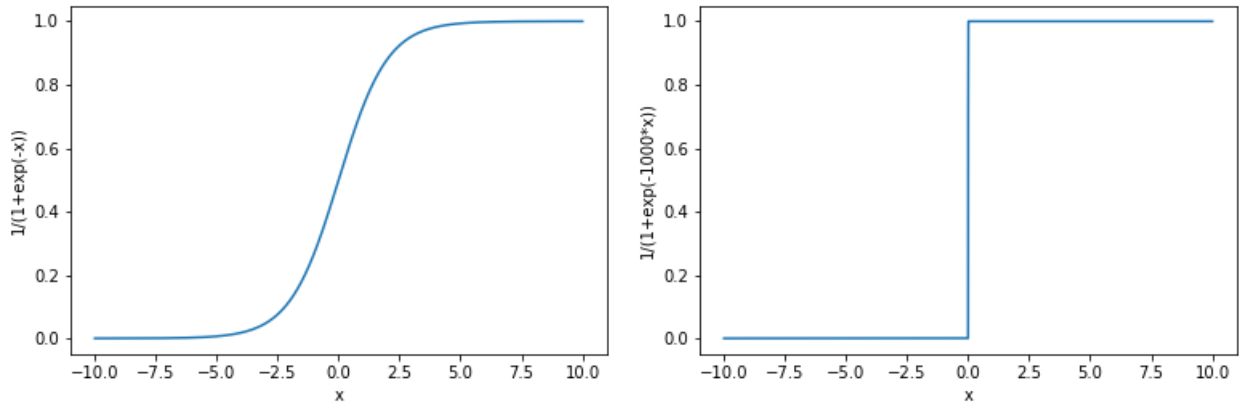

Figure 5: Comparison of standard logistic functions for differently scaled weights. The graph on the right has x initially scaled by 1000.