Artificial Neural Network

# Exercise 1

**Show the output from your program reproducing the AND learning in the final lecture slides, where the weights are outputted at every application of the weight-change rule.**

When we first started our perceptron, we decided upon 6 points that could be separated by the line . These 6 points were as follows:

* ( 6, 9), ( 1, 8), ( -4, 2), ( 1, -1), ( -3, -4), ( 7, 2)

The original graph with points can be found in Appendix A.

Here we can see our perceptron updating the weights in order to create a line which will be similar enough to x = y to still separate out points. The original weights are randomly generated values between 0 and 1:

Text

Description automatically generated

A visual representation of our graph along with 5 lines drawn by our perceptron shows that randomly generated values for the weights will result in the perceptron outputting various different lines which separate our points. Pictured in red is the line from the example above:

Chart

Description automatically generated

# Exercise 2

**Adjust your program so that it calculates and show that it works.**

To explain this question we dove deeper into what the significance of the weights in our perceptron are, and we figured out that they can be used to create a line.

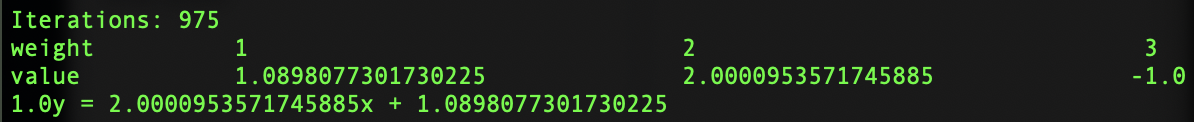
If we have a weight vector [1,2,3], it would represent the following line:

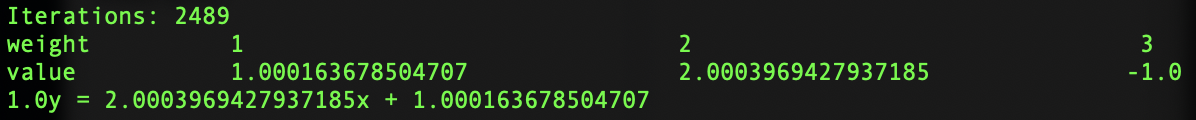
This can be rearranged to:

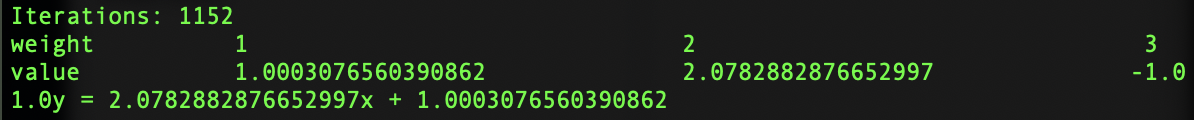
In order to obtain a line in the for , where we have exactly one y, we decided to not allow our perceptron to edit the weight that represents y. In other words, we will only let our perceptron update the weights for x and the constant (b).

Now in order to let our perceptron find the line , we would have to generate two groups of inputs which are linearly separable by the line . To create our points, we used a random point generator which assigned half of the points above the line, and half of the points under the line, and put our perceptron to work. As is shown in exercise 3, we would need only two points to create the line, however, the more points we use, the more accurate our line is likely to be. For the purpose of this exercise we will be using 100 points; 50 above the line, and 50 below. Furthermore, to make sure out points are close enough to the line to make a relevant impact, we are only generating the points with max/min values of 10/-10.

The resulting weights generated by our perceptron were both expected and unexpected. Although we anticipated that our perceptron would generate the line , it never quite reached the target. The use of double types in our weight vector made it nearly impossible for our perceptron to reach exactly, however, most of our results amounted to lines which were incredibly close to the desired line.







These examples demonstrate that there are actually an infinite amount of lines which could separate our points accurately if you just add enough decimal places to the coefficient of x and the constant.

# Exercise 3

**What is the effect when you add more target values (if there is any effect...)? Try to show data/results of running your program that support your answer.**

For this exercise, we will be using the same random point generator as above, but for the sake of simplicity, our target line this time will simply be . To find out the effect of adding more target values, we will simply be increasing the amount of points that our point generator will make, and then we will run the perceptron 1000 times to see what that average weights will be, and what line they will draw.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # of points | W1 (constant) | W2 (x-value) | W3 (y-value) | Line |
| 2 | 0.45914 | 0.79083 | -0.85018 | 0.85018y = 0.79082x + 0.45914 |
| 10 | 0.34967 | 4.23365 | -5.17335 | 5.17335y = 4.23365x + 0.34967 |
| 100 | -0.55502 | 49.10631 | -49.4799 | 49.4799y = 49.10631x - 0.55502 |
| 1000 | 0.00728 | 24.17015x | -24.17355y | 24.17355y=24.17015x+-0.00728 |

At first glance, it’s not very clear what all these number mean, so let’s break it down into the two components of a line in 2D: the slope and the y-intercept. For reference, the line has a slope of 1 and a y-intercept of 0.

|  |  |  |
| --- | --- | --- |
| # of points | Slope | y-intercept |
| 2 | 0.93018 | 0.54005 |
| 10 | 0.81836 | 0.06758 |
| 100 | 0.99245 | 0.01122 |
| 1000 | 0.99986 | 0.00030 |

As you can see from the processed data, the average line given by the weights of the perceptron over 1000 runs clearly shows that the line gets gradually closer to the target line when we increase the amount of target values.

# Appendix

### Appendix A

A picture containing diagram

Description automatically generated