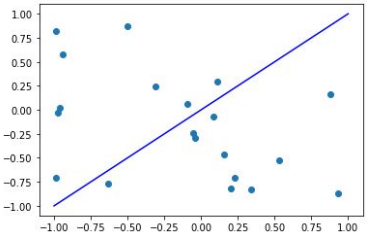
Homework 1: Perceptron algorithm

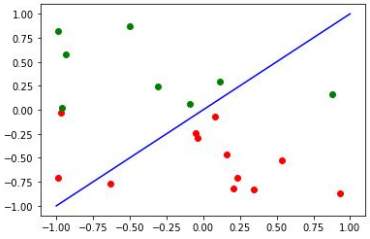
Generate a dataset of two-dimensional points, and choose a random line in the plane as your target function f , where one side of the line maps to Å1 and the other side to ¡1. Let the inputs xn 2 R2 be random points in the plane, and evaluate the target function f on each xn to get the corresponding output yn Æ f (xn). Experiment with the perceptron algorithm in the following settings:

TEST	Size	Dims	Iterations	Execution time (s)	Execution time (s) without any plot	Accuracy
B	20	2	5	0.959	0.00099	100%
C	20	2	13	2.385	0.00099	100%
D	100	2	3	1.039	0.00099	100%
E	1000	2	9	25.007	0.00702	100%
F	1000	10	4594	-	**7.924779**	**99.80%**

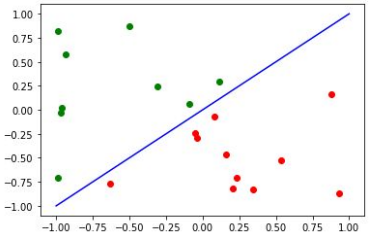
Results for: Size = 20 and dimensions = 2



First look

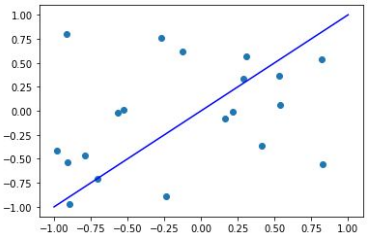


First iteration

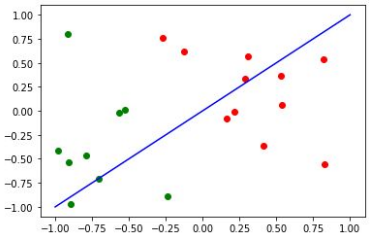


Last iteration (5)

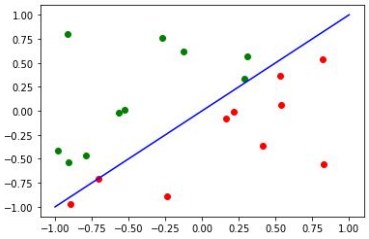
Results for: Size = 20 and dimensions = 2



First look

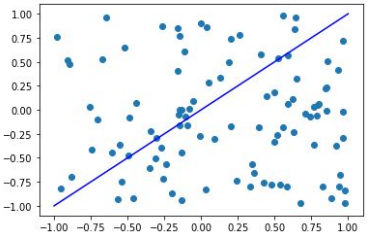


First iteration

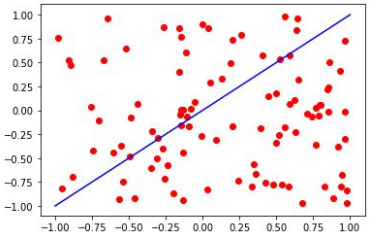


Last iteration (13)

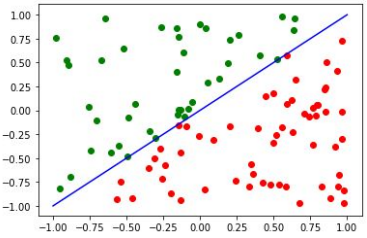
Results for: Size = 100 and dimensions = 2



First look

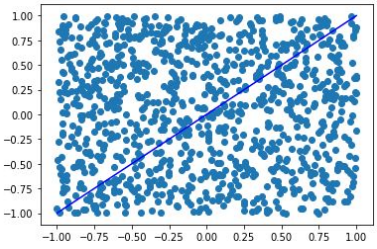


First iteration

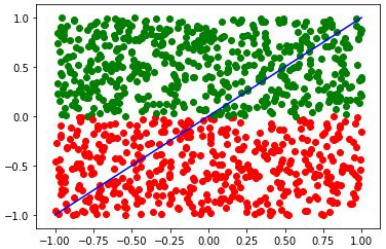


Last iteration (3)

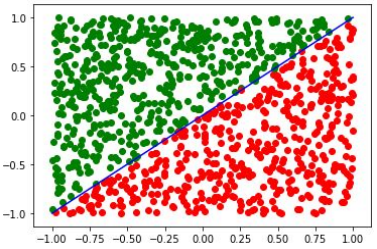
Results for: Size = 1000 and dimensions = 2



First look



First iteration



Last iteration (9)

Conclusions:

- By initializing the weights to a random number and iterating through them, we can see that the optimal weight coefficients are automatically learned.
- If we take a look at the number of iterations, we can see that between the same dimension (2), the number of iterations doesn't vary that much regardless of the input size. For the same size (20) I got first 5 iterations to converge, and then with different random numbers, I got 13. However, for size = 100, my result converged in 3 iterations and for size = 1000 it converged in 9 iterations. At first it may look like the bigger the size, the larger the number of iterations but this doesn't seem to be the case. On the other hand, when we increase the number of dimensions to 10, the iterations value increases significantly. For 10 dimensions and size 1000, the algorithm took 4594 iterations to converge. Which is way distant from the previous values obtained, specially the one with the same size (1000) but different number of dimensions (2).

- Regarding the execution time, two different measures have been taken since plotting the results per every iteration is time consuming. The execution time that has been taken into consideration is the one where no results are being visually shown. On this basis, it can be seen that the higher the size, the more time it takes to converge. For small sizes like 20 or 100, the convergence is almost immediate. For higher values, the number increases a bit. However, as we increase the number of dimensions to 10, the execution value does affect. In this case it changes from being almost 0s in the previous cases to being almost 8.
- Maybe for some cases, especially depending on the number of dimensions, a maximum number of iterations could be defined. This would reduce the execution time but would obtain worse results.
- In this case, the accuracy is always nearly 100% except for the last case where it is 99.80%. I assume that with a larger number of iterations and a large number of points, the chances of getting a wrong classification increases. In addition, maybe some improvements could be done to the algorithm. However, a 99.80% of accuracy is almost perfect.

  This is how accuracy has been calculated:

$$Accuracy_{N,d} = \frac{N - number\ of\ mis\ classifications}{N} \times 100$$