Exercise 4 TDT4171 Sondre Foslien

April 19, 2018

I Gradient decent

The plot from task 1.1 can be seen in figure 1. From inspecction it seems like the minimum cost are for the values $\omega_1 = -5.9$ and $\omega_2 = 2.9$. Here the total cost is $L_{simple}(\omega) = 0.004976671$

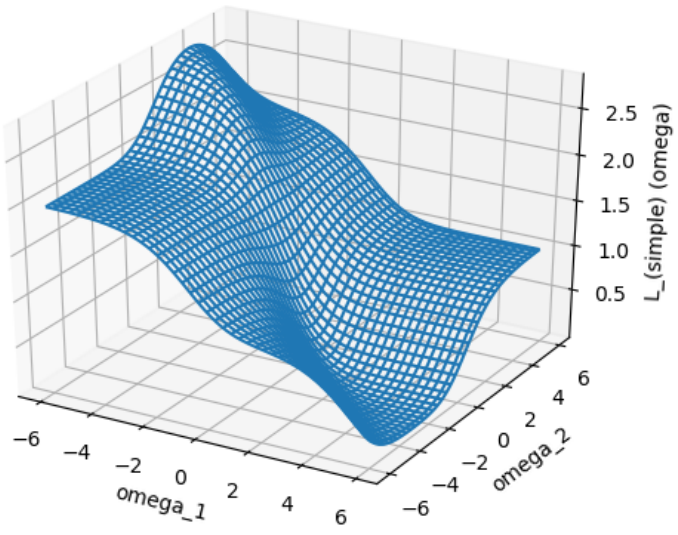

Figure 1: Plot from task 1.1. Shows the cost $L_{simple}(\omega)$ plotted versus the weights ω_1 and ω_2

The derivative of the cost-function was calculated to be:

$$\begin{aligned}\frac{\partial L_{simple}(w)}{\partial w_1} =& 2 \cdot (\sigma(w, [1,0]) - 1) \cdot \sigma(w, [1,0]) \cdot (1 - \sigma(w, [1,0])) + \\ & 2 \cdot (\sigma(w, [1,1]) - 1) \cdot \sigma(w, [1,1]) \cdot (1 - \sigma(w, [1,1]))\end{aligned} \tag{1}$$

$$\begin{aligned}\frac{\partial L_{simple}(w)}{\partial w_2} =& 2 \cdot (\sigma(w, [0,1]) - 1) \cdot \sigma(w, [0,1]) \cdot (1 - \sigma(w, [0,1])) + \\ & 2 \cdot (\sigma(w, [1,1]) - 1) \cdot \sigma(w, [1,1]) \cdot (1 - \sigma(w, [1,1]))\end{aligned} \tag{2}$$

Then the actual gradient decent algorithm were implemented, and tested for different values of η, the learning rate. The learning-rate decides how fast the algorithm will walk towards the steepest decent. It were tried for $\eta = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]$. The results for each iteration can be seen in figure 2. It is clear that the cost is reduced for increasing learning-rates until $\eta = 10$. This is where the cost is at it's lowest with $L_{simple}(\omega) = 0.001310$. For $\eta = 100$ it starts increasing again. Now we have been overambitious and a too fast learning rate results in that the system does noe converge towards a solution and oscillates around the best solution. Running it for more iterations might result in it very slowly converging or simply oscillating out into infinity. For the smaller values The cost gets close to the minimum but since the learning-rate is so slow it is not able to reach the minimum during the given iterations. Given more iterations the will reach it, just awfully slowly.

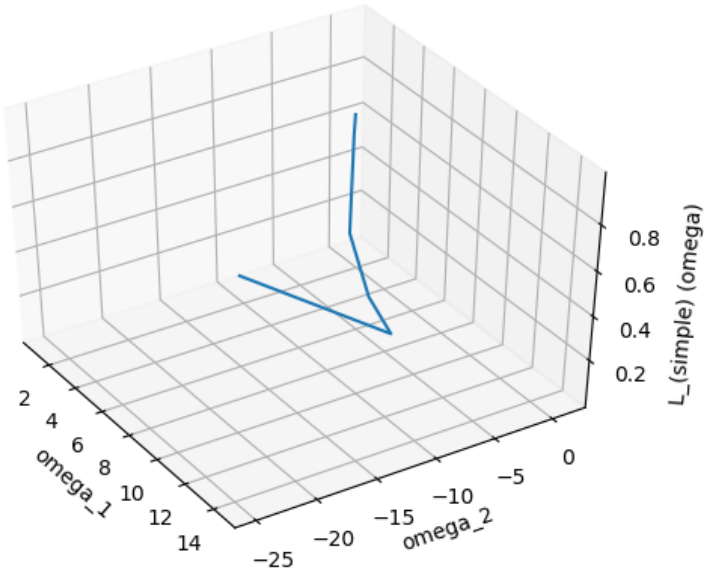

Figure 2: Plot from task 1.2. Shows the cost $L_{simple}(\omega)$ plotted versus the weights ω_1 and ω_2 found with the different learning rates η

II Perceptron algorithm with logistic function

The derivative of the cost function was calculated to be:

$$\frac{\partial L_n(w, x_n, y_n)}{\partial w_i} = [\sigma(w, x_n) - y_n] \cdot x_n \cdot \sigma(w, x_n) \cdot (1 - \sigma(w, x_n)) \tag{3}$$

The perceptron training algorithms were ran for both the separable and the nonseparable dataset with 10 iterations and a learning rate of 0.1. The code was executed five times. The results is shown in the following table.

Separable dataset

Execution no.	Batch training	Stochastic training
1	Time: 6.15 Error: 0.485	Time: 0.58 Error: 0.421
2	Time: 6.16 Error: 0.369	Time: 0.52 Error: 0.436
3	Time: 6.14 Error: 0.388	Time: 0.55 Error: 0.499
4	Time: 6.20 Error: 0.474	Time: 0.54 Error: 0.500
5	Time: 6.19 Error: 0.502	Time: 0.52 Error: 0.348

Nonseparable dataset

Execution no.	Batch training	Stochastic training
1	Time: 6.28 Error: 0.500	Time: 0.53 Error: 0.497
2	Time: 6.29 Error: 0.486	Time: 0.52 Error: 0.500
3	Time: 6.32 Error: 0.500	Time: 0.56 Error: 0.500
4	Time: 6.46 Error: 0.500	Time: 0.61 Error: 0.499
5	Time: 6.33 Error: 0.485	Time: 0.65 Error: 0.500

The batch training uses a lot more time, but is more stable and has at average a better result. Stochastic is very effective, but has worse results. Not that much worse though, so it might be wort it depending on the problem. It is quite clear that the nonseparable dataset was harder to learn to separate. It took longer time for both batch and stochastic and did not perform that much better than just guessing. (50% right)

In figure 3 there is a figure cassifying the different entries in the big separable dataset. The numbers of iterations were set high enough so that it had an error of 0 %.

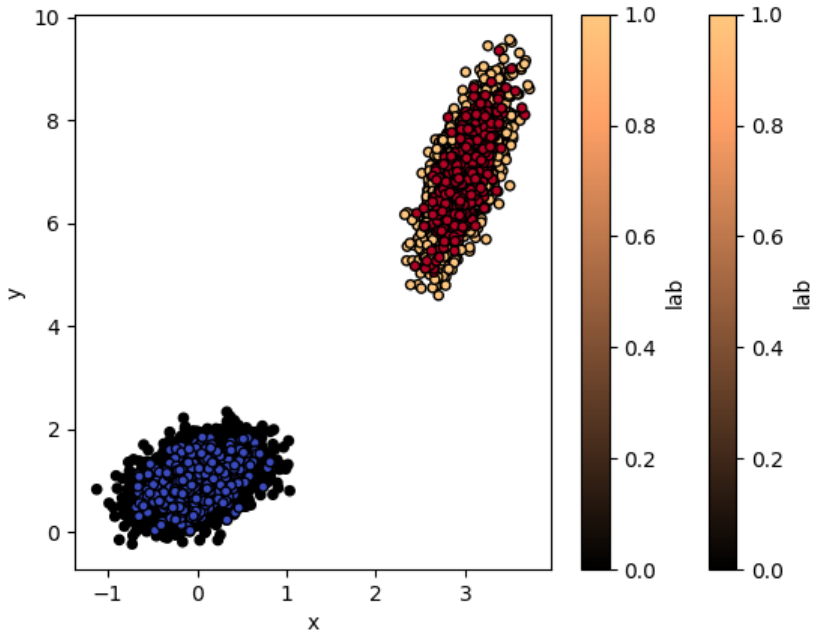

Figure 3: Plot from task 2.3. Shows the separable dataset, both the training-set and the classified test-set.

The stochastic learning algorithm were run on the separable data-set using varying numbers of iterations. (10, 20, 50, 100, 200, 500). The result can be seen in figure 4. As can be seen the error rate rops to zero as the number of iterations gets bigger. The more iterations the more the perceptron will have trained and it will be better to classify. It makes sense that the separable data-set has an error rate of 0 % since it is separable. The execution-time also increases as the number of iterations increases, which also makes sense. More iterations means more computations, which take longer time.

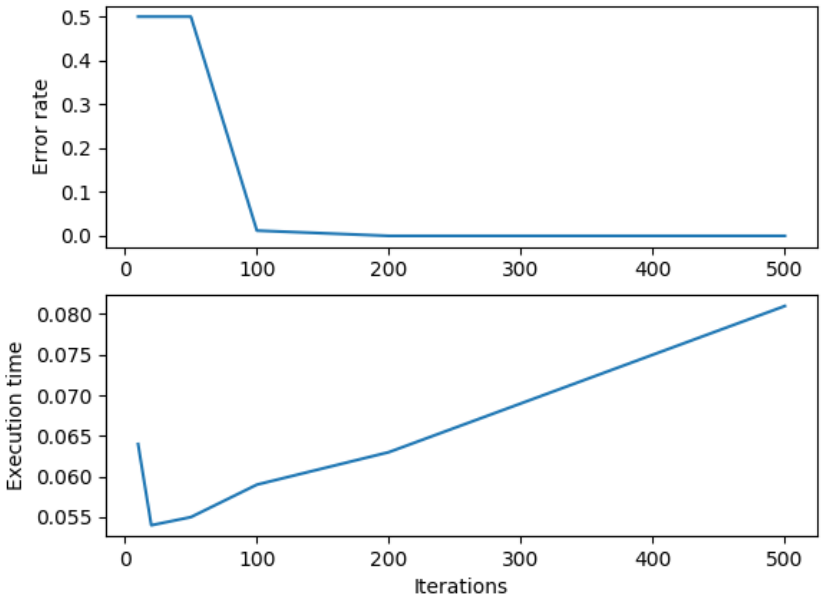

Figure 4: Plot from task 2.3. Show the execution time and error rate for different numbers of iterations.