Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

2nd Lab Assignment

Shift Tuesday Group number 2

Number 81398 Name Maria Carolina Roque

Number 81013 Name José Coelho

Function Optimization – The Gradient Descent Method

1 Introduction

In many situations, it is necessary to optimize a given function, i.e., to minimize or maximize it. Most machine learning methods are based on optimizing a function that measures the performance of the system that we want to train.

This function is generically called *objective function,* because it indirectly defines the objective of the training. Frequently, this function measures how costly are the errors made by the system. In that case, the function is called c*ost function*, and the purpose of training is to minimize it. Since this is the most common case, in this assignment we'll study function minimization. However, all the conclusions can be applied, with the appropriate changes, to the case of function maximization.

In most cases of practical interest, the function that we want to optimize is very complex. Therefore, solving the system of equations that is obtained by setting to zero the partial derivatives of the function with respect to all variables, is not practicable. In fact, these equations are usually highly nonlinear, and the number of variables is often very large, on the order of hundreds, thousands, or even more. In those cases, iterative optimization methods have to be used.

2 The gradient descent method

One of the simplest and most frequently used optimization methods is the method of gradient descent. Consider a function $f(\boldsymbol{x})$ that we want to minimize, where the vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$ is the set of arguments. The gradient of *f*, denoted by ∇f, points in the direction that makes *f* increase fastest. Therefore, it makes sense that, in order to minimize the function, we take steps in the direction of the negative gradient, $-\nabla f$, which is the direction that makes *f* decrease fastest. The gradient method consists of the following steps:

- Choose an initial vector $\boldsymbol{x}^{(0)}$.
- Update $\boldsymbol{x}$ iteratively, according to the equation:

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - \eta \, \nabla f[\boldsymbol{x}^{(n)}].$$

The parameter η is chosen by the user, and must be positive. It is clear, from the previous equation, that the method consists of a succession of steps, each taken in the direction that $-\nabla f$ has at the current location. The iterations stop when a given stopping criterion, chosen by the user, is met.

In this lab we'll study the gradient method in order to gain experience about the way it works. We'll also study some modifications to this method, which are intended to increase its efficiency.

2.1 Minimization of functions of one variable

We'll start by studying the gradient method in the simplest situation, which corresponds to minimizing functions of only one variable. Namely, we'll minimize a function of the form $f(x) = a\, x^2/2$. The parameter *a* controls the functions' curvature.

Start Matlab and change to the directory where you placed the files related to this lab assignment. Then type the command **`quad1ini`**, which initializes the necessary parameters for the tests that you will run. This command initializes the following values:

$$a = 1,\ \eta = 0.1,\ x^{(0)} = -9.$$

Type the command **`quad1`**, which performs the optimization and graphically shows its evolution. The stopping criterion consists in finding a value of f under 0.01 (this value can be controlled by the variable **`threshold`**), with a maximum of 1000 iterations (this value can be controlled by the variable **`maxiter`**).

The variable **`anim`** controls the graphic animation. Setting **`anim=1`** makes the evolution visible as it progresses. This allows us to get a better idea of the evolution, but also makes it take longer. Setting **`anim=0`** shows the plot only at the end of the evolution, which makes it go considerably faster.

1. Fill the following table with the numbers of iterations needed to optimize the function, for different values of a (**`a`** in Matlab) and η (**`eta`** in Matlab). If more than 1000 iterations were needed, write ">1000". If the optimization method diverged, write "div". In the last two lines, instead of the number of iterations, write the approximate values of η that correspond to the fastest optimization and to the threshold of divergence.

η	$a = 0.5$	$a = 1$	$a = 2$	$a = 5$
.001	>1000	>1000	>1000	990
.01	760	414	223	97
.03	252	137	73	31
.1	75	40	21	8
.3	24	12	5	8
1	6	1	threshold	div
3	6	div	div	div
Fastest	2	1	0.5	0.2
Divergence threshold	4	2	1	0.4

Table 1

2. Comment on the results from the table.

À medida que η varia verifica-se que existe uma variação do número de iterações necessárias para o método convergir para o mínimo da função. O método converge mais rapidamente para um dado valor de η_{opt} que se pode demonstrar ser dado por $1/a$, para esta família de funções. Para valores inferiores a η_{opt}, o aumento de η provoca uma diminuição do número de iterações. Quando se ultrapassa o valor de η_{opt}, o número de iterações volta a aumentar, até se atingir o valor do *divergence threshold*, no qual o método não converge nem diverge, ficando a oscilar de forma indefinida até se atingir a condição de >1000 iterações. A partir deste valor, o método diverge. É também possível demonstrar que, para esta família de funções, o valor de η que corresponde ao *divergence threshold* é dado por $2/a$.
O valor de a permite controlar a abertura da curva: ao aumentar o seu valor a curva fica "mais fechada". O gradiente é dado por ax, e portanto, para o mesmo η, quanto maior o valor de a, menos iterações são precisas. Contudo, o aumento de a também faz com que o valor de η_{opt} seja mais baixo.

3. How many steps correspond to the fastest optimization, for each value of *a*? Does there exist, for every differentiable function of one variable (even if the function is not quadratic), and for each given starting point $\mathbf{x}^{(0)}$, a value of η that optimizes the function in that number of steps? Assume that the function grows to $+\infty$ when $\|\mathbf{x}\| \to \infty$.

A otimização mais rápida corresponde a ter apenas uma iteração, assumindo que o valor inicial não é o mínimo.
Para qualquer função de uma variável, diferenciável, é possível determinar o valor de η que permite a optimização em apenas uma iteração. O valor de η_{opt} é dado por $(x^{(0)}-x_{min})/\nabla f(x^{(0)})$.
Se a função for convexa, o valor de x_{min} corresponde ao mínimo global (e único) da função.
Contudo, se a função não for convexa (o que não impede que cresça para $+\infty$ quando $\| x \| \rightarrow \infty$), o valor de η obtido pode fazer a função ser minimizada para um mínimo local e não para o mínimo global, o qual seria a melhor solução. Tal depende do ponto inicial e da sua posição em relação aos máximos e mínimos da função.

2.2. Minimization of functions of more than one variable

When we deal with functions of more than one variable, new phenomena occur in the gradient method. We'll study them by minimizing functions of two variables.

We'll start by studying a simple class of functions: quadratic functions of the form $f(x_1, x_2) = (ax_1^2 + x_2^2)/2$. For these functions, the second derivative with respect to x_1 is *a*, and the second derivative with respect to x_2 is 1.

Type the command **`clear`**, to eliminate the variables used in the previous test, and then type the command **`quad2ini`**, which initializes the necessary parameters for the tests that you'll run next. This command initializes the following values:

$$a = 2,\ \eta = 0.1,\ \mathbf{x}^{(0)} = (-9,9).$$

Type the command **`quad2`**, which performs the optimization and shows the results. The stopping criterion corresponds to finding a value of *f* smaller than 0.01 (this value can be controlled by the variable **`threshold`**), with a maximum of 1000 iterations (this value can be controlled by the variable **`maxiter`**).

Observe that, along the trajectory, the steps are always taken in the direction orthogonal to the level curves of *f*. In fact, the gradient is always orthogonal to these lines.

1. Fill the first column of the following table for the various values of η. Then set $a = 20$ (which creates a relatively narrow valley) and fill the second column. Use the same rules for filling the table as in the preceding case. You may find the values for η that correspond to the fastest optimization and to the threshold of divergence by trial and error.

η	$a = 2$	$a = 20$
.01	448	563
.03	148	186
.1	43	threshold
.3	13	div
1	threshold	div
3	div	div
Fastest	~0.65	~0.095
Divergence threshold	1	0.1

Table 2

2. Comment on the results from the table.

O aumento de *a* origina um vale mais estreito, o que origina oscilações, tornando assim o método do gradiente ineficiente para valores de *a* elevados. Verifica-se, pela Tabela 2, que à medida que se aumenta o parâmetro *a*, para o mesmo η, são necessárias mais iterações para a minimização da função, se não divergir. Tal é também visível pelas curva de nível, as quais ficam aproximadamente paralelas (localmente, junto ao minimo da função [0,0]), o que origina oscilações em torno do mesmo ponto, dado que o gradiente é sempre ortogonal às curva de nível.

3. Is it always possible, for differentiable functions of more than one variable, to achieve, for any given $\mathbf{x}^{(0)}$, the same minimum number of iterations that was reached for functions of one variable? What is the qualitative relationship between the valley's width and the minimum number of iterations that can be achieved?

No caso de funções com apenas uma variável, viu-se que, dependendo da função, seria possível convergir com apenas uma iteração. No caso de funções com mais que uma variável, o mesmo não se verifica para todas as condições iniciais. Sabe-se que o gradiente de uma função é sempre ortogonal às curvas de nível, e por isso, o ponto vai deslocar-se segundo uma linha com a direção do gradiente. No caso da família de funções estudadas, apenas será possível convergir com apenas uma iteração caso o ponto inicial $\mathbf{x}^{(0)}$ seja escolhido de forma a que uma das suas componentes seja igual a zero. Dessa forma, o simétrico do gradiente aponta no sentido do mínimo e, escolhendo um valor de η_{opt} é possível que se obtenha o mínimo em apenas uma iteração. Como, na generalidade, não se conhece o mínimo da função, é de extrema dificuldade encontrar o mínimo com apenas uma iteração.
Quanto mais estreito for o vale, maior o número de iterações necessárias, pois as curvas de nível são mais "paralelas". No caso da largura e comprimento do vale serem iguais (*a*=1), é possível encontrar um η tal que se atinja o mínimo com apenas uma iteração.

3. Momentum term

In order to accelerate the optimization in situations in which the function has narrow valleys (situations which are very common in machine learning problems), one of the simplest solutions is to use the so called *momentum term*. The previous examples showed how the divergence in the gradient descent method is normally oscillatory. The aim of the momentum term is to attenuate

the oscillations by using, at each step, a fraction of the previous one. The iterations are described by:

$$\Delta \mathbf{x}^{(n+1)} = \alpha \Delta \mathbf{x}^{(n)} - \eta \nabla f[\mathbf{x}^{(n)}]$$
$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \Delta \mathbf{x}^{(n+1)}$$

or, alternatively, by

$$\Delta \mathbf{x}^{(n+1)} = \alpha \Delta \mathbf{x}^{(n)} - (1-\alpha)\eta \nabla f[\mathbf{x}^{(n)}]$$
$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \Delta \mathbf{x}^{(n+1)}$$

.

We'll use this second version.

The parameter α should satisfy $0 \le \alpha < 1$. Using $\alpha = 0$ corresponds to optimizing without the momentum term. The term $\alpha \Delta \boldsymbol{x}^{(n)}$, in the update equation for $\Delta \boldsymbol{x}^{(\boldsymbol{n+1})}$, attenuates the oscillations and adds a kind of inertia to the process, which explains the name *momentum term*, given to this term.

The students that are knowledgeable on digital filters, can readily verify that the equation that computes $\Delta \boldsymbol{x}^{(n+1)}$ corresponds to filtering the gradient with a first order low-pass filter, with a pole at $z = \alpha$. This low-pass filtering attenuates rapid oscillations.

1. Still using the function $f(x_1, x_2) = (ax_1^2 + x_2^2)/2$, fill the following table, using $a = 20$, and varying the momentum parameter α. (in Matlab, the parameter α corresponds to the variable **`alfa`**). Use the same rules for filling the table as in the preceding cases.

η	$\alpha = 0$	$\alpha = .5$	$\alpha = .7$	$\alpha = .9$	$\alpha = .95$
.003	>1000	>1000	>1000	>1000	>1000
.01	563	558	552	516	448
.03	186	181	174	115	175
.1	threshold	48	35	91	122
.3	div	threshold	29	83	92
1	div	div	div	92	146
3	div	div	div	div	147
10	div	div	div	div	div
Divergence threshold	0.1	0.3	0.5(6)	1.9	3.9

Table 3

2. Comment on the results from the table.

O método do momento contém mais um parâmetro (α), o qual permite dar maior flexibilidade na minimização de funções. Permite evitar oscilações presentes em problemas como o demonstrado anteriormente, onde as funções a minimizar apresentam vales longos e estreitos.
Verifica-se também que para valores de α mais elevados, a gama de valores para η aumenta, havendo portanto menos valores para o qual o método diverge. Pode-se também concluir que, para o mesmo η, o aumento de α origina menos oscilações (visível graficamente) e portanto, causa uma diminuição no número de iterações necessárias.

4. Adaptive step sizes

The previous examples showed how narrow valleys create difficulties in the gradient descent method, and how the momentum term alleviates these problems. However, in complex problems, the optimization can take a long time even when the momentum term is used. Another acceleration technique that is quite effective relies on the use of adaptive step sizes. This technique will not be explained here, given its complexity. Nevertheless, we'll test its efficiency.

As an example of a function which is difficult to optimize, we'll use the Rosenbrock function, which is one of the common benchmarks used for testing optimization techniques. This function is given by:

$$f(x_1, x_2) = (x_1 - 1)^2 + a(x_2 - x_1^2)^2 .$$

This function has a valley along the parabola $x_2 = x_1^2$, with a minimum at (1,1). The parameter a controls the width of the valley. The original Rosenbrock function uses the value $a = 100$, which creates a very narrow valley. Initially, we'll use $a = 20$, which creates a wider valley, so that we can run our tests faster.

Type the command **`clear`**, followed by **`rosenini`**, which initializes the parameters for the tests that will follow. This command disables the adaptive step sizes, and initializes the following values:

$$a = 20, \ \eta = 0.001, \ \alpha = 0, \ \boldsymbol{x}^{(0)} = (-1.5, 1),$$

Type the command **`rosen`**, which performs the optimization. The stopping criterion corresponds to having two consecutive iterations with f smaller than 0.001, with a maximum of 1000 iterations.

1. Try to find a pair of values for α and η that leads to a number of iterations below 200. If, the number of tests is becoming too large, stop and use the best result obtained so far. Write down how many tests you performed in order to find that pair of values, and fill the following table, using the best value that you found for η, and also values 10% and 20% higher and lower than the best.

N. of tests	α	$\eta \rightarrow$0,065	-20%	-10%	best	+10%	+20%
20	0,9	N. of iterations→	152	166	135	div	div

Table 4

2. Basing yourself on the results that you obtained in the table above, give a qualitative explanation of why it is hard to find values of the parameters that yield a relatively small number of iterations.

Como se pode ver pelos resultados da Tabela 4, verifica-se que pequenas variações de η originam variações significativas no número de iterações necessárias, podendo mesmo o método divergir. Desta forma, é difícil encontrar um par de valores para α e η que origine um número de iterações pequeno, pois, ao testar variações dos parâmetros, por mais pequenas que sejam, obtêm-se valores de iterações diferentes, como se pode observar dos resultados obtidos.

Note that the total time that it takes to optimize a function corresponds to both the time it takes to perform the tests that needed to find a fast enough optimization, plus the time it takes for that optimization to run.

3. Next, we'll test the optimization using adaptive step sizes. Type the command **`assact`**, which activates the adaptive step sizes (**`assdeact`** deactivates them). Fill the following table with the numbers of iterations necessary for the optimization under different situations.

η	$\alpha = 0$	$\alpha = .5$	$\alpha = .7$	$\alpha = .9$	$\alpha = .95$	$\alpha = .99$
.001	596	298	236	140	198	167
.01	565	287	221	190	200	165
.1	769	389	214	183	172	152
1	729	396	233	160	137	173
10	672	383	239	173	124	133

Table 5

Observe how the number of iterations depends little on the value of η, contrary to what happened when fixed step sizes were used (note that, in the table above, η varies by four orders of magnitude). The relative insensitivity to the value of η is due to the fact that the step sizes vary automatically during the minimization (the value given to η represents only the initial value of the step size parameter). The little dependency on the initial value of η makes it easier to choose values that result in efficient optimization.

4. Finally, we'll test the optimization of the Rosenbrock function with the original value of a. Set **a=100**. Try to find values of η and α such that the convergence is reached in less than 500 steps, first without adaptive step sizes, and then with adaptive step sizes. Write down, for each case, the number of tests required to find the final values of η and α. If, in any case, the number of tests is becoming too large, stop and use the best result obtained so far.

   For both cases change the best value of eta by about 10% up and down, without changing α, and write down the corresponding numbers of iterations. Fill table 6 with the values that you obtained.

	N. of tests	η	α	N. of iterations
Without adaptive step sizes	13	-10%	0,95	471
	η=0,021	best		321
		+10%		278
With adaptive step sizes	12	-10%	0,96	405
	η=0,04	best		234
		+10%		364

Table 6

5. Comment on the efficiency of the acceleration methods that you have tested in this assignment.

Ao longo deste trabalho de laboratório foi possível explorar um pouco melhor os métodos do gradiente, do momento, e dos passos adaptativos. Verifica-se que o método do gradiente, embora permita minimizar funções, é, de todos, o menos eficiente, pois a forma da função obtida tem um grande impacto no desempenho do método. O método do momento surge como uma alternativa melhorada do método do gradiente e permite que haja maior flexibilidade, sendo este mais eficiente, necessitando de um menor número de iterações e apresentando o valor do mínimo da função onde o primeiro método estudado não o conseguiu. O aparecimento de um parâmetro adicional permite que a forma da função a minimizar não seja tão relevante, pois os pontos das sucessivas iterações deixam de se mover ao longo da recta definida pelo vetor gradiente, o qual é ortogonal às curvas de nível. Tal é um desafio para o caso de curvas de nível elípticas, onde estão presentes oscilações fortes.
Por fim, estudou-se um pouco do método dos passos adaptativos, no qual o valor de η varia conforme o sinal do gradiente em iterações anteriores, podendo assim acelerar ou diminuir o ritmo de variação do valor de x. Neste método, verifica-se que o valor de η definido inicialmente é pouco relevante, pois o valor vai sendo "corrigido" pelo método de forma a obter os melhores resultados. Podemos então concluir que o método menos eficiente, de forma geral, é o método do gradiente. O método dos passos adaptativos foi o que apresentou melhores resultados.

Do not write below this line.