Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

2nd Lab Assignment

Shift Fri 17h Group number 7

Number 83916 Name João Manuel Prata Morais

Number 97236 Name Francisco Rabaça Möller Freiria

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

$\boldsymbol{x} = (x_1, x_2, \mathsf{L}, x_N)$ 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:

$$x^{(n+1)} = x^{(n)} - 5\,\nabla \mathbf{f}[x^{(n)}].$$

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.

Use function **`quad1`**, which performs the optimization and graphically shows its evolution (to view it in Spyder go to `Tools > Preferences > IPython Console > Graphics > Backend and`

choose Automatic). The stopping criterion consists in finding a value of f under 0.01 with a maximum of 1000 iterations (these values can be controlled by the input parameters **threshold** and **maxiter**).

By default, function **quad1**, initializes the parameters to the following values:

$$x^{(0)} = -9,\ a = 1,5 = 0.1\ .$$

These values can be changed by varying parameters **x_0**, **a** and **eta**.

Parameter **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 and η (**a** and **eta** in **quad1** function). 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. What is the relationship between a (the function's second derivative) and the value of η that corresponds to the fastest optimization? Prove that relationship for this class of functions.

Analisando a influência do parâmetro a nos resultados, pode concluir-se que, para o mesmo valor de η, quanto maior o valor de a, menor será o número de iterações necessárias e logo corresponde a um a otimização mais rápida. O valor de η que leva o método a convergir para o mínimo da função em apenas uma iteração é conseguido igualando $x^{(1)}$ a 0. Resolvendo esta igualdade, e sabendo que a função é diferenciável, conclui-se que η terá de tomar o valor de 1/a.

3. What is the relationship between the function's second derivative and the value of η that corresponds to the divergence threshold? Prove that relationship for this class of functions.

Analisando a função dada, é possível verificar que o método converge mais rapidamente para η igual a 1/a. É também verificável analiticamente que o valor de η que corresponde à divergência threshold é dado por 2/a.

4. Comment on the results from the table.

Analisando os resultados obtidos é possível concluir que o número de iterações se reduz até atingir o valor de η que corresponde ao seu valor ótimo, ou seja, o valor de η para o qual o método converge mais rapidamente. Depois de atingido o valor atrás referido, o número de iterações mantém-se relativamente constante até se verificar a divergência threshold, no qual o método nem converge nem diverge. Quando os valores de η são ainda superiores ao valor que leva à divergência threshold, o método diverge.

5. 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 $x^{(0)}$, a value of η that optimizes the function in that number of steps? Assume that the function has a global minimum.

Assumindo que o ponto inicial não é o mínimo, a otimização mais rápida é conseguida com apenas uma iteração.

Como referido anteriormente, o valor de η que leva o método a convergir para o mínimo da função apenas numa iteração é conseguido igualando x(1) a 0. Resolvendo esta igualdade, e sabendo que a função é diferenciável, conclui-se que η terá de tomar o valor de 1/a.

No caso de a função ser uma parábola com a concavidade voltada para cima, terá apenas um mínimo, de modo que o método vai sempre convergir para o seu mínimo em apenas uma iteração. No que toca a funções que não sejam convexas o método pode não convergir em apenas uma iteração, dependendo assim do ponto inicial.

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.

Use function **quad2**, which performs the optimization and shows the results. The stopping criterion corresponds to finding a value of f smaller than 0.01, with a maximum of 1000 iterations (these values can be controlled by parameters **threshold** and **maxiter**).

By default, function **quad2** initializes the parameters to the following values:

$$x^{(0)} = (-9,9), \qquad a = 2,\ 5 = 0.1.$$

These values can be changed through parameters **x_o**, **a** and **eta**.

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 5 that correspond to the fastest optimization and to the threshold of divergence by trial and error.

η	$a = 2$	$a = 20$
.01	414	414
.03	137	137
.1	40	threshold
.3	12	div
1	Threshold	div
3	div	div
Fastest	0,65	0.092
Divergence threshold	1	0.1

Table 2

2. Comment on the results from the table. What is the qualitative relationship between the valley's width and the minimum number of iterations that can be achieved? Why?

Analisando os resultados da tabela 2, é possível concluir que a influência de η é semelhante ao observado na pergunta 2.1.4. Já com os valores de a, pode concluir-se que com o aumento deste parâmetro, o "vale" fica mais estreito, tornando-se assim um método ineficiente para valores muito elevados de a, acabando mesmo por divergir.

3. Is it always possible, for differentiable functions of more than one variable, to achieve, for any given $x^{(0)}$, the same minimum number of iterations that was reached for functions of one variable?

No que toca às funções com mais de uma variável, contrariamente ao que sucede com as funções de apenas uma variável, não será possível a convergência para o mínimo para todas as condições iniciais. Neste caso, só é possível convergir em uma única iteração no caso em que o ponto inicial tenha uma das componentes nula. Uma vez que o gradiente de uma função é sempre ortogonal às curvas de nível, o seu simétrico irá apontar no sentido do mínimo. Escolhendo assim o valor de η otimizado é possível conseguir atingir o mínimo em apenas uma iteração. Este caso é bastante improvável tendo em conta que geralmente não é conhecido o mínimo da função em estudo.

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 $\alpha\Delta x^{(n)}$, in the update equation for $\Delta x^{(n+1)}$, 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 $\Delta\mathbf{x}^{(n+1)}$ 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 $f(x_1, x_2) = (ax_1^2 + x_2^2)/2$ and **quad2**, fill the following table, using $a = 20$, and varying the momentum parameter α. (parameter α corresponds to variable **alpha**). 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	414	411	406	382	338
.03	137	134	129	96	171
.1	threshold	36	31	85	122
.3	div	threshold	31	67	148
1	div	div	div	74	146
3	div	div	div	div	172
10	div	div	div	div	div
Divergence threshold	0.1	0.3	0.567	1.9	3.9

Table 3

2. Comment on the results from the table.

O método do momento insere um novo parâmetro α. Este método pretende reduzir problemas como os observados nas questões anteriores onde era possível observar várias oscilações e consequentemente um número elevado de iterações. Assim, com o aumento dos valores de α , para o mesmo valor de η, é possível verificar uma redução do numero de oscilações e consequente numero de iterações. O método torna-se assim otimizado quando comparado com os resultados anteriores em que α tomava valor nulo.

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 $x_2 = x^2$, with a minimum at (1,1). The parameter a controls the width of the valley. The original Rosenbrock[1] 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.

Use function **`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.

By default, these function uses the following parameters:

$$a = 20,\ 5 = 0.001,\ \alpha = 0,\ x^{(0)} = (-1.5,\ 1),$$

1. Try to find a pair of values for α and η that leads to a number of iterations below 200 (do it manually, do not use grid search). 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	$\eta\rightarrow$ 0.06	-20%	-10%	Best	+10%	+20%
10	0.9	N. of iterations→	110	123	82	116	117

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 verificar na tabela 4, é dificil obter um par de α e η que minimize o número de iterações abaixo de 200. Este facto e provado pelos resultados acima representados, onde é possível verificar que pequenas variações dos parâmetros atrás referidos resultam em variações significativas do número de iterações.

Para os parâmetros escolhidos, η igual a 0.06 e α igual a 0.9, é possível observar na figura obtida que antes de entrar no vale são realizadas poucas iterações. No entanto, quando se aproxima do mínimo são realizadas muitas iterações. Assim pode concluir-se que pequenas variações destes parâmetros provocam grandes variações no número de iterações a realizar para convergir para o mínimo.

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. Set the following input parameters for function **`rosen`**, and fill the table with the numbers of iterations necessary for the optimization under different situations.

`up` = 1.1, **`down`** = 0.9, **`reduce`** = 0.5

η	$\alpha = 0$	$\alpha = .5$	$\alpha = .7$	$\alpha = .9$	$\alpha = .95$	$\alpha = .99$
.001	401	215	171	101	160	158
.01	384	201	168	165	145	139
.1	575	306	159	149	138	144
1	522	305	169	135	132	123
10	470	292	190	146	113	108

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 (**`up`** = 1, **`down`** = 1, **`reduce`** = 1), and then with adaptive step sizes (**`up`** = 1.1, **`down`** = 0.9, **`reduce`** = 0.5). 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	25	-10%	0.95	314
		final η =0.024		147
		+10%		348
With adaptive step sizes	20	-10%	0.95	294
		final η =0.0349		162
		+10%		354

Table 6

5. Final Remarks

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

   Neste trabalho de laboratório foram testados dois métodos de aceleração de forma a serem comparados com o método sem aceleração ou método do gradiente.
   Tal como foi referido acima, o método do gradiente foi o que originou piores resultados, uma vez que, de uma forma geral e para os mesmos valores de a e η, são realizadas mais iterações do que nos restantes métodos.
   O método do momento consiste numa otimização do método do gradiente, conseguindo atenuar as oscilações com a adição do novo parâmetro α e assim reduzir o número de iterações.
   Finalmente foi analisado o comportamento do método dos passos adaptativos onde o valor de η varia consoante o sinal das iterações anteriores.
   Depois de analisar os resultados obtidos para os vários métodos, conclui-se que o método dos passos adaptativos foi o que apresentou melhores resultados, já que, para os mesmos valores de α, a e η, é o método que converge para o mínimo da função com o menor número de iterações.