Optimization methods for data analysis

Lab 1: Univariate Optimization

Michał Kempka

March 10, 2022

1 Refresher

Note that this section is meant as a refresher and often neglects strict formality in favor of not taking up 50 pages.

1.1 Helpful terms

convex function - segments joining **any** 2 points of the graph lie above the graph. There are several (more precise, formal and general) definitions but we leave it for later. Every **local minimum** of a convex function is also a **global minimum** which makes the optimization less difficult. Also note that most prominent convex functions are shaped like a smile which reminds us that convex functions are desirable.

concave function - a convex function that is 'flipped upside down'; not every function that is not convex is concave and there exist functions that are both convex and concave at the same time.

minimizer - argument for which the function reaches the minimum i.e. x^* such that $f(x^*) = \min_{x \in \mathcal{X}} f(x)$.

local minimum - the lowest value the function reaches in the immediate neighbourhood.

global minimum - the lowest value the function reaches in the whole domain, does not need to be unique, i.e. the same global minimum can be reached in several different points.

univariate optimization - a problem of finding an optimal (usually minimal) value of the function whose argument is a single variable, typically a real number. i.e. finding minimum of $f(x)$ where $x \in \mathcal{X} \subseteq \mathrm{R}$.

unimodal function - a function with one 'mode', i.e. one peak, one valley.

constrained optimization - optimization where domain is constrained e.g. $x \in [0, 1)$.

unconstrained optimization - optimization where domain is not constrained e.g. $x \in \mathcal{R}^n; n \in \mathcal{Z}$.

derivative - a measure of how fast the value of the function changes with infinitesimal change of the argument; $f'(x) = \lim_{\Delta \to 0} \frac{f(x+\Delta)-f(x)}{\Delta}$; in geometric terms it also gives the slope of a tangent to the graph.

stationary point - a point where derivative reaches zero.

1.2 Algorithms

uniform search - sample function at points separated by fixed interval and the choose point with the smallest value

random search - sample function randomly and choose the point with the smallest value

dichotomous search - in each step find the middle of the domain $[a, b]$: $m = \frac{a+b}{2}$ and choose $x_l = m - \delta \& x_r = m + \delta)$ for some small $\delta > 0$; evaluate the function at $x_l \& x_r$ if $x_l > x_r$ then set $a = x_l$; set $b = x_r$ otherwise. Recall that $\frac{f(m+\delta)-f(m-\delta)}{2\delta}$ is a local numerical approximation of the derivative so the algorithm could be described as 'approximate the derivative in the middle of the current interval and cut out the almost-half depending on the sign of the approximation'

golden-section search - Similar to dichotomous search but instead of dividing the interval on 2 equal halves we divide it in a specific point which is determined by golden ratio. It let's us reuse one value for each iteration at a cost of decreasing the interval slower.

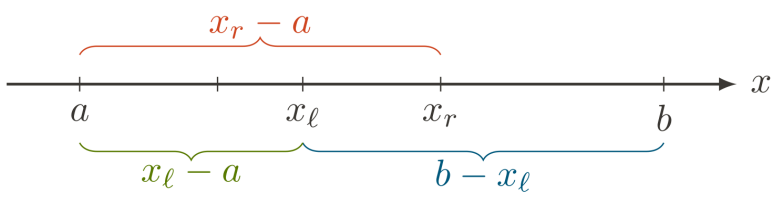
bisection - works similar to dichotomous search but instead of approximating the derivative (see dichotomous search) we evaluate the derivative itself. It makes half as many function calls but requires access to the derivative.

2 Exercises

2.1 Analytical solutions

For this section, unless stated otherwise, consider only functions that are infinitely continuously differentiable in the whole domain (which is implicitly $\mathcal{R}$).

1. What is the necessary condition for a function to reach a local minimum at x? The derivative at x must be 0 i.e. $f'(x) = 0$
2. Assuming we found x that fulfills the necessary condition? Did we find a minimum? Not necessarily. We could have found either **minimum**, **maximum** or a **inflection point**. If the first non-zero derivative (at $x : f'(x) = 0$) is of **odd** degree it's an **inflection point**. If the degree is even and the value is positive (e.g. $f''(x) > 0$) it's a minimum, otherwise it's a maximum.
3. What is the procedure for finding a solution analytically for **constrained** case? Find all stationary points and points on the boundaries of the domain and choose the one with the smallest value
4. What is the procedure for finding a solution analytically for **unconstrained** case? Like unconstrained but you must consider limits of the function for $x \to \pm\infty$. If the values are greater than the ones we already have - ignore them. If they are smaller or tend to $-\infty$ than with, minuscule loss of generality, there is no minimum.
5. Find a minimum of given functions:
   - $f(x) = x^2$ $f'(x) = 2x = 0 \implies x = 0$; and $\lim_{x \to \pm\infty} = \infty > f(0)$ so minimum is at $x = 0$
   - $f(x) = x^3$ $f'(x) = 3x^2 = 0 \implies x = 0$; however the first non-zero derivative (at point $x = 0$) is of degree 3 as $f''(x = 0) = 6 * 0 = 0 \& f'''(x = 0) = 6$ so we have an inflation point thus there is no minimum. Anyway we need to check the limits at infinities and it turns out that $\lim_{x \to -\infty} = -\infty$ so we have no minimum.
   - $f(x) = -x^2 + 2x + 5; x \in [-2, 2]$ $f'(x) = -2x + 2 = 0 \implies x = 1; f(1) = 6$; boundaries: $f(-2) = -3 \& f(2) = 5 \implies f(-2) = -3$ is the minimum
   - $f(x) = x^3 - 3x + 9; x \in [-3, 2]$ $f'(x) = 3x^2 - 3 \implies x = \pm 1; f(1) = 7; f(-1) = 11$; boundaries: $f(-3) = -9; f(2) = 11; \implies f(-3) = -9$ is the minimum
   - $f(x) = \frac{\ln^2(x)}{x}; x \in [0.4, 12]$ $f'(x) = \frac{2\ln(x)\frac{1}{x}x - \ln^2(x) * 1}{x^2} = \frac{\ln(x)(2 - \ln(x))}{x^2} = 0 \implies \ln(x) = 0 \implies x = 1$ or $\ln(x) = 2 \implies x = e^2 \approx 7.39; f(1) = 0; f(e^2) \approx 0.54$; boundaries: $f(0.4) \approx 2.10; f(12) \approx 0.51 \implies f(1) \approx 0$ is the minimum.
   - $f(x) = e^{x^2}$ $f'(x) = e^{x^2} * 2x = 0 \implies x = 0$ since the function goes to infinity for $x \to \pm\infty$ the minimum is $f(0)$
6. Derive the golden-section algorithm (find the golden-ratio for the algorithm to make only one evaluation at a step).



   (a) we are looking for ratio $r = \frac{x_r - a}{b - a}$.

   (b) since we want the ratio to be constant regardless of the evaluation we have $x_r - a = b - x_l$.

(c) since we want the ratio to be constant for each iteration we have: $\frac{x_r-a}{b-a} = \frac{x_l-a}{x_r-a} \implies x_l = b-x_r+a$

(d) plugging equation b into equation c we get: $r = \frac{x_r-a}{b-a} = \frac{b-x_r+a-a}{x_r-a} = \frac{b-a-(x_r-a)}{x_r-a} = \frac{b-a}{x_r-a} - 1 = \frac{1}{r} - 1$

(e) we can multiply both sides by r to get a quadratic equation: $r^2 + r - 1 = 0$

(f) using well known formula for the solutions of quadratic equation $x_0 = \frac{-b\pm\sqrt{b^2-4ac}}{2a} : r = \frac{-1\pm\sqrt{5}}{2}$

(g) since only $r = \frac{-1+\sqrt{5}}{2} \approx 0.618$ is positive and less than 1 we get our solution.

Optimization methods for data analysis

Lab 2: Mathematical Basis

Michał Kempka

March 17, 2022

1 Definitions

1.1 Cheatsheet/refresher

vector norm - loosely (and by default) vectors norm is defined as $\|x\| = \sqrt{\sum_{i=0}^{n-1} x_i^2}$

More broadly it's a function defined on vectors $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}_+$ that satisfies this three conditions:

1. for any two vectors $u, v \in \mathbb{R}$ $\|u+v\| \leq \|u\| + \|v\|$ (triangle inequality)
2. for any scalar a and vector v $\|av\| = |a|\|v\|$
3. $\|v\| = 0 \iff v = 0$

Lp-norm - a norm defined as $\|x\|_p = (\sum_{i=0}^{n-1} x_i^p)^{\frac{1}{p}}$ for any $p \in [1, \infty)$

dot product - known as scalar product or inner product, given vectors $a, b \in \mathbb{R}^n$ dot product is defined as

$$dot(a,b) = a^T b = \langle a, b \rangle = \sum_{i=0}^{n-1} a_i b_i$$

In particular: $dot(x,x) = \|x\|^2$

Additionally: $dot(a,b) = \|a\|\|b\|cos\theta$ where θ is the angle between two vectors.

gradient - a vector of partial derivatives of a function $f : \mathbb{R}^n \to \mathbb{R}$ denoted as $\nabla f(x)$, it is a function itself $\mathbb{R}^n \to \mathbb{R}^n$

$$\nabla f(x) = \nabla f(\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_0} \\ \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_{n-1}} \end{bmatrix}$$

hessian - a square (symmetric) matrix of second-order derivatives of $f : \mathbb{R}^n \to \mathbb{R}$ denoted as $\nabla^2 f(x)$ or $H(x)$, it is a function itself $\mathbb{R}^n \to \mathbb{R}^{n \times n}$

$$H(x) = \nabla^2 f(x) = \nabla^2 f(\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial^2 x_0} & \frac{\partial^2 f(x)}{\partial x_0 \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_0 \partial x_{n-1}} \\ \frac{\partial^2 f(x)}{\partial x_1 \partial x_0} & \frac{\partial^2 f(x)}{\partial^2 x_1} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_{n-1} \partial x_0} & \cdots & \cdots & \frac{\partial^2 f(x)}{\partial^2 x_{n-1}} \end{bmatrix}$$

Disclosure: In the lecture and on Wikipedia the second order partial derivative of the same variable is denoted as $\frac{\partial^2 f(x)}{\partial x_i^2}$. I think that this notation is strange and looks like an error, that is why I used a different one.

chain rule - given $h(x) = f(g(x)); h'(x) = f'(g(x)) * g'(x)$

1.2 Things that might have not been in the lecture but might be useful to know

Hadamard product - known more commonly as **elementwise/componentwise/coordinatewise/pointwise** product, is a more intuitive and simple multiplication scheme for vectors and matrices (unlike standard matrix product). It takes 2 matrices (or vectors) of the same shape and returns a matrix of the same shape with corresponding elements multiplied:

$$H(A, B) = A \circ B = C : A, B, C \in \mathbb{R}^{N \times M}, c_{i,j} = a_{i,j} * b_{i,j}$$

This is usually the default implementation of multiplication sign in programming languages and libraries (e.g. **numpy**; it's different in matlab), however when we speak of matrix/vector product it's by default **not** Hadamard product.

Frobenius norm - square root of a sum all matrix elements squared (L2 norm of a flattened verion of the matrix):

$$\|A\|_F = \sqrt{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} a_{ij}^2}$$

Tensor - generalization of a vector/matrix to more dimensions.

- $x \in \mathbb{R}$ - 0-order tensor (scalar)
- $x \in \mathbb{R}^N$ - first order tensor (vector)
- $x \in \mathbb{R}^{N_1 \times N_2}$ - second order tensor (matrix)
- $x \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ - third order tensor
- $x \in \mathbb{R}^{N_1 \times N_2 \times \ldots N_m}$ - mth order tensor

In many popular machine learning libraries (e.g. PyTorch, Tensorflow), a term tensor is used instead of scalar/matrix/vector by default. NumPy calls tensors **array**s.

What are some example where tensors of orders higher than 2 are more natural/convenient to use or consider?

- 2/3d rgb pictures - 2/3 spacial dimensions and color channel
- our space-time continuum - at least 3 spacial dimensions and at least 1 time dimension
- 'indexed object search' (don't quote me on that, I've just invented the term) - imagine objects that are described with n-dimensional vectors (one vector per object) and each is assigned a number (e.g. uid). A data structure that would let you find uid of any object based on its features (regardless of how many objects there are) is a tensor of order n-1. E.g. for n=5 we want to find red, shirt, xl, cheap, unisex, so we query clothes[red][shirt][xl][cheap][unisex]. Note that words were used here, but assuming they all belong to discrete categories we can treat them as aliases for natural numbers.

Note that this examples differ only semantically and the third example basically covers previous 2.

Jacobian matrix - it is another matrix of derivatives which is significantly less common in optimization than Hessian, so this mention is mostly to point to the difference between them, as they are really 2 very different matrices. Jacobian is a matrix of **first** order partial derivatives but of a **vector-valued** function. So given function $f : \mathbb{R}^n \to \mathbb{R}^m$ The Jacobian matrix is a function $J : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ which effectively

returns m gradients of length n. Jacobian of a gradient is a Hessian.

$$J(x) = \begin{bmatrix} \nabla f_0(x) & \nabla f_1(x) & \dots & \nabla f_{m-1}(x) \end{bmatrix}) = \begin{bmatrix} \frac{\partial f_0(x)}{\partial x_0} & \frac{\partial f_1(x)}{\partial x_0} & \dots & \frac{\partial f_{m-1}(x)}{\partial x_0} \\ \frac{\partial f_0(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_1} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_0(x)}{\partial x_{n-1}} & \dots & \dots & \frac{\partial f_{m-1}(x)}{\partial x_{n-1}} \end{bmatrix}$$

2 Exercises

2.1 General?

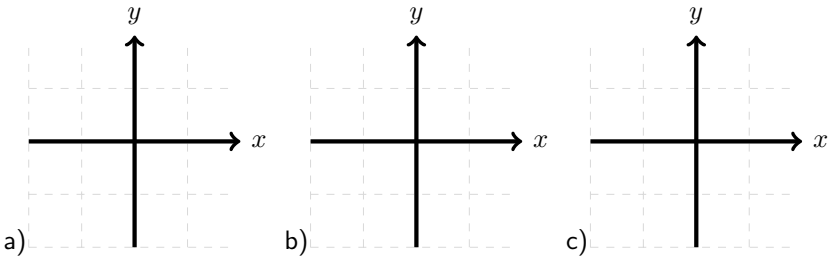
1. Assuming a norm is defined on Real numbers (not positive real) prove that the three conditions imply non-negativity nonetheless (actually first 2 suffice).
   From the second property (homogeneity) we have:

$$\| - x\| = \| - 1 * x\| = \|x\| * | - 1| = \|x\|$$

   From the triangle inequality we have:

$$\begin{aligned} \|x\| + \| - x\| &\geq \|x + (-x)\| \\ 2\|x\| &\geq \|0\| = 0 \\ \|x\| &\geq 0 \end{aligned}$$

2. Draw a unit circle for different norms: a) euclidean (L2) b) L1 (manhattan) , c) Chebyshev (L∞)



a) b) c)

3. Consider an optimization problem:

$$\min_x f(x) + \lambda R(x); x \in \mathbb{R}^n, \lambda \in \mathbb{R}$$

   How does R(x) (regularizer) influence the optimization (compared to $R(x) = const$) when:

   - $R(x) = L_2(x) = \sqrt{\sum_{i=0}^{n-1} x_i^2}$ Coordinates of the solution vector will generally be closer to 0. Effectively we can treat it is constraining the domain implicitly as there always exists sufficiently big x (for many popular algorithms with reasonable conditions) that won't be reached because regularization term 'overwhelms' the original function. We'd like this behavior, for instance, when relative (to each other) size of vector entries matter, not their absolute magnitude. Also, we obviously don't want x to blow to infinity because we have finite precision.
   - $R(x) = L_1(x) = \sum_{i=0}^{n-1} |x_i|$ L1 regularizer has similar effect regarding solution magnitude but additionally encourages sparse solutions (more zeros in solution) in most of useful models. Intuitively, L2 norm penalizes (in relative terms) coordinates that are greater more. E.g. going from 0 to 1 incurs the same penalty as going from 100 to 100.005. In terms of L1 norm the penalty is linear which results in diminishing less important constituents (setting them to 0) in favor of more contributing ones.

4. Prove Cauchy-Schwartz inequality ($\langle x, y\rangle \leq \|x\|\|y\|$)

$$\langle x, y\rangle = \|x\|\|y\|cos\theta \leq \|x\|\|y\| \overbrace{|cos\theta|}^{\leq 1} \leq \|x\|\|y\| * 1 = \|x\|\|y\|$$

2.2 Matrices

1. Why do we care about matrices and optimization at all?

2. Consider a matrix $M \in \mathbb{R}^{n \times m}$ that has exactly one 1 in each row and 0 everywhere else. What is the result of multiplying this matrix by an arbitrary matrix **A** (assuming it's legal) e.g.:

$$\overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{M} \overbrace{\begin{bmatrix} a_{0,0} & \dots & a_{0,5} \\ a_{1,0} & \dots & a_{1,5} \\ a_{2,0} & \dots & a_{2,5} \\ a_{3,0} & \dots & a_{3,5} \end{bmatrix}}^{A} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_2 \\ a_1 \\ 0 \end{bmatrix}$$

Each raw in matrix M selects a row in matrix A depending on index of is. So if $m_{ij} = 1$ then ith row of the resulting matrix will be the same as the jth row of matrix A.

3. What if there are multiple ones in each row? Then each row in matrix M chooses a sum of rows of matrix A depending on where ones are, e.g.

$$\overbrace{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{M} \overbrace{\begin{bmatrix} a_{0,0} & \dots & a_{0,5} \\ a_{1,0} & \dots & a_{1,5} \\ a_{2,0} & \dots & a_{2,5} \\ a_{3,0} & \dots & a_{3,5} \end{bmatrix}}^{A} = \begin{bmatrix} a_0 + a_1 \\ a_1 + a_2 \\ a_2 + a_3 \\ a_0 + a_1 + a_2 + a_3 \\ 0 \end{bmatrix}$$

4. What if instead of ones we had real numbers? Each row in M chooses a **linear combination** of rows in matrix A. And that's what multiplying from left is. , e.g.

$$\overbrace{\begin{bmatrix} 0.5 & 1.5 & 0 & 0 \\ 0 & 2 & 3 & 0 \\ 0 & 0 & 8 & 1.1 \end{bmatrix}}^{M} \overbrace{\begin{bmatrix} a_{0,0} & \dots & a_{0,5} \\ a_{1,0} & \dots & a_{1,5} \\ a_{2,0} & \dots & a_{2,5} \\ a_{3,0} & \dots & a_{3,5} \end{bmatrix}}^{A} = \begin{bmatrix} 0.5a_0 + 1.5a_1 \\ 2a_1 + 3a_2 \\ 8a_2 + 1.1a_3 \end{bmatrix}$$

5. What if we changed order of multiplication (and dimensions) and assume that M has only one 1 in each column? Then each column of M chooses a row from A like analogously to previous example. The same reasoning can be applied to multiple entries of arbitrary values, each column of M on the right side chooses a linear combination of rows from matrix A, e.g.

$$\overbrace{\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ a\_0 & a\_1 & a\_2 & a\_3 \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}}^{M} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a\_0 & a\_1 & a\_2 & a\_3 & a\_2 & a\_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

6. What's an inverse of A (A^{-1})? Inverse is defined only for nonsingular (by extension square) matrices. Matrix multiplied by it's inverse gives an identity matrix.

$$AA^{-1} = A^{-1}A = I$$

Side note: it's more obvious in current setting when we do not use Polish, but please remember that **reverse** and **inverse** are two different things, especially regarding built-in python function **reversed**.

7. What about matrices that are singular? There exist a generalizations so called pseudo-inverses, the most famous of which is Moore-Penrose pseudo-inverse.
8. What is an inverse of an identity matrix? It's identity matrix itself.

9. Derive a formula for an $inverse$ of a 2×2 matrix: $\mathcal{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

   We'd like to find matrix $X = A^{-1}$ so:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \overbrace{\begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix}}^{A^{-1}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

   This let's as construct a system of simple linear equations:

$$ax_{00} + bx_{10} = 1 \tag{1}$$
$$cx_{00} + dx_{10} = 0 \tag{2}$$
$$ax_{01} + bx_{11} = 0 \tag{3}$$
$$cx_{01} + dx_{11} = 1 \tag{4}$$

   By multiplying (1) by c and (2) by $-a$ and suming them them we get:

$$acx_{00} - acx_{00} + bcx_{10} - adx_{10} = c \implies x_{10} = -\frac{c}{ad - bc} \tag{5}$$

   By repeating this for (3) and (4):

$$acx_{01} - acx_{01} + bcx_{11} - adx_{11} = -a \implies x_{11} = \frac{a}{ad - bc} \tag{6}$$

   By using (2) with (5) and (3) with (6) we get:

$$cx_{00} - \frac{dc}{ad - bc} = 0 \implies x_{00} = \frac{d}{ad - bc}$$
$$ax_{01} + \frac{ab}{ad - bc} = 0 \implies x_{01} = -\frac{b}{ad - bc}$$

   This gives us

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

   Note that we have, implicitly made an assumption that $ad - bc \neq 0$. coincidentally this expression is a determinant of matrix A, so if it was 0 the matrix would be singular (not invertible)

10. Find the inverse matrix of: $\mathcal{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 \end{bmatrix} = diag([1, 2, 3, 4, 5, 6, 7])$

    From answers to previous questions you can see that multiplying by a matrix from right 'selects' and scales columns of the left matrix, thus to create I we need to select each row and multiply it by it's only element. This is achieved by a diagonal matrix: $diag([\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}])$

11. Invert the matrix using Gauss-elimination method (There will be no ready answer here):

$$\mathcal{A} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

12. Why do we need an inverse/determinant at all? Consider this equation (system of equations actually): $Ax + b = 0$ if A is invertible we can solve it as $x = -A^{-1}b$ and life is beautiful. If it's not we have a problem and it's a longer story.

2.3 Derivatives

If not stated otherwise assume:

- $x, a \in \mathbb{R}^n$
- $f : \mathbb{R}^n \to \mathbb{R}$
- $A \in R^{n \times n}$

2.3.1 Compute a gradient $\nabla f(x)$ with respect to (w.r.t.) x

1. $f(x) = \sum_{i=0}^{n-1} x_i \quad [1, 1, 1, 1, 1 \dots 1]^T$
2. $f(x) = a^T x = x^T a = \sum_{i=0}^{n-1} a_i x_i; \quad [a_0, a_1 \dots a_{n-1}]^T = a$
3. $f(x) = \sum_{i=0}^{n-1} a_i^2 x_i^2; \quad [2a_0^2 x_0, 2a_1^2 x_1 \dots 2a_{n-1}^2 x_{n-1}]^T = 2a \circ a \circ x$
4. $f(x) = x^T A x$
   After a second of applying some ideas from previous answers you can see that:

$$f(x) = \sum_{j=0}^{n-1} x_j \sum_{i=0}^{n-1} a_{ji} x_i = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} a_{ji} x_j x_i =$$

   We can now compute a single partial derivative:

$$\frac{\partial f(x)}{\partial x_k} = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} \frac{\partial (a_{ji} x_j x_i)}{\partial x_k}$$

   The derivative is nonzero only when $j = k, i = k$ or both

$$\begin{aligned} \frac{\partial f(x)}{\partial x_k} &= \frac{\partial}{\partial x_k} (\overbrace{\sum_{i=0, i \neq k}^{n} a_{ki} x_k x_i}^{j=k} + \overbrace{\sum_{j=0, j \neq k}^{n} a_{jk} x_j x_k}^{i=k} + \overbrace{a_{kk} x_k^2}^{\text{both at once}}) = \\ &= \sum_{i=0, i \neq k}^{n} a_{ki} x_i + \sum_{j=0, j \neq k}^{n} a_{jk} x_j + 2 a_{kk} x_k = \\ &= \sum_{i=0}^{n} a_{ki} x_i + \sum_{j=0}^{n} a_{jk} x_j = \sum_{i=0}^{n-1} (a_{ik} + a_{ki}) x_i \end{aligned}$$

   Notice that vector filled with $x_k = \sum_{i=0}^{n-1} a_{ki} x_i$ is a product of multiplication of A by x. Also notice that if we flip the indicies k and i we get a similar product, namely $A^T x$.
   This sums up to:

$$\nabla f(x) = (A^T + A) x$$

   Which, for **symmetric** matrices, ends up as:

$$\nabla f(x) = 2Ax$$

2.3.2 Compute a Hessian $\nabla^2 f(x)$ w.r.t x

1. $f(x) = \sum_{i=0}^{n-1} x_i \quad 0$ matrix $n \times n$
2. $f(x) = \sum_{i=0}^{n-1} x_i^2 \quad 2I$

3. $f(x) = x^T Ax$ Using previously computed partial derivative:

$$\frac{\partial f(x)}{\partial x_k} = \sum_{i=0}^{n-1}(a_{ik} + a_{ki})x_i \frac{\partial^2 f(x)}{\partial x_i, \partial x_j} = \sum_{i=0}^{n-1}(a_{ik} + a_{ki})$$

Which means that:

$$\nabla^2(f(x)) = H(x^T x) = A + A^T$$

Optimization methods for data analysis

Lab 1b: Univariate Optimization - Newton Method

Michał Kempka

March 17, 2022

1 Taylor's Expansion

Any function can $f(x)$ can be represented as an infinite series using values of derivatives of the function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i$$

It lets us compute approximations of function around some point x_0 with arbitrary precision dependant on degree of used approximation and distance between x and x_0: Approximation of function f at point x around point x_0 of degree m is given by the formula:

$$P_m(x, x_0) = \sum_{i=0}^{m} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i = f(x_0) + \frac{f^{'}(x_0)}{1!}(x - x_0) + \frac{f^{''}(x_0)}{2!}(x - x_0)^2 + \ldots \frac{f^{(m)}(x_0)}{m!}(x - x_0)^m$$

2 Newton method

Imagine that you want to implement a method that approximates a function with second degree polynomial and finds its minimum but do not know Taylor series expansion.
Given a procedural description of an iteration in Newton step (for minimization of function $f(x)$, derive the update ($x_{t+1} = g(x_t) = ?$):

1. Find the first and second derivative of function f at x_t.
2. Pretend that function f is a second order polynomial (quadratic function) and find it's equation.
3. Set x_{t+1} as the minimizer of the newly found quadratic function (approximation).

- Assume we have values of $f(x_t), f'(x_t), f''(x_t)$.
- Assume our function has form of $h(x) = ax^2 + bx + c$
- $h'(x_t) = 2ax_t + b = f'(x_t) \implies \boxed{b = f'(x_t) - 2a_x t(1)}$
- $h''(x_t) = 2a = f''(x_t) \implies \boxed{a = \frac{f''(x_t)}{2}}(2)$
- Combining (1) and (2): $\boxed{b = f'(x_t) - f''(x_t)x_t}$
- Since we want to find the minimizer of the parabola we want: $h'(x) = 0$. At this point it is apparent that c coefficient is not needed to do this because $h'(x) = 2ax + b = 0 \implies x = -\frac{b}{2a}$.
- We know **a** and **b** already so: $-\frac{b}{2a} = -\frac{f'(x_t) - f''(x_t)x_t}{2\frac{f''(x_t)}{2}} = x_t - \frac{f'(x_t)}{f''(x_t)}$
- Set $x_{t+1} = x_t - \frac{f'(x_t}{f''(x_t)}$

Optimization methods for data analysis

Lab 6: SGD

Michał Kempka

April 21, 2022

1 Gradient Descent Variations

Note that, there exist different notations and naming conventions concerning what is considered stochastic/online gradient descent but the one presented seems most popular and straightforward.

1.1 Gradient Descent

Algorithm 1 shows a pseudo code for gradient descent applied to a a problem with a dataset. In each of T iterations (usually called **epochs**) sweeps through the whole dataset, computes the gradient and applies a single update to parameters. This kind of training regim is often reffered to as **batch** mode since the update is applied after computing the whole batch.

1.2 Online Gradient Descent

Online gradient descent (Algorithm 2 presents a different regime - where gradient is applied after each row separately. This approach lets us use true online data (e.g. learning user behaviors from an endless stream of data). It, must be noted however, that these small updates do not represent true gradient and reflect it only **in expectation** which means that the algorithm will make highly erratic moves but will go in the correct direction **on average**.

1.3 Stochastic Gradient Descent

Stochastic Gradient Descent (Algorithm 3 is, essentially, a generalization of previous two cases. 'Stochastic' refers ti the fact that (like in OGD) each direction is just a stochastic estimation of the true gradient. In each step we sample so called **mini-batch** (a subset) from the data set of a given size (a hyper parameter to choose) and average the gradient over this minibatch. Choosing batch size 1 gives us OGD and choosing the biggest possible one results in the regular GD.

Algorithm 1: (Batch) Gradient Descent (GD)

Input:
Dataset x of length n
Parameters θ
Loss function $\mathcal{L}(x_i, \theta) \in \mathbb{R}$
Number of iterations $T \in \mathbb{N}$
Set of decreasing learning rates: $\{\eta_0, \eta_1 \dots \eta_{T-1}\}$
for $t \leftarrow 0$ **to** $T - 1$ **do**
    $g_t \leftarrow 0$
    for $i \leftarrow 0$ **to** $n - 1$ **do**
        $g_t \leftarrow g_t + \frac{1}{n} \nabla \mathcal{L}(x_i, \theta)$ (w.r.t. θ)
    $\theta \leftarrow \theta - \eta_t g_t$

Question: What happens when GD encounters no gradient (equal zero)?

Algorithm 2: Online (Stochastic) Gradient Descent (OGD)

Input:
Dataset x of length n
Parameters θ
Loss function $\mathcal{L}(x_i, \theta) \in \mathbb{R}$
Number of iterations $T \in \mathbb{N}$
Set of decreasing learning rates: $\{\eta_0, \eta_1 \dots \eta_{T-1}\}$
for $t \leftarrow 0$ **to** $T-1$ **do**
    shuffle dataset to change the order
    for $i \leftarrow 0$ **to** $n-1$ **do**
        $g_{ti} \leftarrow \frac{1}{n}\nabla\mathcal{L}(x_i, \theta)$ (w.r.t. θ)
        $\theta \leftarrow \theta - \eta_t g_{ti}$

Algorithm 3: Mini-batch Stochastic Gradient Descent (SGD)

Input:
Dataset x of length n
Parameters θ
Loss function $\mathcal{L}(x_i, \theta) \in \mathbb{R}$
Number of iterations $T \in \mathbb{N}$
Set of decreasing learning rates: $\{\eta_0, \eta_1 \dots \eta_{T-1}\}$
mini-batch size $b \in \mathbb{N}$
for $t \leftarrow 0$ **to** $T-1$ **do**
    shuffle dataset to change the order
    for $b_i \leftarrow 0$ **to** $\lceil \frac{n}{b} \rceil - 1$ **do**
        $g_{tb_i} \leftarrow 0$
        for $j \leftarrow bi \cdot b$ **to** $\max((bi+1) \cdot b, n) - 1$ **do**
            $g_{tb_i} \leftarrow \frac{1}{n}\nabla\mathcal{L}(x_j, \theta)$ (w.r.t. θ)
        $\theta \leftarrow \theta - \eta_t g_{tb_i}$

Question: What happens when SGD encounters no gradient?

Question: What is the influence of batch size?

Question: In all algorithms, the gradient is multiplied by $\frac{1}{n}$. Does it matter?

1.4 Gradients

Given 2d plane (Figure **??** illustrate the steps of GD/SGD/OGD assuming that the learning rate is chosen 'reasonably' (the algorithm does not diverge or gets stack jumping from two points).

2 Newton-Raphson Method

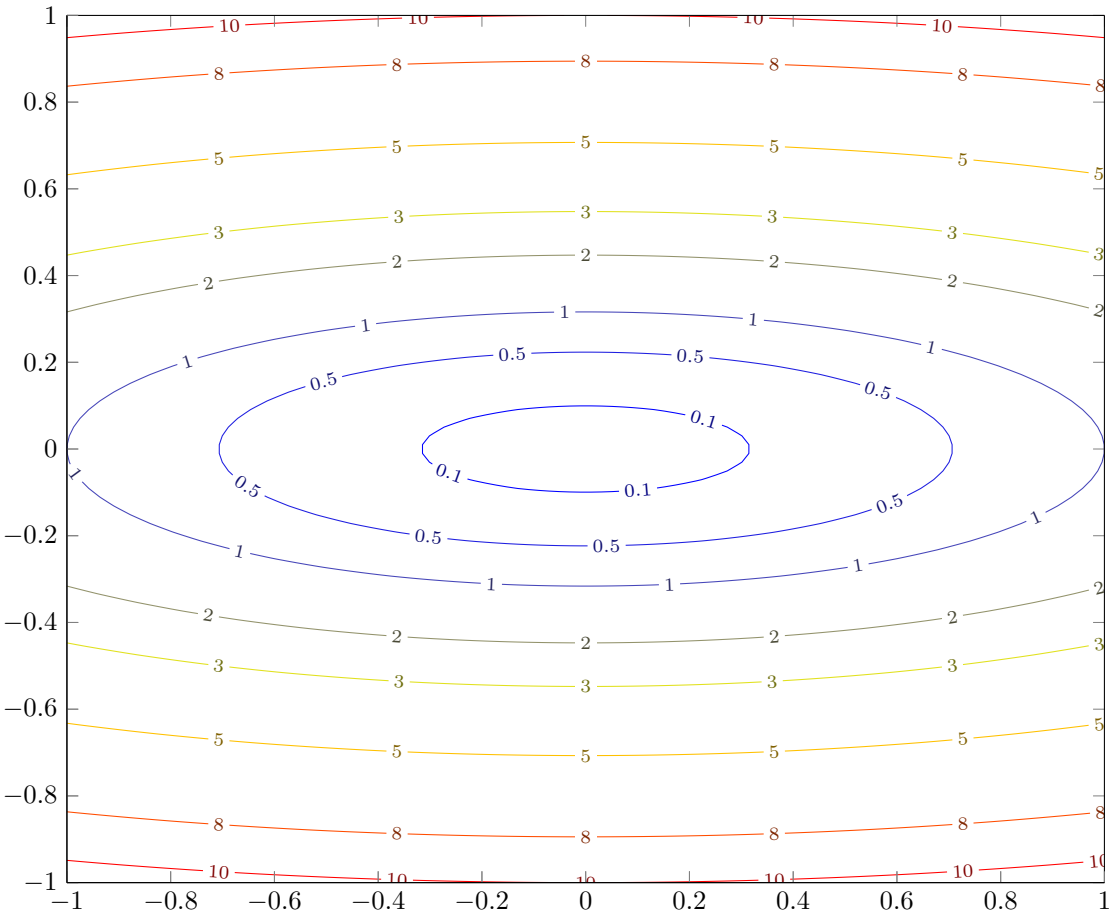
Reminder: given a function $f : \mathbb{R}^n \to \mathbb{R}$ and $x_t \in \mathbb{R}^n$, Newton-Raphson method sets the next point x_{t+1} as a minimizer of the quadratic approximation of the function from the current point:

$$x_{t+1} \leftarrow x_t - (\nabla^2 f(x_t)^{-1} \nabla f(x_t) \tag{1}$$

Question: What will happen if Newton-Raphson method is run on a quadratic function?

2.1 Exercise

Given a function: $f(x_0, x_1) = (5(x_0^2 - x_1)^2 + 2(1 - x_0)^2$ make a step of Newton-Raphson method starting with $x_0, x_1 = (2, 0)$.


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
10
8
5
3
2
1
0.5
0.1

Optimization methods for data analysis

Lab 4: Gradient Descent

Michał Kempka

March 31, 2022

1 Gradient Descent Algorithm (GD)

To find a minimizer: $\arg\min_{x \in \mathbb{R}^n} f(x); f : \mathbb{R}^n \to \mathbb{R}$, in each step we move our current point along the direction of the gradient. The magnitude of the step to make in each iteration is regulated by a real value **step-size** (or **learning rate** depending on who you ask) and is denoted as α_t or η_t. In the simplest formulation of the algorithm the learning rate is **constant** and the same for each coordinate but most (all?) popularly used extensions of GD (AdaGrad, Adam ...) use varying (adaptive) learning rate which is separate coordinate-wise.

$$x_{t+1} = x_t - \eta_t \nabla f(x_t); \eta \in \mathbb{R} \tag{1}$$

Question: What is a good choice of step-size η_t?

2 Fastest Descent Algorithm

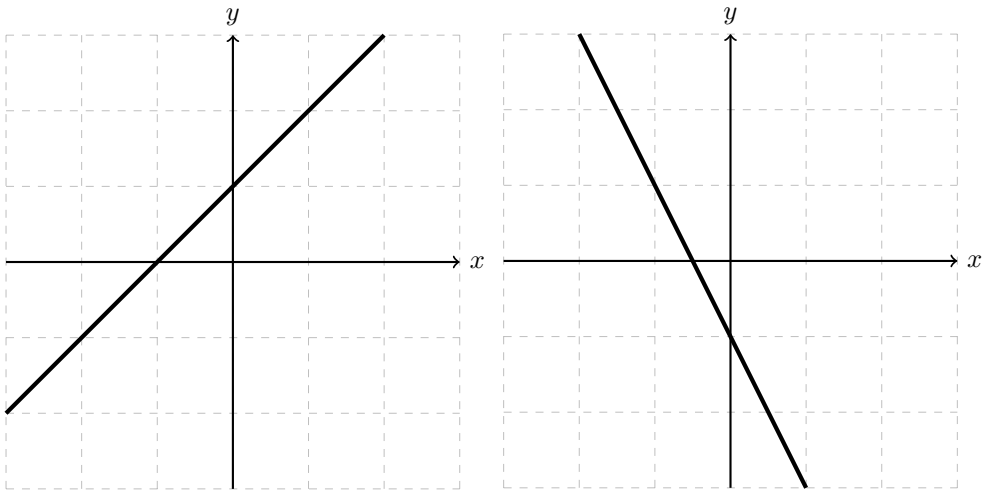
Gradient Descent modification which, in each step chooses the learning rate that minimizes the function value along the gradient.

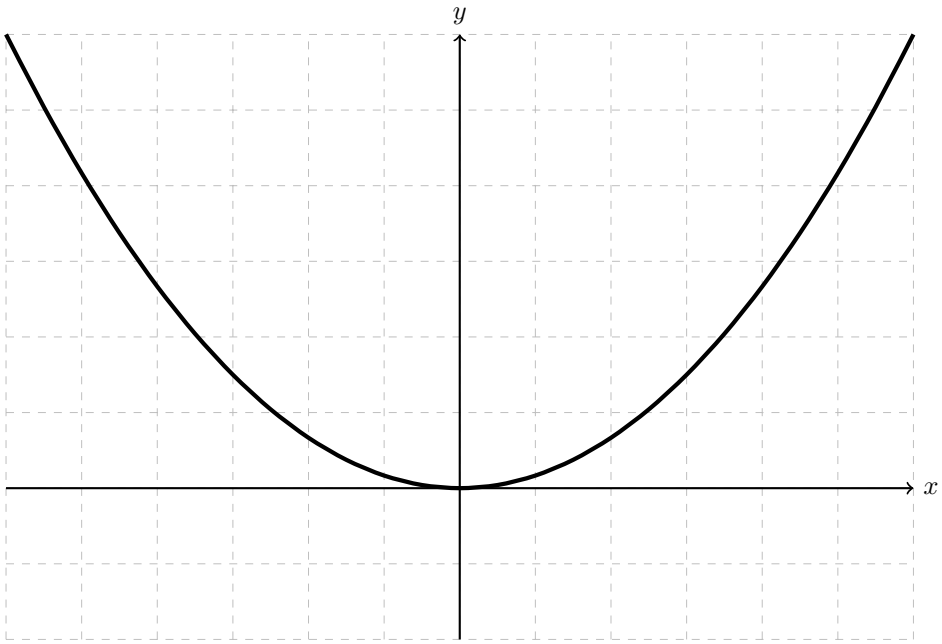
$$\eta_t = \arg\min_{\eta \in \mathbb{R}} f(x_t - \eta_t \nabla f(x_t)) \tag{2}$$

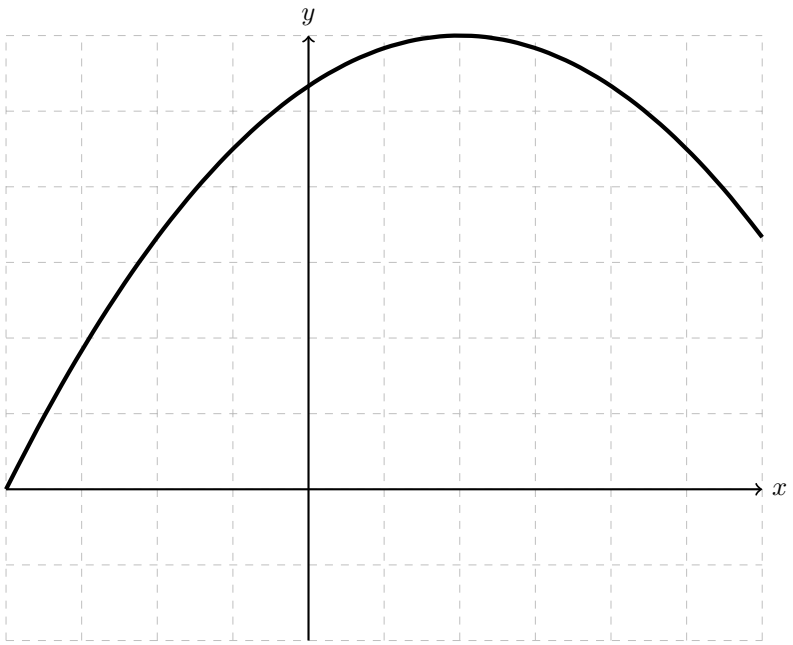
$$x_{t+1} = x_t - \eta_t \nabla f(x_t); \tag{3}$$

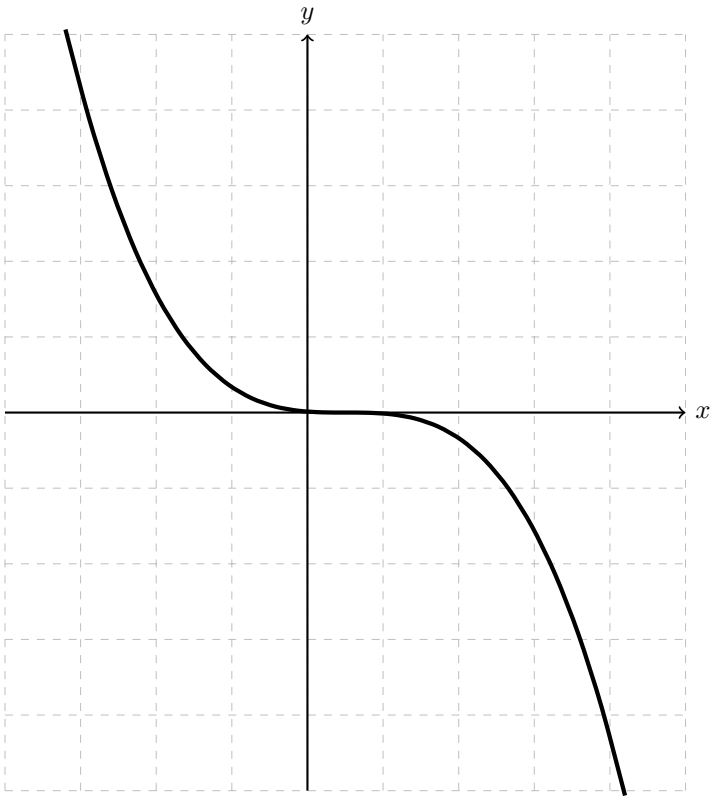
3 Exercise

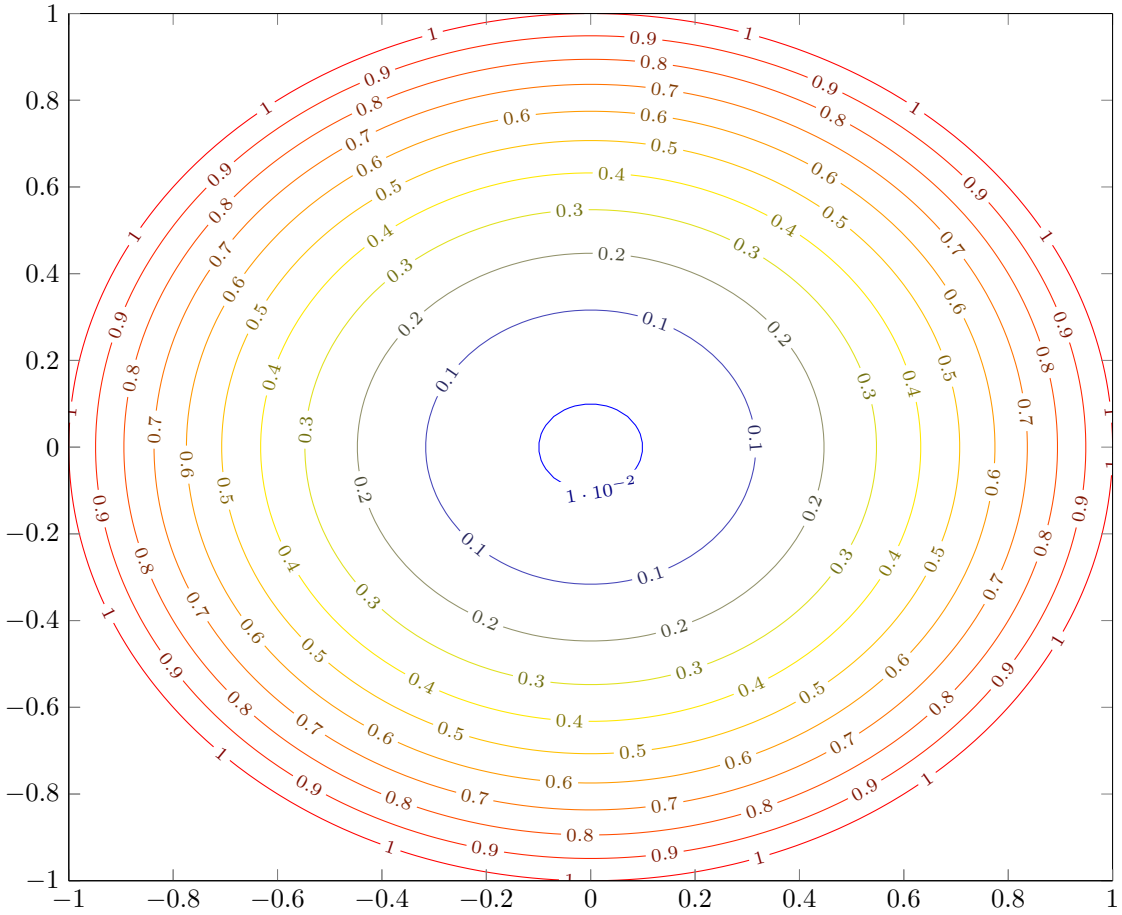
For given functions draw directions of gradients (use arrows) at different points. Neglect the absolute magnitude but try to retain relative magnitude between different points.

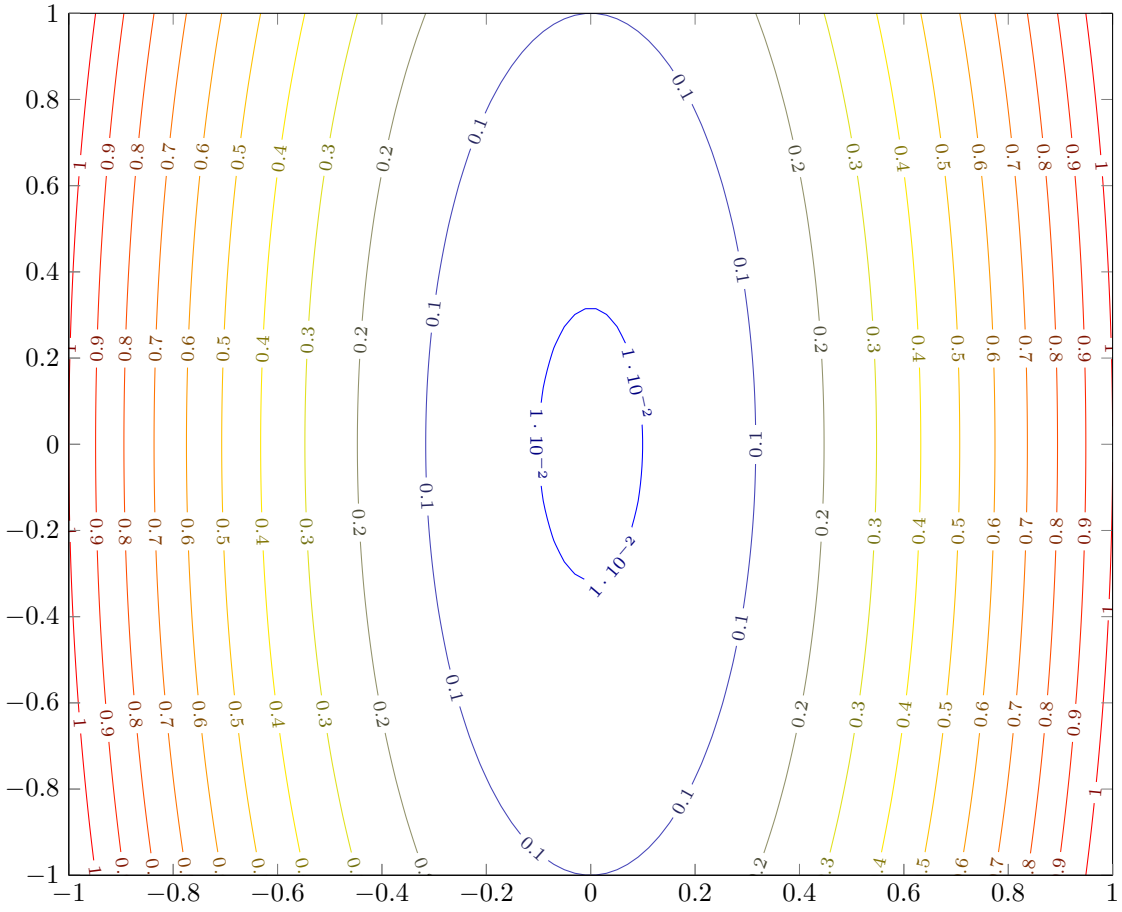



y
x


y
x


y
x


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
1 · 10−2


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
1 · 10−2
1 · 10−2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Optimization methods for data analysis

Lab 5: Regression

Michał Kempka

April 15, 2022

1 One-dimensional Linear Regression

We are given a vector $x \in \mathbb{R}^n$, where each entry represents a single data point, and a vector $y \in \mathbb{R}^n$ where each entry is a decision variable associated with corresponding x_i. We would like to find a **model** (function of x, often referred to as **hypothesis**) that best predicts y_i based on x_i ($\forall_{i \in \{0,1 \dots n-1\}} h(x_i) \approx y_i$). For instance let x be house sizes and y corresponding prices.

Question: Formulate the problem as a minimization of **mean squared error** for a **linear model** and solve it analytically.

Answer:

$$\arg\min_{a,b \in \mathbb{R}} \mathcal{L}(a,b) = \frac{1}{2n} \sum_{i=0}^{n-1} (\overbrace{x_i^T W + b}^{h(x_i)} - y_i)^2 = (ax + b - y)^T (xW + b - y) \tag{1}$$

'a' is the **slope** of the line and 'b' is usually denoted as a **bias** (term) or **intercept** (term).

The equation is just a quadratic expression so we just need to find arguments where the derivatives are 0.

For simpler notation let $\sum \equiv \sum_{i=0}^{n-1}$

$$\begin{aligned}
\frac{\partial \mathcal{L}(a^*, b^*)}{\partial b^*} &= \frac{\partial}{\partial b^*} \frac{1}{2n} \sum (a^* x_i + b^* - y_i)^2 = 0 \\
\frac{1}{n} \sum (a^* x_i + b^* - y_i) &= 0 \\
\frac{1}{n} \sum a^* x_i + \frac{1}{n} \sum b^* - \frac{1}{n} \sum y_i &= 0 \\
\frac{1}{n} a^* \sum x_i + b^* - \frac{1}{n} \sum y_i &= 0 \\
b^* &= \frac{1}{n} \sum y_i - \frac{1}{n} a^* \sum x_i \\
\frac{\partial \mathcal{L}(a^*, b^*)}{\partial a^*} &= \frac{\partial}{\partial a^*} \frac{1}{2n} \sum (a^* x_i + b^* - y_i)^2 = 0 \\
\frac{1}{n} \sum x_i (a^* x_i + b^* - y_i) &= 0 \\
a^* \sum x_i^2 + b^* \sum x_i - \sum x_i y_i &= 0 \\
a^* \sum x_i^2 + (\frac{1}{n} \sum y_i - \frac{1}{n} a^* \sum x_i) \sum x_i - \sum x_i y_i &= 0 \\
a^* (\sum x_i^2 - \frac{1}{n} \sum x_i \sum x_i) &= \sum x_i y_i - \frac{1}{n} \sum y_i \sum x_i \\
a^* &= \frac{\sum x_i y_i - \frac{1}{n} \sum y_i \sum x_i}{\sum x_i^2 - \frac{1}{n} \sum x_i \sum x_i} \\
&= \frac{\sum x_i y_i - \frac{1}{n} \sum y_i \sum x_i + \frac{1}{n} \sum y_i \sum x_i - \frac{1}{n} \sum y_i \sum x_i}{\sum x_i^2 - \bar{x} \sum x_i} \\
&= \frac{\sum x_i y_i - \bar{y} \sum x_i + n \bar{x} \bar{y} - \sum y_i \bar{x}}{\sum x_i^2 - 2\bar{x} \sum x_i + \bar{x} \sum x_i} \\
&= \frac{\sum (x_i y_i - \bar{y} x_i + \bar{x} \bar{y} - y_i \bar{x})}{\sum x_i^2 - 2\bar{x} \sum x_i + n \bar{x}^2} \\
&= \frac{\sum (x_i (y_i - \bar{y}) - \bar{x} (y_i - \bar{y}))}{\sum (x_i^2 - 2\bar{x} x_i + \bar{x}^2)} \\
&= \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \\
&= \frac{Cov(x, y)}{Var(x)}
\end{aligned}$$

Note that the last 5 lines might be sped up with the commonly known properties of variance and covariance: $Var(x) = \mathbb{E}[x^2] - \mathbb{E}[x]^2$ and $Cov(x, y) = \mathbb{E}[xy] - \mathbb{E}[x]\mathbb{E}[y]$

2 Multidimensional Linear Regression

Let's extend the regression to multiple dimensions, that is, where each x_i is a **feature vector**.

We are given a matrix $x \in \mathbb{R}^{n \times m}$, where each row is a vector of m features and represents a single data point, and a vector $y \in \mathbb{R}^n$ where each entry is some decision variable associated with corresponding row in the x matrix. We would like to find a **model** (function of x, often referred to as **hypothesis**) that best predicts y based on x ($\forall_{i \in \{0,1 \ldots n-1\}} h(x_i) \approx y_i$).

Question: Formulate the problem as a minimization of **mean squared error** for a **linear model** and solve it analytically.

Answer:

$$\underset{W \in \mathbb{R}^m; b \in \mathbb{R}}{\arg\min} \mathcal{L}(W, b) = \frac{1}{2n} \sum_{i=0}^{n-1} (\overbrace{x_i^T W + b}^{h(x_i)} - y_i)^2 = (xW + b - y)^T (xW + b - y) \tag{2}$$

'W' is usually used to mean **weights** and 'b' to mean **bias**. Alternatively, sometimes the bias term is omitted and a 'dummy' feature containing 1 is introduced to each row. For simplicity the solution will assume this approach

The equation is just a quadratic expression so we just need to find arguments where the gradient is 0.

$$\begin{aligned}
\nabla\mathcal{L}(W^*) = \nabla\frac{1}{2n}\sum_{i=0}^{n-1}(x_i^T W^* - y_i)^2 = 0 \\
\frac{1}{n}\sum_{i=0}^{n-1} x_i(x_i^T W^* - y_i) = 0 \\
\sum_{i=0}^{n-1} x_i x_i^T W^* = \sum_{i=0}^{n-1} x_i y_i \\
W^* = (\sum_{i=0}^{n-1} \overbrace{x_i x_i^T}^{\text{m×m matrix}})^{-1} \sum_{i=0}^{n-1} x_i y_i \\
= (x^T x)^{-1} \sum_{i=0}^{n-1} x_i y_i
\end{aligned}$$

Question: What happens when number of features m is greater than the number of rows n?
Answer: The leftmost matrix of the equation (the one to be inverted) is a sum of n matrices of at most rank 1 each, so it has at most rank n, thus is sure to be singular (noninvertible) if $m > n$. To circumvent this problem the a pseudoinverse might be used, which satisfies the equation and does not fail for singular matrices. Alternatively we could use an iterative gradient method to solve it.

Question: Assume we want to solve the linear regression problem with **gradient descent**. In time t we have weights W_t. What is an update rule for the iteration (use learning rate η_t).
Answer:

$$\begin{aligned}
W_{t+1} = W_t - \eta_t \nabla\frac{1}{2n}\sum_{i=0}^{n-1}(x_i^T W_t - y_i)^2 \\
= W_t - \frac{\eta_t}{n}\sum_{i=0}^{n-1}(x_i x_i^T W_t - x_i y_i)
\end{aligned}$$

Question: What if we want to have multiple optimization goals at once?
Answer: We assume that the goals are independent which turns the problem into multiple problems of the same nature. We could, however, solve it in parallel making W a matrix instead of a vector.

3 Classification

In a classification problem we are given objects with features, similarly to aforementioned regression case, and a set of **discrete classes**. Each object (vector of features) is assigned a single class out of C. For simplicity we will consider a binary classification where $C = 2$ so $y \in \{0, 1\}^n$ (**labels**). Analogously to regression we'd like to create a model that best predicts class based on features. For binary classification it's choosing 0 or 1 - or maybe something in between? Perhaps we could modify linear regression to solve binary classification?

3.1 Logistic Function

The first problem we encounter is the fact that linear regressions model $h(x_i)$ gives an arbitrary real output and $y_i \in 0, 1$ is given - so we want to find a function, that is preferable **continuously differentiable** that

squeezes our output. The **logistic function** (a.k.a. **sigmoid function**) satisfies our requirements:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

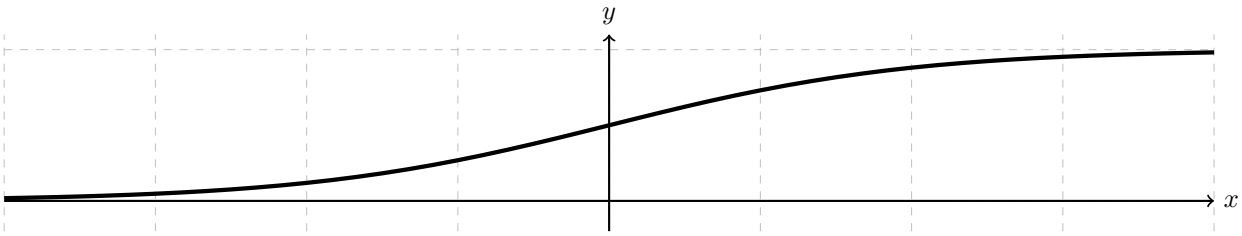


Figure 1: Logistic function

Note, that the sigmoid defines a valid **probability** distribution of a binary variable. Since $\sigma(0) = 0.5$, to obtain 0 or 1 it's sufficient to round it to 1 if $x > 0$ and to 0 otherwise.

The model then becomes:

$$h(x_i) = \sigma(x_i^T W + b) = \frac{1}{1 + e^{-(x_i^T W + b)}} \tag{4}$$

Answer: And our minimization problem becomes:

$$\arg\min_{W \in \mathbb{R}^m} \mathcal{L}(W) = \frac{1}{2n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-(x_i^T W)}} - y_i \right)^2 \tag{5}$$

Question: Is mean squared error of logistic function convex (w.r.t. W)?

Answer: A function is convex if it's Hessian is positive semi-definite. Since our expression is a sum it is sufficient to show that any (by extension all) element of the sum is convex (let's denote i'th component of the sum as $\mathcal{L}_i(W)$).

First, let's compute the derivative of the sigmoid:

$$\begin{aligned}
\frac{\partial \sigma(x)}{\partial x} &= \frac{\partial}{\partial x} \frac{1}{1 + e^{-(x)}} \\
&= -\frac{1}{(1 + e^{-(x)})^2} \cdot (-e^{-x}) \\
&= \sigma(x) \frac{e^{-x}}{(1 + e^{-(x)})} \\
&= \sigma(x) \frac{(1 + e^{-x}) - 1}{(1 + e^{-(x)})} \\
&= \sigma(x)(1 - \frac{-1}{(1 + e^{-(x)})}) \\
&= \sigma(x)(1 - \sigma(x))
\end{aligned}$$

We have 2 possible situations since y_i is either 0 or one. Let's compute partial derivative w.r.t. W_j for $y_i = 0$, For shorter notation let's denote $\sigma_i = \sigma(x_i^T W)$

$$\begin{aligned}
\frac{\partial \mathcal{L}_i(W | y_i = 0)}{\partial W_j} &= \frac{\partial}{\partial W_j} \frac{1}{2} \sigma_i^2 \\
&= \sigma_i \frac{\partial \sigma(x_i^T W)}{\partial W} \\
&= \sigma_i^2 (1 - \sigma_i) x_{ij}
\end{aligned}$$

Now let's compute the second partial derivative w.r.t W_j and W_k:

$$\begin{aligned}\frac{\partial^2 \mathcal{L}_i(W|y_i=0)}{\partial W_j \partial W_k} &= \frac{\partial}{\partial W_k}\left(\frac{\partial \mathcal{L}_i(W|y_i=0)}{\partial W_j}\right) \\ &= \frac{\partial}{\partial W_k}\left(\sigma_i^2(1-\sigma_i)x_{ij}\right) \\ &= \left(2\sigma_i^2(1-\sigma_i)x_{ik} - 3\sigma_i^3(1-\sigma_i)x_{ik}\right)x_{ij} \\ &= x_{ij}x_{ik}\sigma_i^2(1-\sigma_i)(2-3\sigma_i)\end{aligned}$$

Let's consider diagonal entries $i = k$. Then $x_{ij}x_{ik}\sigma_i^2(1-\sigma_i)$ is always positive and $(2-3\sigma_i)$ might be negative. This implies that the hessian diagonal entries might be negative which means it cannot be positive semi-definite so the function is **not convex**. □

3.2 Logloss

As derived before, squared loss of a sigmoid function is not convex which is quite unfortunate. However, we can do better by using **logloss** which is defined as follows:

$$logloss(W) = -\frac{1}{n}\sum_{i=0}^{n-1} y_i \log(\sigma(x_i^T W)) + (1-y_i)\log(1-\sigma(x_i^T W)) \tag{6}$$

The logloss is named so, because it uses the logarithm and is used as a **loss function** (the function to minimize). It contains two elements, but for $y_i \in \{0,1\}$ only one of them is non-zero. Effectively the formula amounts to taking a negative logarithm of the **probability** of the correct class. If probability is high we are mildly penalized(if at all). If probability is low, the penalty can be very heavy (including infinity).
Question: Is logloss of logistic function a convex function?
Answer: Sum of convex functions and a convex function of a linear functions are convex so it's sufficient to check if $-\log(\sigma(x))$ is convex

$$\begin{aligned}\frac{\partial^2}{\partial x}\big(-\log(\sigma(x))\big) &= \frac{\partial}{\partial x}\big(-\frac{1}{\sigma(x)}\cdot\sigma(x)(1-\sigma(x))\big) = \\ &= \frac{\partial}{\partial x}\big((\sigma(x)-1)\big) = \\ &= \sigma(x)(1-\sigma(x)) \geq 0 \implies \text{it's convex}\end{aligned}$$

3.3 Logistic Regression

Using logistic function and logloss in tandem is called **logistic regression** and is a standard starting point for many modern machine learning solutions. However, solving logistic regression analytically (closed form) is not as simple as for linear regression. Thus emerges a need to use alternative methods (e.g. gradient descent).

Question: Assume we want to solve the logistic regression with **gradient descent**. In time t we have weights W_t. What is an update rule for the iteration (use learning rate η_t).

Answer:

$$
\begin{aligned}
W_{t+1} &= W_t - \eta_t \nabla logloss(W_t) \\
&= W_t - \eta_t \nabla \Big(-\frac{1}{n} \sum_{i=0}^{n-1} y_i \log(\sigma(x_i^T W_t)) + (1-y_i) \log(1-\sigma(x_i^T W_t)) \Big) \\
&= W_t - \eta_t \Big(-\frac{1}{n} \sum_{i=0}^{n-1} y_i (1-\sigma(x_i^T W_t)) x_i - (1-y_i)\sigma(x_i^T W_t) x_i \Big) \\
&= W_t - \eta_t \Big(-\frac{1}{n} \sum_{i=0}^{n-1} (y_i - \sigma(x_i^T W_t)) x_i \Big) \\
&= W_t - \eta_t \Big(\frac{1}{n} \sum_{i=0}^{n-1} (\sigma(x_i^T W_t) - y_i) x_i \Big)
\end{aligned}
$$

□

3.4 Linearity

Logistic function is not linear, however the model it squeezes is linear. Effectively, logistic regression 'draws a straight line' which divides 'positive' and 'negative' data points (it's a hyper plane in higher dimensions). The consequence of it this is lack of ability of logistic regression to classify data that is not **linearly separable**.

3.5 Presentation

To see, and play with linear/logistic regression and gradient descent got to: https://playground.tensorflow.org/. Admittedly it's a 'neural networks' playground but, as you'll probably soon find out, neural networks for classification are logistic regression on steroid (sometimes pretty heavy ones but still).

3.6 Softmax

In general we'd like to have more classes than 2. In this case, we can have a separate linear model for each class and choose the one that outputs the greatest value (argmax). However, max and argmax are not continuous which is not very convenient for optimization what is why a softer alternative was proposed, aptly named **softmax**. Given C classes and C linear models ($logits = xW + b; x \in \mathbb{R}^{n \times m}; W \in \mathbb{R}^{m \times C}; b \in \mathbb{R}^{C}$) **softmax**:

$$
softmax_i(logits) = \frac{e^{logits_i}}{\sum_{j=\{0,1 \ldots C-1\}} e^{logits_j}} \tag{7}
$$

In this case we can define logloss as:

$$
multinomial\_logloss = -\sum_{i=0}^{n-1} \sum_{j \in \{0,1 \ldots C-1\}} \mathbb{1}_{y_i=j} \log(softmax_j) \tag{8}
$$

Question: Show that softmax for C=2 is equivalent to logistic function.
Answer:

$$
\begin{aligned}
softmax_0(x) &= \frac{e^{x_0}}{e^{x_0} + e^{x_1}} \\
&= \frac{1}{1 + e^{x_1} e^{-x_0}} \\
&= \frac{1}{1 + e^{x_1 - x_0}} = \sigma(x_0 - x_1)
\end{aligned}
$$

Since x_0 and x_1 are both linear models, their difference is one as well. Effectively we obtain the same model but with 2 times more parameters.

3.7 Cross-entropy

We can consider that the labels (y) are drawn from some distribution $p(y = C_i|x)$. Logistic regression already outputs values that can be interpreted as a probability distribution - let's say q. We'd like our model (the probability it entails) to **diverge** as little as possible from the target distribution p. That is how we arrive at **cross-entropy** loss:

$$ce(p, q, x) = -\sum_{i=0}^{n-1} \sum_{j \in \{0,1...C-1\}} p(y_i = c_j|x_i) \log(q(h_i = c_j|x_i)) \tag{9}$$

The formula is very similar to logloss shown in equation 8 and it's not a coincidence because they are essentially the same. However, most of libraries and references in deep learning will use 'cross entropy' instead of 'logloss' e.g. tensorflow or pytorch