Optimization for Big Data:

A friendly approach for elementary algorithms

Vlad-Stefan Tudor
coordinator: Ion Necoara

2020

***Abstract*—Efficiently processing large quantities of data is more relevant than ever before. Acquisition is facilitated by the increasing number of platforms that gather, store and sort information and processing power unveils new possibilities for algorithms that can lead to novel results. In this context, old methods of practice have to be adapted and new methods have to be all-together developed.**

I. Introduction

The problem of optimization essentially translates to a question of minimization. In general, everything that humans do, and nature, as well, tends to the least effort, that is least energy consumed. Data optimization may lead to a minimization of cost, be that cost any variable that is considered undesirable for a system. The classical optimal solution is the shortest path, but what does that translate to when the problem is n-dimensional, that is, we try to take into account a very large number of criteria we want to minimize?

Moreover, the cost may refer to less tangible concepts. The cost is also the difference between an estimate and a measured output. It's a measure for how far away we are from correctly predicting a behaviour. It's understandable that it's something that needs to be minimized but precision for the model requires a cost of multiple dimensions, one for each independent behaviour-variable.

This sort of rationale is strongly used in machine learning. For systems far too complex to be estimated through analytical calculation, a net of interconnected points, referred to as neurons, are generated with a certain level of randomness. The inputs are transmitted and propagated through this network to the outputs. Since the real behavior is priory known, the outputs can be compared to empirical data and a value of their deviation from the desired result can be calculated. The question is now to estimate a minimum for this total deviation and so finding the internal layout that will best approximate the real-life system and all this can be achieved through algorithms described in this article.

II. Defining the problem

In order to build a degree of intuition, let us visualize an example of a cost function. The plot shows 3 axis: 2 of them represent the variables in the function, composing the horizontal plain, both if them with a range of $[-3; 3]$. The vertical axis shows the output of the function for each possible pair of input values. This is the what we are interested in: We have to find the pair of inputs (x_1, x_2) that yield the minimal value when computed through the function.

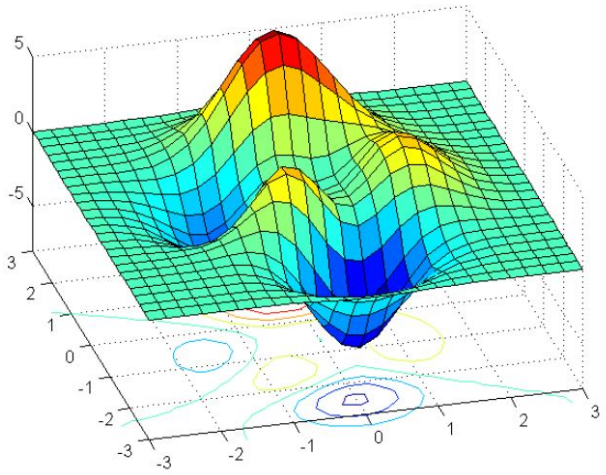

Fig. 1: Example of a cost function [2]

Looking at a graphical representation as such, it seems obvious where to find this point. In our case it's at $(0, -2.5)$. But it's redundantly complicated to find the minimum by plotting all possible values and then selecting the one we see to be the lowest. This is an oversimplified problem. In reality there will be $(x_1, ..., x_n)$ variables and $(y_1, ..., y_m)$ different outputs.

III. The solution, in broad terms

The algorithms starts from an arbitrary point, let it be random or chosen. At each current point the cost-function can be evaluated. It's important to note that the value of the function for the respective point is not the only information we can derive and use, but more in that follows. So the algorithm has now to decide which is the following point that needs to be evaluated, that is the next step that hopefully brings it closer to the point of minimum cost. Intuitively, imagine a man sitting on top of a mountain trying to get down in the valley. It's evident how he will determine the correct direction: he will look at the *slope* of his surroundings. Following a descending slope will naturally get him to the point of lowest elevation. He will consider to have arrived once he notices that the slope is approximately equal to zero.

IV. Essential notions of mathematics

A. Derivatives

The first-order derivative is the mathematical equivalent for the *slope*, mentioned earlier. It is a measure of the rate of change, meaning that negative values of the derivative indicate that the function is decreasing and vice-versa. A zero value shows that in a close vicinity the function neither grows nor decreases, that is a sign that we have found a minimum or a maximum.

Similarly, **The second-order derivative** is a measure for the change in the first-order derivative. It tells us whether the first order derivative tends to rise or decrease. For the original function, this will be a measure of it's shape. A positive second-order derivative indicates a concave function whereas a negative second-order derivative indicates a convex one.

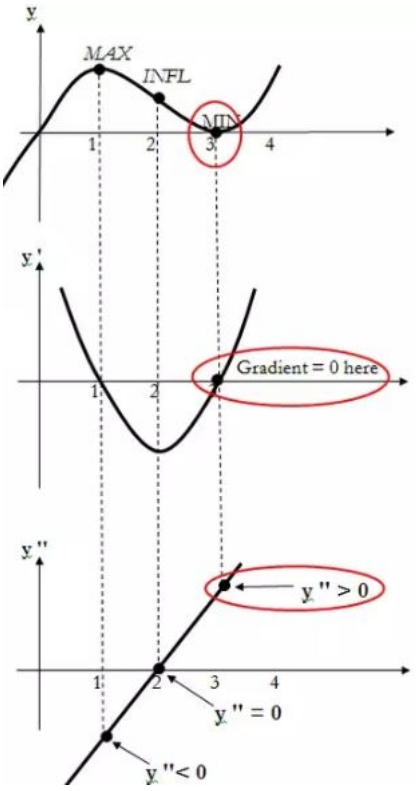

Fig. 2: Visualization of first and second derivatives for $x(x-3)^2$ [5]

B. Matrix Algebra

Generally we describe our model with arrays and matrices of data. The problem is to estimate a set of parameters on the basis of a number of empirical observations and measurements.

- $x \in R_D$ is the set of D different parameters of our model. They are **not known**.
- $b \in R_n$ are the measured outputs of n distinct experiments.
- $A \in R_{m \times D}$ represents the observations. More specifically, for each one of the n experiments (the lines of A), a set of D values (the columns of A) is provided that will be individually multiplied with each of the D components of x. A line a_i of A is thus the set of inputs for the experiment i.

We can write the minimization problem as follows:

$$\min_{x \in R_D} \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

$$f_i(x) = a_i x - b$$

This translates to finding the set of parameters x that describe the system for which the cost function is minimum, where the cost function is defined to be the difference between the output of the model ($a_i x$) and the actual result (b_i).

However, if we thoroughly think of the meaning, minimizing f will give a best-fit approximation, based on the data *we have*. In reality, this estimation may not fit as well other data. An idea would be to somehow penalize a too-close data-fitting, a trade-off for better fitting future data. An example of this necessity is shown below. If we were to have only 2 available measurements (shown in red), a estimation of a system that would best fit the data would be a line crossing the 2 points. However we would better choose a 'not-so-great' estimation of the available data in order to have a better chance of fitting any future data (shown in green).

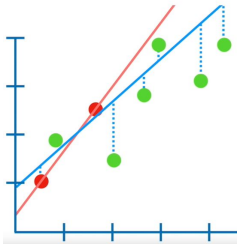

Fig. 3: Fitting the available data vs. future data [1]

We can achieve this if we add another element in the minimization:

$$\min_{x \in R_D} \frac{1}{n} \sum_{i=1}^{n} f_i(x) + \lambda r(x)$$

C. The Lasso Problem

Adding up all this, we can formulate the following problem:

$$\min_{x \in R_D} ||Ax - b||_2^2 + \lambda |x|_1$$

$$||x||_1 = \sum_{i=1}^{D} |x_i|$$

$$||y||_2 = \left(\sum_{i=1}^{D} x_i^2 \right)^{\frac{1}{2}}$$

V. Gradient Method

The gradient method provides the most basic solution for optimization $\min_{x \in R_D} f(x)$. It returns a step for a future point by examining all the directions, calculating a step proportional to their slopes. Naturally, this will result in larger steps for areas of great slopes and smaller steps for area where the derivatives are close to zero. This means that the algorithm 'jumps' smaller steps the more close it gets to the solution. Therefore the next step $(t+1)$ of the gradient method is

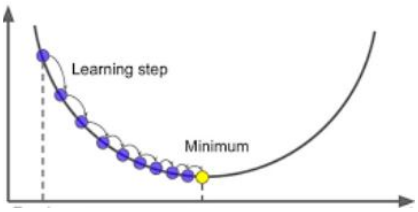

Fig. 4: Gradient method for 1 dimension [3]

calculated as:

$$x^{t+1} = x^t - \alpha_t \nabla f(x^t)$$

$\nabla f(x)$ represents the gradient, meaning the derivatives of all dimensions in the point x. For:

$$f = 0.5||Ax - b||_2^2$$

$$f = 0.5(Ax-b)^T(Ax-b) = 0.5(x^T A^T - b^T)(Ax-b)$$

$$\nabla(x^T A^T Ax) = sA^T Ax$$

$$\nabla(b^T x) = \nabla(x^T b) = b$$

Therefore the gradient of f in x is:

$$\nabla f = A^T(Ax - b)$$

VI. Proximal Gradient

The proximal gradient method provides a solution for optimization $\min_{x \in R_D} f(x) + g(x)$ (f convex, L-smooth, g only convex). The method is used for optimization involving multiple criteria. The steps are calculated, as before, based on the gradient, but now we have to apply the $proximal$ operator in order to find the projection of x on the set of g.

$$prox_{\alpha f}(v) = arg\min_x (f(x) + 0.5\alpha||x - v||_2^2)$$

For $f = ||x||_1$, the proximal operator is called the soft thresholding.

$$prox_{\alpha||x||}(v) = \begin{cases} v_i - \alpha, & v_i > \alpha \\ 0, & v_i = \alpha \\ v_i + \alpha, & v_i < \alpha \end{cases}$$

This makes it possible to apply the gradient to the Lasso problem:

$$\min_{x \in R_D} ||Ax - b||_2^2 + \lambda|x|_1$$

Define the step $(t+1)$ for the algorithm:

$$x^{t+1} = prox_{\alpha^t \lambda||\cdot||_1}(x^t - \alpha^t A^T(Ax^t - b))$$

$$\alpha^t \in (0, \frac{1}{\sigma_{max}(A)}$$

VII. Newton's method

Newton's Method is a highly visual solution. A each step, it evaluates the first and second order derivatives of the function in the current point. The leap will be given by d_t, being the solution for:

$$\nabla^2 f(x_t) d_t = \nabla f(x_t)$$

What this means is that at point x_t we will look for the tangent hyper-plane (first order derivative ∇f) We will then see what quadratic shape has the same tangent in the point x_t as our original function f. The minimum of this quadratic, which can be analytically determined will act as our next step x_{t+1}.

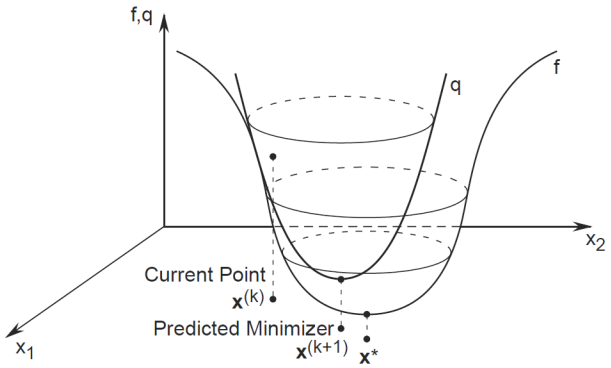

Fig. 5: Newton method for 2 dimensions [4]

Therefore, the Newton iteration is given by:

$$x^{t+1} = x^t - \alpha_t(\nabla^2 f(x^t))^{-1} \nabla f(x^t)$$

For the Lasso problem $\min_{x \in R_D} ||Ax-b||_2^2 + \lambda|x|_1$ note that:

$$\nabla f(x^t) = A^T(Ax^t - b)$$

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

References

[1] Best-fit of current data vs. future data. https://www.youtube.com/watch?v=Q81RR3yKn30. Last accesed: 04/05/2020.

[2] Example of a cost function. https://www.researchgate.net/figure/The-peaks-function-2-has-several-minima$_fig1_253164210$. $Lastaccesed$: $04/05/2020$.

[3] Gradient descent for 1 dimension. https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/. Last accesed: 04/05/2020.

[4] Newton's Method. https://suzyahyah.github.io/calculus/optimization/2018/04/06/Taylor-Series-Newtons-Method.html. Last accesed: 04/05/2020.

[5] Visualization of first and second derivatives. https://www.quora.com/Why-is-a-second-order-derivative-negative-for-a-point-of-minimum. Last accesed: 04/05/2020.