Optimization Techniques with Applications in Machine Learning

Konstantina Maria Argyropoulou

Department of Mathematics and Applied Mathematics
University of Crete

E-mail: temp62@math.uoc.gr

Introduction

Due to the widespread (and increasing) use of optimization algorithms in science, engineering, economics, and industry this subject has found renewed interest. This paper aims to present a description of the most powerful techniques for solving optimization problems. In practice, optimization does not depend only on efficient ones and powerful algorithms, but also from good modeling techniques and careful interpretation of results. In this paper we also try to highlight the aspects, namely, modeling, optimality conditions, the implementation of algorithms and the interpretation of results.

The method of steepest descent

The steep descent method is an iterative first-order optimization algorithm used to find a local minimum of a differentiable function. Given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, *the direction of steepest descent* is the vector $-\nabla f(x_0)$, where x_0 is the starting point. To see this, consider the function $\varphi(t) = f(x_0 + tu)$ where $\mathbf{u}$ is a *unit* vector in $\mathbb{R}^n$. Then, by the Chain Rule we have

$$\varphi'(0) = \nabla f(x_0) \cdot u = ||\nabla f(x_0)|| \cos(\theta),$$

where θ is the angle between $\nabla f(x_0)$ and u. It follows that $\phi'(0)$ is minimized when $\theta = \pi$, which yields

$$u = -\frac{\nabla f(x_0)}{\|\nabla f(x_0)\|}, \qquad \varphi'(0) = -\|\nabla f(x_0)\|.$$

We can therefore reduce the problem of minimizing a function of **several** variables to a *singlevariable minimization problem*, by finding the minimum of $\phi(t)$ for this choice of u. That is, we find the value of t, for $t > 0$, that minimizes

$$\varphi_0(t) = f(x_0 - t\nabla f(x_0)).$$

After finding the minimizer t_0, we can set $x_1 = x_0 - t_0\nabla f(x_0)$, and continue the process, by searching from x_1 in the direction of $-\nabla f(x_1)$ to obtain x_2 by minimizing $\phi_1(t) = f(x_1 - t\nabla f(x_1)$, and so on. Thus, the steepest descent method, starting at x_0, computes a sequence of iterations (x_k), where for $k \geq 0$, $x_{k+1} = x_k - t_k\nabla f(x_k)$ and $t_k > 0$ minimizes the function

$$\varphi_k(t) = f(x_k - t_k\nabla f(x_k)). \tag{1}$$

Line search with backtracking

It is often only possible, and computationally more efficient, to use *approximations* of the minimum of t_k of the (1) function than to compute exactly at every iteration, especially when the computations of the function we are minimizing and its first derivative are computationally expensive. Thus, various methods have been created that use at each iteration an *approximation* of the minimum t_k. One of these methods is the method of *liine search with backtracking* . Given a function $f : \mathbb{R}^n \to \mathbb{R}$ which is differentiable, we know that the steep descent direction in the k step of the algorithm, is the $p_k = -\nabla f(x_k)$, where x_k is the k-in approximation of an iterative minimization process, such as the steep descent method. We will say that step t_k in the relation (1) is *accepted* if the following, so-called *Wolfe conditions* are satisfied:

$$\text{for } \alpha \in (0,1) \qquad f(x_k + t \cdot p_k) \leq f(x_k) + \alpha t\, \nabla f(x_k)^T \cdot p_k, \tag{2}$$

$$\text{for } \beta \in (\alpha,1) \qquad \beta \cdot \nabla f(x_k)^T \leq \nabla f(x_k + tp_k)^T \cdot p_k. \tag{3}$$

We initially choose $t = 1$ and then, as long as the relation (2) is not satisfied, we backtrack, that is, we reduce the step t by a constant $\rho \in (0,1)$. When (2) is satisfied, we calculate the next term of the sequence of approximations as $x_{k+1} = x_k + t_k p_k$. In the first backtracking we construct an approximation of φ using a polynomial $q \in \mathbb{P}_2$ such that

$$q(0) = \varphi(0), \quad q(1) = \varphi(1), \quad q'(0) = \varphi'(0).$$

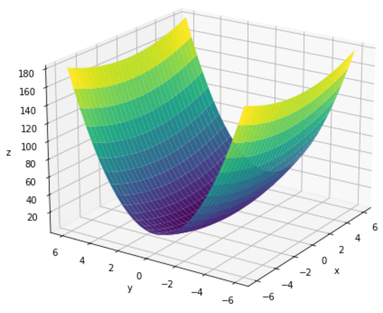
If $f(x_k + p_k)$ is not admissible (ie, the first Wolfe condition is not satisfied), then $\varphi(1) > \varphi(0) + \alpha\varphi'(0)$. We easily see that

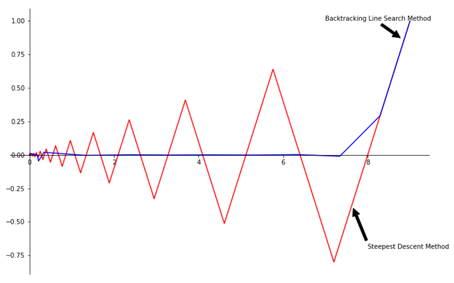
$$q(t) = [\varphi(1) - \varphi(0) - \varphi'(0)]t^2 + \varphi'(0) \cdot t + \varphi(0),$$

and the previous relation shows that the polynomial q is indeed quadratic. Setting the first derivative to zero, we arrive at the choice of t

$$t^* = \frac{\varphi'(0)}{2[\varphi(1) - \varphi(0) - \varphi'(0)]}. \tag{4}$$

The fact that $q''(t^*) = \varphi(1) - \varphi(0) - \varphi'(0) > 0$ shows that q does have a minimum at t^*.



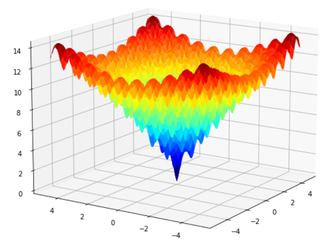


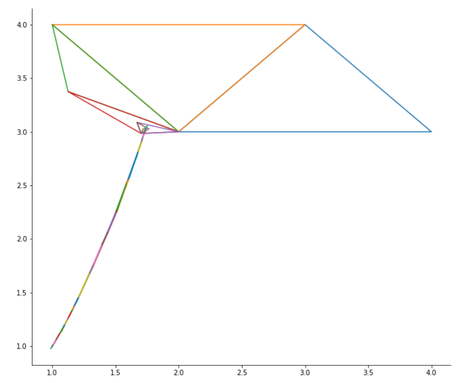
The Nelder-Mead method

The Nelder–Mead method is a direct search method often applied to nonlinear optimization problems for which derivatives of the objective funtion may not be known or are expensive to compute. The idea of the method is to calculate the objective function at the vertices of a simplex and replace the vertex with the largest value by another point, usually the symmetric to the side with vertices the points where the function takes the smallest values.
Of particular interest to the Nelder-Mead method are functions with many local minima, as there is a possibility that the method will get trapped in any of them. One such example is the Ackley function

$$f(x,y) = -20e^{0.2\sqrt{\frac{1}{2}(x^2+y^2)}} - e^{\frac{1}{2}(\cos(2\pi x)+(\cos 2\pi y))} + e + 20,$$

which has a global minimum at $(0,0)$ and many local The Nelder Mead method transforms an initial simplex with vertices the points $(2,3)$, $(4,3)$ and $(3,3)$ in a simplex in which the Ackley function has a minimum at the vertex with coordinates $x = (-0.96851623, -0.96850533)$.





Neural networks

Suppose we have a network with L layers and n_l is the number of neurons in layer l, $l = 1, \ldots, N$. Thus the network maps from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$. We can succintly summarize the action of the network by letting $a_j^{[l]}$ denote the output fron neuron j at level l, so that

$$a^{[1]} = x \in \mathbb{R}^{n_1},$$
$$a^{[l]} = \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l} \quad \text{for } l = 2, 3, \ldots, L.$$

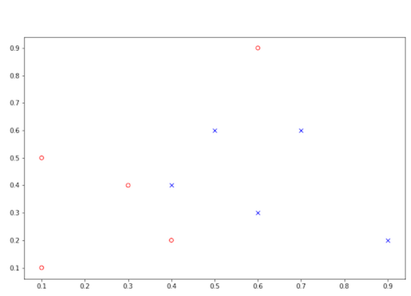
Suppose we have N training points in $\mathbb{R}^{n_1}$, $\{x^{(i)}\}_{i=1}^N$ and given target outputs $\{y(x^{(i)})\}_{i=1}^N$. We wish to minimize a const function depending on all of the weights and biases, such as the quadratic cost function

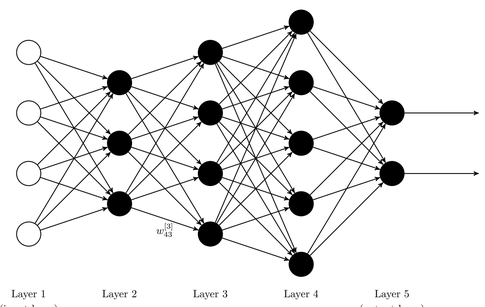
$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left\|y(x^{(i)}) - a^{[l]}(x^{(i)})\right\|_2^2. \tag{5}$$

If p is the current estimate of the minimum of the objective function (5) we use the steepest descent method to produce an improvement $p \leftarrow p - \eta\,\nabla\text{Cost}(p)$, with η a small stepsize, known as the *learning rate*. When a have a large number of parameters and a large number of training points the computation of the gradient at every iteration of the steepest descent method may be prohibitively expensive. A cheaper alternative may be the *stochastic gradient method*. N steps of this method, the so-called *epoch*, may be:

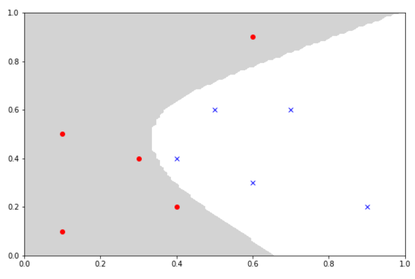
1. Shuffle the integers $\{1, 2, \ldots, N\}$ into a new order $\{k_1, k_2, \ldots, k_N\}$
2. for $i = 1, \ldots, N$, update $p \leftarrow p - \eta\,\nabla\text{Cost}_{x^{(k_i)}}(p)$

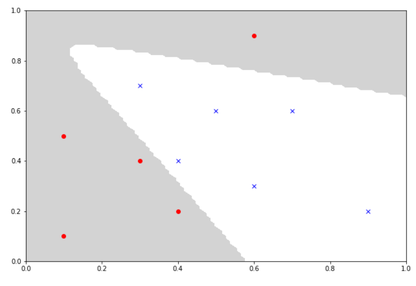
As an example, the set of points in the figure below left are in either category A (circles) or category B (crosses). Training a neural network as the one shown below right to classify points in $\mathbb{R}^2$, say according to the largest component of $a^{[L]}$





results in the classification of the points of $[0,1]^2$ as shown below left. Adding an additional training point and re-training the neural network results in the categorization shown below right.





The master thesis on which this poster presentation is based was presented and approved on October 20 2022. The committee members were Michael Plexousakis (plex@uoc.gr), Theodoros Katsaounis (thodoros.katsaounis@uoc.gr), and Panagiotis Chatzipantelides (p.chatzipa@uoc.gr).