6.883 Science of Deep Learning – Spring 2018 February 12, 2018

Lecture 2: Overview of Continuous Optimization Methods

Lecturer: Aleksander Mądry *Scribes: Scott Foster, Luke Kulik, Kevin Li*
(Revised by Andrew Ilyas and Dimitris Tsipras)

1 Introduction

Continuous Optimization is an important toolkit for deep learning. It aims to solve the canonical problem is solving the unconstrained minimization problem:

$$\underset{x}{\text{minimize}}\ f(x), \qquad \text{where } f \text{ is continuous and smooth.} \tag{1}$$

(For us, smooth means that derivatives of all orders exist.) This problem is (of course) intractable in general, so we will make additional assumptions in order to be able to design algorithms. It turns out that all of our algorithms will be iterative. So that we will need to specify a primitive that, given the current solution x^t determines in what direction and how far we should go to obtain an even better solution x^{t+1}.

2 Gradient Descent Method

The most fundamental tool for solving continuous optimization problems is gradient descent method. The basic idea here relies on the Taylor expansion of our function around our current point x (in order to find the best step Δ to take to arrive to the new solution). This expansion states that

$$f(x+\Delta) = \underbrace{f(x) + \nabla f(x)^\intercal \Delta}_{\varphi_x(\Delta)} + \underbrace{\frac{1}{2}\Delta^\intercal \nabla^2 f(x) \Delta + \ldots}_{\varrho_x(\Delta)}, \tag{2}$$

where $\varphi_x(\Delta) = f(x) + \nabla f(x)^\intercal \Delta$ can be viewed as the linear approximation of our function at the point x and $\varrho_x(\Delta)$ is the "tail error" of this approximation. Note that

$$\|\varphi_x(\Delta)\| \sim \|\Delta\| \quad \text{and} \quad \|\varrho_x(\Delta)\| \leq O(\|\Delta\|^2). \tag{3}$$

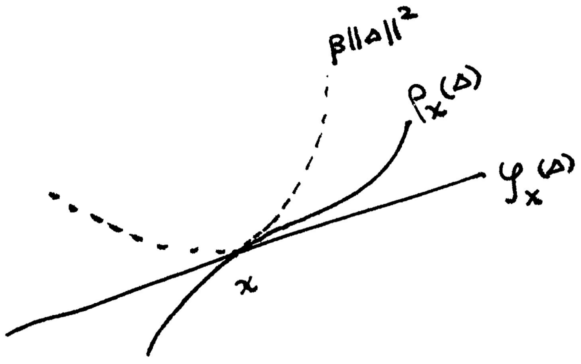
for sufficiently small Δ. So, in principle, by taking Δ small enough we can always ensure that the error of our linear approximation is smaller than the benefit from moving in the direction that minimizes $\varphi_x(\Delta)$.

2.1 Smoothness Assumptions

The second condition in (3) can be seen as a form a of smoothness condition. Specifically, we define the notion of β-smoothness.

Definition 1 *A function f is β-smooth iff*

$$\varrho_x(\Delta) \leq \frac{1}{2}\beta\|\Delta\|^2, \quad \textit{for all } x \textit{ and } \Delta. \tag{4}$$



$$f(x) = \varphi_x(x) + \rho_x(\Delta) \leq \varphi_x(x) + \beta\|\Delta\|^2$$

In particular, this implies that f is dominated by a quadratic function $\varphi_x(\Delta) + \beta\|\Delta\|^2$. That is,

$$f(x+\Delta) \leq \varphi_x(\Delta) + \beta\|\Delta\|^2,$$

for all x and Δ.

Thus we can consider minimizing the proxy $\varphi_x(\Delta) + \beta\|\Delta\|^2$ as a function of Δ in lieu of minimizing $f(x+\Delta)$ directly. Since the minimizer is $\Delta^* = -(2\beta)^{-1}\Delta f(x)$, we arrive at the gradient descent algorithm.

- Pick an initial point, say, $x^0 = 0$.
- For $t = 0, 1, \ldots, T$, set
$$x^{t+1} = x^t - \frac{1}{\beta}\nabla f(x^t). \tag{5}$$

This basic algorithm has the following guarantee on the amount of progress made in each step

$$f(x^t) - f(x^{t+1}) \geq \frac{1}{2\beta}\|\nabla f(x^t)\|. \tag{6}$$

As a result, it eventually is bound to a point $\widehat{x}$ such that

$$\|\nabla f(\hat{x})\| \approx 0. \tag{7}$$

This means that $\hat{x}$ will be a critical point. However, it might be a saddle point and not necessarily a local extremum. So, further assumptions are needed to guarantee convergence to optimality.

2.2 Convexity

If we assume that f is convex, then $\hat{x}$ is indeed guaranteed to be a global minimum. Convexity is equivalent to having the bound

$$0 \leq \varrho_x(\Delta) \leq \frac{1}{2}\beta\|\Delta\|^2, \tag{8}$$

for all x and Δ.

It can then be shown that after $T = O(\beta R^2 \varepsilon^{-1})$ steps, where $R = \|x^0 - x^*\|$, we have that $f(x^T) - f(x^*) \leq \varepsilon$. However, this linear dependence on ε^{-1} and R can be quited inconvenient. So, we need a stronger assumption to get a better bound.

2.3 Strong-α Convexity Assumption

Definition 2 *The function f is said to be strong-α convex iff*

$$\frac{1}{2}\alpha\|\Delta\|^2 \le \varrho_x(\Delta) \quad \textit{for all } x \textit{ and } \Delta \textit{ and } \alpha > 0. \tag{9}$$

So, if we assume that f is strong-α convex and β-smooth, this means that the error $\varrho_x(\Delta)$ is "sandwiched" by two quadratics, i.e.,

$$\frac{1}{2}\alpha\|\Delta\|^2 \le \varrho_x(\Delta) \le \frac{1}{2}\beta\|\Delta\|^2,$$

for all x and Δ. Consequently, the function $f(x+\Delta)$ is "sandwiched" as well by corresponding quadratics too. That is, we have

$$\varphi_x(\Delta) + \frac{1}{2}\alpha\|\Delta\|^2 \le \varrho_x(\Delta) \le \frac{1}{2}\beta\|\Delta\|^2,$$

for all x and Δ. (See the Figure 1.)

Once this assumption is in place, we can attain $f(x^T) - f(x^*) \le \varepsilon$ using only

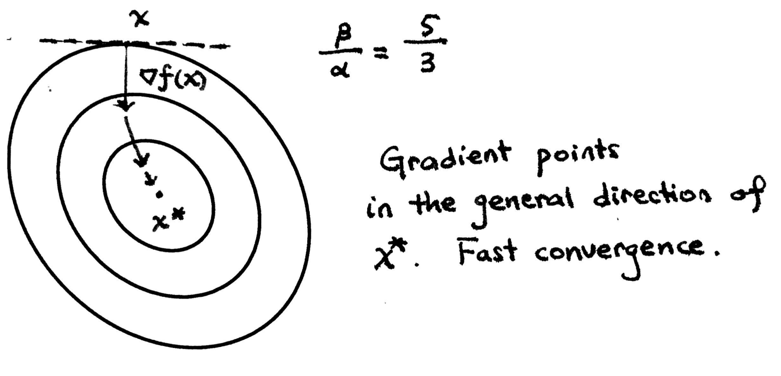
$$T = O\left(\frac{\beta}{\alpha}\log\frac{R}{\varepsilon}\right) \text{ steps. That is, we have a logarithmic dependency in } \varepsilon^{-1}. \tag{10}$$

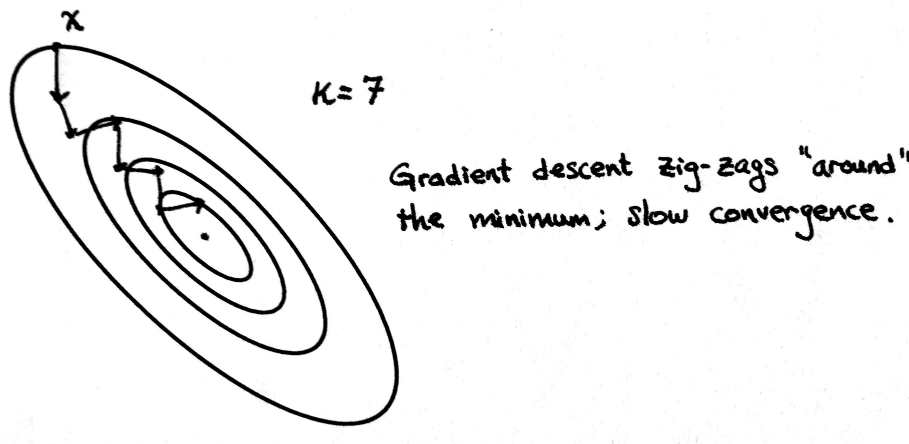
The quantity $\kappa := \beta/\alpha$ is called the condition number of f. It can be viewed as a reflection of the "badness" of the (Euclidean) geometry of f.

In particular, if f is twice differentiable everywhere (which we assumed here), the values of α and β correspond to the bounds on the smallest and largest eigenvalues of the Hessian. That is,

$$\alpha = \inf_x \lambda_{\min}(\nabla^2 f(x)) \quad \text{and} \quad \beta = \sup_x \lambda_{\max}(\nabla^2 f(x)) \tag{11}$$

Now, to get some intuition regarding why this condition number is important one should note that when $\kappa = 1$ (i.e., when f is quadratic), the gradient point directly in the direction of the minimizer. Conversely, when κ is large, the gradient direction does not correlate well with the direction towards minimum. As a result, the optimization path tends to slowly zig-zag toward the minimizer. See the figures below.


x
$\nabla f(x)$
x^*
$\frac{\beta}{\alpha} = \frac{5}{3}$
Gradient points in the general direction of x^*. Fast convergence.


x
$\kappa = 7$
Gradient descent zig-zags "around" the minimum; slow convergence.

2.4 Momentum Gradient Descent

Momentum gradient descent attempts to overcome the zig-zagging problem we alluded to above. The algorithm is given by

$$\begin{aligned} v^{t+1} &= \gamma v^t - \varepsilon \nabla f(x^t) \\ x^{t+1} &= x^t + v^{t+1} \end{aligned} \tag{12}$$

One can view this dynamics as corresponding to a physical system of heavy ball with friction. Specifically, here, v is the velocity of the ball, $\gamma < 1$ is the friction parameter and the gradient corresponds to the applied force. The intuition why this kind of dynamics might be helpful is that zig-zagging along the directions orthogonal to the direction of the minimum point will cancel out, and the velocity will instead build up in the desired direction of the minimum.

Indeed, one can show that a certain "predictive" variant of this dynamics, known as Nesterov's momentum variant of gradient descent, given by

$$\begin{aligned} v^{t+1} &= \gamma_t v^t - \varepsilon_t \nabla f(x^t + \gamma_t v^t) \\ x^{t+1} &= x^t + v^{t+1}, \end{aligned} \tag{13}$$

with a very specific way of setting γ_t and ε_t, can be formally analyzed and shown to achieve ε-optimality in only

$$T = O\left(\sqrt{\kappa} \log \frac{R}{\varepsilon}\right) \text{ steps.} \tag{14}$$

Also, one can show that if the interaction with f is restricted to its gradients only (so-called first order optimization model), then the square root dependence on κ is the best asymptotically possible bound (in the worst case).

3 Generalized gradient descent

In machine learning applications, we typically are able to learn more about f than its gradient—can we use this extra information to achieve better bounds? Recall that the above analysis relied on the sandwiching inequality

$$\frac{1}{2}\alpha\|\Delta\|^2 \leq \varsigma_x(\Delta) \leq \frac{1}{2}\beta\|\Delta\|^2, \quad \text{for all } x \text{ and } \Delta. \tag{15}$$

The norm used in the above equation so far was, naturally, assumed to be an Euclidean norm. However, it does not have to be! We can change the norm $\|\cdot\|$ so that, in effect, the constants α and β are improved. And what we will get in this way is so-called *general gradient descent method.*

There is many norm choices we might consider here. But the one family of norms that will be most useful to us will be so-called A-*norm* $\|\cdot\|_A$ specified by some positive definite matrix $A \succ 0$ and defined as

$$\|v\|_A := v^\intercal A v.$$

Note that if A is an identity matrix, this corresponds directly to the Euclidean norm.

Now, in analogy to the "standard" gradient descent setup, we minimize the objective

$$\underset{\Delta}{\text{minimize}}\ \varphi_x(\Delta) + \beta\|\Delta\|_A^2. \tag{16}$$

Note the solution to (16) is given by

$$\frac{1}{\beta}A^{-1}\nabla f(x) = \arg\min_{\Delta} \varphi_x(\Delta) + \frac{1}{2}\beta\|\Delta\|_A^2. \tag{17}$$

So, it is *not* necessarily corresponding to moving in the direction of the actual gradient.

The right choice of A will be helpful as it can "rescale" the ellipsoidal level sets of f (i.e., which correspond to large κ) to make them much "rounder" and thus enable more rapid convergence. (Although at the cost of having to compute the inverse of A, or solving a linear system in it each time we take a step.)

Now, one could wonder what is the "best" choice of A. This choice, in a sense, turns out to be taking A to be the Hessian $\nabla^2 f(x)$ of our function at the point x. (Note that it means this might be a different norm at different points!) This choice is motivated by noticing that if Δ is "not too big" we have that

$$\varrho_x(\Delta) \lessapprox \frac{1}{2}\Delta^\intercal \nabla^2 f(x)\Delta \approx \frac{1}{2}\|\Delta\|^2_{\nabla^2 f(x)}. \tag{18}$$

So, the (local) condition number here is close to 1 and thus being best possible.

The resulting algorithm is known as *Newton's method* and its update rule ends up being

$$x^{t+1} = x^t - \eta\nabla^2 f(x)^{-1}\Delta f(x). \tag{19}$$

Here, the step size η have to be chosen carefully to address the requirement that Δ needs to be "not too big". In general, analyzing the convergence of such methods turns out to be tricky (also, because of the norm potentially changing in each iteration) and is beyond the scope of this class. But it can lead to significant speed ups. Unfortunately, this comes at a price of the need to invert the Hessian (or rather solve linear system in it), which tends to be computationally expensive (and also might require too much space to even store the Hessian explicitly). This is particularly relevant in deep learning context.

3.1 Quasi-Newton methods

The idea behind so-called quasi-Newton methods is to try to strike a compromise between the benefit of being able to improve the geometry of the problem via rescaling and the computation time/space constraints. The basic idea is to use the update rule

$$x^{t+1} = x^t - \eta H_t^{-1}\nabla f(x) \tag{20}$$

for a sequence of matrices $H_t \succ 0$ that approximates the Hessian (c.f. (19)) in some way.

We discuss two choices of approximations, the **BFGS** (as well as its variant **L-BFGS**) and the AdaGrad algorithm.

3.2 BFGS method

In the BFGS method we aim to approximate in certain sense the *inverse* of the Hessian. Specifically, we start with a "trivial" guess for the Hessian: the identity matrix, i.e., $B_0 = I$. Then, in each step, we refine this guess by updating it based on the new "information" we got about the Hessian. (Note that, importantly, we make an implicit assumption here that Hessians at different point are the same/close to each other.)

Note that the inverse Hessian and thus our guess for it B_t at time t, has to satisfy a number of simple conditions. First of all, it has to be positive definite, i.e., $B_t \succ 0$. Secondly, if we observe how the gradients change between two consecutive points, it has to "explain" this change. That is, we need to have that

$$B_t y_t = s_t \tag{21}$$

where $s_t = x^t - x^{t-1}$ and $y_t = \nabla f(x^t) - \nabla f(x^{t-1})$.

This constraints restrict the choice of B_t but do not uniquely define it. To this end, we settle on a choice of B_t that satisfies these constraints and is "minimal" in the sense of its similarity to our previous estimate B_{t-1}. Specifically, we take it to be

$$B_t = \min_B \|B - B_{t-1}\|_F \text{ subject to } B_t y_t = s_t \text{ and } B_t \succ 0 \tag{22}$$

where $\|\cdot\|_F$ is the Frobenius norm.

Once again, the implicit assumption in this algorithm is that the Hessian is globally constant; indeed, it is possible to show that BFGS accelerates the optimization of quadratic functions. This is clearly not true in applications, yet this method works well in practice (sometimes).

In high dimensional problems, even storing the Hessian is not really feasible though. So, a simplified version of the BFGS method, known as L-BFGS ("limited memory" BFGS) is a variant where (22) is replaced with

$$B_{t+1} = \min_B \|B - I\|_F \quad \text{subject to } B_t y_t = s_t \text{ and } B_t \succ 0, \tag{23}$$

i.e., we assume always that B_{t-1} is just an identity matrix. The L-BFGS heuristic is also successfully used in practice, even though its theoretical motivation seems to be much weaker.

4 AdaGrad

The final algorithm we consider is the `AdaGrad` algorithm, which is derived using the so-called online convex optimization (or online learning) framework. In this framework, one considers the following iterative online prediction game:

- For each $t = 1, \dots, T$:
    1. output a choice x_t;
    2. learn a "penalty" function f_t and incur penalty $f_t(x_t)$ corresponding to your choice.

The goal of this game is to choose a sequence of choices so that to minimize the sum of all the corresponding penalties $\sum_{t=1}^{T} f_t(x_t)$.

Of course, since the penalty function f_t is revealed only after the choice x_t was made, the resulting penalty can be arbitrary large. So, there is no hope one would be able to provide any non-trivial guarantees for a given strategy of making the choices in absolute terms. Still, it turns out that there is a useful measure of *relative* performance: the *regret*, which is defined as

$$R(T) := \sum_t f_t(x_t) - \min_x \sum_t f_t(x). \tag{24}$$

That is, R is the difference between the penalty of our algorithm against a hypothetical algorithm which knows the sequence $\{f_t\}$ in advance but is restricted to choosing a single choice x^* in each one of the rounds.

Note that minimizing $R(T)$ with $f_1 = \cdots = f_T$, $f := f_T$, amounts to finding a minimum of f. In particular, if $R(T)$ is sub-linear, i.e., $R(T) \in o(T)$, then x_t is making progress towards the minimum:

$$R(T) \in o(T) \implies \frac{1}{T}\sum_t f(x_t) - f(x^*) \to 0, \quad \text{where } f(x^*) = \min f(x). \tag{25}$$

4.1 Follow the Regularized Leader

One algorithm that achieves sublinear regret is the *regularized follow the leader* algorithm, defined by always playing

$$x^t := \arg\min_x \left[\eta \sum_{s=1}^{t-1} f_s(x) + \|x\|^2 \right], \tag{26}$$

and taking $x_0 = 0$.

For appropriate choices of the parameter η, we can show that

$$R(T) \leq O\left(\|x^* - x^0\| \sqrt{\sum \|\nabla f_t(x_t)\|} \right) \in o(T), \tag{27}$$

as required.

Now, in analogy to what we did above, we could make this algorithm be tuned to the geometry of the problem by introducing a different (to Euclidean) norm $\| \cdot \|_{H_t}$ in each step. Specifically, we could have

$$x^t := \arg\min_x \left[\eta \sum_{s=1}^{t-1} f_s(x) + \|x\|^2_{H_t} \right] \quad \text{in which case} \quad R(T) \leq O\left(\|x^* - x^0\| \sqrt{\sum \|\nabla f_t(x_t)\|_{H_t}} \right). \tag{28}$$

It turns out that we can achieve the tighter bound

$$R(T) \leq \min_{H \succ 0} O\left(\|x^* - x^0\| \sqrt{\sum \|\nabla f_t(x_t)\|_H} \right), \quad \text{the minimum over positive definite } H \tag{29}$$

if we set H_t to the square-root of the empirical covariance matrix

$$H_t = \left(\sum_{s=1}^{t-1} \nabla f_s(x_s) \nabla f_s(x_s)^T \right)^{1/2}. \tag{30}$$

This latter choice is the `AdaGrad` algorithm. In machine learning, the covariance matrix may be too large to fit inside memory, so another choice is to use the only the diagonal entries

$$\tilde{H}_t = \left(\sum_{s=1}^{t-1} \operatorname{diag} H_t \right)^{1/2}. \tag{31}$$

One benefit of (30) (and also (31)) is that it allows different coordinates of x to have different learning rates. This is especially useful in cases coordinates of the data set have different rates of occurrence. Consider a natural language setting where

$$x = (\text{binary vector of 0's and 1's})_{v \in V} \tag{32}$$

with one word v for every word in the vocabulary V. If v is a common word, then there will be many 1's in the v^{th} column (e.g., '*and*') which may not be very informative; on the other hand, a less common word w may be very informative (e.g., '*solipsism*'). A different learning rate for every coordinate allows us to "slow down" the not very helpful learning on the v^{th} coordinate without hampering the helpful learning on the w^{th} one.