COS 597G: Toward Theoretical Understanding of Deep Learning **Fall 2018**

Lecture 3: 20 September 2018

Lecturer: Sanjeev Arora *Scribe: Abhishek Bhrushundi, Pritish Sahu, Wenjia Zhang*

Recall the basic form of an optimization problem:

$$\min (or \max)\ f(x)$$

$$\text{subject to } x \in B,$$

where $B \subseteq \mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$ is the objective function[1]. Gradient descent (ascent) and its variants are a popular choice for finding approximate solutions to these sort of problems. In fact, it turns out that these methods also work well empirically even when the objective function f is non-convex, most notably when training neural networks with various loss functions.

In this lecture we will explore some of the basics ideas behind gradient descent, try to understand its limitations, and also discuss some of its popular variants that try to get around these limitations. We begin with the Taylor series approximation of functions which serves as a starting point for these methods.

3.1 Taylor series approximation

We begin by recalling the Taylor series for univariate real-valued functions from Calculus 101: if $f : \mathbb{R} \to \mathbb{R}$ is infinitely differentiable at $x \in \mathbb{R}$ then the Taylor series for f at x is the following power series

$$f(x) + f'(x)\Delta x + f''(x)\frac{(\Delta x)^2}{2!} + \ldots + f^{(k)}(x)\frac{(\Delta x)^k}{k!} + \ldots$$

Truncating this power series at some power $(\Delta x)^k$ results in a polynomial that approximates f around the point x. In particular, for small Δx,

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + f''(x)\frac{(\Delta x)^2}{2!} + \ldots + f^{(k)}(x)\frac{(\Delta x)^k}{k!}$$

Here the error of the approximation goes to zero at least as fast as $(\Delta x)^k$ as $\Delta x \to 0$. Thus, the larger the k the better is the approximation. This is called the *k^{th}-order Taylor approximation* of f at x. [2]

This can be generalized to the multivariate case. For example, the first-order Taylor approximation of a function $f : \mathbb{R}^d \to \mathbb{R}$ that's differentiable at $x \in \mathbb{R}^d$ is given by

$$f(x + \Delta x) \approx f(x) + \Delta x^T \nabla f|_x.$$

Here $\nabla f|_x$ is the gradient of f at x, i.e. a column vector in $\mathbb{R}^d$ whose i$^{\text{th}}$ coordinate is $\frac{\partial f}{\partial x_i}$. Similarly, the second-order Taylor approximation of a function f that's twice-differentiable at $x \in \mathbb{R}^d$ has the following form:

$$f(x + \Delta x) \approx f(x) + \Delta x^T \nabla f|_x + \frac{1}{2}(\Delta x)^T(\nabla^2 f|_x)(\Delta x).$$

Here $\nabla^2 f|_x$ is the Hessian of f, i.e. a symmetric $d \times d$ matrix whose (i, j) entry is $\frac{\partial^2 f}{\partial x_i \partial x_j}$.

[1] For the sake of simplicity, we will assume that we are dealing with a minimization problem and that $B = \mathbb{R}^d$ throughout the lecture.

[2] Of course, for the k$^{\text{th}}$ order Taylor approximation to exist, we only need f to be k-times differentiable at x.

If f satisfies slightly stronger assumptions than just differentiability we can bound the error of approximation using Taylor's theorem. We will only state the result for first-order Taylor approximation since we will use it in later sections to analyze gradient descent.

Theorem 1 (Multivariate Taylor's theorem (first-order))**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be such that f is twice-differentiable and has continuous derivatives in an open ball B around the point $x \in \mathbb{R}^d$. Then for any small enough $\Delta x \in \mathbb{R}^d$ such that $x + \Delta x$ is also contained in the ball B, we have the following:*

$$f(x + \Delta x) = f(x) + \Delta x^T \nabla f|_x + \frac{1}{2}(\Delta x)^T (\nabla^2 f|_w)(\Delta x),$$

where $\nabla^2 f|_w$ is the Hessian of f evaluated at a point $w \in \mathbb{R}^d$ that lies on the line connecting x and $x + \Delta x$.

3.2 Gradient descent

Recall that, at any step $t \geq 0$, when at a point $x_t \in \mathbb{R}^d$, gradient descent tries to move in a direction $\Delta x \in \mathbb{R}^d$ such that $f(x_t + \Delta x) < f(x_t)$. This is typically done by choosing $\Delta x = -\eta \nabla f|_{x_t}$ for a small η. The intuition for this choice of Δx comes from the first-order Taylor approximation of f around x_t. Assuming f is differentiable at x_t, we know that for any small enough (in magnitude) Δx,

$$f(x_t + \Delta x) \approx f(x_t) + \Delta x^T \nabla f|_{x_t}.$$

Now it's clear that in order to get maximum leverage out of moving along Δx we should align Δx along $-\nabla f|_{x_t}$. This explains why gradient descent sets chooses $\Delta x = -\eta \nabla f|_{x_t}$: if $x_{t+1} = x_t - \eta \nabla f|_{x_t}$ then

$$f(x_{t+1}) = f(x_t) - \eta \left((\nabla f|_{x_t})^T \nabla f|_{x_t} \right) < f(x_t)$$

since $(\nabla f|_{x_t})^T \nabla f|_{x_t} > 0$. The parameter $\eta > 0$, if you recall, is called the *learning rate*. Intuitively, since the first-order approximation is good only for small Δx, we want to choose a small $\eta > 0$ to make Δx small in magnitude. Additionally, it has been observed empirically that a high learning rate leads to "overshooting" past the local minima/stationary point and may even lead to the algorithm diverging.

On the other hand, a small η increases the time the algorithm takes to converge. Choosing an η that balances between the two extremes can often be tricky in practice. In fact, in some cases it helps to make the learning rate adaptive, i.e. it can change with t!

3.2.1 Limitations of gradient descent

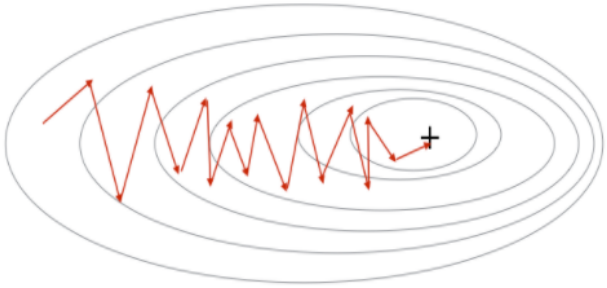
The downside of using a first-order approximation and moving in the direction of the negative of the gradient is that, for any finite η, this direction may be almost orthogonal to the optimal direction, i.e. the direction $x^* - x_t$ where x^* is the local minima/stationary point we are trying to get to. This can lead to a lot of "zig-zag" movement and thus a slower rate of convergence (See Figure 3.1).

It may be noted that this kind of behavior results from the level sets of f being somewhat "ellipsoidal" in the region around the local minima/stationary point and that this information about the behavior of the level sets is captured by the Hessian of f in the region.[3].

Then, a possible solution to this, though it's unclear if it helps, is to use the second-order Taylor approximation instead of the first-order approximation. Unfortunately, computing the Hessian at every point x_t can be prohibitively

[3] In particular, if the Hessian is ill-conditioned then the level sets tend to have an ellipsoidal shape. We define what ill-conditioned means later in the lecture.

Figure 3.1: Movement along the negative of the gradient for finite η



computationally expensive[4], although it turns out that we can design methods that use information about the Hessian *implicitly*.

3.3 Algorithms that use second-order information implicitly

3.3.1 Some basic facts and definitions from linear algebra

Fact 2. Let A be an $n \times n$ real symmetric matrix. Then A has all real eigenvalues.

Fact 3. Let $\lambda_{max}(A)$ and $\lambda_{min}(A)$ denote the largest and smallest eigenvalue of a real symmetric matrix A respectively. Then

$$\lambda_{\max}(A) = \max_{||x||_2=1} x^T Ax = \max_{x \neq 0} \frac{x^T Ax}{||x||_2}$$

$$\lambda_{\min}(A) = \min_{||x||_2=1} x^T Ax = \min_{x \neq 0} \frac{x^T Ax}{||x||_2}$$

Definition 4 (Smoothness of real symmetric matrix)**.** Given a real symmetric matrix A, its *smoothness*, denoted by $\beta(A)$, is defined as

$$\beta(A) \coloneqq \max\{|\lambda_{\max}(A)|, |\lambda_{\min}(A)|\}.$$

The following is an easy consequence of the above facts and definitions.

Fact 5. If A is a real symmetric matrix with smoothness $\beta(A)$ then for every unit vector x we have that

$$|x^T Ax| \leq \beta(A).$$

Definition 6 (stationary point)**.** Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. Then any point $x \in \mathbb{R}^d$ where $\nabla f|_x = 0$ is called a *stationary point*.

3.3.2 Gradient descent under smoothness assumptions

One way to use second-order information implicitly is to use a bound on the smoothness of the Hessian of the objective function f and then use it to set the learning rate appropriately. In this case, we can prove that gradient descent quickly

[4]Especially when the function f is the loss function for training a deep neural network with a large number of parameters; the size of the Hessian will be huge !

converges to a region where $||\nabla f||_2$ is small, i.e. close to a stationary point of f.
We will now state and prove a lemma that captures this formally. We remark that the lemma holds for weaker conditions on f than those required by the statement below. We use $||\cdot||$ to denote $||\cdot||_2$.

Lemma 1 (Descent lemma)**.** *Suppose f is a twice-differentiable function with continuous derivatives everywhere such that, for all $x \in \mathbb{R}^d$,*

$$\beta(\nabla^2 f|_x) \leq \beta.$$

If the learning rate η is set to $\frac{1}{\beta}$ then for every step $t \geq 0$ of gradient descent we have that

$$f(x_t) - f(x_{t+1}) \geq \frac{1}{2\beta}||\nabla f|_{x_t}||^2,$$

where $x_{t+1} = x_t - \eta \nabla f|_{x_t}$.

Proof. For the rest of the proof, let us denote $\nabla f|_{x_t}$ by ∇f, and let $\Delta x = -\eta \nabla f = \frac{-\nabla f}{\beta}$. Then $x_{t+1} = x_t + \Delta x$.

We now use Theorem 1 to get a Taylor approximation of f around x_t:

$$f(x_t + \Delta x) = f(x_t) + (\Delta x)^T \nabla f + \frac{1}{2}\Delta x^T \left(\nabla^2 f|_w\right) \Delta x,$$

where w is some point on the line joining x and $x + \Delta x$. Since $\Delta x = -\frac{\nabla f}{\beta}$, it follows that

$$(\Delta x)^T \nabla f = -\frac{||\nabla f||^2}{\beta}.$$

Furthermore, since $\nabla^2 f|_w$ has smoothness β (by assumption on f), we have that

$$\Delta x^T \left(\nabla^2 f|_w\right) \Delta x = ||\Delta x||^2 \left(\frac{\Delta x}{||\Delta x||}\right)^T \left(\nabla^2 f|_w\right) \left(\frac{\Delta x}{||\Delta x||}\right) \leq ||\Delta x||^2 \cdot \beta = \frac{||\nabla f||^2}{\beta}$$

Substituting these values in the expression for $f(x_t + \Delta x)$, we get

$$\begin{aligned} f(x_{t+1}) &= f(x_t + \Delta x) \\ &\leq f(x_t) - \frac{1}{\beta}||\nabla f||^2 + \frac{1}{2\beta}||\nabla f||^2 \\ &= f(x_t) - \frac{1}{2\beta}||\nabla f||^2 \end{aligned}$$

□

Now suppose we keep running gradient descent till the $||\nabla f||$ is not too small, the above lemma guarantees that the algorithm "makes a descent" in every step. If our objective function f has a lower bound on the minimum value it can take (which would most likely be the case if f were a loss function) then the algorithm can "descend" only a finite number of times and must eventually arrive close to a stationary point where the gradient is small.

Note that, for non-convex functions, a stationary point may be a *saddle point*[5]. It's impossible to tell whether a point with zero gradient is a saddle point or a local extremum without information about second-order derivatives.

Task: An interesting experiment to do is to check if $||\nabla f||$ becomes "small" (perhaps relative to $||x||$?) towards the end of the training phase in deep learning when the loss becomes small.

[5] A point that is not a local extremum yet has zero gradient is called a saddle point.

3.3.3 Heavy-ball method and Nesterov's accelerated gradient

Heavy-ball method, which is also referenced as *momentum* in deep learning, was proposed by Polyak [4] and is a modification of vanilla gradient descent. The main idea here is to move in a direction given by a linear combination of past gradients in each step of the algorithm. More precisely, it maintains $x^{(t)}$, the current position in space, and $u^{(t)}$, the "momentum" which is a linear combination of past gradients, using the following update rules:

$$u^{(t+1)} = u^{(t)} - \beta^{(t)} \nabla f|_{x^{(t)}}$$
$$x^{(t+1)} = x^{(t)} + \gamma^{(t)} u^{(t)}.$$

Here the values $\gamma^{(t)}$ and $\beta^{(t)}$ are set based on bounds on the condition number of the Hessian of f (bounds that hold for the entire "landscape" of f). The condition number of a real symmetric matrix A is given by

$$\kappa = \frac{|\lambda_{max}(A)|}{|\lambda_{min}(A)|}.$$

Nesterov later proposed *accelerated gradients* (NAG) [1] which has similar update rules as heavy-ball method:

$$u^{(t+1)} = x^{(t)} - \beta^{(t)} \nabla f|_{x^{(t)}}$$
$$x^{(t+1)} = u^{(t)} + \gamma^{(t)} (u^{(t+1)} - u^{(t)}).$$

If the only information that's available is the gradients and nothing else, then Nesterov's momentum is asymptotically the fastest — there is a convex function for which NAG has the fastest convergence (with rate $\mathcal{O}(t^{-2})$) among all first-order methods. Instead, current analysis only gives a convergence rate of $\mathcal{O}(t^{-1})$ for heavy-ball method [5].

Task: The momentum method takes inspiration from the concept of momentum in physics, and there is a way to view the algorithm from a physics perspective using Lagrangians. A possible task is to understand this explanation and write a 2-3 pages long summary.

3.3.4 Adaptive regularization

These type of methods are a variant of gradient descent that, instead of moving along $-\eta \nabla f|_{x_t}$ and using the same learning rate η for every coordinate, use different learning rates for different coordinates, i.e. an update looks like $x_{t+1} = x_t - G_t \nabla f_{x_t}$ where G_t is a $d \times d$ diagonal matrix whose diagonal entries are the coordinate-wise learning rates $\eta_1^t, \ldots, \eta_d^t$. The diagonal matrix itself is computed using information about past gradients. We will see this in detail in a future lecture. A prominent example of this type of method is AdaGrad [2].

There are also other methods that combine the idea of momentum as in Nesterov's method and adaptive regularization. An example of such a method is ADAM [3].

3.4 Second-order methods

3.4.1 Newton's method

This method uses the second-order Taylor approximation to perform an update. Consider the second-order Taylor approximation of f around the point x_t:

$$f(x_t + \Delta x) \approx f(x_t) + \Delta x^T \nabla f|_{x_t} + \frac{\Delta x^T \nabla^2 f|_{x_t} \Delta x}{2}.$$

We now want to find a Δx that minimizes the RHS. Let $z = \Delta x$, $\nabla f = \nabla f|_{x_t}$ and $H = \nabla^2 f|_{x_t}$, then we want to find

$$\operatorname*{argmin}_z \left(f(x_t) + z^T \nabla f + \frac{1}{2} z^T H z \right).$$

Since it's not clear how to do this, we instead find an Δx such that $x_t + \Delta x$ is a stationary point by differentiating the expression with respect to z and setting it to zero:

$$\frac{\partial}{\partial z} \left(f(x_t) + z^T \nabla f + \frac{1}{2} z^T H z \right) = \nabla f + Hz = 0$$

$$\implies z = -H^{-1} \nabla f.$$

Thus, the update equation in Newton's method looks like

$$x_{t+1} = x_t - H^{-1} \nabla f.$$

3.5 Possible solution concepts

There are various possibilities for what a satisfactory solution to an optimization problem might look like, other than a global optimum (which might not always be possible to find).

1. **Stationary points**: These are points where $\nabla f = 0$. In the case of non-convex functions, these points may be local extrema or saddle points. The descent lemma says that, under smoothness assumptions, gradient descent can reach a stationary point in time $\mathcal{O}(\beta)$, where β is a bound on the smoothness. Newton's method (under technical assumptions) can be shown to converge to a stationary point.
2. **Second-order local minima**: These are points where $\nabla f = 0$ and $\nabla^2 f$ is PSD. In future lectures, we will see variants of gradient descent that can be used to find such points.
3. **Higher-order local minima**: The general conditions for such local minima in higher dimensions involve higher-order derivatives and it's not clear how to find such points efficiently.
4. **Global optima**: There is no compact way to express conditions for global minima under complexity-theoretic assumptions.

References

[1] Yurii Nesterov, *A method for solving the convex programming problem with convergence rate $O(1/k^2)$,* Dokl. Akad. Nauk SSSR, Vol. 269, 1983.

[2] John Duchi, Elad Hazan, Yoram Singer, *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,* Journal of Machine Learning Research, 2011

[3] Diederik P Kingma and Jimmy Ba, *Adam: A method for stochastic optimization,* 3rd International Conference for Learning Representations (ICLR), 2015

[4] Polyak, Boris T. *Some methods of speeding up the convergence of iteration methods,* USSR Computational Mathematics and Mathematical Physics 4.5, 1964.

[5] Ghadimi, Euhanna, Hamid Reza Feyzmahdavian, and Mikael Johansson. *Global convergence of the heavy-ball method for convex optimization.*, Control Conference (ECC), 2015 European. IEEE, 2015.