ECE 901: Large-scale Machine Learning and Optimization **Fall 2016**

Lecture 5 — 09/20

Lecturer: Dimitris Papailiopoulos *Scribe: Zachary Charles*

5.1 Convergence Rates for Gradient Descent

5.1.1 Why are convergence rates useful?

Say we want to solve the following optimization problem:

$$\min_{x \in X} f(x)$$

Here, we assume that X is a convex set and f is a convex function. In general, we want a *provable* worst-case convergence rate that guarantees how close we can get to the optimum value f^* in T iterations of our optimization algorithm.

Say we use gradient descent to produce iterates x_k. Oftentimes, for a given class of functions, we will be able to prove a convergence rate of the form $f(x_T) - f* \leq p(T)s(f, X)$. Here, $p(T)$ is a function only of T, while $s(f, X)$ is a function of f and its feasible set X. For example, if f is a β-smooth function with domain given by all of $\mathbb{R}$, $s(f, X)$ could be a function of β.

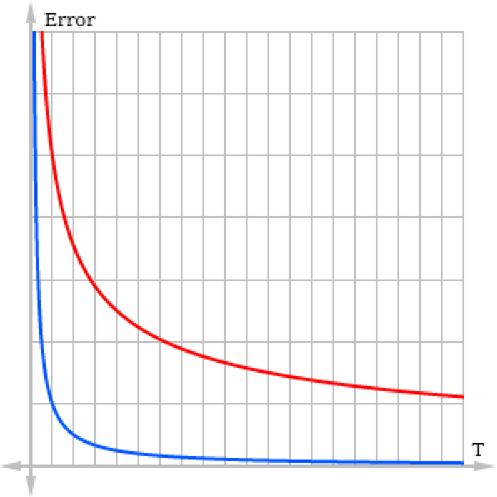

Figure 5.1. An idealized convergence rate (red) versus the actual convergence rate (blue)

The graph above shows an idealized provable convergence rate (in red) versus an idealized actual convergence rate of a function f. The important idea here is that while provable convergence rates may be strictly larger than the convergence rate of a given function, they allow us to get a handle on large classes of functions and their convergence rates. The convergence rates are often determined by which class of functions we are looking at.

5.1.2 Convergence rate for convex, Lipschitz functions

We assume that f is a convex, L-Lipschitz function. We further assume that f achieves its minimum value f^* in the interior of the (convex) feasible set at the point x^*. Recall that gradient descent is an iterative method that uses the following update method:

$$x_{k+1} = x_k - \gamma \nabla f(x_k)$$

Here, $\gamma > 0$ is some constant we define. It is referred to as the *learning rate*. Last time, we derived the following inequality in gradient descent for convex, L-Lipschitz functions:

$$\begin{aligned} f(x_k) - f^* &\leq \frac{1}{2\gamma}\bigg(||x_k - x^*||^2 - ||x_{k+1} - x^*||_2 + \gamma^2||\nabla f(x_k)||^2\bigg) \\ &\leq \frac{1}{2\gamma}\bigg(||x_k - x^*||^2 - ||x_{k+1} - x^*||_2 + \gamma^2 L^2\bigg) \end{aligned}$$

For simplicity, we define $\Delta_i = ||x_i - x^*||$. We then get the following theorem:

Theorem 5.1. *Assume $f(x)$ is a convex, L-Lipschitz function and that we produce T iterates x_k via gradient descent. Choosing $\gamma = \frac{\Delta_1}{L\sqrt{T}}$, we can achieve a convergence rate given by:*

$$f\bigg(\frac{1}{T}\sum_{k=1}^{T} x_k\bigg) - f^* \leq \frac{\Delta_1 L}{\sqrt{T}}$$

Proof: Looking at the first T iterates, we get the following inequalities:

$$\begin{aligned} f(x_1) - f^* &\leq \frac{1}{2\gamma}(\Delta_1^2 - \Delta_2^2) + \frac{\gamma}{2}L^2 \\ f(x_2) - f^* &\leq \frac{1}{2\gamma}(\Delta_2^2 - \Delta_3^2) + \frac{\gamma}{2}L^2 \\ &\vdots \\ f(x_T) - f^* &\leq \frac{1}{2\gamma}(\Delta_T^2 - \Delta_{T+1}^2) + \frac{\gamma}{2}L^2 \\ &\leq \frac{1}{2\gamma}\Delta_T^2 + \frac{\gamma}{2}L^2 \end{aligned}$$

Note that the last step followed by removing a nonpositive term. Summing these all up and using the fact that the resulting sum is telescoping, we get:

$$\sum_{k=1}^{T}(f(x_k) - f^*) \leq \frac{1}{2\gamma}\Delta_1^2 + \frac{T\gamma L^2}{2}$$

Using the convexity of the function f, we then find:

$$\begin{aligned} f\left(\frac{1}{T}\sum_{k=1}^{T} x_k\right) - f^* &\leq \frac{1}{T}\sum_{k=1}^{T}\Big(f(x_k)\Big) - f^* \\ &\leq \frac{1}{T}\left(\frac{1}{2\gamma}\Delta_1^2 + \frac{T\gamma L^2}{2}\right) \\ &= \frac{1}{2\gamma T}\Delta_1^2 + \frac{\gamma}{2}L^2 \end{aligned}$$

We want to pick γ that minimizes the right-hand side. Taking a derivative wrt γ and setting it equal to 0, we get $\gamma = \frac{\Delta_1}{L\sqrt{T}}$. Plugging this in to our inequality above, we find:

$$f\left(\frac{1}{T}\sum_{k=1}^{T} x_k\right) - f^* \leq \frac{\Delta_1 L}{\sqrt{T}}$$

□

5.1.3 Improving this result

The above result on the convergence rate of gradient descent essentially relied on two assumptions:

1. Convexity of $f(x)$
2. Boundedness of $||\nabla f(x_k)||$ for our iterates x_k

We saw last time that any convex, Lipschitz function $f(x)$ will satisfy these two properties. Of course, to prove any kind of guarantee on the convergence rate of the algorithm, we needed to make some assumptions about $f(x)$. However, without assuming more structure on f, we cannot prove a better convergence rate. Theory has shown that the bound above is asymptotically tight, ie. that for some convex L-Lipschitz function, gradient descent observes a convergence rate that is asymptotically like $\frac{1}{\sqrt{T}}$. What if we want to do better? This line of questioning leads to the following mantra in optimization and machine learning: If we impose more structure on f, we can improve the convergence rate.

5.1.4 Convergence rates for smooth, strongly convex functions

Assume that f is β-smooth and λ-strongly convex. We then have the following lemma, the proof of which can be found in Sebastien Bubeck's book, Lemma 3.11 of *Convex Optimization: Algorithms and Complexity.*

Lemma 5.2 (Coercivity of gradients). *Let $f(x)$ be a λ-strongly convex, β-smooth function on a convex set X. Then for any $x, y \in X$, we have:*

$$\langle \nabla f(x) - \nabla f(y), x - y\rangle \geq \frac{\lambda\beta}{\lambda+\beta}||x-y||^2 + \frac{1}{\lambda+\beta}||\nabla f(x) - \nabla f(y)||^2$$

We will use this lemma to prove a convergence rate for these functions. The gradient descent algorithm remains unchanged from the one above. The theorem below involves the condition number $\kappa = \frac{\beta}{\lambda}$. Note that for λ-strongly convex, β-smooth functions, it is straightforward to show that $\beta \geq \lambda$, implying that $\kappa \geq 1$.

Theorem 5.3. *Let $\gamma = \frac{2}{\lambda+\beta}$ and let $\kappa = \frac{\beta}{\lambda}$. Using gradient descent to produce $T+1$ iterates x_k, we get:*

$$f(x_{T+1}) - f^* \leq \frac{\beta}{2} \exp\left(\frac{-T}{\kappa}\right)\Delta_1^2$$

Proof: Using the update rule for gradient descent and expanding, we get the following equality:

$$\Delta_{k+1}^2 = ||x_{k+1} - x^*||^2 = ||x_k - x^*||^2 - 2\gamma\langle \nabla f(x_k), x_k - x^* \rangle + \gamma^2 ||\nabla f(x_k)||^2$$

By Lemma 5.2 above and the fact that $\nabla f(x^*) = 0$, we have:

$$\begin{aligned}\langle \nabla f(x_k), x_k - x^* \rangle &= \langle \nabla f(x_k) - \nabla f(x^*), x_k - x^* \rangle \\ &\geq \frac{\lambda\beta}{\lambda+\beta}\Delta_k^2 + \frac{1}{\lambda+\beta}||\nabla f(x_k) - \nabla f(x^*)||^2\end{aligned}$$

Plugging this in to our equation for Δ_{k+1}^2 above we get:

$$\begin{aligned}\Delta_{k+1}^2 &\leq \Delta_k^2 - 2\gamma\left(\frac{\lambda\beta}{\lambda+\beta}\Delta_k^2 + \frac{1}{\lambda+\beta}||\nabla f(x_k)||^2\right) + \gamma^2||\nabla f(x_k)||^2 \\ &= \left(1 - \frac{2\gamma\lambda\beta}{\lambda+\beta}\right)\Delta_k^2 + \left(\frac{-2\gamma}{\lambda+\beta} + \gamma^2\right)||\nabla f(x_k) - \nabla f(x^*)||^2 \\ &\leq \left(1 - \frac{2\gamma\lambda\beta}{\lambda+\beta}\right)\Delta_k^2 + \left(\frac{-2\gamma}{\lambda+\beta} + \gamma^2\right)\beta^2\Delta_k^2\end{aligned}$$

Note that this last step holds by β-smoothness of f and the fact that $\nabla f(x^*) = 0$. Plugging in $\gamma = \frac{2}{\lambda+\beta}$, some tedious arithmetic shows:

$$\Delta_{k+1}^2 \leq \left(1 - \frac{4\lambda\beta}{(\lambda+\beta)^2}\right)\Delta_k^2$$

Some more manipulation shows that we can bound this above by $\frac{\kappa-1}{\kappa+1}\Delta_k^2$. Applying this bound repeatedly, we get the equation:

$$\Delta_{k+1}^2 \leq \left(\frac{\kappa-1}{\kappa+1}\right)^k \Delta_1^2$$

Therefore, we get a constant contraction factor of $\frac{\kappa-1}{\kappa+1}$ for each iteration performed. Some straightforward bounds show that this is bounded above further by $\frac{\beta}{2}\exp(-k/\kappa)$. □

5.1.5 Comparison of Gradient Descent Convergence Rates

The following table contains the convergence rate of gradient descent for various classes of functions for an appropriate choice of the learning rate γ. We assume that all of the functions are convex and have a feasible set that is convex.

Function class	**Convergence rate for T iterations**
L-Lipschitz	$\frac{\Delta_1 L}{\sqrt{T}}$
β-smooth	$\frac{\beta\Delta_1^2}{T}$
λ-strongly convex and L-Lipschitz	$\frac{L^2}{\lambda T}$
λ-strongly convex and β-smooth	$\beta\Delta_1^2 \exp(\frac{-T\lambda}{\beta})$

Remark: The bounds in the table above are tight. That is, for each function class, there is some function f that exhibits that convergence rate using gradient descent. For an arbitrary function in that function class, the bound above is an upper bound on the convergence rate.

Remark: In general, it is difficult to determine λ, β, L for λ-strongly convex functions, β-smooth functions, and L-Lipschitz functions. In particular, while the proofs above typically determine the best γ in terms of parameters like λ, β, L, doing so in practice is difficult. Instead, we rely on various heuristics and tricks to determine good learning rates γ for an application of gradient descent.

Moral: If the above table demonstrates one thing, it's the fact that structure helps. If you know that the function you are optimizing belongs to a more restrictive class of functions, you can often use the extra structure to prove nicer results on things such as convergence rates of various algorithms.

5.2 Computational Complexity of Gradient Descent

In the above section, we determined the number of iterations needed to reach certain error levels. This does not determine how much it will cost to run gradient descent. In general, we need to know the number of iterations needed and the cost of each iteration.

For a given function f, let $i_f(\epsilon)$ denote the number of iterations x_k of gradient descent required such that $\Delta_k^2 \leq \epsilon$. Let c_f denote the cost of each iteration of gradient descent. Then the overall cost (called the computational complexity of running gradient descent on f) to achieve an error of at most ϵ is $i_f(\epsilon)c_f$.

The cost of each iteration, c_f, will depend on our function. Once we have the gradient $\nabla f(x_k)$, we simply need to comput $x_k - \gamma \nabla f(x_k)$. Since this cost is often dwarfed by the complexity of finding the gradient, we often compute the cost of gradient in terms of the number of gradient evaluated (equivalently, the number of calls to a gradient oracle). In this case, the complexity of gradient descent on f would be $i_f(\epsilon)$ multiplied by the number of oracle calls.

5.2.1 Gradient Descent Costs for Machine Learning

We now wish to understand how expensive gradient descent can be for machine learning. Say that our function $f(x) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(a_i^T x)$. For given loss functions ℓ_i and fixed training set $a_i \in \mathbb{R}^d$, this is the average error of a classifier with weight x and loss functions ℓ_i. We wish to determine the computational complexity of using gradient descent to train our logistic regression classifier.

For a given x, to compute $\nabla f(x)$ we would need to compute the gradient $\nabla \ell_i(a_i^T x)$ for each i. By the chain rule, we have:

$$\nabla \ell_i(a_i^T x) = \ell_i(a_i^T x) a_i$$

The cost of computing this is proportional to the number of non-zero entries in a_i, which we will denote $nnz(a_i)$. Let A denote the matrix whose columns are given by the a_i. Then to compute $\nabla f(x)$, we would incur a cost proportional to $nnz(A)$.

5.2.2 Example - Logistic Regression

In order to do a more detailed computation, we analyze the cost of gradient descent to train a logistic regression classifier. Say our objective function has the form:

$$f(x) = \sum_{i=1}^{n} \log(1 + \exp(-y_i a_i^T x)) + 0.1||x||_2^2$$

Here, $y_i \in \{-1, 1\}$ and $a_i \in \mathbb{R}^d$. We wish to know how much gradient descent on this function would cost. First, note that (as discussed in a previous lecture) $\log(1 + e^x)$ is β-smooth with $\beta = \frac{1}{4}$. We compose this function with $y_i a_i^T x$ which is smooth with $\beta = ||a_i||^2$. Composing, the summation term of $f(x)$ is smooth with $\beta = \max_i \frac{1}{4}||a_i||$. The regularizer is β-smooth with $\beta = 2$. We assume that the $\frac{1}{4}||a_i||$ term is larger, so $f(x)$ is smooth with $\beta = \max_i \frac{1}{4}||a_i||$. A similar analysis shows that $f(x)$ is 0.2-strongly convex.

We will assume for the sake of simplifying the analysis that for all i, $||a_i||_\infty = O(1)$. That is, all the entries in the matrix A are bounded by some constant. By norm properties, this implies that $||a_i|| = O(d)$, so $\beta = O(d)$. We will also assume that $\Delta_1^2 = O(d)$ (in other words, we are optimizing within the ball of radius $\sqrt{d}$). Since $\lambda = 0.2$, we know that $\lambda = O(1)$. This implies that $\kappa = O(d)$ as well.

Say that we perform $T + 1$ iterations of gradient descent. By our analysis above, this shows that:

$$f(x_{T+1}) - f^* \leq \beta\Delta_1^2 \exp(-T\lambda/\kappa)$$

Using our assumptions above, we get:

$$f(x_{T+1}) - f^* \leq O(d)O(d)\exp(-O(\frac{T}{d}))$$

If we want $f(x_{T+1} - f^*) \leq \epsilon$, we can solve for T to find:

$$T = O(d\log(\frac{d}{\epsilon}))$$

Note that this follows purely from our upper bound on $f(x_{T+1}) - f^*$ in the above equation and basic logarithm properties. As shown above, the cost of each gradient evaluation will be $O(nnz(A))$, where A has columns given by the a_i. Since each $a_i \in \mathbb{R}^d$, $nnz(A) = O(nd)$. Multiplying the two, we get a cost that is $O(nd^2\log(\frac{d}{\epsilon}))$.

In particular, our cost of gradient descent will be at least $O(nd^2)$ in the worst case. This is a *cubic* cost for our algorithm. While this implies that it is doable in polynomial time, cubic complexity is bad when dealing with huge amounts of data. The issue above is that while the convergence of gradient descent is fast, the constants in our convergence rates (such as β) scale with the problem. This leads to an overall cubic cost.

In the example above, however, our function $f(x)$ is a sum over data points. This is often the case in machine learning. Instead of doing a full gradient evaluation at each step (which requires n separate gradients for each of the ℓ_i) we could instead perform each iteration by just doing a local update with respect to one of the ℓ_i. This leads to a variant on gradient descent called *Stochastic Gradient Descent.* As we will see in the next lecture, SGD can be n times faster than gradient descent if the desired error is not too small.