10-725/36-725: Convex Optimization **Fall 2019**

Lecture 24: November 18

Lecturer: Yuanzhi Li *Scribes: Stefani Karp, Chirag Pabbaraju*

24.1 Non-convex Optimization

In convex optimization problems, if the criterion is differentiable, we can follow the direction of the negative gradient; if we arrive at a point with zero gradient, we have found a minimum. In contrast, in non-convex optimization problems, the criterion landscape has many local minima as well as saddle points (points with zero gradient that are not local minima). Naively following the direction of the negative gradient may not lead us to even a local minimum. In fact, in high dimensions, one can construct a function where gradient descent *almost always* gets stuck at a saddle point. The rise of deep learning has increased the motivation to study non-convex optimization, since the loss landscape of these neural networks is typically highly non-convex.

The goals in non-convex optimization are therefore the following:

- Find at least one local minimum
- If possible, find the *global* minimum

The second goal is generally hard, but it can be achieved in certain special settings. In this lecture, however, we focus on how to achieve the first goal *efficiently*. The algorithm we discuss is called Neon2 [AL18].

24.2 Definitions

Given a second-order differentiable function $f : \mathbb{R}^d \to \mathbb{R}$

- L-Lipschitzness: $L = \sup_{x\in\mathbb{R}^d} \|\nabla f(x)\|_2$. This implies that for all $x, y \in \mathbb{R}^d$,

  $$|f(x) - f(y)| \leq L\|x - y\|_2.$$

  In non-convex optimization, though, Lipschitzness usually doesn't matter as much as the function's smoothness matters.

- β-Smoothness: $\beta = \sup_{x\in\mathbb{R}^d} \|\nabla^2 f(x)\|_{sp}$ where $\|\cdot\|_{sp}$ is the spectral norm. Smoothness implies:
  - (Upper quadratic bound) For every $x, y \in \mathbb{R}^d$,

    $$f(y) \leq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\beta}{2}\|y - x\|_2^2.$$

- (Lower quadratic bound) For every $x, y \in \mathbb{R}^d$,

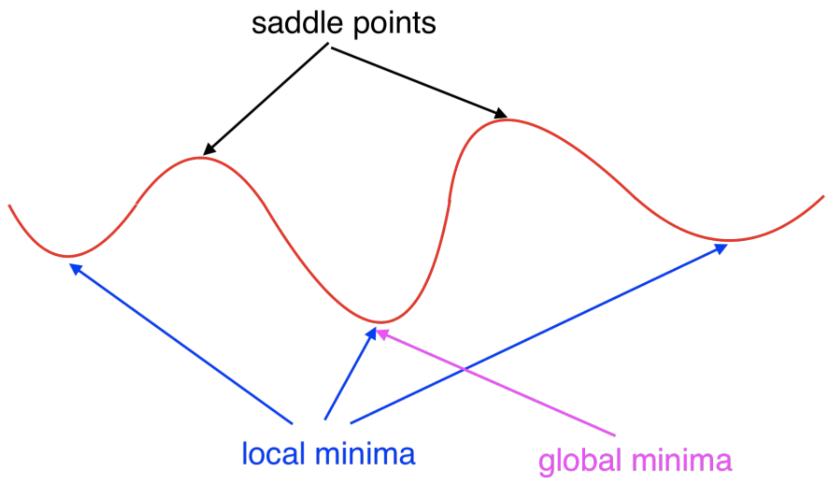
$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle - \frac{\beta}{2}\|y - x\|_2^2.$$

Note: For convex functions, the same upper bound applies, but the lower bound is instead linear: $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle$.

- γ-Lipschitzness of Hessian: For all $x, y \in \mathbb{R}^d$, $\|\nabla^2 f(x) - \nabla^2 f(y)\|_{sp} \leq \gamma \|x - y\|_2$.

  This implies that for all $x, \tau \in \mathbb{R}^d$, the quadratic approximation is tight up to $\pm\gamma\|\tau\|_2^3$. Formally:

$$f(x + \tau) = f(x) + \langle \nabla f(x), \tau \rangle + \frac{1}{2}\tau^T \nabla^2 f(x) \tau \pm \gamma \|\tau\|_2^3$$



- (Second order) Local minima: $\nabla f(x) = 0$ and $\nabla^2 f(x)$ is positive semidefinite - i.e., $\nabla^2 f(x) \succeq 0$.
- Saddle Point: $\nabla f(x) = 0$, $\nabla^2 f(x)$ is not positive semidefinite - i.e., $\exists v \in \mathbb{R}^d$ such that $v^T \nabla^2 f(x) v < 0$.

With the above definitions, our non-convex optimization goal can be formally stated as follows:
Given $f : \mathbb{R}^d \to \mathbb{R}$ that is $\beta-$smooth and has a $\gamma-$Lipschitz Hessian, find a point x in time poly$(\frac{1}{\epsilon}, \frac{1}{\delta}, \gamma, \beta, d)$ such that:

- $\|\nabla f(x)\|_2 \leq \epsilon$ (gradient is approximately 0)
- $\nabla^2 f(x) \succeq -\delta I$ (Hessian is almost PSD)

for all $\epsilon, \delta > 0$.

Can any algorithm achieve this? We now analyze two candidate algorithms, one well-known in the non-convex optimization community but lacking a formal name (hence the affectionately-dubbed name "Algorithm Folklore"). The other is Neon2.

24.3 Approach 1: Algorithm "Folklore"

- Do gradient descent until we arrive at a point x' with $\|\nabla f(x')\|_2 \leq \epsilon$.
  - Recall the quadratic upper bound for β-smooth functions:
    $$f(x - \eta\nabla f(x)) \leq f(x) - \eta\|\nabla f(x)\|_2^2 + \eta^2\beta^2\|\nabla f(x)\|_2^2$$
  - Thus, large gradient $\implies$ decrease function value using gradient descent.
- Check if $\nabla^2 f(x') \succeq -\delta I$.
  - If yes, we are done.
  - If not, find a unit eigenvector v such that $v^T\nabla f(x')v \leq -\delta$. Can be done efficiently via eigenvector solvers.
  - Do one step of Hessian Descent: For a step size η, if $f(x' + \eta v) \leq f(x' - \eta v)$, go to $x'' = x' + \eta v$. Otherwise, go to $x'' = x' - \eta v$.
- Repeat gradient descent.

<u>Notes</u>:

- Since $f(x' + \tau) = f(x') + \langle \nabla f(x'), \tau\rangle + \frac{1}{2}\tau^T\nabla^2 f(x)\tau \pm \gamma\|\tau\|_2^3$,
  $$\begin{aligned}\min(f(x' + \eta v), f(x' - \eta v)) \leq \frac{1}{2}\left(f(x' + \eta v) + f(x' - \eta v)\right) &\leq f(x') + \frac{\eta^2}{2}v^T\nabla^2 f(x')v + \gamma\eta^3 \\ &\leq f(x') - \frac{\eta^2\delta}{2} + \gamma\eta^3\end{aligned}$$
  Taking $\eta = \frac{\delta}{4\gamma}$, the function value is decreased by at least $\frac{\delta^3}{64\gamma^2}$, i.e., $f(x'') \leq f(x') - \frac{\delta^3}{64\gamma^2}$.
- In other words, Hessian descent will decrease the function value by $\Omega(\delta^3)$ when the negative eigenvalue of the Hessian is $\leq -\delta$; i.e., the more non-convex the function is at this point, the more descent we get.
- Suppose f is non-negative and the initial point x^{init} satisfies $f(x^{\text{init}}) \leq 1$.
- Each iteration of gradient descent decreases the function value on the order ϵ^2 and each iteration of Hessian descent decreases the function value on the order δ^3.
- Thus, this algorithm achieves the goals (i.e., $\|\nabla f(x)\|_2 \leq \epsilon$ and $\nabla^2 f(x) \succeq -\delta I$) within a total of $O\left(\frac{1}{\epsilon^2}\right)$ gradient evaluations and $O\left(\frac{1}{\delta^3}\right)$ Hessian eigenvector solver steps.

24.4 Approach 2: Algorithm Neon2

It turns out that we can achieve the goals faster by both *reducing* the number of gradient evaluations and completely *getting rid of* eigenvector solvers. The intuitive workflow to achieve this is via the following loopy argument:

- First reduce the number of gradient evaluations at the cost of increasing the number of Hessian eigenvector solvers (purely for analysis purposes).
- Then reduce the number of Hessian eigenvector solvers at the cost of increasing the number of gradient evaluations.

Inspiration from convex optimization

For a *convex* function f, we can reduce the number of gradient evaluations to find a point x with $\|\nabla f(x)\|_2 \leq \epsilon$ using Nesterov Accelerated Gradient Descent (AGD) [N83]. AGD finds a point x such that $f(x) \leq \min_{y \in \mathbb{R}^d} f(y) + \epsilon^2$ in:

- $O\left(\frac{1}{\epsilon}\right)$ gradient evaluations for any smooth, convex function f (vs. $O\left(\frac{1}{\epsilon^2}\right)$ for GD without acceleration).
- $O\left(\frac{1}{\sqrt{\alpha}} \log \frac{1}{\epsilon}\right)$ gradient evaluations if f is α-strongly convex.

Note: If f is 1-smooth, then $f(x) \leq \min_{y \in \mathbb{R}^d} f(y) + \epsilon^2$ implies $\|\nabla f(x)\|_2 \leq \epsilon$.

So for smooth, convex f, AGD can find a point x with $\|\nabla f(x)\|_2 \leq \epsilon$ in $O\left(\frac{1}{\epsilon}\right)$ iterations, which is faster than GD's $O\left(\frac{1}{\epsilon^2}\right)$ iterations.

Algorithm

Consider 2 different cases –

- The function is *truly* non-convex locally (i.e., $\nabla^2 f(x)$ has at least one very negative eigenvalue): do Hessian descent (which works better/converges very fast when the function is very non-convex).
- The function is not very non-convex locally (i.e., it is approximately convex locally, in the sense that $\nabla^2 f(x)$ has only small negative eigenvalues): can we use AGD?

This yields the following **algorithm**:

1. Find the eigenvector of $\nabla^2 f(x)$ with the most negative eigenvalue. (using 1 Hessian eigenvector solver)
2. If the eigenvalue is sufficiently negative, do one step of Hessian descent.
3. Otherwise, do accelerated gradient descent (AGD) until convergence.
4. Repeat.

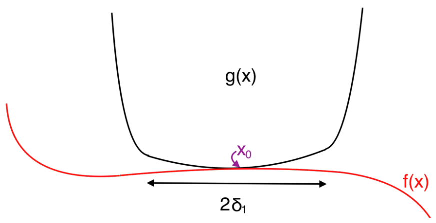
Compared to Algorithm "Folklore", this algorithm has fewer gradient evaluations (Step 3) at the cost of more Hessian eigenvector solvers (Step 1).

Analysis

For simplicity, assume $\beta = \gamma = 1$. Set $\delta_1 = \frac{1}{100}\epsilon^{0.5}$ (threshold for "sufficiently negative" eigenvalues). So if $\nabla^2 f(x_0) \succeq -\delta_1 I$, do AGD. Otherwise, do Hessian descent, which decreases the function value by $\Omega(\delta_1^3) = \Omega(\epsilon^{1.5})$. This means we can do at most $O\left(\frac{1}{\epsilon^{1.5}}\right)$ iterations of Hessian descent.

The trickier part of the analysis is for Step 3 (AGD). Define the δ_1-strongly convex function $g(x)$ as follows:

$$g(x) = f(x) + 4\delta_1 \|x - x_0\|_2^2 + 4 \times \mathbf{1}_{\|x-x_0\|_2 \geq \delta_1} \left(\|x - x_0\|_2 - \delta_1\right)^2$$



Source: [AL18]

Note: It's not true in general that $g(x)$ is minimized at x_0; it's true here due to the shape of $f(x)$ at x_0.

Inside the $2\delta_1$ region around x_0, the function is like a quadratic function with quadratic coefficient δ_1. Outside the $2\delta_1$ region around x_0, the coefficient is a constant (so curvature is much higher).

Thus, $g(x)$ is a δ_1-strongly convex function (globally).

Do AGD on $g(x)$ and find a point x_1 such that:

$$g(x_1) \leq g(x_0) + \epsilon^2$$

and thus

$$\|\nabla g(x_1)\|_2 \leq \epsilon^2$$

in $O\left(\frac{1}{\sqrt{\delta_1}} \log \frac{1}{\epsilon}\right)$ gradient evaluations.

We now consider two cases for x_1.

1) $\|x_1 - x_0\|_2 \geq \delta_1$:

$$\begin{aligned}
f(x_1) &\leq g(x_1) - 4\delta_1\|x_1 - x_0\|_2^2 && \text{by definition of } g(x)\\
&\leq g(x_1) - 4\delta_1^3 && \text{by assumption on } \|x_1 - x_0\|_2\\
&\leq g(x_0) + \epsilon^2 - 4\delta_1^3 && \text{because AGD converges to } x_1\\
&= f(x_0) + \epsilon^2 - 4\delta_1^3 && \text{by definition of } g\\
&= f(x_0) - \Omega(\epsilon^{1.5}) && \text{because } \epsilon^{1.5} \text{ dominates } \epsilon^2 \text{ for } \epsilon < 1
\end{aligned}$$

Thus, $f(x_1) \leq f(x_0) - \Omega(\epsilon^{1.5})$, so the function value decreases. In a handwavy way, this makes sense. At x_0, g and f are equal. However, outside the δ_1 range, g is much larger than f. So if $g(x_1)$ is still ϵ^2-close to $g(x_0)$ (or significantly lower), then it must be the case that $f(x_1)$ is less than $f(x_0)$.

2) $\|x_1 - x_0\|_2 \leq \delta_1$ (intuitively, f is close to g in this region, since the indicator component of g is 0):

By definition of $g(x)$, with the indicator component of g set to 0, we have:

$$\begin{aligned}
&\nabla f(x_1) = \nabla g(x_1) - 8\delta_1(x_1 - x_0)\\
\implies\quad &\|\nabla f(x_1)\|_2 \leq \|\nabla g(x_1)\|_2 + 8\delta_1\|x_1 - x_0\|_2\\
\implies\quad &\|\nabla f(x_1)\|_2 \leq \|\nabla g(x_1)\|_2 + 8\delta_1^2 \leq \epsilon \qquad \text{since } \|x_1 - x_0\|_2 \leq \delta_1
\end{aligned}$$

i.e., the gradient of f at x_1 is small.

Thus, at every step of the algorithm, we need 1 Hessian eigenvector solver and up to $\tilde{O}\left(\frac{1}{\sqrt{\delta_1}}\right) = \tilde{O}\left(\frac{1}{\epsilon^{0.25}}\right)$ gradient evaluations to arrive at one of the following cases: We either reach a point x_1 for which $\|\nabla f(x_1)\|_2 \leq \epsilon$, in which case, we are done. Or, if we don't reach a such a point x_1, we necessarily decrease the function value by at least $\Omega\left(\epsilon^{1.5}\right)$ (either through Step 2 of the algorithm or Part 1 of Step 3). Also, this can only then happen at most $O(\frac{1}{\epsilon^{1.5}})$ times. Thus, there are $O\left(\frac{1}{\epsilon^{1.5}}\right)$ steps (with one Hessian eigenvector solver per step) of $\tilde{O}(\frac{1}{\epsilon^{0.25}})$ gradient evaluations, to account for a final total of $\tilde{O}\left(\frac{1}{\epsilon^{1.75}}\right)$ gradient evaluations before necessarily finding a point x such that $\|\nabla f(x)\|_2 \leq \epsilon$.

Eliminating Hessian eigenvector solvers

At this point, we still require $O(\frac{1}{\epsilon^{1.5}})$ calls to the Hessian eigenvector solver to achieve our overall goal. We can state the goal of the Hessian eigenvector solver as the following (it suffices to allow approximation up to constant factors): Find a unit vector w such that

$$w^T \nabla^2 f(x) w \leq -0.9\delta_1.$$

Can we do this efficiently? In particular, **can we do it within $O(\frac{1}{\epsilon^{0.25}})$ gradient evaluations?**

We recall the power method at this juncture. The power method can give us the most negative eigenvector-value pair in $O(\frac{1}{\delta_1})$ iterations of computing a matrix-vector multiplication. However, this is inefficient, and it turns out that we can further speed this process up to $O(\frac{1}{\sqrt{\delta_1}})$ by the Lanzos method/Chebyshev polynomial methods. With this optimization in place, we are at a point where we can compute the most negative eigenvector-value pair in $O(\frac{1}{\sqrt{\delta_1}})$ iterations of computing a matrix-vector multiplication. The final inefficient piece of computation that then remains is the matrix-vector multiplication, namely $\nabla^2 f(x) z$.

Recalling the definition of the Hessian, we have that

$$\nabla^2 f(x) z = \lim_{\eta \to 0} \frac{\nabla f(x + \eta z) - \nabla f(x)}{\eta}.$$

Thus, we only require 2 evaluations of the gradient, namely at at $x + \eta z$ and x, and provided that η is small, we should be good.

<u>Critical lemma of Neon2</u>: η only needs to be $\frac{1}{\text{poly}(1/\delta_1)}$ small, and the approximation error won't mess up the eigenvector solver (via "stability analysis" of optimization algorithms, which is very hard in general).

Overall, this means we need $O\left(\frac{1}{\epsilon^{1.75}}\right) = O\left(\frac{1}{\delta_1^{3.5}}\right)$ gradient evaluations to replace the Hessian eigenvector solvers.

Conclusion

Neon2 achieves the goal within (ignoring poly(γ, β) factors, which we set to 1 at the beginning):

$$\tilde{O}\left(\frac{1}{\epsilon^{1.75}}\right) + O\left(\frac{1}{\delta^{3.5}}\right) \text{ gradient evaluations of } f$$

24.5 Parting Thoughts

Unlike convex optimization, non-convex optimization is rarely considered as the general optimization problem: $\min f(x)$. Instead, we should leverage the structure/properties of f (e.g., f is a neural network) to optimize f more efficiently. Thus, Neon2 - although it's the most efficient known algorithm (to this date) for general non-convex optimization - only addresses a very, very small fraction of non-convex optimization (i.e., the cases where we know nothing about f that could help us optimize f more efficiently).

References

[AL18] Z. Allen-Zhu and Y. Li, "Neon2: Finding Local Minima via First-Order Oracles," *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 3720–3730.

[N83] Nesterov, Yurii E. "A method for solving the convex programming problem with convergence rate $O(1/k^2)$." Dokl. akad. nauk Sssr. Vol. 269. 1983.