Stochastic, Non-convex and Composite Optimization from the Machine Learning's Perspective

Qinghua Ding
Computer Science Department
Tsinghua University

Abstract

Recently I've been reading quite a lot papers and books in convex optimization (including operator theory) and non-convex optimization. However, I found it hard to keep track of different methods. Each method seem to have its own constraints and results. Here I will sort ALL IMPORTANT methods out, either in convex optimization and non-convex optimization. These methods are critical to our understanding of optimization methodology and will be helpful for designing better algorithms.

Stochastic & Online Convex Optimization

1. Stochastic Gradient Descent

Big data created the problem for computation. Consider the problem of minimizing a function which has the sum-of-functions structure, that is,

$$\min f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x).$$

It's hard to compute an overall gradient $f'(x)$, which would require computing the gradient for n directions and summing them up, especially when n is large and even infinite. Thus we make a compromise, i.e., we construct an unbiased estimator of the overall derivative to save the computational power.

$$\tilde{f}'(x) = f_i'(x)$$

Here i is chosen uniform randomly from $i = 1...n$. And we have the following unbiasedness and bounded variance assumption.

$$E(\tilde{f}'(x)) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) = f'(x)$$

$$var(\tilde{f}'(x)) = \frac{1}{n} f_i(x)^2 - \frac{1}{n^2}(\sum_{i=1}^{n} f_i(x))^2 = \sigma^2 < +\infty$$

Usually, we assume $\sigma = o(1)$. Then the stochastic gradient descent works as follows.

- sample i from $1...m$,
- gradient descent $x_{k+1} = x_k - h_k f_i'(x_k)$.

In face of non-convex functions, the stochastic gradient descent will take $\mathcal{O}(\epsilon^{-4})$ iterations to get to a ϵ-stationary point. And the step size performs optimal when $h_k = \frac{1}{\sqrt{k+1}}$. It's easy to see from the fact that

$$\begin{aligned} f(x_{k+1}) \leq & f(x_k) - \langle f(x_k) | x_{k+1} - x_k \rangle + \frac{L}{2} \parallel x_{k+1} - x_k \parallel^2 \\ \leq & f(x_k) - h_k \langle f(x_k) | \tilde{f}'(x_k) \rangle + \frac{h_k^2 L}{2} \parallel x_{k+1} - x_k \parallel^2 \end{aligned} \tag{1}$$

Taking the expectation and then we have

$$\min_k \mathbb{E} \parallel f'(x_k) \parallel^2 \leq \frac{2(f(x_0) - f^*) + (\sum h_k^2)\sigma^2 L}{2(\sum h_k) - (\sum h_k^2)L}$$

Assume $\sum h_k \to +\infty$, $\sum h_k^2 = \tilde{\mathcal{O}}(1)$, $\sigma = o(1)$, then we have that this sequence always converges and at a rate of $n^{-\frac{1}{2}}$. And this leads to the complexity result.

Moreover, if the function is (strongly) convex, the complexity is $\mathcal{O}(\epsilon^{-2})$ (or $\mathcal{O}(\epsilon^{-1})$). An interesting observation is that, when we take $h_k = \frac{1}{L}$ as in classical gradient descent, we will have

$$\min_k \mathbb{E} \parallel f'(x_k) \parallel^2 \leq \mathcal{O}(n^{-\frac{1}{2}}) + \sigma^2.$$

And the process will never converge, even if $\sigma = o(1)$. Thus we can perceive what stochastic optimization is suffering from essentially - the power of the variance, which could make the algorithm performs worse an order of magnitude.

2. Momentum SGD

The momentum SGD in fact makes use of the momentum method in stochastic gradient descent. Consider the following update rule.

$$\begin{aligned} g_{k+1} = & \tilde{f}'(x_k) \\ m_{k+1} = & \gamma_k m_k + g_{k+1} \\ \Delta x_{k+1} = & x_k - h_k m_{k+1} \end{aligned} \tag{2}$$

The momentum method incoperates the historic information into the gradient, thus enhance the directions which

it has been moving along, while weakening the directions which it has been oscillating along. This method hence makes the SGD perform more robustly and reduce the variance of gradient estimation.

Another form of acceleration (with little changes) is the Nesterov's accelerated gradient as follows.

$$\begin{aligned} g_{k+1} =& \tilde{f}'(x_k - \gamma_k h_k m_k) \\ m_{k+1} =& \gamma_k m_k + g_{k+1} \\ \Delta x_{k+1} =& x_k - h_k m_{k+1} \end{aligned} \tag{3}$$

The performance of this family of method is proved to improve the convergence rate of the SGD only when the function is convex or strongly convex. And this analysis do not extend to non-convex cases.

3. Adaptive Subgradient Descent (AdaGrad)

The AdaGrad tries to solve the per-dimension learning rate problem, since a global learning rate is too coarse to capture the difference between dimensions. We give the AdaGrad algorithm as below.

$$\begin{aligned} g_{k+1} =& \tilde{f}'(x_k) \\ n_{k+1} =& n_k + g_{k+1} \odot g_{k+1} \\ \Delta x_{k+1} =& x_k - h_k g_{k+1} \odot^{-1} (\sqrt{n_{k+1}} + \epsilon) \end{aligned} \tag{4}$$

Note that $\odot$ is per-dimension multiply and $\odot^{-1}$ is per-dimension division. In this way, the AdaGrad tracks the per-dimension information, and temper the learning rate of the dimensions that has already changed a lot, while enhance the others that only have changed mildly.

Although it showed to achieve $\mathcal{O}(\sqrt{T})$ regret bound for online convex optimization, no assertion has been made to the general non-convex settings.

4. RMSProp

Anyway, AdaGrad still suffers from lower learning rate as the learning process goes on. Indeed, the per-dimension accumulative quadratic norm will become quite large in the end, hindering any further improvements the gradient descent hopes to make. Thus a decaying accumulative term is used in RMSProp to solve this problem.

$$\begin{aligned} g_{k+1} =& \tilde{f}'(x_k) \\ n_{k+1} =& \gamma n_k + (1-\gamma) g_{k+1} \odot g_{k+1} \\ \Delta x_{k+1} =& x_k - h_k g_{k+1} \odot^{-1} (\sqrt{n_{k+1}} + \epsilon) \end{aligned} \tag{5}$$

The RMSProp thus makes the learning procedure less hindered, and preserve some agility of the model throughout the process.

5. Adaptive Moment Estimation (Adam)

To take the advantage of both the momentum-based methods and the L_2-norm based methods, Adam progresses the learning procedure as follows.

$$\begin{aligned} g_{k+1} =& \tilde{f}'(x_k) \\ m_{k+1} =& \mu m_k + (1-\mu) g_{k+1} \\ \bar{m}_{k+1} =& \frac{m_{k+1}}{1-\mu^{t+1}} \\ n_{k+1} =& \gamma n_k + (1-\gamma) g_{k+1} \odot g_{k+1} \\ \bar{n}_{k+1} =& \frac{n_{k+1}}{1-\gamma^{t+1}} \\ \Delta x_{k+1} =& x_k - h_k \bar{m}_{k+1} \odot^{-1} (\sqrt{\bar{n}_{k+1}} + \epsilon) \end{aligned} \tag{6}$$

A very interesting thing to notice is the bias-correction step, $\bar{m}_{k+1} = \frac{m_{k+1}}{1-\mu^t}$, which seems not so straight forward. In fact, we have the following for the decaying estimation,

$$\begin{aligned} \mathbb{E}(m_{k+1}) =& \mu \mathbb{E}(m_k) + (1-\mu)\mathbb{E}(g_k) \\ =& (1-\mu) \sum_{i=0}^{k} \mu^i \mathbb{E}(g_i) \\ =& (1-\mu^{k+1}) \mathbb{E}(g) \end{aligned} \tag{7}$$

Note that we are somewhat not serious and take $\mathbb{E}(g_i) = \mathbb{E}(g), \forall i = 1...k$ for analyzing our intuition. However, the bias correction step is then clear itself. The $\mathcal{O}(\sqrt{T})$ regret bound is proved. Anyway, we still know nothing about the convergence rate.

In fact, we can never expect get an convergence rate bound better than SGD from these online optimization methods, which are essentially designed for online convex regret minimization, and do not expect the regret to be non-convex nor the optimization function to be the same over iterations.

6. Stochastic Variance Reduced Gradient Descent (SVRG)

In stochastic gradient descent, we have seen that the convergence rate is greatly influenced by the variance of the gradient oracle $\tilde{f}'(x)$. However, we still seek for better methods to reduce the variance of this oracle. One very useful method widely used in statistics is the variance reduction method.

One straight forward way would be averaging a set of sample gradients, which directly reduces the gradient variance.

$$\tilde{g}(x) = \frac{1}{b} \tilde{f}'(x)$$

It's clear that the variance of $\tilde{f}(x)$ is reduced by b^2, and this method could effectively reduce the variance. But calculations proved that this simple sampling method is not satisfying enough. Since it also causes the calculation multiply by b.

Now we consider another way by constructing a better designed gradient estimator. Consider we take one step to calculate the full gradient information. Then we do not need to calculate the full gradient information any more for the next few iterations.

To clarify our idea, we compute $\mu = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i'(x_0)$ at first, and then we update m times using variance reduced gradient estimator, i.e., we choose $i_t \in \{1, ..., n\}$ for $t = 1, ..., m$, and update x_0 as follows.

$$
\begin{aligned}
g_t =& f'_{i_t}(x_t) - f'_{i_t}(x_0) + \mu \\
x_{t+1} =& x_t - h_t g_t
\end{aligned}
\tag{8}
$$

And intuitively, we have the integrated the previous full gradient to the gradient estimator at current. This utilize the fact that, in the neighborhood of some point, the gradient won't change too much. And when we have no information about some derivative, we may just use the previous gradients to substitute the unknown gradients and descent along these gradients. An intuitive substitute would be

$$
\begin{aligned}
g_t =& \frac{1}{n} f'_{i_t}(x_t) + \frac{1}{n} \sum_{j \neq i_t} f'_j(x_0) \\
=& \frac{1}{n} (f'_{i_t}(x_t) - f'_{i_t}(x_0)) + \mu
\end{aligned}
\tag{9}
$$

However, this estimator could be extremely unbiased, since the renewed direction has only $\frac{1}{n}$ weight. We then magnify this weight in order to make an unbiased gradient estimator. This gives the variance reduced gradient estimator above.

To understand why the variance of the gradient estimator is reduced, we change the representation of the gradient estimator at some point as follows.

$$
\begin{aligned}
g_t =& (f'_{i_t}(x_t) - f'_{i_t}(x^*)) \\
& - (f_{i_t}(x_0) - f_{i_t}(x^*)) \\
& + (f'(x_0) - f'(x^*))
\end{aligned}
\tag{10}
$$

Note that we used the fact that $f'(x^*) = 0$. Using $\| a + b \|^2 \leq 2 \| a \|^2 + 2 \| b \|^2$ and $\mathbb{E}((X - \mathbb{E}(X))^2) = \mathbb{E}(X^2) - \mathbb{E}(X)^2 \leq \mathbb{E}(X^2)$ to bound $\| g_t \|^2$, we have

$$
\begin{aligned}
\| g_t \|^2 \leq & 2\mathbb{E} \| f'_{i_t}(x_t) - f'_{i_t}(x^*) \|^2 \\
& + 2\mathbb{E} \| f'_{i_t}(x_0) - f'_{i_t}(x^*) \|^2 \\
\leq & 4L(f(x_0) - f(x^*) + f(x_0) - f(x^*))
\end{aligned}
\tag{11}
$$

Via further analysis, we can obtain linear convergence rate for strongly convex function.

Non-convex Optimization

In fact, the society of non-convex optimization has been quite mild during the past few decades. However, since 2014, the machine learning society begin to concentrate on the non-convex optimization, since most problems in machine learning research are non-convex. And simple, robust and fast non-convex optimization techniques are really needed.

To illustrate some of the modern approaches of non-convex optimization in the viewpoint of the computer science society, we specially reserved for it this section. And we will discuss some recently proposed methods that solve real problems in practice.

Warmup

Exponential saddle points In this section, we illustrate the arguement of Razvan Pascanu, et. al., about the exponential number of saddle points that could impede the optimization progress. We first consider the information hidden in the Hessian of a stationary point.

- If $\lambda_n(f''(x_0)) > 0$, then it's a local minimum,
- If $\lambda_1(f''(x_0) < 0$, then it's a local maximum,
- Otherwise if $|f''(x_0)| \neq 0$, i.e., then it's a min-max saddle point,
- Finally if the Hessian is degenerate, it's a saddle point or a monkey saddle.

The arguement is based on the conclusion that for generic function chosen from a random Gaussian ensemble of functions, local minima with high error are exponentially rare in the dimensionality of the problem, while saddle points with many negative and approximate plateau directions are exponentially likely at high error. Thus the proliferation of saddle points could be a predominant obstruction to rapid non-convex optimizations.

Local minima can't be too bad Under certain assumptions, it can be proved via spin-glass models, random matrix theory, etc., that for large-size decoupled networks the lowest critical values of the random loss function form a layered structure and they are located in a well-defined band lower-bounded by the global minimum. The number of local minima outside that band diminishes exponentially with the size of the network.

It is also empirically verified that the mathematical model exhibits similar behavior as the computer simulations. And most critical points found there are local minima of high quality measured by the test error. Thus, the main concern for the optimization over a large-size neural network should be finding the local minima, instead of finding the global minima.

Limitation of traditional methods Based on the arguements above, we are concerned about algorithms that converges to local minima, rather than saddle points. This target aligns well with the first-order and second-order conditions in nonlinear optimization.

- first-order condition: $\| f'(x_0) \| \leq \epsilon$,
- second-order condition: $-\lambda_n(f'(x_0)) \leq \delta$.

Moreover, gradient descent, Newton method and trusted region methods are used widely in nonlinear optimization, and we rarely care about wether they get stuck in saddle points. In face of exponential saddle points, we are more concerned about that how these classical methods perform with the abundance of saddle points.

The answer available is pessimistic - to avoid divergence, the step size of the gradient descent won't exceed $\frac{1}{|\lambda_1|}$. And if there is a large discrepancy between the eigenvalues, the gradient descent will have to take very small steps in some directions. This indicates that gradient descent could take quite long time to move away from the saddle points, and it also could take quite long time to get to the local minima.

And it's even worse for the Newton method. The Hessian-gradient product could change the sign gradient along some eigenvector that indicates the negative curvature. This will make the saddle point an attractor for the Newton method, and Newton method will move towards a saddle point rather than escaping from it.

Finally, trusted region methods use damped Hessian to remove negative curvatures. And a damping coefficient multiplied by identity matrix is added to the Hessian, thus making the most negative eigenvalue positive, i.e., $\lambda_n + \alpha > 0$. And this cause the step size in other directions fold by $\frac{\lambda_i}{\lambda_i+\alpha}$. This will also make the steps too small, and hence impedes the convergence.

To this end, we make it clear the needs for new non-convex optimization algorithms that move away from saddle points and make progresses fast enough.

I.1 Generalized Trust Region [convergence proof?]

The first tryout in the computer science literature to handle this problem is the design of generalized trusted region methods, and we'll discuss the ideas below.

In order to make the Newton method more robust, we consider how to preserve the signs of the gradient while making steps that is secure. A somewhat innovative method is to move the second-order term in the target to the constraints. We consider the generalized trust region method as follows.

$$\begin{aligned} s &= \arg\min_s m_k\{f, x_k, s\} \\ &\text{s. t. } d(x_k, x_k + s) \leq \Delta \end{aligned} \tag{12}$$

Notice that we make the approximation $m_k(x)$ first-order, thus the gradient is guaranteed, and we then make the constraint second-order, which measures the accuracy of approximation.

$$\frac{1}{2}|s^T H s| \leq \Delta$$

This is better than gradient descent in the sense that it take steps adaptively. And the emperical results show that this method is superior to gradient descent and Newton methods. However, no convergence guarantee or convergence rate analysis has been made to this method.

I.2 Noisy Gradient Descent

The generalized trust region method does not have good analytical property, and hence we turn to find algorithms that have robust theoretical guarantee for non-convex optimization. In fact, it could be hard to handle the degenerate case, and we first consider how we can overcome the problem of strict saddles.

Definition 1. (*strict saddle*) A twice differentiable function f satisfies strict saddle property if all its local minima have $f''(x) > 0$, and all its other stationary points satisfy $\lambda_n(f''(x)) < 0$.

And it's been verified that many other problems, e.g., SVD, fourth-order tensor decomposition, etc., all have this property. And we make use of a robust version of this strict saddle property.

Definition 2. A twice differentiable $f(x)$ is $(\alpha, \gamma, \epsilon, \delta)$-*strict saddle*, if for any point x at least one of the following is true

- $\| f'(x) \| \geq \epsilon$,
- $\lambda_n(f''(x)) \leq -\gamma$,
- there is a local minimum x^* thus $\| x - x^* \| \leq \delta$, and $f(x)$ restricted to 2δ neighborhood of w^* is α-strongly convex.

Essentially, this condition says for any point whose gradient is small, it is wither close to a robust local minimum, or is a saddle point with a significant negative eigenvalue. Now we consider the technical issues. In fact, for case 1, the gradient descent is sufficient for decreasing till this condition is broken. For case 3, we can easily verify the local convergence rate of gradient descent. The most tricky thing is the second case, where we've encountered some saddle point.

Since there exists negative curvature, we still hope that we can find this curvature and use this direction, but how? One seemingly brute solution is to add noise to the translation at each iteration. Due to the random noise, there will be cases where the point jumps out of the saddle region, and flee away via the negative curvature.

$$x_{k+1} = x_k + h_k f'(x_k) + \xi$$

Here ξ is a random guassian noise on unit sphere. The assumption of this method is quite strong, i.e., $f(x)$ is bounded by $|f(x)| \leq B$, and is β-smooth and has ρ-Lipschitz Hessian. Moreover, it has to satisfy strict saddle property in Def2. We then have the following theorem about its convergence.

Theorem 1. $\forall \zeta > 0, \eta \leq \mathcal{O}(\frac{1}{\log(\eta^{-1})})$, with prob. $1 - \zeta$, noisy gradient descent ouputs a point that is $\tilde{\mathcal{O}}(\sqrt{\eta \log(\eta^{-1}\zeta^{-1})})$ close to some local minimum in $t = \tilde{\mathcal{O}}(\eta^{-2}\log(\zeta^{-1}))$ iterations.

I.3 Third Order Optimization

In fact, this algorithm follows the idea of cubic regularization to search for higher order optimality. The novelty of this work lies in its generalization of this group of methods. In fact, we have the following extension of the notion of p-th order optima.

Definition 3. (*higher order optima*) A point is a p-th order local minimum if for any nearby point y, $f(x) - f(y) \leq o(\| x - y \|^p)$.

Note that in order to attain higher order optima, we have to bound higher order continuity as well.

$$\| f^{(3)}(x) - f^{(3)}(y) \|_F \leq R \| x - y \|$$

Also we have to guarantee second order Lipschitz continuity, i.e., $\| f''(x) - f''(y) \| \leq L \| x - y \|$. And we have the third-order necessary condition as follows.

- $f'(x) = 0$, $f''(x) > 0$,
- for any u satisfying $u^T f''(x) u = 0$, we have $[f^{(3)}(x)](u, u, u) = 0$.

The convergence rate on first order and second order is $\mathcal{O}(\epsilon^{-\frac{3}{2}})$ and $\mathcal{O}(\epsilon^{-3})$, which matches the cubic regularization, with a special term indicating the third order condition. The rest of the work is just old stories in optimization theory. Another result that's worth noticing is that fourth order or higher order optima is NP hard to obtain.

I.4 Gradient Descent

We still list gradient descent here mainly for illustrating how it performs in fact of saddle points.

Theorem 2. Gradient Descent converges to local minima for strict saddle functions, but it could take exponential time.

The analysis of convergence is based on the Stable Manifold Theorem in dynamic system. And the complexity result is derived by designing a octopus-like function which have a great many saddle points hindering the progress of the gradient descent optimizer, leading to exponential time for overcoming the obstacles.

I.5 AGD-guilty

The accelerated gradient descent has been used in convex optimization and it improves the convergence rate of gradient descent by an order of magnitude. However, it is not clear how we can use this method in the non-convex setting. In fact, the momentum is originally designed for the convex case, inspired by a heavy ball falling down a hill. However, we are not sure wether this trick can be extended to the non-convex case.

Anyway, the AGD-guilty has a very simple idea - we will keep using AGD, until the convexity assumption is broken. But if it's broken, we can still make use of the non-convexity found to move along the negative curvature. We illustrate the two cases.

- there is no guilt made, then we can utilize the strong convexity arguement to prove better convergence than stochastic gradient descent,
- some guilt is made, then we make use of this guilt to improve along the negative curvature.

The idea is simple, however, developing theoretical analysis can be quite complex. We skip the lengthy analysis and give the main result. AGD-guilty finds an approximate stationary point only using gradient oracle, but still require first and second order continuity, in $\mathcal{O}(\epsilon^{-\frac{7}{4}})$ iterations.

Still adding more ...