Natasha 2: Faster Non-Convex Optimization Than SGD – How to Swing By Saddle Points (Literature Review)

Charlie Hou

May 8, 2018

1 Abstract

This is a report for the paper [All17]. This, along with the project I did with Hrishikesh Khandeparkar and Gene Li on the effect of depth on training speed, is my final project for the course. This report serves as a summary and contains a few thoughts on [All17].

2 Introduction

We know that a lot of different optimization problems are non-convex, not least of which include optimizing neural networks. For the most part, we use SGD to train these neural networks, and it works very well. There exists a lot of literature out there that explains why SGD works so well: for example, it's been shown that SGD has implicit regularization built into it, and that it tends to find solutions with good generalization ability. However, it still remains to be seen whether SGD actually is the best algorithm for non-convex optimization, as it doesn't really use information that could be available to us about the non-convex optimization problem. With this in mind, [All17] introduces the Natasha 2 algorithm, which is an optimization algorithm for non-convex online optimization.

There exists heuristics out there that can turn local minima into global minima. In addition, for a lot of applications, local minima is good enough. So the objective here is to avoid saddle points. The usual way to escape saddle points is to use random perturbation. This paradigm has been very successful, and random perturbation has been shown to provably escape from saddle points in a reasonable amount of time. However, this way of getting away from saddle points somehow feels a little unsatisfactory. We want to be able to use more information to make a more educated guess on which way to escape instead of choosing a random direction. What this paper does is introduce an algorithm that efficiently uses Hessian information to escape from saddle points.

3 The Idea

Ideally, what you would do at a saddle point is take the negative eigenvector of the hessian $\nabla^2 f(x)$ and go in that direction. This is illustrated in figure 1. However, calculating this is really intensive. So what we do instead is use Oja's algorithm, which is an an online approximation of the power method, which gives the most negative eigenvector of a matrix. Oja's algorithm at each step is as cheap as a gradient computation.

So what we have, loosely speaking, is an algorithm that has two paths it can go down at each step. If Oja's algorithm can find a negative enough curvature direction, we take that direction. If it cannot, then we proceed as if there is no saddle point nearby, and this can be done with an algorithm that is very similar to SGD which [All17] calls Natasha 1.5. The reason we do not use SGD exactly is

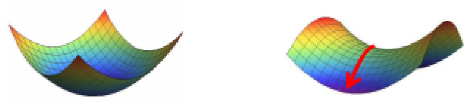
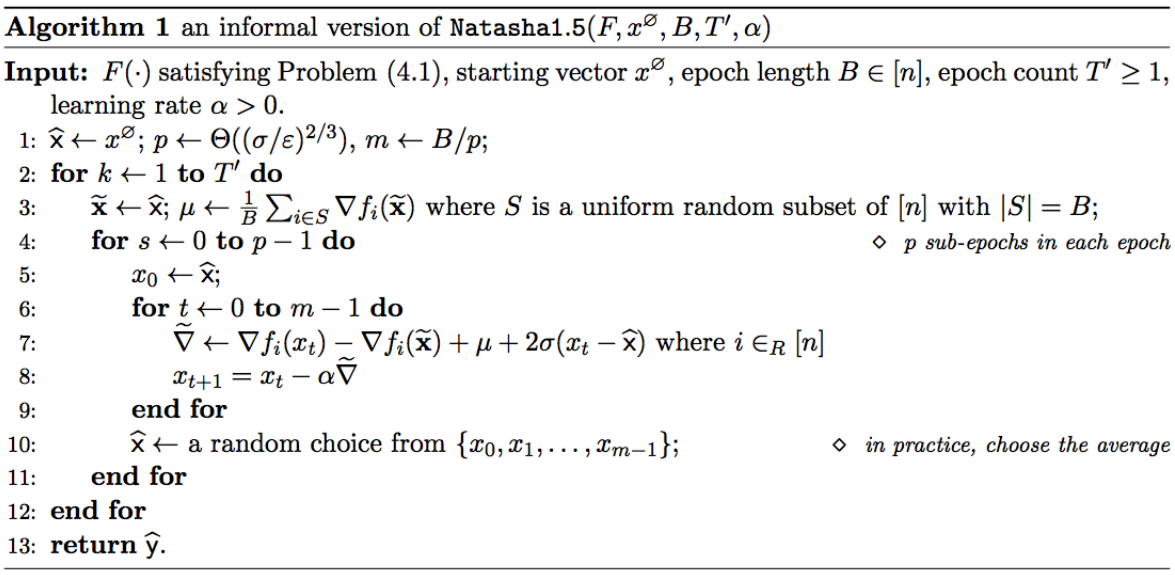


Figure 1: On the left is a local minimum, and the right is a saddle point. Note that at a saddle point, there is always a direction of negative curvature, under mild regularity conditions.



```
Algorithm 1 an informal version of Natasha1.5($F, x^{\varnothing}, B, T', \alpha$)
Input: $F(\cdot)$ satisfying Problem (4.1), starting vector $x^{\varnothing}$, epoch length $B \in [n]$, epoch count $T' \geq 1$,
   learning rate $\alpha > 0$.
 1: $\widehat{\mathsf{x}} \leftarrow x^{\varnothing}$; $p \leftarrow \Theta((\sigma/\varepsilon)^{2/3})$, $m \leftarrow B/p$;
 2: for $k \leftarrow 1$ to $T'$ do
 3:     $\widetilde{\mathbf{x}} \leftarrow \widehat{\mathsf{x}}$; $\mu \leftarrow \frac{1}{B}\sum_{i \in S} \nabla f_i(\widetilde{\mathsf{x}})$ where $S$ is a uniform random subset of $[n]$ with $|S| = B$;
 4:     for $s \leftarrow 0$ to $p - 1$ do                                   ◇ p sub-epochs in each epoch
 5:         $x_0 \leftarrow \widehat{\mathsf{x}}$;
 6:         for $t \leftarrow 0$ to $m - 1$ do
 7:             $\widetilde{\nabla} \leftarrow \nabla f_i(x_t) - \nabla f_i(\widetilde{\mathsf{x}}) + \mu + 2\sigma(x_t - \widehat{\mathsf{x}})$ where $i \in_R [n]$
 8:             $x_{t+1} = x_t - \alpha\widetilde{\nabla}$
 9:         end for
10:         $\widehat{\mathsf{x}} \leftarrow$ a random choice from $\{x_0, x_1, \ldots, x_{m-1}\}$;        ◇ in practice, choose the average
11:     end for
12: end for
13: return $\widehat{\mathsf{y}}$.
```

Figure 2

because we are trying to solve the online optimization problem, which can introduce some issues with the convergence analysis, and because we want to use some knowledge about "how non-convex" the problem is.

Definition 3.1 (σ-strongly non-convex).
f is σ-strongly non-convex if all the eigenvalues of $\nabla^2 f > -\sigma$

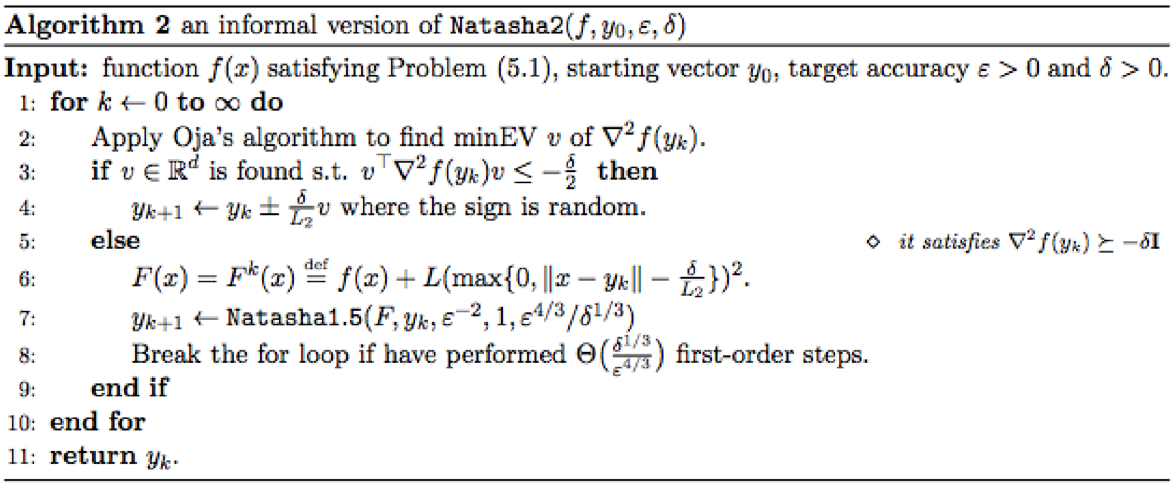
We want to use the parameter σ to inform our algorithm, so that it can use it to perform better than algorithms that don't use it.

4 The Algorithm

4.1 Natasha 1.5

In figure 2, we show the algorithm for Natasha 1.5. First note how it only sees a random B number of points, with which it must perform its gradient updates. Now we go through and note all the interesting things about this algorithm.

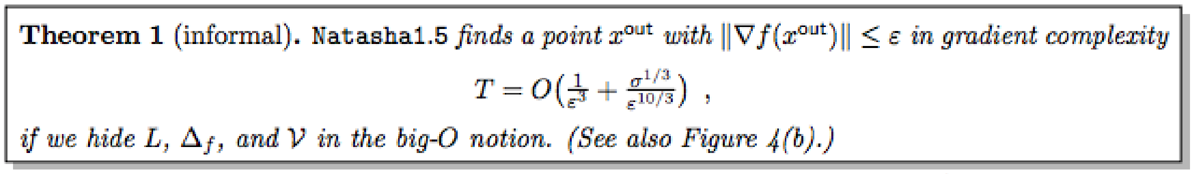
First note that we use a strange way to calculate our gradient, on line 7. The first thing that is strange about it is the $2\sigma(x_t - \hat{x})$. This is there because we are not trying to calculate the gradient of f directly but rather the gradient of $f + \sigma\|x - \hat{x}\|^2$ The reason we do this is to make the objective



Algorithm 2 an informal version of $\texttt{Natasha2}(f, y_0, \varepsilon, \delta)$

```
Input: function $f(x)$ satisfying Problem (5.1), starting vector $y_0$, target accuracy $\varepsilon > 0$ and $\delta > 0$.
 1: for $k \leftarrow 0$ to $\infty$ do
 2:     Apply Oja's algorithm to find minEV $v$ of $\nabla^2 f(y_k)$.
 3:     if $v \in \mathbb{R}^d$ is found s.t. $v^\top \nabla^2 f(y_k) v \leq -\frac{\delta}{2}$ then
 4:         $y_{k+1} \leftarrow y_k \pm \frac{\delta}{L_2} v$ where the sign is random.
 5:     else                                              ◇ it satisfies $\nabla^2 f(y_k) \succeq -\delta \mathbf{I}$
 6:         $F(x) = F^k(x) \overset{\text{def}}{=} f(x) + L(\max\{0, \|x - y_k\| - \frac{\delta}{L_2}\})^2$.
 7:         $y_{k+1} \leftarrow \texttt{Natasha1.5}(F, y_k, \varepsilon^{-2}, 1, \varepsilon^{4/3}/\delta^{1/3})$
 8:         Break the for loop if have performed $\Theta(\frac{\delta^{1/3}}{\varepsilon^{4/3}})$ first-order steps.
 9:     end if
10: end for
11: return $y_k$.
```

Figure 3



Theorem 1 (informal). $\texttt{Natasha1.5}$ *finds a point* x^{out} *with* $\|\nabla f(x^{\text{out}})\| \leq \varepsilon$ *in gradient complexity*

$$T = O\left(\frac{1}{\varepsilon^3} + \frac{\sigma^{1/3}}{\varepsilon^{10/3}}\right) ,$$

if we hide L, Δ_f, *and* $\mathcal{V}$ *in the big-O notion. (See also Figure 4(b).)*

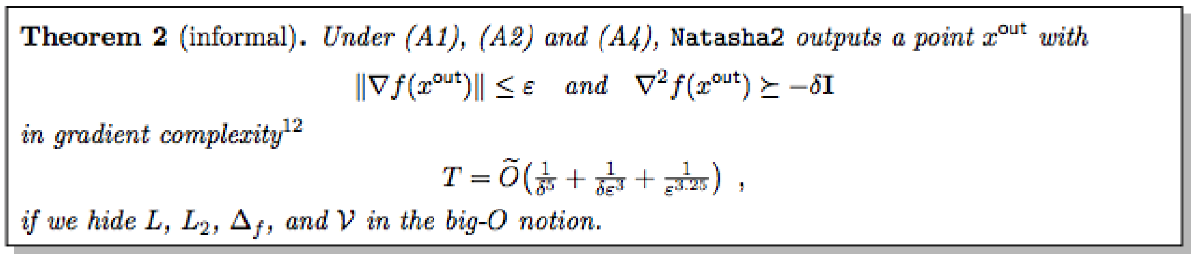
strongly convex, which allows us to use convex optimization techniques.

Next note that we have two extra terms in the middle, which we might think does not really do anything, as in expectation they simply cancel out, which means that the entire expression in expectation is exactly the gradient we want to approximate. In other words, it does what we want but in the roundabout way. This update comes from SVRG, which is a good non-convex optimization algorithm, and it is in there to make the math work out.

Finally, see at line 10 that we make a random choice from the previous iterates for $\hat{x}$. The reason this is here is to allow us to bound the approximation error between $\tilde{\nabla}$ and $\nabla^2 f$, because this error is dominated by $\|x - \hat{x}\|$.

4.2 Natasha 2

In figure 3, we show the algorithm for Natasha 2. Note that we start with using Oja's algorithm to compute the minimum eigenvalue of the Hessian, and use it to take negative curvature direction if it is there, and to use Natasha 1.5 if it is not. Take a look at line 6. This part looks a little strange. We optimize this relaxation of the function because (1) it satisfies nice properties about smoothness and convexity (2) using y_k because if we can make sure $\|x - y_k\|$ is small, then $\|x - y_k\| - \frac{\delta}{L_2}$ is small, which would imply that if $\nabla F < \epsilon$, then $\nabla f < \epsilon$. Essentially, using previous iterates ensures us that minimizing the "nice" approximation of f will minimize f as well. All of these things are necessary for [All17] to show that their algorithm converges asymptotically faster than SGD does. They prove the following two things in figures 4 and 5.



Theorem 2 (informal). *Under (A1), (A2) and (A4),* `Natasha2` *outputs a point* x^{out} *with*

$$\|\nabla f(x^{\mathsf{out}})\| \leq \varepsilon \quad \textit{and} \quad \nabla^2 f(x^{\mathsf{out}}) \succeq -\delta \mathbf{I}$$

in gradient complexity[12]

$$T = \widetilde{O}\big(\tfrac{1}{\delta^5} + \tfrac{1}{\delta\varepsilon^3} + \tfrac{1}{\varepsilon^{3.25}}\big) \ ,$$

if we hide L, L_2, Δ_f, *and* $\mathcal{V}$ *in the big-O notion.*

5 Conclusion

This paper contains some interesting ideas about using the non-convexity parameter σ to design an algorithm. However, what would be really helpful to understand this paper would be some numerical simulations using its algorithm. As shown above, the algorithm is a little cumbersome to write (especially since this is online optimization) and one has to deal with a lot of little details which are in the paper. [All17] obscures a lot of the constants using the big-O notation, and it would be instructive to investigate the effects of these constants on the actual running speed. It may be provably asymptotically faster than SGD, but is it empirically?

References

[All17] Z. Allen-Zhu. "Natasha 2: Faster Non-Convex Optimization Than SGD". In: *ArXiv e-prints* (Aug. 2017). arXiv: `1708.08694 [math.OC]` (1, 3, 4).