# A survey on gradient-based methods: Escaping from saddle points

Hongfan Chen
chenhf@umich.edu

Tunan Wang
wtunan@umich.edu

## 1 Introduction

Non-convex function can be much more complicated than convex functions, as the latter usually have only one local minimum. In non-convex problems, traditional gradient-based methods, for example vanilla gradient descent, only guarantees convergence to *first-order stationary point* ($\nabla f = 0$) in $O(\frac{1}{\epsilon^2})$ iterations, but usually those points can be saddle points, local minimum, global minimum and even maximum. Indeed in practice, optimization problems on architectures such as deep neural networks often involve first-order stationary solutions that turned out to be saddle points. Therefore, a more reasonable goal of optimization may be finding *second-order stationary point*($\nabla f = 0$ and $\nabla^2 f \succeq 0$), which rules out many common lower-order saddle points and allows only local minima and higher-order saddle points.

In this project, we surveyed and implemented some of the newly developed methods that focus on convergence to a *second-order stationary point*. We compared those methods to simple gradient method, which also has a chance to escape from saddle points but sometimes may get stuck. According to Ge et al. [3], simple gradient descent method can only be trapped by strict saddle points with starting point in a slim area, but even so the convergence behaviour is questionable, and some newly proposed methods have been proved to have better convergence rate.

Here, we will mainly focus on perturbed gradient decent(PGD), perturbed accelerated gradient descent(PAGD), and their improved versions in the latest work.

Below is a table of convergence behaviours of all the methods we mentioned. The improvement of FPGD and FPAGD occurs at the point that they are more robust against data dimensions, i.e. iterations are depends on $\mathbf{\log n}$ rather than $\mathbf{\log^4 n}$ or $\mathbf{\log^6 n}$, where $n$ is the dimension of the data.

| **Method** | **Oracle** | **Iterations** | **Stationary** |
|---|---|---|---|
| GD | Gradient | $O(1/\epsilon^2)$ | first-order |
| AGD | Gradient | $\tilde{O}(1/\epsilon^2)$ | first-order |
| PGD | Gradient | $\tilde{O}(\log^4 n/\epsilon^2)$ | second-order |
| PAGD | Gradient | $\tilde{O}(\log^6 n/\epsilon^{1.75})$ | second-order |
| FPGD | Gradient | $\tilde{O}(\log n/\epsilon^{1.75})$ | second-order |
| FPAGD | Gradient | $\tilde{O}(\log n/\epsilon^{1.75})$ | second-order |

**Table 1:** Convergence behaviours of all the methods mentioned

## 2 Preliminaries

In this section, we listed several notations and definitions that will be used in the following discussion.

**Definition 1.** *A differentiable function $f(\cdot)$ is called* ***$\mathcal{L}$-smooth*** *if:*

$$\|\nabla f(x_1) - \nabla f(x_2)\| \leq l\|x_1 - x_2\|, \quad \forall x_1, x_2$$

**Definition 2.** *$x$ is an $\epsilon$-**first-order stationary point** of function $f(\cdot)$ if $\|\nabla f(x_t)\| \leq \epsilon$*

**Definition 3.** *A twice-differentiable function $f(\cdot)$ is called $\rho$-**Hessian Lipschitz** if:*

$$\|\nabla^2 f(x_1) - \nabla^2 f(x_2)\| \leq \rho\|x_1 - x_2\|, \quad \forall x_1, x_2$$

**Definition 4.** *For a $\rho$-**Hessian Lipschitz** function $f(\cdot)$, $x$ is an $\epsilon$-**second-order stationary point** if:*

$$\|\nabla f(x_t)\| \leq \epsilon \quad \textit{and} \quad \lambda_{min}\left(\nabla^2 f(x)\right) \geq -\sqrt{\rho\epsilon}$$

**Definition 5.** *A twice-differentiable function $f(\cdot)$ is called $\mu$-**strongly convex** if:*

$$\nabla^2 f \succeq \mu I_d$$

*or equivalent,*

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

## 3 Algorithms

The first algorithm we are going to overview is the vanilla gradient descent. To minimize the function $f : R^n \to R$, the most popular way is to follow the opposite direction of gradient $\nabla f$, which is easy and straightforward to implement. We update the parameter by

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

where $\eta$ is the step size and $x_t$ is the current iterate.

As long as the gradient $\nabla f$ is nonzero, following the negative gradient direction always leads us to local progress as we move towards smaller function value in a narrow region. However, when $\nabla f$ is equal to zero, we reach a critical point, where gradient descent algorithms seems to get stuck. For a strongly convex function this is good enough, as this point is also the global minimum. However unfortunately, in the non-convex setting, a mere guarantee of convergence to first-order stationary point is not satisfactory, as the point may possibly be saddle point in this case. Thus there is limitation in vanilla gradient descent method.

**Algorithm 1** Perturbed Gradient Descent (PGD) [6]

**Require:** $x_0$, step size $\eta$, perturbation radius $r$, time interval $\mathscr{T}$, tolerance $\epsilon$.

$t_{\text{perturb}} \leftarrow 0$
**for** $t = 0, 1, \ldots, T$ **do**
  **if** $\|\nabla f(x_t)\| \leq \epsilon$ and $t - t_{\text{perturb}} > \mathscr{T}$ **then**
    $x_t \leftarrow x_t - \eta\xi_t \quad \xi_t \sim \text{Uniform}(\mathbb{B}_0(r))$
    $t_{\text{perturb}} \leftarrow t$
  **end if**
  $x_{t+1} \leftarrow x_t - \eta\nabla f(x_t)$
**end for**

The perturbed Gradient descent method is designed based on the vanilla gradient descent, but an additional step is introduced when the algorithm reaches the **$\epsilon$-first-order stationary point**. The intuition here is that when we are about to get stuck, we add a perturbation to the current iterate. More specifically, the perturbation is generated from a uniform open ball with radius $r$. As in Jin et al. (2017)[3], they proved that if we divide the ball into "stuck region", starting from which gradient descent does not escape the saddle, and its complement, from which GD escape quickly, then we can show that the volume of the "stuck region" is reasonably small when compared to its complement. So adding an perturbation will help escape the saddle point with large probability, but what is also ideal is that according to Ge et al.[2015], perturbed GD in fact finds second-order stationary points as fast as vanilla GD finds first-order stationary point and only up to a logarithmic factors in dimension. Therefore, perturbed GD has certain advantage over vanilla GD. Besides, setting a threshold value for the frequency of perturbation also help control the variation in the descent process.

**Algorithm 2** Perturbed Accelerated Gradient Descent(PAGD)

**Require:** $x_0$, step size $\eta$, iterative parameter $\theta$, negative curvature parameter $\gamma$, momentum upper bound $s$, perturbation radius $r$, time interval $\mathscr{T}$, tolerance $\epsilon$.

$v_0 \leftarrow 0$
**for** $t = 0, 1, \ldots, T$ **do**
  **if** $\|\nabla f(x_t)\| \leq \epsilon$ and *no perturbation in last* $\mathscr{T}$ *steps* **then**
    $x_t \leftarrow x_t + \xi_t \quad \xi_t \sim \text{Uniform}(\mathbb{B}_0(r))$
  **end if**
  $y_t \leftarrow x_t + (1-\theta)v_t$
  $x_{t+1} \leftarrow y_t - \eta\nabla f(y_t)$
  $v_{t+1} \leftarrow x_{t+1} - x_t$
  **if** $f(x_{t+1}) \leq f(y_{t+1}) + \langle \nabla f(y_{t+1}), x_{t+1} - y_{t+1}\rangle - \frac{\gamma}{2}\|x_{t+1} - y_{t+1}\|^2$ **then**
    $(x_{t+1}, v_{t+1}) \leftarrow \text{Negative-Curvature-Exploitation}(x_{t+1},\, v_{t+1},\, s)$
  **end if**
**end for**

Just like Nesterov's standard accelerated gradient descent method, the Perturbed accelerated gradient descent also falls into the category of momentum-based methods. The standard accelerated method is considered to be faster than gradient descent(Nesterov[1983]) in the convex case. However, as in Jin et al.(2017)[6], they showed that in the presence of saddle points, the momentum-

based methods also yield faster convergence than gradient descent. The algorithm they proposed, as showed in algorithm 2, takes the advantage of momentum and can be viewed as an improved version of Nesterov's AGD method. The improvement consists of two parts: Perturbation parts(lines 3-4) and Negative curvature exploitation(NCE, lines 9-10, also Algorithm 4 in appendix). The intuition of perturbation here is the same as in algorithm 1, while NCE part tells that, if the function is "too non-convex" in the current direction, then we abort it by resetting the momentum to zero vector and then decide whether to exploit negative curvature depending on the magnitude of the current momentum.

**Algorithm 3** Faster Perturbed Gradient Descent (FPGD) [6]

**Require:** $x_0$, step size $\eta$, perturbation radius $r$, time interval $\mathscr{T}$, tolerance $\epsilon$, sub-iteration's step size $\eta_{sub}$.

1: **for** $t = 0, 1, \ldots, T$ **do**
2:  **if** $\|\nabla f(x_t)\| \leq \epsilon$ **then**
3:  $e \leftarrow$ Negative-Curvature-Finding$(x_t, r, \mathscr{T})$
4:  $x_t \leftarrow x_t - \eta_{\text{sub}} e$
5:  **end if**
6:  $x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$
7: **end for**

Having the same overall structure as PGD, FPGD improves the performance of PGD using the idea that, in a small neighborhood of a saddle point, gradients can be viewed as Hessian-vector products plus some bounded deviation. By the gradients' approximation of Hessian-vector products, the algorithm performs better than the previous methods theoretically.

Rather than try directions randomly, FPGD chooses to iterate to the good direction first, and then take the step.

The approximation of Hessian-vector can be illustrated as:

$$h_f(x) \coloneqq f(x) - \langle \nabla f(0), x \rangle$$

Then, with $\rho$**-Hessian Lipschitz**,

$$\|\nabla^2 f(\xi x) - \nabla^2 f(0)\| \leq \rho \|x\| \Rightarrow \|\nabla h_f(x) - \nabla^2 f(0) \cdot x\| \leq \rho \|x\|^2$$

Note that, $\nabla^2 h_f(x) = \nabla^2 f(x)$.

FPAGD below gives a "improved" version of PAGD. The basic idea of FPAGD is also the Hessian-vector's approximation. Rather than always try to seek the negative curvature exploitation step, FPAGD separate the direction searching and filtering apart just like FPGD.

# 4 Numerical Simulation

In this section, we implemented algorithms and performed numerical simulation on a simple function,

$$f(x_1, x_2) = \frac{1}{2} x_1^2 + \frac{1}{2} sin(3x_1) + x_2^2$$

which has one local minimum, one global minimum, and one saddle point.

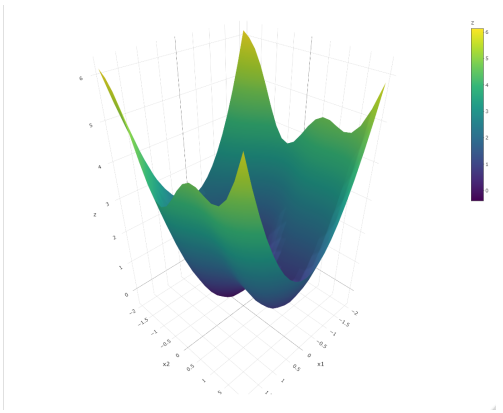

**Figure 1:** 3D plot

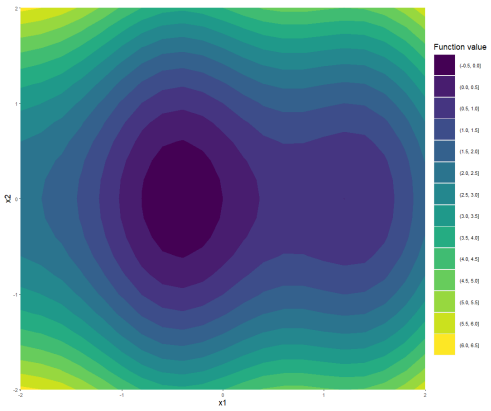


**Figure 2:** Contour plot

Figure 1 and 2 give us the image of simulation function. And it is easy to find that all the local minimums and saddle points are located along $x_2 = 0$.

Since perturbed methods are designed to overcome the influence of saddle points, we set the initial point at $(0.6806, 0)$, which is very close to the saddle point with gradient less than $10^{-4}$.

Some important initial values should be pointed out:

**Step size:** $\eta = 0.1$ **Time interval:** $\mathscr{T} = 1$ **Epoch:** $T = 36$

Here, we compare all the algorithms with the same number of gradient evaluations, i.e. the number of calling gradient calculation function. When $\mathscr{T} = 1$, the sub-loop in FPGD only take 1 step, which makes the number of gradient evaluations in one loop to be 2. All the other algorithms suffer the same number of evaluations in one loop, hence, we can say that the comparison is fair.

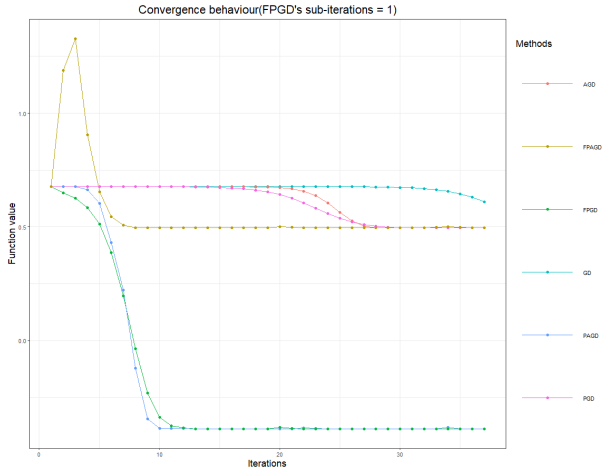


**Figure 3:** Performance of algorithms

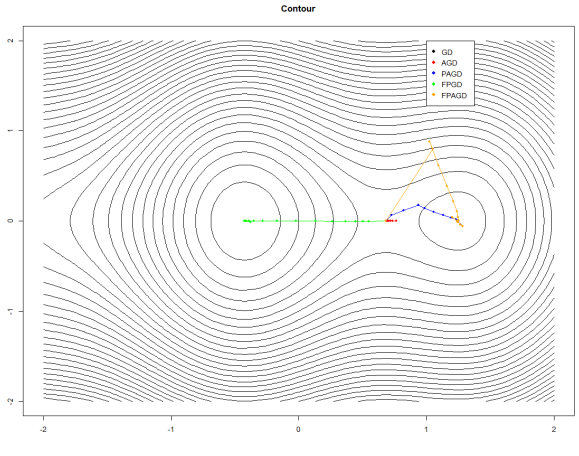


**Figure 4:** Trace of different methods

Figure 3 and 4 show the convergence behaviour of different methods. From both figures, we can see that while GD still did not converge after 36 epochs, the other algorithms all managed to converge to local minimums. However, the convergence speeds are different:

$$\text{GD} < \text{AGD} \approx \text{PGD} < \text{FAGD} \approx \text{PAGD} \approx \text{FPAGD}$$

This result meets our expectation from table 1. Note that, since the dimension of simulation data is low and the curvature of the function is rather simple, those algorithms tend to have similar behaviors here, but the results may differ if they were applied in a more complicated setting.

## 5 Software

All the work combined together were implemented in out new R package PertGD, which is distributed via Github(R-package). Currently our package only supports objective function that has explicit analytical form for both gradient and function itself, but we may extend the availability to general functions later.

## 6 Contributions

- **Hongfan Chen**: I set up the general framework of implementation, implemented and commented on algorithm 1, 3, 6 and part of algorithm 2. I also help design the numerical simulation, and help knit up the final write-up and R package.
- **Tunan Wang**: I organized the topics of papers, implemented and commented on algorithm 4, 5, 7 and part of algorithm2. I also help implement the numerical simulation, and contributed to the final write-up and R packages.

# References


[1] Ge R, Huang F, Jin C, et al. Escaping from saddle points—online stochastic gradient for tensor decomposition[C]. Conference on learning theory. PMLR, 2015: 797-842.

[2] Lee J D, Simchowitz M, Jordan M I, et al. Gradient descent only converges to minimizers[C]. Conference on learning theory. PMLR, 2016: 1246-1257.

[3] Jin C, Ge R, Netrapalli P, et al. How to escape saddle points efficiently[C]. International Conference on Machine Learning. PMLR, 2017: 1724-1732.

[4] Jin C, Netrapalli P, Jordan M I. Accelerated gradient descent escapes saddle points faster than gradient descent[C]. Conference On Learning Theory. PMLR, 2018: 1042-1085.

[5] Chenyi Zhang and Tongyang Li. Escape saddle points by a simple gradient-descent based algorithm. Advances in Neural Information Processing Systems. 2021.

[6] Jin C, Netrapalli P, Ge R, et al. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points[J]. Journal of the ACM (JACM), 2021, 68(2): 1-29.


# Appendix

**Algorithm 4** Negative Curvature Exploitation

**Require:** $x_t$, momentum $v_t$, momentum lower-bound $s$.

**if** $\|v_t\| \geq s$ **then**
  $x_{t+1} \leftarrow x_t$
**else**
  $\delta \leftarrow s \cdot v_t / \|v_t\|$
  $x_{t+1} \leftarrow \underset{x \in \{x_t+\delta, x_t-\delta\}}{\operatorname{argmin}} f(x)$
**end if**
**return** $(x_{t+1}, 0)$

**Algorithm 5** Negative Curvature Finding [5]

**Require:** $\tilde{x}$, perturbation radius $r$, time interval $\mathscr{T}$, step size $\eta$.

$y_0 \leftarrow \text{Uniform}(\mathbb{B}_{\tilde{x}}(r))$
**for** $t = 1, \ldots, \mathscr{T}$ **do**
  $y_t \leftarrow y_{t-1} - \frac{\eta \|y_{t-1}\|}{r} (\nabla f(\tilde{x} + r y_{t-1}/\|y_{t-1}\|) - \nabla f(\tilde{x}))$
  $y_t \leftarrow r \cdot y_t / \|y_t\|$
**end for**
**return** $\mathscr{T} y / r$

**Algorithm 6** Faster Perturbed Accelerated Gradient Descent(FPAGD)

**Require:** $x_0$, step size $\eta$, iterative parameter $\theta$, negative curvature parameter $\gamma$, momentum upper bound $s$, perturbation radius $r$, time interval $\mathscr{T}$, tolerance $\epsilon$.

$t_{\text{perturb}} \leftarrow 0, z_0 \leftarrow x_0, \tilde{x} \leftarrow x_0, \zeta \leftarrow 0$
**for** $t = 0, 1, \ldots, T$ **do**
  **if** $\|\nabla f(x_t)\| \leq \epsilon$ and $t - t_{\text{perturb}} > \mathscr{T}$ **then**
    $\tilde{x} \leftarrow x_t$; $x_t \leftarrow \text{Uniform}(\mathbb{B}_{\tilde{x}}(r))$
    $z_t \leftarrow x_t$; $\zeta \leftarrow \nabla f(\tilde{x})$; $t_{\text{perturb}} \leftarrow t$
  **end if**
  **if** $t - t_{\text{perturb}} = \mathscr{T}'$ **then**
    $e := \frac{x_t - \tilde{x}}{\|x_t - \tilde{x}\|}$
    $x_t \leftarrow \tilde{x} - \eta_{\text{sub}} e$; $z_t \leftarrow x_t$; $\zeta \leftarrow 0$
  **end if**
  $x_{t+1} \leftarrow z_t - \eta(\nabla f(z_t) - \zeta)$
  $v_{t+1} \leftarrow x_{t+1} - x_t$
  $z_{t+1} \leftarrow x_{t+1} + (1 - \theta) v_{t+1}$
  **if** $t_{\text{perturb}} \neq 0$ and $t - t_{\text{perturb}} < \mathscr{T}'$ **then**
    $z_{t+1} \leftarrow \tilde{x} + r \cdot \frac{z_{t+1} - \tilde{x}}{\|z_{t+1} - \tilde{x}\|}$; $x_{t+1} \leftarrow \tilde{x} + r \cdot \frac{x_{t+1} - \tilde{x}}{\|z_{t+1} - \tilde{x}\|}$
  **end if**
  **else**
  **if** $f(x_{t+1}) \leq f(z_{t+1}) + \langle \nabla f(z_{t+1}), x_{t+1} - z_{t+1} \rangle - \frac{\gamma}{2} \|z_{t+1} - x_{t+1}\|^2$ **then**
    $(x_{t+1}, v_{t+1}) \leftarrow \text{Negative-Curvature-Exploitation}(x_{t+1}, v_{t+1}, s)$
    $z_{t+1} \leftarrow x_{t+1} + (1 - \theta) v_{t+1}$
  **end if**
**end for**