How to Escape Saddle Points Efficiently[8]

Jing Ma
Boston University
jingma@bu.edu

1. Problem Setting and Backgrounds

1.1. Problem Setting

In most learning problems, we need to construct a loss function, and then find the best parameters to minimize it. This paper investigates the most general form of optimization problems, i.e. for any function $f : \mathbb{R}^d \to \mathbb{R}$, which doesn't have to be convex, we need to find the $\mathbf{x}$ so that $\mathbf{x} = \operatorname{argmin}_x f(\mathbf{x})$.

1.2. Notations and Definitions

We use $\|\cdot\|$ of a vector to denote its l_2-norm, and use $\|\cdot\|$ of a matrix to denote its spectral norm.
$f(\cdot)$ is called l-gradient Lipschitz if $\forall \mathbf{x}_1, \mathbf{x}_2, \|\nabla f(\mathbf{x}_1) - \nabla f(\mathbf{x}_2)\| \leq l\|\mathbf{x}_1 - \mathbf{x}_2\|$; $f(\cdot)$ is called ρ-Hessian Lipschitz if $\forall \mathbf{x}_1, \mathbf{x}_2, \|\nabla^2 f(\mathbf{x}_1) - \nabla^2 f(\mathbf{x}_2)\| \leq \rho\|\mathbf{x}_1 - \mathbf{x}_2\|$.
A point $\mathbf{x}$ is called a first-order stationary point if $\|\nabla f(\mathbf{x})\| = 0$; a point $\mathbf{x}$ is called an ϵ-first-order stationary point if $\|\nabla f(\mathbf{x})\| \leq \epsilon$.
A point $\mathbf{x}$ is called a second-order stationary point if $\|\nabla f(\mathbf{x})\| = 0$ and $\lambda_{\min}(\nabla^2 f(\mathbf{x})) \geq 0$, where $\lambda_{\min}(\cdot)$ means the smallest eigenvalue of a matrix; for a ρ-Hessian Lipschitz function $f(\cdot)$, a point $\mathbf{x}$ is called an ϵ-second-order stationary point if $\|\nabla f(\mathbf{x})\| \leq \epsilon$ and $\lambda_{\min}(\nabla^2 f(\mathbf{x})) \geq -\sqrt{\rho\epsilon}$.

1.3. Backgrounds

In most cases, there are no closed-form solutions to the minimum of a function. Then the simplest and most standard way is to use gradient descent. Gradient descent works perfectly in a convex optimization problem, but may be less effective in a non-convex optimization problem, which is most of the case.
In a non-convex optimization problem, gradient descent is only able to look for first-order stationary points, but is not capable of distinguishing whether it's a second-order stationary point or not. For non-convex functions, a first-order stationary point can be a global minimum, a local minimum, a saddle point, or even a local maximum. The global minimum is the exact point we are looking for, and most of local minima are almost as good as global minima especially in deep networks[4]. So the first two cases, i.e. the second-order stationary points, are usually acceptable. However, saddle points or local maxima, which are not second-order stationary points, are far from what we want, so we want to escape from them. In this paper, both saddle points and local maxima are referred to as saddle points.
As an example of a saddle point, Figure 1 shows the behavior of function $f(x_1, x_2) = x_1^2 - x_2^2$ around its saddle point at the origin. Basically, a saddle point is a local minimum along some dimensions, but is a local maximum along the other. In high dimensional non-convex problems, a first-order stationary point is more likely to be a saddle point than a local minimum, since it's more likely that the point is a local maximum along some directions than it has to be a local minimum for all directions.

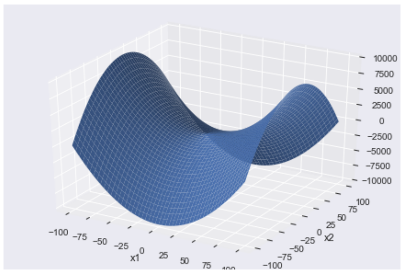


Figure 1: The behavior of function $f(x_1, x_2) = x_1^2 - x_2^2$, with a saddle point at the origin.

So especially in high dimensional non-convex problems, it's essential to have methods that can escape from saddle points, and to look for second-order stationary points efficiently.

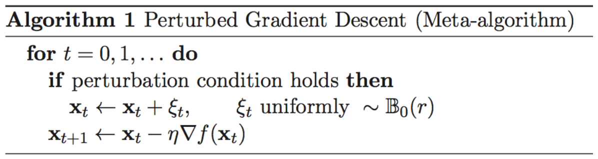
1.4. Related Works

For gradient descent, using the well-known update formula $x_{t+1} = x_t - \eta\nabla f(x_t)$ with $\eta = 1/l$, where l is the Lipschitz constant for the gradient of $f(\cdot)$, the algorithm is guaranteed to converge to a first-order stationary point within $\frac{l(f(x_0)-f^*)}{\epsilon^2}$ iterations, where f^* denotes the global minimum of f. This iteration complexity clearly doesn't depend on the input dimension d, and thus the gradient

descent algorithm is called dimension-free.
As to algorithms looking for second-order stationary points, there are mainly three categories of methods. The most straightforward category is to look at the Hessian matrix. This kind of methods are able to give dimension-free iteration complexities, i.e. $O(1/\epsilon^{1.5})$ in [10, 5]. However, for each iteration we'll have to calculate the full Hessian matrix, which is too time-consuming.
Instead of calculating the whole Hessian matrix, the second kind of methods calculate the product of Hessian matrix with given vectors. This kind of methods give iteration complexities of $O(\log(d) \cdot \text{poly}(1/\epsilon))$[1, 3, 2], which we call almost-dimension-free, since they only grow with a speed of polylog(d). However, they suffer from the same problem as the first category of methods, i.e. each iteration takes too much time to compute.
A third category of methods don't look at the Hessian matrix at all, and just make modifications to the gradient descent algorithm, and thus called gradient-based methods. Those methods take a similar amount of time per iteration as the traditional gradient descent. However, the best iteration complexity previous work within this category can give grows with poly(d)[7, 9], which is too much for high dimensional problems.
The main contribution of this paper is that they give a gradient-based algorithm that searches for second-order stationary points, with an almost-dimension-free iteration complexity, i.e. $\tilde{O}(\frac{l(f(x_0)-f^*)}{\epsilon^2})$, where $\tilde{O}$ hides any log factors.

2. Algorithm

2.1. Meta-algorithm



Algorithm 1 Perturbed Gradient Descent (Meta-algorithm)

```
for $t = 0, 1, \ldots$ do
    if perturbation condition holds then
        $\mathbf{x}_t \leftarrow \mathbf{x}_t + \xi_t, \quad \xi_t$ uniformly $\sim \mathbb{B}_0(r)$
    $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$
```

Figure 2: Meta-algorithm[8]

The meta-algorithm is shown in Figure 2. The basic idea is as long as you are around a first-order stationary point, i.e. you satisfy the "perturbation condition", then you add a perturbation and continue; otherwise, i.e. if you are not even around a first-order stationary point, you just do the traditional gradient descent. The perturbation is randomly uniformly distributed within a zero-centered d-dimensional ball with a radius of r calculated based on the properties of the function $f(\cdot)$.
The meta-algorithm comes from an observation about the geometry around saddle points, i.e. points from where traditional gradient descent gets stuck are usually within a thin band.

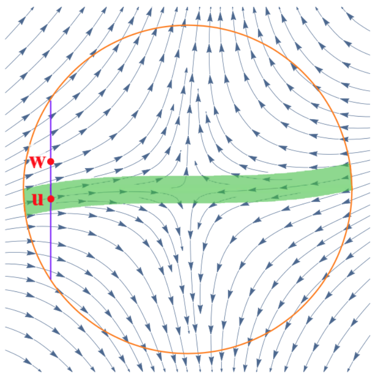


Figure 3: An illustration of the stuck band in 2 dimension[8].

Figure 3 gives an illustration of the stuck band in 2 dimension. If you are within the band, gradient descent will lead you gradually towards the saddle point; otherwise, you will keep going away from the saddle point along the gradient descent direction. Thus by bounding the volume of the stuck band, we can show that a random point within a big enough d-dimensional ball is more likely to be out of the band, i.e. a big enough random perturbation around a saddle point will be highly likely to help you escape from it.

2.2. Lemmas

To put everything in a mathematical way, the above intuitions are summed up into the following lemmas.

Lemma 1 *(informal) Assume $f(\cdot)$ is l-gradient Lipschitz and ρ-Hessian Lipschitz, suppose $\tilde{\mathbf{x}}$ satisfies $\|\nabla f(\tilde{\mathbf{x}})\| \leq g_{thres}$ and $\lambda_{min}(\nabla^2 f(\tilde{\mathbf{x}})) \leq -\sqrt{\rho\epsilon}$, where g_{thres} is defined later in the detailed algorithm in Figure 4, and let $\mathbf{e}_1$ be the direction corresponding to the smallest eigenvalue of $\nabla^2 f(\tilde{\mathbf{x}})$, then $\forall\delta \in (0, 1/3]$ and $\forall \mathbf{w}, \mathbf{u} \in \mathbb{B}_{\tilde{x}}(r)$, if $\mathbf{w} - \mathbf{u} = \mu r \mathbf{e}_1$ and $\mu \geq \delta/(2\sqrt{d})$, then at least one of $\mathbf{w}, \mathbf{u}$ is not in the stuck band.*

Lemma 1 bounds the width of the stuck band. Based on it, we can further prove Lemma 2.

Lemma 2 *(informal) Assume $f(\cdot)$ is l-gradient Lipschitz and ρ-Hessian Lipschitz, suppose $\mathbf{x}_t$ satisfies $\|\nabla f(\mathbf{x}_t)\| \leq g_{thres}$ and $\lambda_{min}(\nabla^2 f(\mathbf{x}_t)) \leq -\sqrt{\rho\epsilon}$, then after one perturbation step followed by t_{thres} gradient descent steps, we will have $f(\mathbf{x}_{t+t_{thres}}) \leq f(\mathbf{x}_t) - f_{thres}$, where $g_{thres}, t_{thres}, f_{thres}$ are defined later in the detailed algorithm in Figure 4, with high probability.*

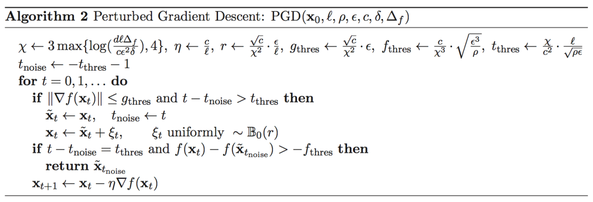
Lemma 2 together with Lemma 3 shown later contribute as the main elements to prove the main Theorem 4.

2.3. Perturbed Gradient Descent

Suppose the objective function $f(\cdot)$ is l-gradient Lipschitz and ρ-Hessian Lipschitz, and we are not around an ϵ-second order stationary point, then there are only two possibilities. First, the norm of the gradient is large, i.e. $\|\nabla f(\mathbf{x}_t)\| \geq g_{\text{thres}}$. In this case, we just do traditional gradient descent. Lemma 3 gives a guarantee on how much the function $f(\cdot)$ is going to decrease after one step of gradient descent.

Lemma 3 *Assume $f(\cdot)$ is l-gradient Lipschitz and ρ-Hessian Lipschitz, then we have $f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) - \frac{\eta}{2}\|\nabla f(\mathbf{x}_t)\|^2$ with $\eta < \frac{1}{l}$.*

The second possibility is when we are around a saddle point, i.e. $\|\nabla f(\mathbf{x}_t)\| \leq g_{\text{thres}}$ and $\lambda_{\min}(\nabla^2 f(\mathbf{x}_t)) \leq -\sqrt{\rho\epsilon}$. In this case, we do one perturbation step followed by t_{thres} steps of gradient descent, and Lemma 2 gives a guarantee on how much the function $f(\cdot)$ is going to decrease after those steps.
Combine the two cases together, the detailed version of Perturbed Gradient Descent algorithm is given in Figure 4. The algorithm is going to stop and output when the function $f(\cdot)$ decreases less than the amount predicted by Lemma 2.



Algorithm 2 Perturbed Gradient Descent: PGD($\mathbf{x}_0, \ell, \rho, \epsilon, c, \delta, \Delta_f$)

```
$\chi \leftarrow 3\max\{\log(\frac{d\ell\Delta_f}{c\epsilon^2\delta}), 4\}$, $\eta \leftarrow \frac{c}{\ell}$, $r \leftarrow \frac{\sqrt{c}}{\chi^2}\cdot\frac{\epsilon}{\ell}$, $g_{\text{thres}} \leftarrow \frac{\sqrt{c}}{\chi^2}\cdot\epsilon$, $f_{\text{thres}} \leftarrow \frac{c}{\chi^3}\cdot\sqrt{\frac{\epsilon^3}{\rho}}$, $t_{\text{thres}} \leftarrow \frac{\chi}{c^2}\cdot\frac{\ell}{\sqrt{\rho\epsilon}}$
$t_{\text{noise}} \leftarrow -t_{\text{thres}} - 1$
for $t = 0, 1, \ldots$ do
    if $\|\nabla f(\mathbf{x}_t)\| \leq g_{\text{thres}}$ and $t - t_{\text{noise}} > t_{\text{thres}}$ then
        $\tilde{\mathbf{x}}_t \leftarrow \mathbf{x}_t$,    $t_{\text{noise}} \leftarrow t$
        $\mathbf{x}_t \leftarrow \tilde{\mathbf{x}}_t + \xi_t$,    $\xi_t$ uniformly $\sim \mathbb{B}_0(r)$
    if $t - t_{\text{noise}} = t_{\text{thres}}$ and $f(\mathbf{x}_t) - f(\tilde{\mathbf{x}}_{t_{\text{noise}}}) > -f_{\text{thres}}$ then
        return $\tilde{\mathbf{x}}_{t_{\text{noise}}}$
    $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta\nabla f(\mathbf{x}_t)$
```

Figure 4: The detailed version of Perturbed Gradient Descent[8]

The main theorem states that:

Theorem 4 *Assume $f(\cdot)$ is l-gradient Lipschitz and ρ-Hessian Lipschitz, then there exists a constant c_{max} s.t. $\forall\delta > 0, \epsilon \leq \frac{l^2}{\rho}, \Delta_f \geq f(x_0) - f^*, c < c_{max}$, $PGD(x_0, l, \rho, \epsilon, c, \delta, \Delta_f)$ will output an ϵ-second-order stationary point, w.p.$\geq 1-\delta$, and terminate in the following number of iterations:*

$$O\left(\frac{l(f(x_0) - f^*)}{\epsilon^2} log^4(\frac{dl\Delta_f}{\epsilon^2\delta})\right).$$

We can see the iteration complexity given in Theorem 4 depends on polylog(d). So the paper proposes a gradient-based algorithm that searches for second-order stationary points, with an almost-dimension-free iteration complexity, i.e. $\tilde{O}(\frac{l(f(x_0)-f^*)}{\epsilon^2})$, where $\tilde{O}$ hides any log factors.

3. Experiments

To discuss the effectiveness of this proposed algorithm, here we look at two examples. The first experiment is a casually designed simple example, while the second experiment needs a carefully designed non-convex function, the details of which are given in another paper by the same group[6].

3.1. A Simple Example

To construct a simple example, we define $f(x_1, \ldots, x_d) =$ sigmoid$\left((\sum_{i=1}^{d-1} x_i^2) - x_d^2\right)$. The $(\sum_{i=1}^{d-1} x_i^2) - x_d^2$ part makes the origin become a saddle point, while the sigmoid makes sure the function is bounded, so that the parameters defined in Figure 4 are easier to estimate. The behavior of this function with $d = 2$ is shown in Figure 5, and there is a clear saddle point at the origin. Initialize the d-dimensional input as the following: $\forall i = 1, \ldots, d-1, x_i = 0, x_d = 10^{-20}$, so that it's within the stuck band. All other parameters of the Perturbed Gradient Descent algorithm can be estimated as $l = d, \rho = 2, \epsilon = 0.05, c = 1, \delta = 0.05, \Delta_f = 1$. Let $\mathbf{x}$ go through both Gradient Descent algorithm, and PGD algorithm. The performance graphs in Figure 6 show that PGD uses less iterations to escape from the saddle point. However, the dependence of iteration complexity on the input dimension d is not clear, since the polylog dependence on d is the upper bound, so a random non-convex function with a random initialization doesn't have to be the worst case so that it needs the maximal possible iterations. Our study only shows tests with $d = 2, 4, 8$. Under the same setting but with $d = 16$, $f(\cdot)$ will not decrease by an observable amount until around 10^5 iterations, and doesn't make a significant difference from traditional gradient descent.

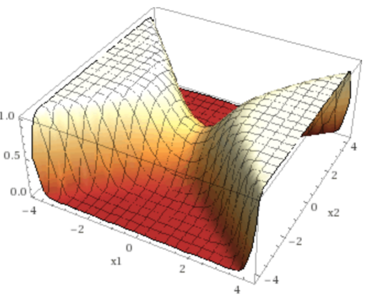
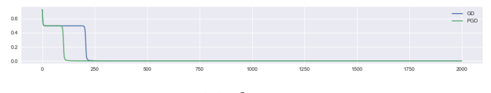
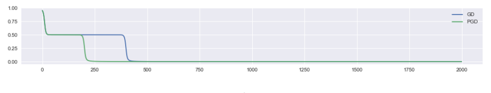


Figure 5: The behavior of $f(x_1, x_2) = \text{sigmoid}(x_1^2 - x_2^2)$.

By looking at the parameters in Figure 4 we can see that both ϵ and δ are small numbers, so χ is significantly large. Thus, $r, g_{\text{thres}}, f_{\text{thres}}$ are extremely small, while t_{thres} is large. This means we have to be extremely close to a saddle point, and make a small perturbation, and after rel-


(a) $d = 2$


(b) $d = 4$

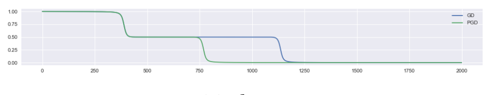

(c) $d = 8$

Figure 6: Performance of GD and PGD algorithms on the function $f(x_1, \ldots, x_d) = \text{sigmoid}\left((\sum_{i=1}^{d-1} x_i^2) - x_d^2\right)$, with $d = 2, 4, 8$.

atively lots of steps, we are guaranteed to decrease by a small amount.
So even though the paper gives an almost-dimension-free guarantee on the iteration complexity, it doesn't mean that it is always significantly better than gradient descent.

3.2. Carefully Designed Experiments

Under extreme conditions, however, i.e. by using a carefully designed non-convex function, gradient descent can face big trouble so that the perturbed gradient descent algorithm has a sound effect. In their another paper[6], a super complicated function $f(d, L, \gamma)$ (its behavior in 2 dimension is shown in Figure 7) is designed so that gradient descent will make its trajectory closer and closer to the saddle points each time, and it takes exponentially increasing time to escape from them (Figure 8). On the contrary, the perturbed gradient descent takes a relatively constant time to escape from those saddle points, with the upper bound guaranteed by Theorem 4.

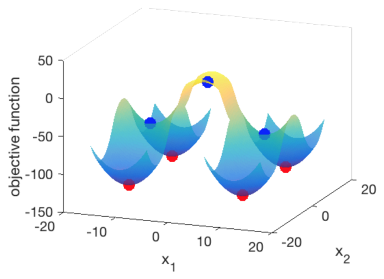
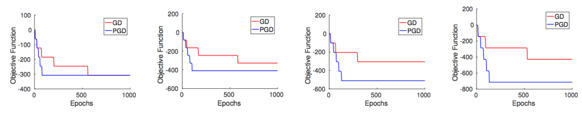


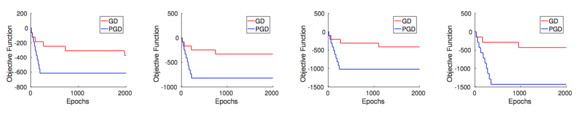
Figure 7: The behavior of $f(d, L, \gamma)$ with $d = 2$.

4. Discussion

In this report, we investigate on the paper *How to Escape Saddle Points Efficiently*. We introduce the problem setting, and why a gradient-based method looking for second-order stationary points within polylog(d) iteration complexity is important. Then we briefly talk about the basic intuitions of this paper, and informally show that the PGD method satisfies our requirement.
However, our experiments show that usually we don't have to resort to the perturbation step to escape from saddle points. Empirically, it's unusual for us to be that close to a saddle point, and it only takes slightly more steps for traditional gradient descent to escape. The guarantee on the decreasing speed is a loose bound in most cases, and is going to be useful only under extreme situations.


(a) $L = 1, \gamma = 1$ (b) $L = 1.5, \gamma = 1$ (c) $L = 2, \gamma = 1$ (d) $L = 3, \gamma = 1$

(a) $d = 5$


(a) $L = 1, \gamma = 1$ (b) $L = 1.5, \gamma = 1$ (c) $L = 2, \gamma = 1$ (d) $L = 3, \gamma = 1$

(b) $d = 10$

Figure 8: Performance of GD and PGD algorithms on the carefully designed function $f(d, L, \gamma)$[6], with $d = 5, 10$.

References

[1] N. Agarwal, Z. Allen-Zhu, B. Bullins, E. Hazan, and T. Ma. Finding approximate local minima for nonconvex optimization in linear time. *arXiv preprint arXiv:1611.01146*, 2016.

[2] Y. Carmon and J. C. Duchi. Gradient descent efficiently finds the cubic-regularized non-convex newton step. *arXiv preprint arXiv:1612.00547*, 2016.

[3] Y. Carmon, J. C. Duchi, O. Hinder, and A. Sidford. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization*, 28(2):1751–1772, 2018.

[4] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[5] F. E. Curtis, D. P. Robinson, and M. Samadi. A trust region algorithm with a worst-case iteration complexity of $o(\epsilon^{-3/2})$ for nonconvex optimization. *Mathematical Programming*, 162(1-2):1–32, 2017.

[6] S. S. Du, C. Jin, J. D. Lee, M. I. Jordan, A. Singh, and B. Poczos. Gradient descent can take exponential time to escape saddle points. In *Advances in neural information processing systems*, pages 1067–1077, 2017.

[7] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle pointsonline stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015.

[8] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan. How to escape saddle points efficiently. In *Proceedings of*

the 34th International Conference on Machine Learning-Volume 70, pages 1724–1732. JMLR. org, 2017.

[9] K. Y. Levy. The power of normalization: Faster evasion of saddle points. *arXiv preprint arXiv:1611.04831*, 2016.

[10] Y. Nesterov and B. T. Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.