Escape saddle points by a simple gradient-descent based algorithm

Anonymous Author(s)
Affiliation
Address
email

Abstract

Escaping saddle points is a central research topic in nonconvex optimization. In this paper, we propose a simple gradient-based algorithm such that for a smooth function $f\colon \mathbb{R}^n \to \mathbb{R}$, it outputs an ϵ-approximate second-order stationary point in $\tilde{O}(\log n/\epsilon^{1.75})$ iterations. Compared to the previous state-of-the-art algorithms by Jin et al. with $\tilde{O}(\log^4 n/\epsilon^2)$ or $\tilde{O}(\log^6 n/\epsilon^{1.75})$ iterations, our algorithm is polynomially better in terms of $\log n$ and matches their complexities in terms of $1/\epsilon$. For the stochastic setting, our algorithm outputs an ϵ-approximate second-order stationary point in $\tilde{O}(\log^2 n/\epsilon^4)$ iterations. Technically, our main contribution is an idea of implementing a robust Hessian power method using only gradients, which can find negative curvature near saddle points and achieve the polynomial speedup in $\log n$ compared to the perturbed gradient descent methods. Finally, we also perform numerical experiments that support our results.

1 Introduction

Nonconvex optimization is a central research area in optimization theory, since lots of modern machine learning problems can be formulated in models with nonconvex loss functions, including deep neural networks, principal component analysis, tensor decomposition, etc. In general, finding a global minimum of a nonconvex function is NP-hard in the worst case. Instead, many theoretical works focus on finding a local minimum instead of a global one, because recent works (both empirical and theoretical) suggested that local minima are nearly as good as global minima for a significant amount of well-studied machine learning problems; see e.g. [4, 12, 13, 14, 15, 16]. On the other hand, saddle points are major obstacles for solving these problems, not only because they are ubiquitous in high-dimensional settings where the directions for escaping may be few (see e.g. [5, 8, 11]), but also saddle points can correspond to highly suboptimal solutions (see e.g. [17, 24]).

Hence, one of the most important topics in nonconvex optimization is to *escape saddle points*. Specifically, we consider a twice-differentiable function $f\colon \mathbb{R}^n \to \mathbb{R}$ such that

- f is ℓ-smooth: $\|\nabla f(\mathbf{x}_1) - \nabla f(\mathbf{x}_2)\| \le \ell\|\mathbf{x}_1 - \mathbf{x}_2\| \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$,
- f is ρ-Hessian Lipschitz: $\|\mathcal{H}(\mathbf{x}_1) - \mathcal{H}(\mathbf{x}_2)\| \le \rho\|\mathbf{x}_1 - \mathbf{x}_2\| \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$;

here $\mathcal{H}$ is the Hessian of f. The goal is to find an ϵ-approximate second-order stationary point $\mathbf{x}_\epsilon$:[1]

$$\|\nabla f(\mathbf{x}_\epsilon)\| \le \epsilon, \quad \lambda_{\min}(\mathcal{H}(\mathbf{x}_\epsilon)) \ge -\sqrt{\rho\epsilon}. \tag{1}$$

In other words, at any ϵ-approx. second-order stationary point $\mathbf{x}_\epsilon$, the gradient is small with norm being at most ϵ and the Hessian is close to be positive semi-definite with all its eigenvalues $\ge -\sqrt{\rho\epsilon}$.

[1] We can ask for an (ϵ_1, ϵ_2)-approx. second-order stationary point s.t. $\|\nabla f(\mathbf{x})\| \le \epsilon_1$ and $\lambda_{\min}(\nabla^2 f(\mathbf{x})) \ge -\epsilon_2$ in general. The scaling in (1) was adopted as a standard in literature [1, 6, 10, 18, 19, 20, 22, 25, 26, 27].

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Algorithms for escaping saddle points are mainly evaluated from two aspects. On the one hand, considering the enormous dimensions of machine learning models in practice, dimension-free or almost dimension-free (i.e., having $\mathrm{poly}(\log n)$ dependence) algorithms are highly preferred. On the other hand, recent empirical discoveries in machine learning suggests that it is often feasible to tackle difficult real-world problems using simple algorithms, which can be implemented and maintained more easily in practice. On the contrary, algorithms with nested loops often suffer from significant overheads in large scales, or introduce concerns with the setting of hyperparameters (see e.g. [1, 7]), making them relatively hard to find practical implementations.

It is then natural to explore simple gradient-based algorithms for escaping from saddle points. The reason we do not assume access to Hessians is because its construction takes $\Omega(n^2)$ cost in general, which is computationally infeasible when the dimension is large. A seminal work along this line was by Ge et al. [12], which found an ϵ-approximate second-order stationary point satisfying (1) using only gradients in $O(\mathrm{poly}(n, 1/\epsilon))$ iterations. This is later improved to be almost dimension-free $\tilde{O}(\log^4 n/\epsilon^2)$ in the follow-up work [18],[2] and the state-of-the-art result [20] based on Nesterov's accelerated gradient descent [23] takes $\tilde{O}(\log^6 n/\epsilon^{1.75})$ iterations. However, these results still suffer from a significant overhead in terms of $\log n$, and it has been an open question whether it is possible to keep both the merits of using only first-order information as well as being close to dimension-free using a simple, gradient-based algorithm [21]. This paper answers this question in the affirmative.

Contributions. Our main contribution is a simple, single-loop, and robust gradient-based algorithm that can find an ϵ-approximate second-order stationary point of a smooth, Hessian Lipschitz function $f\colon \mathbb{R}^n \to \mathbb{R}$. Compared to the previous state-of-the-art results by [20] and [18, 19] using $\tilde{O}(\log^6 n/\epsilon^{1.75})$ or $\tilde{O}(\log^4 n/\epsilon^2)$ iterations, our algorithm achieves a polynomial speedup in $\log n$:

Theorem 1 (informal)**.** *Our algorithm finds an ϵ-approximate second-order stationary point in $\tilde{O}(\log n/\epsilon^{1.75})$ iterations.*

Technically, our work is inspired by the perturbed gradient descent (PGD) algorithm in [18, 19] and the perturbed accelerated gradient descent (PAGD) algorithm in [20]. Specifically, PGD applies gradient descents iteratively until it reaches a point with small gradient, which can be a potential saddle point. Then PGD generates a uniform perturbation in a small ball centered at that point and then continues the GD procedure. It is demonstrated that, with an appropriate choice of the perturbation radius, PGD can shake the point off from the neighborhood of the saddle point and converge to a second-order stationary point with high probability. The PAGD in [20] adopts a similar perturbation idea, but the GD is replaced by Nesterov's AGD [23].

Our algorithm is built upon PGD and PAGD but with one main modification regarding the perturbation idea: it is more efficient to add a perturbation in the *negative curvature direction* nearby the saddle point, rather than the uniform perturbation in PGD and PAGD, which is a compromise since we generally cannot access the Hessian at the saddle due to its high computational cost. Our key observation lies in the fact that we do not have to compute the entire Hessian to detect the negative curvature. Instead, in a small neighborhood of a saddle point, gradients can be viewed as Hessian-vector products plus some bounded deviation. In particular, GD near the saddle with learning rate $1/\ell$ is approximately the same as the power method of the matrix $(I - \mathcal{H}/\ell)$. As a result, the most negative eigenvalues stand out in GD because they have leading exponents in the power method, and thus it approximately moves along the direction of the most negative curvature nearby the saddle point. Following this approach, we can escape the saddle points more rapidly than previous algorithms: for a constant ϵ, PGD and PAGD take $O(\log n)$ iterations to decrease the function value by $\Omega(1/\log^3 n)$ and $\Omega(1/\log^5 n)$ with high probability, respectively; on the contrary, we can first take $O(\log n)$ iterations to specify a negative curvature direction, and then add a larger perturbation in this direction to decrease the function value by $\Omega(1)$. See Proposition 1 and Proposition 2.

Our algorithm is also applicable to the stochastic setting, where we can only access stochastic gradients and the stochasticity is not under the control of our algorithm. We further assume that the stochastic gradients are Lipschitz (or equivalently, the underlying functions are gradient-Lipschitz, see Assumption 2), which is also adopted in most of the existing works; see e.g. [9, 18, 19, 29]. We demonstrate that a simple extended version of our algorithm takes $O(\log^2 n)$ iterations to detect a negative curvature direction using only stochastic gradients, and then obtain an $\Omega(1)$ function value

[2] The $\tilde{O}$ notation omits poly-logarithmic terms, i.e., $\tilde{O}(g) = O(g\,\mathrm{poly}(\log g))$.

decrease with high probability. On the contrary, the perturbed stochastic gradient descent (PSGD) algorithm in [18, 19], the stochastic version of PGD, takes $O(\log^{10} n)$ iterations to decrease the function value by $\Omega(1/\log^5 n)$ with high probability. Our result is summarized as follows:

Theorem 2 (informal)**.** *In the stochastic setting, our algorithm finds an ϵ-approximate second-order stationary point using $\tilde{O}(\log^2 n/\epsilon^4)$ iterations via stochastic gradients.*

Setting	Reference	Iterations	Simplicity
Non-stochastic	[18, 19]	$\tilde{O}(\log^4 n/\epsilon^2)$	Single-loop
Non-stochastic	[20]	$\tilde{O}(\log^6 n/\epsilon^{1.75})$	Single-loop
Non-stochastic	**this work**	$\tilde{O}(\log n/\epsilon^{1.75})$	Single-loop
Stochastic	[18, 19]	$\tilde{O}(\log^{15} n/\epsilon^4)$	Single-loop
Stochastic	[10]	$\tilde{O}(\log^5 n/\epsilon^{3.5})$	Single-loop
Stochastic	[3]	$\tilde{O}(\log^2 n/\epsilon^{3.5})$	Nested-loop
Stochastic	[9]	$\tilde{O}(\log^2 n/\epsilon^3)$	Nested-loop
Stochastic	**this work**	$\tilde{O}(\log^2 n/\epsilon^4)$	Single-loop

Table 1: A summary of the state-of-the-art results on finding approximate second-order stationary points by the first-order (gradient) oracle. Iteration numbers are highlighted in terms of the dimension n and the precision ϵ.

It is worth highlighting that our gradient-descent based algorithm enjoys the following nice features:

- *Simplicity:* Some of the previous algorithms have nested-loop structures with the concern of practical impact when setting the hyperparameters. In contrast, our algorithm based on negative curvature finding only contains a single loop with two components: gradient descent (including AGD or SGD) and perturbation. As mentioned above, such simple structure is preferred in machine learning, which increases the possibility of our algorithm to find real-world applications.
- *Robustness:* Our algorithm is robust from two aspects. On the one hand, Theorem 2 essentially shows the robustness of our gradient-based negative curvature finding algorithm using even stochastic gradients, which is more prevalent than exact gradients in practice. On the other hand, we can replace the GD part by other momentum methods including PAGD and enjoy their faster convergence rates (see Theorem 3).

Finally, we perform numerical experiments that support our polynomial speedup in $\log n$. We perform our negative curvature finding algorithms using GD or SGD in various landscapes, and use comparative studies to show that our algorithms can achieve a higher probability of escaping from saddle points using much fewer iterations than PGD and PSGD (typically less than $1/3$ times of the iteration number of PGD and $1/2$ times of the iteration number of PSGD, respectively).

Open questions. This work leaves a couple of natural open questions for future investigation:

- Can we achieve the polynomial speedup in $\log n$ for more advanced stochastic optimization algorithms with complexity $\tilde{O}(\mathrm{poly}(\log n)/\epsilon^{3.5})$ [2, 3, 10, 25, 27] or $\tilde{O}(\mathrm{poly}(\log n)/\epsilon^3)$ [9, 28]?
- How is the performance of our algorithms for escaping saddle points in real-world applications, such as tensor decomposition [12, 15], matrix completion [13], etc.?

Broader impact. This work focuses on the theory of nonconvex optimization, and as far as we see, we do not anticipate its potential negative societal impact. Nevertheless, it might have a positive impact for researchers who are interested in understanding the theoretical underpinnings of (stochastic) gradient descent methods for machine learning applications.

Organization. In Section 2, we introduce our gradient-based Hessian power method algorithm for negative curvature finding, and present how our algorithms provide polynomial speedup in $\log n$ for both PGD and PAGD. In Section 3, we present the stochastic version of our negative curvature finding algorithm using stochastic gradients and demonstrate its polynomial speedup in $\log n$ for PSGD. Numerical experiments are presented in Section 4. We provide detailed proofs and additional numerical experiments in the supplementary material.

2 A simple algorithm for negative curvature finding

We show how to find negative curvature near a saddle point using a gradient-based Hessian power method algorithm, and extend it to a version with faster convergence rate by replacing gradient descents by accelerated gradient descents. The intuition works as follows: in a small enough region nearby a saddle point, the gradient can be approximately expressed as a Hessian-vector product formula, and the approximation error can be efficiently upper bounded, see Eq. (6). Hence, using only gradients information, we can implement an accurate enough Hessian power method to find negative eigenvectors of the Hessian matrix, and further find the negative curvature nearby the saddle.

2.1 Negative curvature finding based on gradient descents

We first present an algorithm for negative curvature finding based on gradient descents. Specifically, for any $\tilde{\mathbf{x}} \in \mathbb{R}^n$ with $\lambda_{\min}(\mathcal{H}(\tilde{\mathbf{x}})) \le -\sqrt{\rho\epsilon}$, it finds a unit vector $\hat{\mathbf{e}}$ such that $\hat{\mathbf{e}}^T\mathcal{H}(\tilde{\mathbf{x}})\hat{\mathbf{e}} \le -\sqrt{\rho\epsilon}/4$.

Algorithm 1: Negative Curvature Finding($\tilde{\mathbf{x}}, r, \mathscr{T}$).

$\mathbf{y}_0 \leftarrow$Uniform($\mathbb{B}_{\tilde{\mathbf{x}}}(r)$) where $\mathbb{B}_{\tilde{\mathbf{x}}}(r)$ is the ℓ_2-norm ball centered at $\tilde{\mathbf{x}}$ with radius r;
for $t = 1, ..., \mathscr{T}$ **do**
  $\mathbf{y}_t \leftarrow \mathbf{y}_{t-1} - \frac{\|\mathbf{y}_{t-1}\|}{\ell r}\left(\nabla f(\tilde{\mathbf{x}} + r\mathbf{y}_{t-1}/\|\mathbf{y}_{t-1}\|) - \nabla f(\tilde{\mathbf{x}})\right)$;
Output $\mathbf{y}_{\mathscr{T}}/r$.

Proposition 1. *Suppose the function $f\colon \mathbb{R}^n \to \mathbb{R}$ is ℓ-smooth and ρ-Hessian Lipschitz. For any $0 < \delta_0 \le 1$, we specify our choice of parameters and constants we use as follows:*

$$\mathscr{T} = \frac{8\ell}{\sqrt{\rho\epsilon}} \cdot \log\left(\frac{\ell}{\delta_0}\sqrt{\frac{n}{\pi\rho\epsilon}}\right), \quad r = \frac{\epsilon}{8\ell^2}\sqrt{\frac{\pi}{n}}\delta_0. \tag{2}$$

Suppose that $\tilde{\mathbf{x}}$ satisfies $\lambda_{\min}(\nabla^2 f(\tilde{\mathbf{x}})) \le -\sqrt{\rho\epsilon}$. Then with probability at least $1-\delta_0$, Algorithm 1 outputs a unit vector $\hat{\mathbf{e}}$ satisfying

$$\hat{\mathbf{e}}^T\mathcal{H}(\mathbf{x})\hat{\mathbf{e}} \le -\sqrt{\rho\epsilon}/4, \tag{3}$$

using $O(\mathscr{T}) = \tilde{O}\left(\frac{\log n}{\sqrt{\rho\epsilon}}\right)$ iterations, where $\mathcal{H}$ stands for the Hessian matrix of function f.

Proof. Without loss of generality we assume $\tilde{\mathbf{x}} = \mathbf{0}$ by shifting $\mathbb{R}^n$ such that $\tilde{\mathbf{x}}$ is mapped to $\mathbf{0}$. Define a new n-dimensional function

$$h_f(\mathbf{x}) := f(\mathbf{x}) - \langle \nabla f(\mathbf{0}), \mathbf{x}\rangle, \tag{4}$$

for the ease of our analysis. Since $\langle \nabla f(\mathbf{0}), \mathbf{x}\rangle$ is a linear function with Hessian being 0, the Hessian of h_f equals to the Hessian of f, and $h_f(\mathbf{x})$ is also ℓ-smooth and ρ-Hessian Lipschitz. In addition, note that $\nabla h_f(\mathbf{0}) = \nabla f(\mathbf{0}) - \nabla f(\mathbf{0}) = 0$. Then for all $\mathbf{x} \in \mathbb{R}^n$,

$$\nabla h_f(\mathbf{x}) = \int_{\xi=0}^{1} \mathcal{H}(\xi\mathbf{x}) \cdot \mathbf{x}\,\mathrm{d}\xi = \mathcal{H}(\mathbf{0})\mathbf{x} + \int_{\xi=0}^{1} (\mathcal{H}(\xi\mathbf{x}) - \mathcal{H}(\mathbf{0})) \cdot \mathbf{x}\,\mathrm{d}\xi. \tag{5}$$

Furthermore, due to the ρ-Hessian Lipschitz condition of both f and h_f, for any $\xi \in [0,1]$ we have $\|\mathcal{H}(\xi\mathbf{x}) - \mathcal{H}(\mathbf{0})\| \le \rho\|\mathbf{x}\|$, which leads to

$$\|\nabla h_f(\mathbf{x}) - \mathcal{H}(\mathbf{0})\mathbf{x}\| \le \rho\|\mathbf{x}\|^2. \tag{6}$$

Observe that the Hessian matrix $\mathcal{H}(\mathbf{0})$ admits the following eigen-decomposition:

$$\mathcal{H}(\mathbf{0}) = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i\mathbf{u}_i^T, \tag{7}$$

where the set $\{\mathbf{u}_i\}_{i=1}^n$ forms an orthonormal basis of $\mathbb{R}^n$. Without loss of generality, we assume the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ corresponding to $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ satisfy

$$\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n, \tag{8}$$

in which $\lambda_1 \le -\sqrt{\rho\epsilon}$. If $\lambda_n \le -\sqrt{\rho\epsilon}/2$, Proposition 1 holds directly. Hence, we only need to prove the case where $\lambda_n > -\sqrt{\rho\epsilon}/2$, in which there exists some p, p' with

$$\lambda_p \le -\sqrt{\rho\epsilon} < \lambda_{p+1}, \quad \lambda_{p'} \le -\sqrt{\rho\epsilon}/2 < \lambda_{p'+1}. \tag{9}$$

We use $\mathfrak{S}_{\|}$, $\mathfrak{S}_{\perp}$ to separately denote the subspace of $\mathbb{R}^n$ spanned by $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_p\}$, $\{\mathbf{u}_{p+1}, \mathbf{u}_{p+2}, \ldots, \mathbf{u}_n\}$, and use $\mathfrak{S}'_{\|}$, $\mathfrak{S}'_{\perp}$ to denote the subspace of $\mathbb{R}^n$ spanned by $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{p'}\}$, $\{\mathbf{u}_{p'+1}, \mathbf{u}_{p+2}, \ldots, \mathbf{u}_n\}$. Furthermore, we define $\mathbf{y}_{t,\|} := \sum_{i=1}^{p} \langle \mathbf{u}_i, \mathbf{y}_t \rangle \mathbf{u}_i$, $\mathbf{y}_{t,\|} := \sum_{i=p}^{n} \langle \mathbf{u}_i, \mathbf{y}_t \rangle \mathbf{u}_i$, $\mathbf{y}_{t,\|} := \sum_{i=1}^{p'} \langle \mathbf{u}_i, \mathbf{y}_t \rangle \mathbf{u}_i$, $\mathbf{y}_{t,\|} := \sum_{i=p'}^{n} \langle \mathbf{u}_i, \mathbf{y}_t \rangle \mathbf{u}_i$ respectively to denote the component of $\mathbf{y}_t$ in Line 3 in the subspaces $\mathfrak{S}_{\|}$, $\mathfrak{S}_{\perp}$, $\mathfrak{S}'_{\|}$, $\mathfrak{S}'_{\perp}$, and let $\alpha_t := \|\mathbf{y}_{t,\|}\|/\|\mathbf{y}_t\|$. Observe that

$$\Pr\left\{\alpha_0 \ge \delta_0\sqrt{\pi/n}\right\} \ge \Pr\left\{|y_{0,1}|/r \ge \delta_0\sqrt{\pi/n}\right\}, \tag{10}$$

where $y_{0,1} := \langle \mathbf{u}_1, \mathbf{y}_0 \rangle$ denotes the component of $\mathbf{y}_0$ along $\mathbf{u}_1$. Consider the case where $\alpha_0 \ge \delta_0\sqrt{\pi/n}$, which can be achieved with probability

$$\Pr\left\{\alpha_0 \ge \sqrt{\frac{\pi}{n}}\delta_0\right\} \ge 1 - \sqrt{\frac{\pi}{n}}\delta_0 \cdot \frac{\mathrm{Vol}(\mathbb{B}_0^{n-1}(1))}{\mathrm{Vol}(\mathbb{B}_0^{n}(1))} \ge 1 - \sqrt{\frac{\pi}{n}}\delta_0 \cdot \sqrt{\frac{n}{\pi}} = 1 - \delta_0. \tag{11}$$

We prove that there exists some t_0 with $1 \le t_0 \le \mathscr{T}$ such that

$$\|\mathbf{y}_{t_0,\perp'}\|/\|\mathbf{y}_{t_0}\| \le \sqrt{\rho\epsilon}/(8\ell). \tag{12}$$

Assume the contrary, for any $1 \le t \le \mathscr{T}$, we all have $\|\mathbf{y}_{t,\perp'}\|/\|\mathbf{y}_t\| > \sqrt{\rho\epsilon}/(8\ell)$. Then $\|\mathbf{y}'_{t,\perp}\|$ satisfies the following recurrence formula:

$$\|\mathbf{y}_{t+1,\perp'}\| \le (1 + \sqrt{\rho\epsilon}/(2\ell))\|\mathbf{y}_{t,\perp'}\| + \|\Delta_{\perp'}\| \le (1 + \sqrt{\rho\epsilon}/(2\ell) + \|\Delta\|/\|\mathbf{y}_{t,\perp'}\|)\|\mathbf{y}_{t,\perp'}\|, \tag{13}$$

where $\Delta := \frac{\|\mathbf{y}_t\|}{r}(\nabla h_f(r\mathbf{y}_t/\|\mathbf{y}_t\|) - \mathcal{H}(0) \cdot (r\mathbf{y}_t/\|\mathbf{y}_t\|))$ and $\|\Delta\|/\|\mathbf{y}_t\| \le \rho r$ due to (6). Hence,

$$\|\mathbf{y}_{t+1,\perp'}\| \le \Big(1 + \frac{\sqrt{\rho\epsilon}}{2\ell} + \frac{\|\Delta\|}{\|\mathbf{y}_{t,\perp'}\|}\Big)\|\mathbf{y}_{t,\perp'}\| \le \Big(1 + \frac{\sqrt{\rho\epsilon}}{2\ell} + \rho r \cdot \frac{8\ell}{\sqrt{\rho\epsilon}}\Big)\|\mathbf{y}_{t+1,\perp'}\|, \tag{14}$$

which leads to

$$\|\mathbf{y}_{t,\perp'}\| \le \|\mathbf{y}_{0,\perp'}\|(1 + \sqrt{\rho\epsilon}/(2\ell) + 8\rho r\ell/\sqrt{\rho\epsilon})^t \le \|\mathbf{y}_{0,\perp'}\|(1 + 5\sqrt{\rho\epsilon}/(8\ell))^t, \quad \forall t \in [\mathscr{T}]. \tag{15}$$

Similarly, we can have the recurrence formula for $\|\mathbf{y}_{t,\|}\|$:

$$\|\mathbf{y}_{t+1,\|}\| \ge (1 + \sqrt{\rho\epsilon}/(2\ell))\|\mathbf{y}_{t,\|}\| - \|\Delta_{\|}\| \ge (1 + \sqrt{\rho\epsilon}/(2\ell) - \|\Delta\|/(\alpha_t\|\mathbf{y}_t\|))\|\mathbf{y}_{t,\|}\|. \tag{16}$$

Considering that $\|\Delta\|/\|\mathbf{y}_t\| \le \rho r$ due to (6), we can further have

$$\|\mathbf{y}_{t+1,\|}\| \ge (1 + \sqrt{\rho\epsilon}/(2\ell) - \rho r/\alpha_t)\|\mathbf{y}_{t,\|}\|. \tag{17}$$

Intuitively, $\|\mathbf{y}_{t,\|}\|$ should have a faster increasing rate than $\|\mathbf{y}_{t,\perp}\|$ in this gradient-based Hessian power method, ignoring the deviation from quadratic approximation. As a result, the value the value $\alpha_t = \|\mathbf{y}\|_{t,\|}/\|\mathbf{y}_t\|$ should be non-decreasing. It is demonstrated in Lemma 6 in Appendix B that, even if we count this deviation in, α_t can still be lower bounded by some constant $\alpha_{\min}$:

$$\alpha_t \ge \alpha_{\min} = \frac{\delta_0}{4}\sqrt{\frac{\pi}{n}}, \qquad \forall 1 \le t \le \mathscr{T}. \tag{18}$$

by which we can further deduce that

$$\|\mathbf{y}_{t,\|}\| \ge \|\mathbf{y}_{0,\|}\|(1 + \sqrt{\rho\epsilon}/\ell - \rho r/\alpha_{\min})^t \ge \|\mathbf{y}_{0,\|}\|(1 + 7\sqrt{\rho\epsilon}(8\ell))^t, \quad \forall 1 \le t \le \mathscr{T}. \tag{19}$$

Observe that

$$\frac{\|\mathbf{y}_{\mathscr{T},\perp'}\|}{\|\mathbf{y}_{\mathscr{T},\|}\|} \le \frac{\|\mathbf{y}_{0,\perp'}\|}{\|\mathbf{y}_{0,\|}\|} \cdot \Big(\frac{1 + 5\sqrt{\rho\epsilon}/(8\ell)}{1 + 7\sqrt{\rho\epsilon}/(8\ell)}\Big)^{\mathscr{T}} \le \frac{1}{\delta_0}\sqrt{\frac{n}{\pi}}\Big(\frac{1 + 5\sqrt{\rho\epsilon}/(8\ell)}{1 + 7\sqrt{\rho\epsilon}/(8\ell)}\Big)^{\mathscr{T}} \le \frac{\sqrt{\rho\epsilon}}{8\ell}. \tag{20}$$

Since $\|\mathbf{y}_{\mathscr{T},\|}\| \le \|\mathbf{y}_{\mathscr{T}}\|$, we have $\|\mathbf{y}_{\mathscr{T},\perp'}\|/\|\mathbf{y}_{\mathscr{T}}\| \le \sqrt{\rho\epsilon}/(8\ell)$, contradiction. Hence, there here exists some t_0 with $1 \le t_0 \le \mathscr{T}$ such that $\|\mathbf{y}_{t_0,\perp'}\|/\|\mathbf{y}_{t_0}\| \le \sqrt{\rho\epsilon}/(8\ell)$. Consider the normalized

vector $\hat{\mathbf{e}} = \mathbf{y}_{t_0}/r$, we use $\hat{\mathbf{e}}_{\perp'}$ and $\hat{\mathbf{e}}_{\|'}$ to separately denote the component of $\hat{\mathbf{e}}$ in $\mathfrak{S}'_{\perp}$ and $\mathfrak{S}'_{\|}$. Then, $\|\hat{\mathbf{e}}_{\perp'}\| \le \sqrt{\rho\epsilon}/(8\ell)$ whereas $\|\hat{\mathbf{e}}_{\|'}\| \ge 1 - \rho\epsilon/(8\ell)^2$. Then,

$$\hat{\mathbf{e}}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}} = (\hat{\mathbf{e}}_{\perp'} + \hat{\mathbf{e}}_{\|'})^T\mathcal{H}(\mathbf{0})(\hat{\mathbf{e}}_{\perp'} + \hat{\mathbf{e}}_{\|'}) = \hat{\mathbf{e}}_{\perp'}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\perp'} + \hat{\mathbf{e}}_{\|'}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\|'} \tag{21}$$

since $\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\perp'} \in \mathfrak{S}'_{\perp}$ and $\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\|'} \in \mathfrak{S}'_{\|}$. Due to the ℓ-smoothness of the function, all eigenvalue of the Hessian matrix has its absolute value upper bounded by ℓ. Hence,

$$\hat{\mathbf{e}}_{\perp'}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\perp'} \le \ell\|\hat{\mathbf{e}}_{\perp'}^T\|_2^2 = \rho\epsilon/(64\ell^2). \tag{22}$$

Further according to the definition of $\mathfrak{S}_{\|}$, we have

$$\hat{\mathbf{e}}_{\|'}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\|'} \le -\sqrt{\rho\epsilon}\|\hat{\mathbf{e}}_{\|'}\|^2/2. \tag{23}$$

Combining these two inequalities together, we can obtain

$$\hat{\mathbf{e}}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}} = \hat{\mathbf{e}}_{\perp}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\perp'} + \hat{\mathbf{e}}_{\|'}^T\mathcal{H}(\mathbf{0})\hat{\mathbf{e}}_{\|'} \le -\sqrt{\rho\epsilon}\|\hat{\mathbf{e}}_{\|'}\|^2/2 + \rho\epsilon/(64\ell^2) \le -\sqrt{\rho\epsilon}/4. \tag{24}$$

□

Remark 1. *In practice, the value of $\|\mathbf{y}_t\|$ can become large during the execution of Algorithm 1. To fix this, we can renormalize $\mathbf{y}_t$ to have ℓ_2-norm r at the ends of such iterations, and this does not influence the performance of the algorithm.*

2.2 Faster negative curvature finding based on accelerated gradient descents

In this subsection, we replace the GD part in Algorithm 1 by AGD to obtain our Accelerated Negative Curvature Finding algorithm (Algorithm 2) with similar effect and faster convergence rate.

Algorithm 2: Accelerated Negative Curvature Finding($\tilde{\mathbf{x}}, r', \mathscr{T}'$).

$\mathbf{y}_0 \leftarrow$Uniform($\mathbb{B}_{\tilde{\mathbf{x}}}(r')$) where $\mathbb{B}_{\tilde{\mathbf{x}}}(r')$ is the ℓ_2-norm ball centered at $\tilde{\mathbf{x}}$ with radius r'; $\mathbf{v}_0 \leftarrow \mathbf{0}$;
for $t = 1, ..., \mathscr{T}'$ **do**
  $\mathbf{v}'_t \leftarrow \mathbf{v}_{t-1} - \nabla h_f(\mathbf{y}_t)$, $\mathbf{y}'_t \leftarrow \mathbf{y}_{t-1} + \mathbf{v}'_t$;
  $\mathbf{v}_t \leftarrow \mathbf{v}_t \cdot \frac{r'}{\|\mathbf{y}'_t\|}$, $\mathbf{y}_t \leftarrow \mathbf{y}'_t \cdot \frac{r'}{\|\mathbf{y}'_t\|}$;
Output $\mathbf{y}_{\mathscr{T}'}/r'$.

The function $h_f(\mathbf{x})$ appearing in Algorithm 2 is defined as:

$$h_f(\mathbf{x}) := \frac{\|\mathbf{x}\|^2}{r'^2} \cdot \big(f(r'\mathbf{x}/\|\mathbf{x}\| + \tilde{\mathbf{x}}) - f(\tilde{\mathbf{x}}) - \langle \nabla f(\tilde{\mathbf{x}}), r'\mathbf{x}/\|\mathbf{x}\|\rangle\big). \tag{25}$$

Intuitively, $h_f(\mathbf{x})$ contains only second and higher order terms of the Taylor expansion of $f(\mathbf{x})$ at $\tilde{\mathbf{x}}$. In addition, along any axis across $\mathbf{0}$, the value of h_f is quadratic, which enables us to renormalize $\mathbf{y}_t$ during the algorithm without affecting the output. Such renormalization guarantees that the norm of $\mathbf{y}_t$ does not scale exponentially during the execution of the algorithm. Also note that each call of $\nabla h_f(\mathbf{x})$ can be implemented using 2 calls of $\nabla f(\mathbf{x})$. The following proposition exhibits the effectiveness of Algorithm 2:

Proposition 2. *Suppose the function $f\colon \mathbb{R}^n \to \mathbb{R}$ is ℓ-smooth and ρ-Hessian Lipschitz. For any $0 < \delta_0 \le 1$, we specify our choice of parameters and constants we use as follows:*

$$\mathscr{T}' = \frac{8}{(\rho\epsilon)^{1/4}} \cdot \ln\Big(\frac{\ell}{\delta_0}\sqrt{\frac{n}{\pi\rho\epsilon}}\Big), \quad r' = \frac{\delta_0}{40\mathscr{T}'}\sqrt{\frac{\pi}{n}}\Big(\frac{\epsilon}{\rho^3}\Big)^{1/4}. \tag{26}$$

Suppose that $\tilde{\mathbf{x}}$ satisfies $\lambda_{\min}(\nabla^2 f(\tilde{\mathbf{x}})) \le -\sqrt{\rho\epsilon}$. Then with probability at least $1 - \delta_0$, Algorithm 2 outputs a unit vector $\hat{\mathbf{e}}$ satisfying

$$\hat{\mathbf{e}}^T\mathcal{H}(\mathbf{x}_0)\hat{\mathbf{e}} \le -\sqrt{\rho\epsilon}/4, \tag{27}$$

where $\mathcal{H}$ stands for the Hessian matrix of function f, using $O(\mathscr{T}') = \tilde{O}\Big(\frac{\log n}{(\rho\epsilon)^{1/4}}\Big)$ iterations.

The proof of Proposition 2 is similar to the proof of Proposition 1, and is deferred to Appendix B.2.

2.3 Escaping saddle points using negative curvature finding

In this subsection, we demonstrate that our negative curvature finding algorithms, Algorithm 1 and Algorithm 2, can help in escaping saddle points of nonconvex functions. The intuition works as follows: we start with gradient descents or accelerated gradient descents until the gradient becomes small. At this position, we compute the negative curvature direction, described by a unit vector $\hat{\mathbf{e}}$, via Algorithm 1 or Algorithm 2. Then, we add a perturbation along this direction of negative curvature and go back to gradient descents or accelerated gradient descents. It has the following guarantee:

Lemma 1. *Suppose the function $f : \mathbb{R}^n \to \mathbb{R}$ is ℓ-smooth and ρ-Hessian Lipschitz. Then for any point $\mathbf{x}_0 \in \mathbb{R}^n$, if there exists a unit vector $\hat{\mathbf{e}}$ satisfying $\hat{\mathbf{e}}^T \mathcal{H}(\mathbf{x}_0)\hat{\mathbf{e}} \le -\frac{\sqrt{\rho\epsilon}}{4}$ where $\mathcal{H}$ stands for the Hessian matrix of function f, the following inequality holds:*

$$f\Big(\mathbf{x}_0 - \frac{f'_{\hat{\mathbf{e}}}(\mathbf{x}_0)}{4|f'_{\hat{\mathbf{e}}}(\mathbf{x}_0)|}\sqrt{\frac{\epsilon}{\rho}} \cdot \hat{\mathbf{e}}\Big) \le f(\mathbf{x}_0) - \frac{1}{384}\sqrt{\frac{\epsilon^3}{\rho}}, \tag{28}$$

where $f'_{\hat{\mathbf{e}}}$ stands for the gradient component of f along the direction of $\hat{\mathbf{e}}$.

Proof. Without loss of generality, we assume $\mathbf{x}_0 = \mathbf{0}$. We can also assume $\langle \nabla f(\mathbf{0}), \hat{\mathbf{e}} \rangle \le 0$; if this is not the case we can pick $-\hat{\mathbf{e}}$ instead, which still satisfies $(-\hat{\mathbf{e}})^T \mathcal{H}(\mathbf{x}_0)(-\hat{\mathbf{e}}) \le -\frac{\sqrt{\rho\epsilon}}{4}$. In practice, to figure out whether we should use $\hat{\mathbf{e}}$ or $-\hat{\mathbf{e}}$, we apply both of them in (28) and choose the one with smaller function value. Then, for any $\mathbf{x} = x_{\hat{\mathbf{e}}}\hat{\mathbf{e}}$ with some $x_{\hat{\mathbf{e}}} > 0$, we have $\frac{\partial^2 f}{\partial x_{\hat{\mathbf{e}}}^2}(\mathbf{x}) \le -\frac{\sqrt{\rho\epsilon}}{4} + \rho x_{\hat{\mathbf{e}}}$ due to the ρ-Hessian Lipschitz condition of f. Hence,

$$\frac{\partial f}{\partial x_{\hat{\mathbf{e}}}}(\mathbf{x}) \le f'_{\hat{\mathbf{e}}}(\mathbf{0}) - \frac{\sqrt{\rho\epsilon}}{4} x_{\hat{\mathbf{e}}} + \rho x_{\hat{\mathbf{e}}}^2, \tag{29}$$

by which we can further derive that

$$f(x_{\hat{\mathbf{e}}}\hat{\mathbf{e}}) - f(\mathbf{0}) \le f'_{\hat{\mathbf{e}}}(\mathbf{0}) x_{\hat{\mathbf{e}}} - \frac{\sqrt{\rho\epsilon}}{8} x_{\hat{\mathbf{e}}}^2 + \frac{\rho}{3} x_{\hat{\mathbf{e}}}^3 \le -\frac{\sqrt{\rho\epsilon}}{8} x_{\hat{\mathbf{e}}}^2 + \frac{\rho}{3} x_{\hat{\mathbf{e}}}^3. \tag{30}$$

Settings $x_{\hat{\mathbf{e}}} = \frac{1}{4}\sqrt{\frac{\epsilon}{\rho}}$ gives (28). □

We give the full algorithm details in Appendix C.1. Along this approach, we achieve the following:

Theorem 3 (informal, full version deferred to Appendix C.3)**.** *For any $\epsilon > 0$ and a constant $0 < \delta \le 1$, Algorithm 6 satisfies that at least one of the iterations $\mathbf{x}_t$ will be an ϵ-approximate second-order stationary point in*

$$\tilde{O}\Big(\frac{(f(\mathbf{x}_0) - f^*)}{\epsilon^{1.75}} \cdot \log n\Big) \tag{31}$$

iterations, with probability at least $1 - \delta$, where f^* is the global minimum of f.

Intuitively, the proof of Theorem 3 has two parts. The first part is similar to the proof of [20, Theorem 3], which shows that PAGD uses $\tilde{O}(\log^6 n/\epsilon^{1.75})$ iterations to escape saddle points. We show that there can be at most $\tilde{O}(\Delta_f/\epsilon^{1.75})$ iterations with the norm of gradient larger than ϵ using almost the same techniques, but with slightly different parameter choices. The second part is based on Algorithm 2, our accelerated negative curvature finding algorithm. Specifically, at each saddle point we encounter, we can take $\tilde{O}(\log n/\epsilon^{1/4})$ iterations to find its negative curvature (Proposition 2), and add a perturbation in this direction to decrease the function value by $O(\epsilon^{1.5})$ (Lemma 1). Hence, the iterations introduced by Algorithm 4 can be at most $\tilde{O}\big(\frac{\log n}{\epsilon^{1.5}} \cdot \frac{1}{\epsilon^{0.25}}\big) = \tilde{O}(\log n/\epsilon^{1.75})$, which is simply an upper bound on the overall iteration number. The detailed proof is deferred to Appendix C.3.

Remark 2. *Although Theorem 3 only demonstrates that our algorithms will visit some ϵ-approximate second-order stationary point during their execution with high probability, it is straightforward to identify one of them if we add a termination condition: once Negative Curvature Finding (Algorithm 1 or Algorithm 2) is applied, we record the position $\mathbf{x}_{t_0}$ and the function value decrease due to the following perturbation. If the function value decrease is larger than $\frac{1}{384}\sqrt{\frac{\epsilon^3}{\rho}}$ as per Lemma 1, then the algorithms make progress. Otherwise, $\mathbf{x}_{t_0}$ is an ϵ-approximate second-order stationary point with high probability.*

3 Stochastic setting

In this section, we present a stochastic version of Algorithm 1 using stochastic gradients, and demonstrate that it can also be used to escape saddle points and obtain a polynomial speedup in $\log n$ compared to the perturbed stochastic gradient (PSGD) algorithm in [19].

3.1 Stochastic negative curvature finding

In the stochastic gradient descent setting, the exact gradients oracle ∇f of function f cannot be accessed. Instead, we only have unbiased stochastic gradients $\mathbf{g}(\mathbf{x};\theta)$ such that

$$\nabla f(\mathbf{x}) = \mathbb{E}_{\theta\sim\mathcal{D}}[\mathbf{g}(\mathbf{x};\theta)] \qquad \forall \mathbf{x}\in\mathbb{R}^n, \tag{32}$$

where $\mathcal{D}$ stands for the probability distribution followed by the random variable θ. Define

$$\zeta(\mathbf{x};\theta) := g(\mathbf{x};\theta) - \nabla f(\mathbf{x}) \tag{33}$$

to be the error term of the stochastic gradient. Then, the expected value of vector $\zeta(\mathbf{x};\theta)$ at any $\mathbf{x}\in\mathbb{R}^n$ equals to $\mathbf{0}$. Further, we assume the stochastic gradient $\mathbf{g}(\mathbf{x},\theta)$ also satisfies the following assumptions, which were also adopted in previous literatures; see e.g. [9, 18, 19, 29].

Assumption 1. *For any $\mathbf{x}\in\mathbb{R}^n$, the stochastic gradient $\mathbf{g}(\mathbf{x};\theta)$ with $\theta\sim\mathcal{D}$ satisfies:*

$$\Pr[(\|\mathbf{g}(\mathbf{x};\theta)-\nabla f(\mathbf{x})\|\geq t)] \leq 2\exp\big(-t^2/(2\sigma^2)\big), \quad \forall t\in\mathbb{R}. \tag{34}$$

Assumption 2. *For any $\theta\in supp(\mathcal{D})$, $\mathbf{g}(\mathbf{x};\theta)$ is $\tilde{\ell}$-Lipschitz for some constant $\tilde{\ell}$:*

$$\|\mathbf{g}(\mathbf{x}_1;\theta)-\mathbf{g}(\mathbf{x}_2;\theta)\| \leq \tilde{\ell}\|\mathbf{x}_1-\mathbf{x}_2\|, \qquad \forall \mathbf{x}_1,\mathbf{x}_2\in\mathbb{R}^n. \tag{35}$$

Assumption 2 emerges from the fact that the stochastic gradient $\mathbf{g}$ is often obtained as an exact gradient of some smooth function,

$$\mathbf{g}(\mathbf{x};\theta) = \nabla f(\mathbf{x};\theta). \tag{36}$$

In this case, Assumption 2 guarantees that for any $\theta\sim\mathcal{D}$, the spectral norm of the Hessian of $f(\mathbf{x};\theta)$ is upper bounded by $\tilde{\ell}$. Under these two assumptions, we can construct the stochastic version of Algorithm 1, as shown in Algorithm 3.

Algorithm 3: Stochastic Negative Curvature Finding($\mathbf{x}_0, r_s, \mathscr{T}_s, m$).

$\mathbf{y}_0 \leftarrow 0,\ L_0 \leftarrow r_s$;
for $t = 1, ..., \mathscr{T}_s$ **do**
  Sample $\{\theta^{(1)},\theta^{(2)},\cdots,\theta^{(m)}\}\sim\mathcal{D}$;
  $\mathbf{g}(\mathbf{y}_{t-1}) \leftarrow \frac{1}{m}\sum_{j=1}^{m}\big(\mathbf{g}(\mathbf{x}_0+\mathbf{y}_{t-1};\theta^{(j)})-\mathbf{g}(\mathbf{x}_0;\theta^{(j)})\big)$;
  $\mathbf{y}_t \leftarrow \mathbf{y}_{t-1} - \frac{1}{\tilde{\ell}}(\mathbf{g}(\mathbf{y}_{t-1})+\xi_t/L_{t-1}), \qquad \xi_t\sim\mathcal{N}\Big(0,\frac{r_s^2}{d}I\Big)$;
  $L_t \leftarrow \frac{\|\mathbf{y}_t\|}{r_s}L_{t-1}$ and $\mathbf{y}_t \leftarrow \mathbf{y}_t\cdot\frac{r_s}{\|\mathbf{y}_t\|}$;
Output $\mathbf{y}_{\mathscr{T}}/r_s$.

Similar to the non-stochastic setting, Algorithm 3 can be used to escape from saddle points and obtain a polynomial speedup in $\log n$ compared to PSGD algorithm in [19]. This is quantitatively shown in the following theorem:

Theorem 4 (informal, full version deferred to Appendix D)**.** *For any $\epsilon>0$ and a constant $0<\delta\leq 1$, our algorithm[3] based on Algorithm 3 using only stochastic gradient descent satisfies that at least one of the iterations $\mathbf{x}_t$ will be an ϵ-approximate second-order stationary point in*

$$\tilde{O}\Big(\frac{(f(\mathbf{x}_0)-f^*)}{\epsilon^4}\cdot\log^2 n\Big) \tag{37}$$

iterations, with probability at least $1-\delta$, where f^* is the global minimum of f.

[3]Our algorithm based on Algorithm 3 has similarities to the Neon2^{online} algorithm in [3]. Both algorithms find a second-order stationary point for stochastic optimization in $\tilde{O}(\log^2 n/\epsilon^4)$ iterations, and we both apply directed perturbations based on the results of negative curvature finding. Nevertheless, our algorithm enjoys simplicity by only having a single loop, whereas Neon2^{online} has a nested loop for boosting their confidence.

4 Numerical experiments

In this section, we provide numerical results that exhibit the power of our negative curvature finding algorithm for escaping saddle points. More experimental results can be found in Appendix E. All the experiments are performed on MATLAB R2019b on a computer with Six-Core Intel Core i7 processor and 16GB memory, and their codes are given in the supplementary material.

Comparison between Algorithm 1 and PGD. We compare the performance of our Algorithm 1 with the perturbed gradient descent (PGD) algorithm in [19] on a test function $f(x_1, x_2) = \frac{1}{16}x_1^4 - \frac{1}{2}x_1^2 + \frac{9}{8}x_2^2$ with a saddle point at $(0, 0)$. The advantage of Algorithm 1 as illustrated in Figure 1.

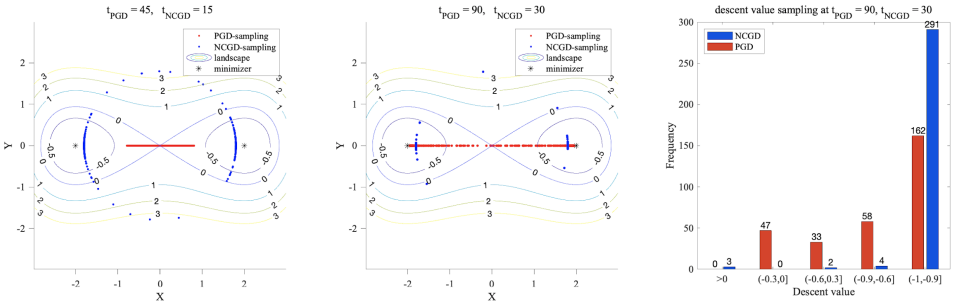

Figure 1: Run Algorithm 1 and PGD on landscape $f(x_1, x_2) = \frac{1}{16}x_1^4 - \frac{1}{2}x_1^2 + \frac{9}{8}x_2^2$. Parameters: $\eta = 0.05$ (step length), $r = 0.1$ (ball radius in PGD and parameter r in Algorithm 1), $M = 300$ (number of samplings). **Left**: The contour of the landscape is placed on the background with labels being function values. Blue points represent samplings of Algorithm 1 at time step $t_{\text{NCGD}} = 15$ and $t_{\text{NCGD}} = 30$, and red points represent samplings of PGD at time step $t_{\text{PGD}} = 45$ and $t_{\text{PGD}} = 90$. Algorithm 1 transforms an initial uniform-circle distribution into a distribution concentrating on two points indicating negative curvature, and these two figures represent intermediate states of this process. It converges faster than PGD even when $t_{\text{NCGD}} \ll t_{\text{PGD}}$.
Right: A histogram of descent values obtained by Algorithm 1 and PGD, respectively. Set $t_{\text{NCGD}} = 30$ and $t_{\text{PGD}} = 90$. Although we run three times of iterations in PGD, there are still over 40% of gradient descent paths with function value decrease no greater than 0.9, while this ratio for Algorithm 1 is less than 5%.

Comparison between Algorithm 3 and PSGD. We compare the performance of our Algorithm 3 with the perturbed stochastic gradient descent (PSGD) algorithm in [19] on a test function $f(x_1, x_2) = (x_1^3 - x_2^3)/2 - 3x_1x_2 + (x_1^2 + x_2^2)^2/2$. Compared to the landscape of the previous experiment, this function is more deflated from a quadratic field due to the cubic terms. Nevertheless, Algorithm 3 still possesses a notable advantage compared to PSGD as demonstrated in Figure 2.

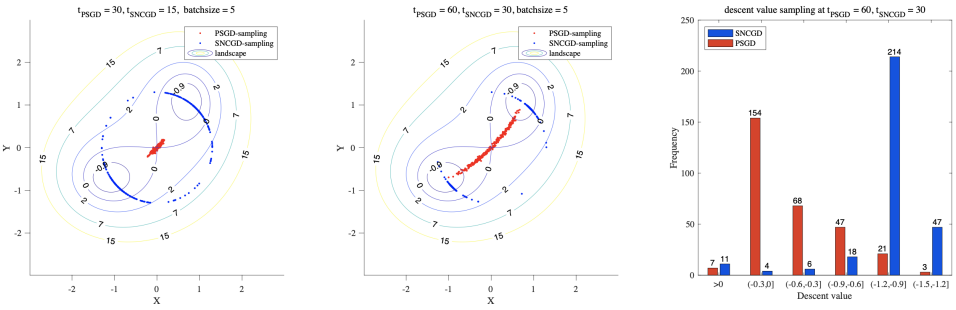

Figure 2: Run Algorithm 3 and PSGD on landscape $f(x_1, x_2) = \frac{x_1^3 - x_2^3}{2} - 3x_1x_2 + \frac{1}{2}(x_1^2 + x_2^2)^2$. Parameters: $\eta = 0.02$ (step length), $r = 0.01$ (variance in PSGD and r_s in Algorithm 3), $M = 300$ (number of samplings). **Left**: The contour of the landscape is placed on the background with labels being function values. Blue points represent samplings of Algorithm 3 at time step $t_{\text{SNCGD}} = 15$ and $t_{\text{SNCGD}} = 30$, and red points represent samplings of PSGD at time step $t_{\text{PSGD}} = 30$ and $t_{\text{PSGD}} = 60$. Algorithm 3 transforms an initial uniform-circle distribution into a distribution concentrating on two points indicating negative curvature, and these two figures represent intermediate states of this process. It converges faster than PSGD even when $t_{\text{SNCGD}} \ll t_{\text{PSGD}}$.
Right: A histogram of descent values obtained by Algorithm 3 and PSGD, respectively. Set $t_{\text{SNCGD}} = 30$ and $t_{\text{PSGD}} = 60$. Although we run two times of iterations in PSGD, there are still over 50% of SGD paths with function value decrease no greater than 0.6, while this ratio for Algorithm 3 is less than 10%.

References

[1] Naman Agarwal, Zeyuan Allen-Zhu, Brian Bullins, Elad Hazan, and Tengyu Ma, *Finding approximate local minima faster than gradient descent*, Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp. 1195–1199, 2017, arXiv:1611.01146.

[2] Zeyuan Allen-Zhu, *Natasha 2: Faster non-convex optimization than SGD*, Advances in Neural Information Processing Systems, pp. 2675–2686, 2018, arXiv:1708.08694.

[3] Zeyuan Allen-Zhu and Yuanzhi Li, *Neon2: Finding local minima via first-order oracles*, Advances in Neural Information Processing Systems, pp. 3716–3726, 2018, arXiv:1711.06673.

[4] Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro, *Global optimality of local search for low rank matrix recovery*, Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 3880–3888, 2016, arXiv:1605.07221.

[5] Alan J. Bray and David S. Dean, *Statistics of critical points of Gaussian fields on large-dimensional spaces*, Physical Review Letters **98** (2007), no. 15, 150201, arXiv:cond-mat/0611023.

[6] Yair Carmon, John C. Duchi, Oliver Hinder, and Aaron Sidford, *Accelerated methods for nonconvex optimization*, SIAM Journal on Optimization **28** (2018), no. 2, 1751–1772, arXiv:1611.00756.

[7] Yair Carmon, John C. Duchi, Oliver Hinder, and Aaron Sidford, *Accelerated methods for nonconvex optimization*, SIAM Journal on Optimization **28** (2018), no. 2, 1751–1772.

[8] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, Advances in neural information processing systems, pp. 2933–2941, 2014, arXiv:1406.2572.

[9] Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang, *Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator*, Advances in Neural Information Processing Systems, pp. 689–699, 2018, arXiv:1807.01695.

[10] Cong Fang, Zhouchen Lin, and Tong Zhang, *Sharp analysis for nonconvex SGD escaping from saddle points*, Conference on Learning Theory, pp. 1192–1234, 2019, arXiv:1902.00247.

[11] Yan V. Fyodorov and Ian Williams, *Replica symmetry breaking condition exposed by random matrix calculation of landscape complexity*, Journal of Statistical Physics **129** (2007), no. 5-6, 1081–1116, arXiv:cond-mat/0702601.

[12] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan, *Escaping from saddle points – online stochastic gradient for tensor decomposition*, Proceedings of the 28th Conference on Learning Theory, Proceedings of Machine Learning Research, vol. 40, pp. 797–842, 2015, arXiv:1503.02101.

[13] Rong Ge, Jason D. Lee, and Tengyu Ma, *Matrix completion has no spurious local minimum*, Advances in Neural Information Processing Systems, pp. 2981–2989, 2016, arXiv:1605.07272.

[14] Rong Ge, Jason D. Lee, and Tengyu Ma, *Learning one-hidden-layer neural networks with landscape design*, International Conference on Learning Representations, 2018, arXiv:1711.00501.

[15] Rong Ge and Tengyu Ma, *On the optimization landscape of tensor decompositions*, Advances in Neural Information Processing Systems, pp. 3656–3666, Curran Associates Inc., 2017, arXiv:1706.05598.

[16] Moritz Hardt, Tengyu Ma, and Benjamin Recht, *Gradient descent learns linear dynamical systems*, Journal of Machine Learning Research **19** (2018), no. 29, 1–44, arXiv:1609.05191.

[17] Prateek Jain, Chi Jin, Sham Kakade, and Praneeth Netrapalli, *Global convergence of non-convex gradient descent for computing matrix squareroot*, Artificial Intelligence and Statistics, pp. 479–488, 2017, arXiv:1507.05854.

[18] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan, *How to escape saddle points efficiently*, Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 1724–1732, 2017, arXiv:1703.00887.

[19] Chi Jin, Praneeth Netrapalli, Rong Ge, Sham M. Kakade, and Michael I. Jordan, *On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points*, 2019, arXiv:1902.04811.

[20] Chi Jin, Praneeth Netrapalli, and Michael I. Jordan, *Accelerated gradient descent escapes saddle points faster than gradient descent*, Conference on Learning Theory, pp. 1042–1085, 2018, arXiv:1711.10456.

[21] Michael I. Jordan, *On gradient-based optimization: Accelerated, distributed, asynchronous and stochastic optimization*, `https://www.youtube.com/watch?v=VE2ITg%5FhGnI`, 2017.

[22] Yurii Nesterov and Boris T. Polyak, *Cubic regularization of Newton method and its global performance*, Mathematical Programming **108** (2006), no. 1, 177–205.

[23] Yurii E. Nesterov, *A method for solving the convex programming problem with convergence rate $O(1/k^2)$*, Soviet Mathematics Doklady, vol. 27, pp. 372–376, 1983.

[24] Ju Sun, Qing Qu, and John Wright, *A geometric analysis of phase retrieval*, Foundations of Computational Mathematics **18** (2018), no. 5, 1131–1198, arXiv:1602.06664.

[25] Nilesh Tripuraneni, Mitchell Stern, Chi Jin, Jeffrey Regier, and Michael I. Jordan, *Stochastic cubic regularization for fast nonconvex optimization*, Advances in neural Information Processing Systems, pp. 2899–2908, 2018, arXiv:1711.02838.

[26] Yi Xu, Rong Jin, and Tianbao Yang, *NEON+: Accelerated gradient methods for extracting negative curvature for non-convex optimization*, 2017, arXiv:1712.01033.

[27] Yi Xu, Rong Jin, and Tianbao Yang, *First-order stochastic algorithms for escaping from saddle points in almost linear time*, Advances in Neural Information Processing Systems, pp. 5530–5540, 2018, arXiv:1711.01944.

[28] Dongruo Zhou and Quanquan Gu, *Stochastic recursive variance-reduced cubic regularization methods*, International Conference on Artificial Intelligence and Statistics, pp. 3980–3990, 2020, arXiv:1901.11518.

[29] Dongruo Zhou, Pan Xu, and Quanquan Gu, *Finding local minima via stochastic nested variance reduction*, 2018, arXiv:1806.08782.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See the 'Open questions' part in the introduction.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See the 'Broader impact' part in the introduction.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]