Journal of Computational Mathematics and Data Science 10 (2024) 100090

Contents lists available at ScienceDirect

Journal of Computational Mathematics and Data Science

journal homepage: www.elsevier.com/locate/jcmds

Escaping saddle points efficiently with occupation-time-adapted perturbations

Xin Guo [a], Jiequn Han [b], Mahan Tajrobehkar [a,*], Wenpin Tang [c]

[a] Department of Industrial Engineering and Operations Research, UC Berkeley, Berkeley, CA, USA
[b] Center of Computational Mathematics, Flatiron Institute, NY, USA
[c] Department of Industrial Engineering and Operations Research, Columbia University, New York City, NY, USA

ARTICLE INFO

Keywords:
Vertex-repelling random-walk
Occupation-time-adapted perturbations
Perturbed gradient descent
Perturbed accelerated gradient descent

ABSTRACT

Motivated by the super-diffusivity of self-repelling random walk, which has roots in statistical physics, this paper develops a new perturbation mechanism for optimization algorithms. In this mechanism, perturbations are adapted to the history of states via the notion of occupation time. After integrating this mechanism into the framework of perturbed gradient descent (PGD) and perturbed accelerated gradient descent (PAGD), two new algorithms are proposed: perturbed gradient descent adapted to occupation time (PGDOT) and its accelerated version (PAGDOT). PGDOT and PAGDOT are guaranteed to avoid getting stuck at non-degenerate saddle points, and are shown to converge to second-order stationary points at least as fast as PGD and PAGD, respectively. The theoretical analysis is corroborated by empirical studies in which the new algorithms consistently escape saddle points and outperform not only their counterparts, PGD and PAGD, but also other popular alternatives including stochastic gradient descent, Adam, and several state-of-the-art adaptive gradient methods.

1. Introduction

Gradient descent (GD), which dates back to [1], aims to minimize a function $f : \mathbb{R}^d \to \mathbb{R}$ via the iteration: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t), t = 0, 1, 2, \dots$, where $\eta > 0$ is the step size and ∇f is the gradient of f. Due to its simple form and fine computational properties, GD and its variants (e.g., stochastic gradient descent) are essential for many machine learning tools: principle component analysis [2], phase retrieval [3], and deep neural network [4], just to name a few. In the era of data deluge, many problems are concerned with large-scale optimization in which the intrinsic dimension d is large. GD turns out to be efficient in dealing with high-dimensional convex optimization, where the first-order stationary point $\nabla f(\boldsymbol{x}) = 0$ is necessarily the global minimum point. Algorithmically, it involves finding a point with small gradient $\|\nabla f(\boldsymbol{x})\| < \epsilon$. A classical result of [5] showed that the time required by GD to find such a point in a possibly non-convex problem is of order ϵ^{-2}, independent of the dimension d.

In non-convex settings, applying GD will still lead to an approximate first-order stationary point. This is, however, insufficient: for non-convex functions, first-order stationary points can be either global minimum, local minimum, local maximum, or saddle points. As we will explain, saddle points are the main bottleneck for GD in many non-convex problems. The goal of this paper is therefore to develop efficient algorithms to escape saddle points in high-dimensional non-convex problems, and hence overcome the curse of dimensionality.

Escaping local minima: Inspired by annealing in metallurgy, [6] developed simulated annealing to approximate the global minimum of a given function. [7] proposed a diffusion simulated annealing and proved that it converges to the set of global minimum

* Corresponding author.
E-mail addresses: xinguo@berkeley.edu (X. Guo), jiequnhan@gmail.com (J. Han), mahan_tajrobehkar@berkeley.edu (M. Tajrobehkar), wt2319@columbia.edu (W. Tang).

https://doi.org/10.1016/j.jcmds.2024.100090
Received 22 December 2022; Received in revised form 30 December 2023; Accepted 8 January 2024

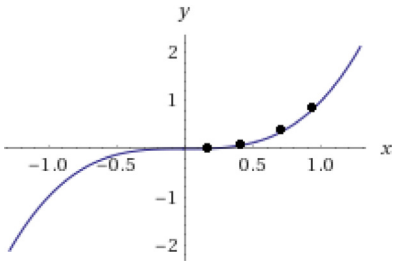

Fig. 1. Illustration of occupation-time-adapted perturbation using $f(x) = x^3$.

points. However, subsequent works [8–13] revealed that it might take an exponentially long time (of order $\exp(d)$) for diffusion simulated annealing to get close to the global minimum. See [14] for a review. Some work, for instance based on Lévy flights [15] or by Cuckoo's search [16] showed empirically faster convergence to the global minimum. Yet the theory of these approaches is far fetched. Closely related to simulated annealing are recent efforts in approximating the global minimum in non-convex problems via Langevin dynamics-based stochastic gradient descent [17,18], where the gradient is evaluated at a randomly selected data point in each iteration. There also exist several variants of Langevin-based stochastic gradient descent using non-reversibility [19] and replica exchange [20,21]. Typically, these algorithms take polynomial time in the dimension d, and thus may scale poorly when d is large.

Escaping saddle points: Fortunately, in many non-convex problems, it suffices to find a local minimum. Indeed, there has been a line of recent work arguing that local minima are less problematic, and that for many non-convex problems there are no spurious local minima. That is, all local minima are comparable in value with the global minimum. Examples include tensor decomposition [22–25], semidefinite programming [26,27], dictionary learning [28], phase retrieval [29], robust regression [30], low-rank matrix factorization [31–34], and certain classes of deep neural networks [35–42]. Nevertheless, as shown in [43–45], saddle points may correspond to suboptimal solutions, and it may take exponentially long time to move from saddle points to a local minimum point. Meanwhile, it has been observed in empirical studies [46,47] that GD and its variants such as stochastic gradient descent (SGD) [48] and Adam [49] may be trapped in saddle points.

[22] took the first step to show that by adding noise at each iteration, GD can escape all saddle points in polynomial time. Additionally, [50,51] proved that with random initialization, GD converges to a local minimizer. Moreover, [52] proposed the perturbed gradient descent (PGD) algorithm, which [53] further improved to the perturbed accelerated gradient descent (PAGD) algorithm. They showed that PGD and PAGD are efficient — the time complexity is almost independent of the dimension d. See also [54] for a summary of results in this direction.

Our idea. Motivated by the "fast exploration" of self-repelling random walk, this paper develops a new perturbation mechanism by adapting the perturbations to the history of states. Recall that [52,54] used the following perturbation update when perturbation conditions hold:

$$\boldsymbol{x}_t' = \boldsymbol{x}_t + \mathrm{Unif}(B^d(\mathbf{0}, r)), \quad \boldsymbol{x}_{t+1} = \boldsymbol{x}_t' - \eta \nabla f(\boldsymbol{x}_t'),$$

where $\mathrm{Unif}(B^d(\mathbf{0}, r))$ is a point picked uniformly in the ball of radius r. On the empirical side, [55,56] applied this idea of GD with noise to train deep neural networks.

Our idea is to replace $\mathrm{Unif}(B^d(\mathbf{0}, r))$ with non-uniform perturbations, whose mechanism depends on the current state $\boldsymbol{x}_t$ and the history of states $\{\boldsymbol{x}_s;\ s \le t\}$. There are conceivably many ways to add non-uniform perturbation based on the current and previous states; here we choose to adapt perturbations to the "occupation time".

The intuition is illustrated by the one-dimensional function $f(x) = x^3$ (see Fig. 1).

There is a saddle point at 0, and imagine GD approaches 0 from the right. It can be shown that GD converges monotonically to a stationary point (see Appendix A). The uniform perturbation will add noise with probability 1/2 both to the right and to the left. To the right, GD will again get stuck at the saddle point 0. However, to the left, there is a possibility of escaping from 0 and finding a local minimum ($-\infty$ in this case). Therefore, it is reasonable to add noise with a larger probability to the left, since it has spent a long time on the right and has yet to explore the left side.

The previous intuition can be quantified via the notion of occupation times L_t (the number of $\{x_s\}_{s<t}$ to the left of x_t) and R_t (the number of $\{x_s\}_{s<t}$ to the right of x_t). By definition, $R_t + L_t = t$, for each $t = 0, 1, \ldots$. If L_t is larger, the perturbation will push the iterate x_t to the right; and if R_t is larger, push to the left. More precisely,

$$x_{t+1} = \begin{cases} x_t - r\,\mathrm{Unif}(0,1) & \text{with probability } p, \\ x_t + r\,\mathrm{Unif}(0,1) & \text{with probability } 1-p, \end{cases} \tag{1}$$

where $p = \frac{w(R_t)}{w(L_t)+w(R_t)}$ and $w : \{0, 1, \ldots\} \to (0, \infty)$ is an increasing weight function on the nonnegative integers (e.g., $w(k) = 1 + k^\alpha$ for $\alpha > 0$).

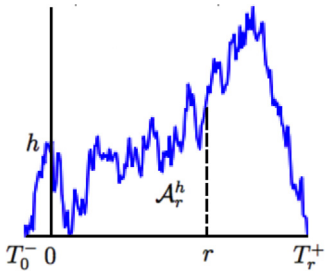

Fig. 2. Illustration of $\mathcal{A}_r^h$.

The dynamics (1) is closely related to the vertex-repelling random walk defined by

$$Z_{t+1} = \begin{cases} Z_t - 1 & \text{with probability } \frac{w(\widetilde{R}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}, \\ Z_t + 1 & \text{with probability } \frac{w(\widetilde{L}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}, \end{cases} \tag{2}$$

where $\widetilde{R}_t := \{s < t : Z_s = Z_t + 1\}$ and $\widetilde{L}_t := \{s < t : Z_s = Z_t - 1\}$. This (non-Markovian) random walk model was introduced by [57] in the statistical physics literature. Based on the scaling arguments and simulations, it was conjectured that for $w(\cdot)$ with a suitable growth, the walk $(Z_t, t \geq 0)$ is recurrent and is further super-diffusive in the sense that $\mathbb{E}Z_t^2 \sim t^{\frac{4}{3}}$, whereas for a simple random walk $(S_t, t \geq 0)$ its exploration range is $\mathbb{E}S_t^2 \sim t \ll t^{\frac{4}{3}}$. These properties have only been proved rigorously for a simpler variant — the edge-repelling random walk, see [58–61]. For instance, it was conjectured that for $w(k) \sim \lambda^k$ with $\lambda > 1$,

$$\left(\frac{Z_{nu}}{(\sigma n)^{\frac{2}{3}}}, u \geq 0 \right) \text{ converges in distribution to } (\mathcal{Z}_u, u \geq 0) \quad \text{as } n \to \infty,$$

where $\sigma > 0$ is a variance parameter depending on $w(\cdot)$, and the scaling limit $(\mathcal{Z}_u, u \geq 0)$ is a (universal) continuous process whose *marginal* distribution $p(u, \cdot)$ is given as follows. Let $\left(|B_s^h|, s \in \mathbb{R}\right)$ be a two-sided reflected Brownian motion with $|B_0^h| = h > 0$. Define

$$T_0^- := \sup\{s < 0 : |B_s^h| = 0\} \quad \text{and} \quad T_r^+ := \sup\{s > r : |B_s^h| = 0\} \text{ for } r > 0,$$

and

$$\mathcal{A}_r^h := \int_{T_0^-}^{T_r^+} |B_s^h| ds \quad (\text{area of } |B^h| \text{ between } T_0^- \text{ and } T_r^+).$$

See Fig. 2 for an illustration.

Denote by $\rho(u, r; h) := \frac{\partial}{\partial u}\mathbb{P}\left(\mathcal{A}_r^h \leq u\right)$ the density of $\mathcal{A}_r^h$. Then

$$p(u, x) = \int_0^\infty \rho\left(\frac{u}{2}, |x|; h\right) dh \quad \text{for } x \in \mathbb{R}. \tag{3}$$

Clearly, the conjecture implies that $\mathbb{E}Z_t^2$ is of order $t^{\frac{4}{3}}$. It is easy to see from (3) that $\mathcal{Z}_u$ has the same distribution as $r^{-\frac{2}{3}}\mathcal{Z}_{ru}$ for each $u > 0$. However, the distribution of $(\mathcal{Z}_u, u \geq 0)$ at the *process level* remains open to date, and we even do not know whether the process $\mathcal{Z}$ is a diffusion process. A counterpart to the vertex-repelling walk is the vertex-reinforced walk [62,63] defined by $Z_{t+1} = Z_t - 1$ with probability $\frac{w(\widetilde{L}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}$, and $Z_{t+1} = Z_t + 1$ with probability $\frac{w(\widetilde{R}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}$. It is well known [63,64] that vertex-reinforced random walk exhibits localization at a finite number of points for some choices of $w(\cdot)$, e.g., $w(k) \sim k^\alpha$ with $\alpha \geq 1$.

Our results. We will first show that vertex-repelling walk will never be localized or stuck at some points in contrast with vertex-reinforced walk (see Theorem 3.1). The non-localization and the (conjectured) super-diffusive properties of the vertex-repelling walk (2) facilitate exploration, and thus the corresponding perturbation scheme (1) makes it more likely to escape from saddle points.

We will then propose a new perturbation mechanism based on the dynamics (1), which can be integrated into the framework of (any) perturbation-based optimization algorithms. In particular, integrating the above-mentioned mechanism into the framework of PGD and PAGD, we propose two new algorithms: perturbed gradient descent adapted to occupation time (PGDOT, Algorithm 1) and its accelerated version, perturbed accelerated gradient descent adapted to occupation time (PAGDOT, Algorithm 2).

We will prove that Algorithm 1 (resp. Algorithm 2) converges to a second-order stationary point at least as fast as PGD (resp. PAGD). Algorithms 1 and 2 are state-dependent adaptive algorithms, perturbing GD and accelerated gradient descent (AGD) [65] non-uniformly according to the history of states.

We will finally corroborate our theoretical analysis by experimental results. Specifically, we will empirically demonstrate that Algorithms 1 and 2 exhibit faster escape from saddle points, surpassing not only their counterparts (PGD and PAGD) but also outperforming SGD and several adaptive gradient methods, including Adam, AMSGrad [66], AdaBelief [67], and STORM [68] in training deep learning models on popular datasets such as MNIST [69], CIFAR-10 [70], and CIFAR-100 [70].

Algorithm 1 Perturbed Gradient Descent Adapted to Occupation Time (Meta Algorithm)

for $t = 0, 1, \dots$ **do**
  if perturbation condition holds **then**
    for $i = 1, \dots, d$ **do**
      $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$
      $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$
      $$x_t^i \leftarrow \begin{cases} x_t^i - \frac{r}{\sqrt{d}}\,\mathrm{Unif}(0,1) & \text{w.p.} \quad p, \\ x_t^i + \frac{r}{\sqrt{d}}\,\mathrm{Unif}(0,1) & \text{w.p.} \quad 1-p, \end{cases}$$
      where $p = \frac{w(R_t^i)}{w(L_t^i)+w(R_t^i)}$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t)$

Algorithm 2 Perturbed Accelerated Gradient Descent Adapted to Occupation Time (Meta Algorithm)

for $t = 0, 1, \dots,$ **do**
  if perturbation condition holds **then**
    for $i = 1, \dots, d$ **do**
      $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$
      $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$
      $$x_t^i \leftarrow \begin{cases} x_t^i - \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p.} \quad p, \\ x_t^i + \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p.} \quad 1-p, \end{cases} \quad \text{where } p = \frac{w(R_t^i)}{w(L_t^i)+w(R_t^i)}$$
  $\boldsymbol{x}_{t+1} \leftarrow \mathrm{Accelerate}(\boldsymbol{x}_t, \boldsymbol{v}_t), \quad \boldsymbol{v}_{t+1} \leftarrow \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$

Notations. Below we collect the notations that will be used throughout this paper. For S a finite set, let $\#S$ denote the number of elements in S. For D as a domain, let $\mathrm{Unif}(D)$ be the uniform distribution on D, e.g., $\mathrm{Unif}(0,1)$ is the uniform distribution on $[0,1]$. For a function $f : \mathbb{R}^d \to \mathbb{R}$, let ∇f and $\nabla^2 f$ denote its gradient and Hessian, and $f^\star := \min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$ denote its global minimum. For $\boldsymbol{A}$ a symmetric matrix, let $\lambda_{\min}(\boldsymbol{A})$ be its minimum eigenvalue.

The notation $\|\cdot\|$ is used for both the Euclidean norm of a vector and the spectral norm of a matrix. For $\boldsymbol{x} = (x^1, \dots, x^d)$ and $r > 0$, let $B^d(\boldsymbol{x}, r) := \{\boldsymbol{y} : \|\boldsymbol{y} - \boldsymbol{x}\| \le r\}$ be the d-dimensional ball centered at $\boldsymbol{x}$ with radius r, and $C^d(\boldsymbol{x}, r) := \{\boldsymbol{y} : |y^i - x^i| \le r \text{ for } 1 \le i \le d\}$ be the d-dimensional hypercube centered at $\boldsymbol{x}$ with distance r to each of its surfaces. We use the symbol $O(\cdot)$ to hide only absolute constants which do not depend on any problem parameter.

The rest of the paper is organized as follows. Section 2 provides background on the continuous optimization and recalls some existing results. Section 3 presents the main results. Section 4 contains numerical experiments to corroborate our analysis. Section 5 concludes.

2. Background and existing results

2.1. Results of GD

We consider non-convex optimization (convex optimization results are recalled in Appendix B). In this case, it is generally difficult to find the global minima. A popular approach is to consider the first-order stationary points instead.

Definition 2.1. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. We say that (i) $\boldsymbol{x}$ is a first-order stationary point of f if $\nabla f(x) = 0$; (ii) $\boldsymbol{x}$ is an ϵ-first-order stationary point of f if $\|\nabla f(x)\| \le \epsilon$.

We say that a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz if $\|\nabla f(\boldsymbol{x}_1) - \nabla f(\boldsymbol{x}_2)\| \le \ell \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$. For gradient Lipschitz functions, GD converges to the first-order stationary points, which is quantified by the following theorem from [5][Section 1.2.3].

Theorem 2.2. *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz. For any $\epsilon > 0$, if we run GD with step size $\eta = \ell^{-1}$, then the number of iterations to find an ϵ-first-order stationary point is $\frac{\ell(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^2}$.*

Note that in Theorem 2.2, the time complexity of GD is independent of the dimension d. For a non-convex function, a first-order stationary point can be either a local minimum, a saddle point, or a local maximum. The following definition is taken from [52][Definition 4].

Definition 2.3. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. We say that (i) $\boldsymbol{x}$ is a local minimum if $\boldsymbol{x}$ is a first-order stationary point, and $f(\boldsymbol{x}) \le f(\boldsymbol{y})$ for all $\boldsymbol{y}$ in some neighborhood of $\boldsymbol{x}$; (ii) $\boldsymbol{x}$ is a saddle point if $\boldsymbol{x}$ is a first-order stationary point but not a local minimum. Assume further that f is twice differentiable. We say a saddle point $\boldsymbol{x}$ is strict if $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) < 0$.

For a twice differentiable function f, note that $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \le 0$ for any saddle point $\boldsymbol{x}$. So by assuming a saddle point $\boldsymbol{x}$ to be strict, we rule out the case $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) = 0$. The next subsection will review two perturbation-based algorithms that allow jumping out of strict saddle points.

2.2. Results of PGD and PAGD

One drawback of GD in non-convex optimization is that it may get stuck at saddle points. [52,53] proposed PGD and PAGD, respectively, to escape saddle points, which we review here. To proceed further, we need some vocabulary regarding the Hessian of the function f.

Definition 2.4. A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is ρ-Hessian Lipschitz if $\|\nabla^2 f(\boldsymbol{x}_1) - \nabla^2 f(\boldsymbol{x}_2)\| \le \rho \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$. Furthermore, we say that (i) $\boldsymbol{x}$ is a second-order stationary point of f if $\nabla f(\boldsymbol{x}) = 0$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \ge 0$; (ii) $\boldsymbol{x}$ is a ϵ-second-order stationary point of f if $\|\nabla f(\boldsymbol{x})\| \le \epsilon$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \ge -\sqrt{\rho\epsilon}$.

To simplify the presentation, assume that all saddle points are strict (Definition 2.3). In this situation, all second-order stationary points are local minima. The basic idea of these two algorithms is as follows. Imagine that we are currently at an iterate $\boldsymbol{x}_t$ which is not an ϵ-second-order stationary point. There are two scenarios: (i) The gradient $\|\nabla f(\boldsymbol{x}_t)\|$ is large and a usual iteration of GD or AGD is enough; (ii) The gradient $\|\nabla f(\boldsymbol{x}_t)\|$ is small but $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}_t)) \le -\sqrt{\rho\epsilon}$ (large negative). So $\boldsymbol{x}_t$ is around a saddle point, and a perturbation ξ is needed to escape from the saddle region: $\widetilde{\boldsymbol{x}}_t = \boldsymbol{x}_t + \xi$.

The main result for PGD, Theorem 3 in [52], and for PAGD, Theorem 3 in [53], are stated below showing that the time complexity of these two algorithms are almost dimension-free (with a log factor).

Theorem 2.5 (*[52]*). *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz and ρ-Hessian Lipschitz. Then there exists $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^*$, and $c \le c_{\max}$, PGD outputs an ϵ-second-order stationary point with probability $1 - \delta$, terminating within the following number of iterations:*

$$O\left(\frac{\ell(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^2} \log^4\left(\frac{d\ell\Delta_f}{\epsilon^2\delta}\right)\right).$$

Compared with Theorem 2.2, PGD takes almost the same order of time to find a second-order stationary point as GD does to find a first-order stationary point.

Theorem 2.6 (*[53]*). *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz and ρ-Hessian Lipschitz. Then there exists an absolute constant $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^*$, and $c \ge c_{\max}$, with probability $1 - \delta$, one of the iterates $\boldsymbol{x}_t$ of PAGD will be an ϵ-second-order stationary point in the following number of iterations:*

$$O\left(\frac{\ell^{1/2}\rho^{1/4}(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^{7/4}} \log^6\left(\frac{d\ell\Delta_f}{\rho\epsilon\delta}\right)\right).$$

3. Main results

In this section, we first prove the non-localization property of the vertex-repelling random walk. Then, we formalize the idea of perturbations adapted to occupation time and provide the full version of PGDOT and PAGDOT in Algorithms 3 and 4, respectively. Our main results show that these algorithms converge rapidly to second-order stationary points.

3.1. Non-localization property of vertex-repelling random walk

The following theorem suggests that the new perturbation mechanism helps perturbation-based algorithms to avoid getting stuck at saddle points, as the dynamics of vertex-repelling random walk prescribed in (1) does not localize.

Theorem 3.1. *Let $\{Z_t, t = 0, 1, \ldots\}$ be the vertex-repelling random walk defined by (2), where $w : \{0, 1, \ldots\} \to (0, \infty)$ is an increasing function such that $w(n) \to \infty$ as $n \to \infty$. Then*

$$\mathbb{P}\left(\exists t_0 > 0,\ k \le \ell : Z_t \in \{k, \ldots, \ell\} \text{ for all } t \ge t_0\right) = 0.$$

The proof of this theorem is given in Appendix C.

3.2. Perturbed gradient descent adapted to occupation time

PGD adds a uniform random perturbation when stuck at saddle points. From the discussion in the introduction, it is more reasonable to perturb with non-uniform noise whose distribution depends on the occupation times. Recall that $w : \{0, 1, \ldots\} \to (0, \infty)$

is an increasing weight function on the nonnegative integers. The following algorithm adapts PGD to random perturbation depending on the occupation dynamics. We follow the parameter setting as in [52]. Our algorithm performs GD with step size η and gets a perturbation of amplitude $\frac{r}{\sqrt{d}}$ near saddle points at most once every t_{thres} iterations. The threshold t_{thres} ensures that the dynamics of the algorithm is mostly GD. The threshold g_{thres} determines if a perturbation is needed, and the threshold f_{thres} decides when the algorithm terminates.

Algorithm 3 Perturbed Gradient Descent Adapted to Occupation Time: PGDOT($\boldsymbol{x}_0, \ell, \rho, \epsilon, c, \delta, \Delta_f, w$)

$\chi \leftarrow 3\max\left\{\log(\frac{d\ell\Delta_f}{c\epsilon^2\delta}), 4\right\}$, $\eta \leftarrow \frac{c}{\ell}$, $r \leftarrow \frac{\epsilon\sqrt{c}}{\chi^2\ell}$

$g_{\text{thres}} \leftarrow \frac{\epsilon\sqrt{c}}{\chi^2}$, $f_{\text{thres}} \leftarrow \frac{c}{\chi^3}\sqrt{\frac{\epsilon^3}{\rho}}$, $t_{\text{thres}} \leftarrow \frac{\chi\ell}{c^2\sqrt{\rho\epsilon}}$

$t_{\text{noise}} \leftarrow -t_{\text{thres}} - 1$

for $t = 0, 1, \ldots$ **do**

  if $||\nabla f(\boldsymbol{x}_t)|| \le g_{\text{thres}}$ and $t - t_{\text{noise}} > t_{\text{thres}}$ **then**

    $\widetilde{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t$, $t_{\text{noise}} \leftarrow t$

    for $i = 1, \ldots, d$ **do**

      $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$

      $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$

      $$x_t^i \leftarrow \begin{cases} \widetilde{x}_t^i - \frac{r}{\sqrt{d}}\text{Unif}(0,1) & \text{w.p.} \quad p, \\ \widetilde{x}_t^i + \frac{r}{\sqrt{d}}\text{Unif}(0,1) & \text{w.p.} \quad 1-p, \end{cases} \quad \text{where } p = \frac{w(R_t^i)}{w(L_t^i)+w(R_t^i)}$$

  if $t - t_{\text{noise}} = t_{\text{thres}}$ and $f(\boldsymbol{x}_t) - f(\widetilde{\boldsymbol{x}}_{t_{\text{noise}}}) > -f_{\text{thres}}$ **then**

    return $\widetilde{\boldsymbol{x}}_{t_{\text{noise}}}$

  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta\nabla f(\boldsymbol{x}_t)$

The next theorem gives the convergence rate of Algorithm 3: PGDOT finds a second-order stationary point in the same number of iterations (up to a constant factor) as PGD does.

Theorem 3.2. *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz and ρ-Hessian Lipschitz. Then there exists $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^*$, and $c \le c_{\max}$, PGDOT (Algorithm 3) outputs an ϵ-second-order stationary point with probability $1 - \delta$ terminating within the following number of iterations:*

$$O\left(\frac{\ell(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^2}\log^4\left(\frac{d\ell\Delta_f}{\epsilon^2\delta}\right)\right).$$

The proof of Theorem 3.2 is based on a geometric characterization of saddle points — thin pancake property [52]. In Appendix D, we will discuss this property, and show how it is used to prove Theorem 3.2.

3.3. Perturbed accelerated gradient descent adapted to occupation time

Similar to the way we combined our perturbation mechanism with PGD, we can adapt PAGD to this mechanism as well resulting in the accelerated version of PGDOT (Algorithm 4). We follow the parameter setting as in [53].

Algorithm 4, similar to PAGD, employs a feature called Negative Curvature Exploitation (NCE) which resets the momentum and decides whether to exploit the negative curvature when the function becomes "too convex". See [53][Algorithm 3].

The next theorem gives the convergence rate of Algorithm 4: PAGDOT finds a second-order stationary point in the same number of iterations (up to a constant factor) as PAGD does, and therefore achieves a faster convergence rate than PGD and PGDOT. The proof of Theorem 3.3 is similar to that of Theorem 3.2.

Theorem 3.3. *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz and ρ-Hessian Lipschitz. Then there exists an absolute constant $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^*$, and $c \ge c_{\max}$, one of the iterates $\boldsymbol{x}_t$ of PAGDOT (Algorithm 4) will be an ϵ-second-order stationary point in the following number of iterations, with probability $1 - \delta$:*

$$O\left(\frac{\ell^{1/2}\rho^{1/4}(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^{7/4}}\log^6\left(\frac{d\ell\Delta_f}{\rho\epsilon\delta}\right)\right).$$

It is worth mentioning that the new algorithms can be regarded as generalizations of PGD and PAGD. This is exemplified by the fact that when the weight function w is a constant function (e.g. $w(k) = 1, \forall k \ge 0$), the perturbations become uniform. Additionally, Algorithms 3 and 4 share some spirit with GD with momentum methods such as the heavy ball method [71]. In the heavy ball method, a momentum term, which is a function of the current and previous states, is explicitly added to control the oscillations and accelerate in low curvatures along the direction close to momentum. In Algorithms 3 and 4, however, no explicit momentum term is added. Instead, the perturbation is adapted to the history of states providing the current state with an explicit direction.

Algorithm 4 Perturbed Accelerated Gradient Descent Adapted to Occupation Time: PAGDOT($\boldsymbol{x}_0, \eta, \theta, \gamma, s, r, \mathscr{T}, w$)

$\chi \leftarrow \max\left\{\log(\frac{d\ell\Delta_f}{\rho\epsilon\delta}), 1\right\}$, $\kappa \leftarrow \frac{\ell}{\sqrt{\rho\epsilon}}$, $\eta \leftarrow \frac{1}{4\ell}$
$\theta \leftarrow \frac{1}{4\sqrt{\kappa}}$, $\gamma \leftarrow \frac{\theta^2}{\eta}$, $s \leftarrow \frac{\gamma}{4\rho}$, $r \leftarrow \frac{\eta\epsilon}{\chi^5 c^8}$, $\mathscr{T} \leftarrow \chi c\sqrt{\kappa}$
$\boldsymbol{v}_0 \leftarrow 0$
for $t = 0, 1, \ldots,$ **do**
  if $\|\nabla f(\boldsymbol{x}_t)\| \le \epsilon$ and *no perturbation in last* $\mathscr{T}$ *steps* **then**
    $\widetilde{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t$
    for $i = 1, \ldots, d$ **do**
      $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$
      $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$
      $x_t^i \leftarrow \begin{cases} \widetilde{x}_t^i - \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p.} \quad p, \\ \widetilde{x}_t^i + \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p.} \quad 1-p, \end{cases}$ where $p = \frac{w(R_t^i)}{w(L_t^i)+w(R_t^i)}$
  $\boldsymbol{y}_t \leftarrow \boldsymbol{x}_t + (1-\theta)\boldsymbol{v}_t$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{y}_t - \eta\nabla f(\boldsymbol{y}_t)$
  $\boldsymbol{v}_{t+1} \leftarrow \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$
  if $f(\boldsymbol{x}_t) \le f(\boldsymbol{y}_t) + \langle\nabla f(\boldsymbol{y}_t), \boldsymbol{x}_t - \boldsymbol{y}_t\rangle - \frac{\gamma}{2}\|\boldsymbol{x}_t - \boldsymbol{y}_t\|^2$ **then**
    $(\boldsymbol{x}_{t+1}, \boldsymbol{v}_{t+1}) \leftarrow \mathrm{NCE}(\boldsymbol{x}_t, \boldsymbol{v}_t, s)$

We also remark that Theorem 3.2 (Theorem 3.3) has been proven using the exact same steps and lemmas as the ones employed in proving Theorem 2.5 (Theorem 2.6). As a result, the convergence rate of PGDOT (PAGDOT) is identical to that of PGD (PAGD). This equivalence extends to the hidden constants as well. However, we believe that by exploiting the intrinsic properties of the embedded perturbation mechanism, such as the super-diffusivity of the corresponding random walk, there is a possibility to enhance the theoretical results. Nonetheless, formally establishing this improvement remains an open question.

4. Empirical results

This section presents empirical results to corroborate the theoretical analysis presented in the previous section. The experiments showcase the effectiveness of our new perturbation-based algorithms, not only in escaping saddle points but also in doing so efficiently and rapidly. To comprehensively evaluate their performance, we consider both small-scale and large-scale problems to demonstrate the practicality and scalability of the new algorithms. Here, we compare the algorithms on a per-iteration basis. Refer to Appendix G for the runtime comparison.

The small-scale problems include a synthetic problem, a nonlinear regression problem, a regularized quadratic problem, and a phase retrieval problem, wherein we compare the new algorithms with their counterparts, PGD and PAGD, and vanilla GD. In the large-scale problems, we focus on image classification tasks using popular datasets such as MNIST [69], CIFAR-10 [70], and CIFAR-100 [70]. We compare the performance of the new algorithms against their counterparts, as well as SGD [48] and several state-of-the-art adaptive gradient algorithms including Adam [49], AMSGrad [66], AdaBelief [67], and STORM [68].

In these experiments, we use $L_t^i(h) := \#\{t - t_{\text{count}} \le s < t : x_t^i - h \le x_s^i \le x_t^i\}$ and $R_t^i(h) := \#\{t - t_{\text{count}} \le s < t : x_t^i < x_s^i \le x_t^i + h\}$ instead of L_t^i and R_t^i in Algorithms 3 and 4. Here h is a hyperparameter characterizing the occupation time over a small interval. t_{count} is another hyperparameter prescribing how long one should keep track of the history of $\boldsymbol{x}_t$ in order to approximate the occupation time with a constant memory cost. All the hyperparameter settings used in the experiments and their tuning process are reported in Appendix E. The small-scale problems are run on a commodity machine with Intel® Core™ i7-7500U CPU. The image classification tasks are run on Google Colab, utilizing NVIDIA A100 GPU in the High-RAM setting.

Stair function problem. Given $N \in \mathbb{Z}^+, L \in \mathbb{R}^+$, define a function $\tilde{f} : \mathbb{R}^+ \to \mathbb{R}^+$ as

$$\tilde{f}(r) = \begin{cases} r^3, & r \in [0, \frac{1}{2}L), \\ (r - nL)^3 + \frac{1}{4}nL^3, & r \in [a(n), b(n)), 1 \le n \le N, \\ (r - NL)^3 + \frac{1}{4}NL^3, & r \in [NL + \frac{1}{2}L, \infty), \end{cases}$$

where $a(n) = nL - \frac{1}{2}L$ and $b(n) = nL + \frac{1}{2}L$. For $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, we define $f(\boldsymbol{x}) = \tilde{f}\left(\frac{1}{d}\sum_{i=1}^d x_i^2\right)$.

Fig. 3 presents the visualization of the case $N = 4, L = 1$ as well as the training curves of f given by 5 different algorithms when $d = 4$. The initial values are all the same, and all the algorithms except for GD are run 3 times considering the randomness of perturbations. We observe that while GD becomes trapped at saddle points, all other algorithms successfully navigate away from these points. Furthermore, the newly introduced algorithms, PGDOT and PAGDOT, outperform their respective counterparts.

Nonlinear regression problem. We consider a nonlinear regression problem, adapted from learning time series data with a continuous dynamical system [72]. The loss function is defined as $f(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^N (\hat{y}(s_i; \boldsymbol{x}) - y^*(s_i))^2$, where $\{s_i\}_{i=1}^N$ are N sample points, $y^*(s)$ is the

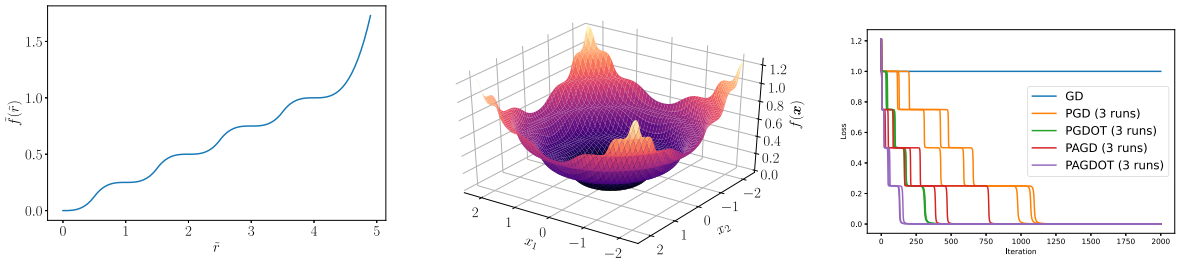

Fig. 3. Graph of $\tilde{f}$ (left) and the landscape of $f(\boldsymbol{x})$ (middle) with $\boldsymbol{x} \in \mathbb{R}^2$ in the case of $N = 4, L = 1$. The figure on the right shows the performance of different algorithms in the stair function problem when $N = 4, L = 1, d = 4$.

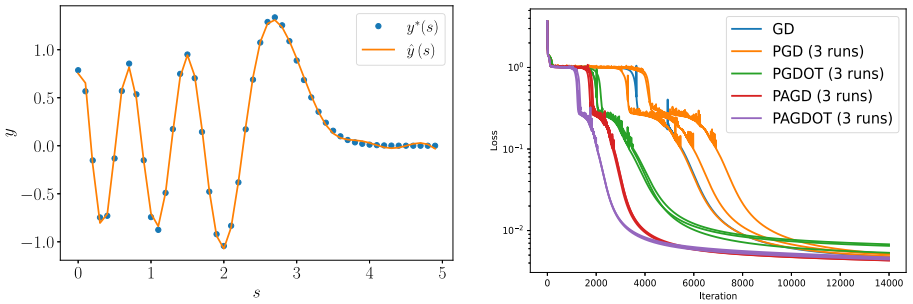

Fig. 4. The target function $y^*(t)$ and fitted function $\hat{y}(t)$ obtained by PGDOT (left), and the performance of different algorithms (right) in the nonlinear regression problem.

target function, and $\hat{y}(s)$ is the function to fit with the form $\hat{y}(s;\boldsymbol{x}) = \sum_{m=1}^{M}(a_m \cos(\lambda_m s) + b_m \sin(\lambda_m s))e^{w_m s}$. Here $\boldsymbol{x} = \{a_m, b_m, \lambda_m, w_m\}_{m=1}^{M}$ and the optimization problem is non-convex. We assume $y^*(s) = \mathrm{Ai}(\omega[s - s_0])$, where $\omega = 3.2, s_0 = 3.0$, and $\mathrm{Ai}(s)$ is the Airy function of the first kind, given by the improper integral $\mathrm{Ai}(s) = \frac{1}{\pi}\int_0^\infty \cos\left(\frac{u^3}{3} + su\right) \mathrm{d}u$.

For the specific regression model, we assume $M = 4$ and use $N = 50$ data points with $s_i = i/10, i = 0, \ldots, 49$. $\{a_m, b_m, \lambda_m\}_{m=1}^{4}$ are initialized via $\mathcal{N}(0,1)$ and $\{w_m\}_{m=1}^{4}$ are initialized via $\mathrm{Unif}(-2,-0.2)$. Fig. 4 shows the target function and the fitted function obtained by PGDOT. Also, the learning curves of 5 different algorithms are plotted. We observe that PGDOT and PAGDOT escape the saddle point faster than GD and outperform PGD and PAGD, respectively.

The next two non-convex optimization problems are taken from [73]. In both problems, all other algorithms escape saddle points faster than GD, with PGDOT and PAGDOT slightly outperforming their counterparts.

Regularized quadratic problem. The first problem is a regularized quadratic problem [74], in which the loss function is defined as

$$f_1(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T H \boldsymbol{x} + \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{b}_i^T \boldsymbol{x} + \|\boldsymbol{x}\|_{10}^{10},$$

where we take $N = 10$, $H = \mathrm{diag}([1, -0.1])$, and $\boldsymbol{b}_i$'s instances of $\mathcal{N}(0, \mathrm{diag}([0.1, 0.001]))$. We initialize this problem at $\boldsymbol{x}_0 = 0$. Different algorithms' performance are shown in Fig. 5.

Phase retrieval problem. The second problem is the phase retrieval problem [75] with loss function

$$f_2(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N}((\boldsymbol{a}_i^T \boldsymbol{x})^2 - (\boldsymbol{a}_i^T \boldsymbol{x}^*)^2)^2,$$

where we choose $N = 200$, $\boldsymbol{x}^*$ an instance of $\mathcal{N}(0, I_d/d)$, and $\boldsymbol{a}_i$'s instances of $\mathcal{N}(0, I_d)$ with $d = 10$. We initialize the problem at $\boldsymbol{x}_0$ sampled from $\mathcal{N}(0, I_d/(10000d))$. Fig. 5 presents the learning curves of different algorithms.

Image classification task. [46] observed that in training multilayer perceptrons (MLPs) on MNIST and CIFAR-10, SGD might get stuck at saddle points. Additionally, [47] demonstrated that Adam gets stuck and performs poorly when MLPs with certain weight initialization are trained on the MNIST dataset. In the following, we conduct image classification task on the MNIST, CIFAR-10, and CIFAR-100 datasets using various deep learning models. The purpose of these experiments is to deliberately subject the algorithms to saddle points through specific weight initialization schemes, drawing inspiration from the abovementioned works.

In each experiment, we compare PGDOT and PAGDOT with PGD, PAGD, SGD, Adam, AMSGrad, AdaBlief, and STORM. The batch size is set as 128 for all algorithms across all experiments. It is worth mentioning that SGD and the adaptive gradient methods exhibit poor performance across all experiments, leading to nearly identical plots. As a result, we only present the training curves of SGD and Adam in this section, while the performance results of AMSGrad, AdaBelief, and STORM can be found in Appendix F.

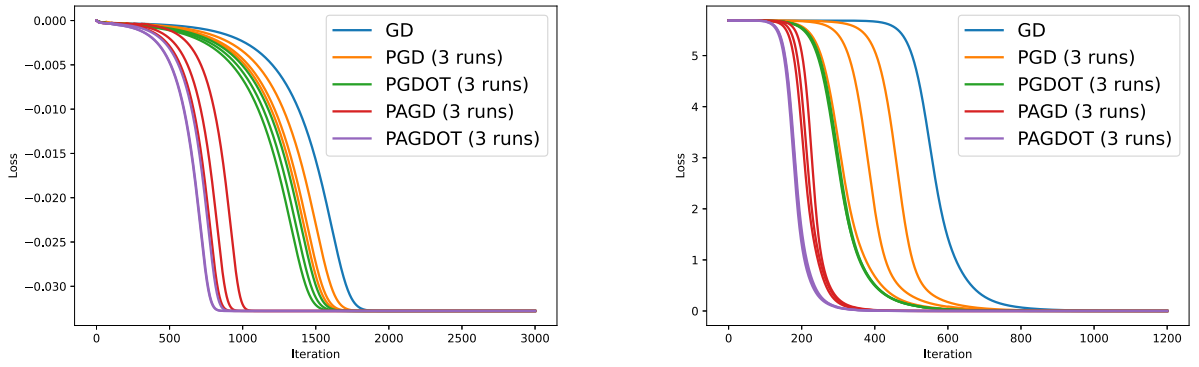

Fig. 5. The performance of different algorithms in the regularized quadratic problem (left) and the phase retrieval problem (right).

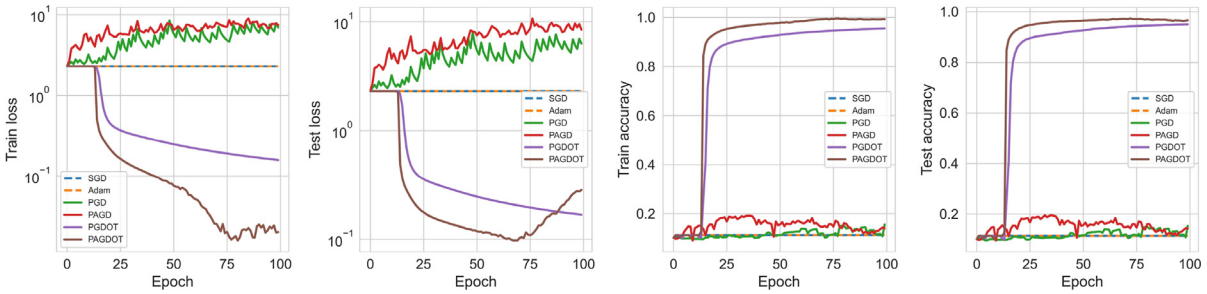

Fig. 6. Experimental results of training the MLP model on MNIST using various algorithms.

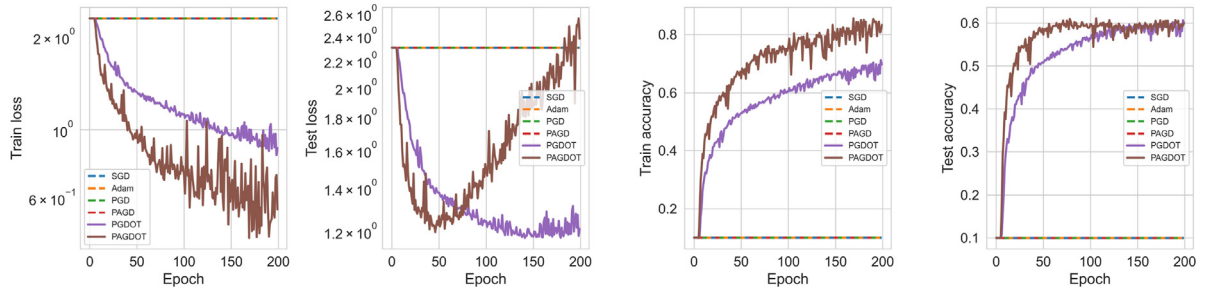

Fig. 7. Experimental results of training SqueezeNet on CIFAR-10 using various algorithms.

In the first experiment, we train an MLP with one hidden layer of size 100 on the MNIST dataset. We use Rectified Linear Unit (ReLu) as the activation function of the neurons in the hidden layer. The MLP has 3 layers of size (28×28)-100-10 totaling in 79,510 parameters considering all the weights and biases. We initialize the weights and biases with $\mathcal{N}(-1, 0.1)$ and run the algorithms for 100 epochs. Fig. 6 shows the train and test losses and accuracy results of different algorithms.

In the second experiment, we train SqueezeNet [76] on the CIFAR-10 dataset. We set the number of output channels in the last convolutional layer as 10 to account for the 10 different classes existing in CIFAR-10. In total, the model has 740,554 parameters. The weights and biases of the convolutional layers are initialized with $\mathcal{N}(-1, 0.1)$ and the algorithms are run for 200 epochs. Fig. 7 shows the train and test losses and accuracy results of different algorithms.

In the third experiment, we train ResNet18 [77] on the CIFAR-100 dataset. We set the number of output channels in the last linear layer as 100 to account for the 100 different classes existing CIFAR-100. In total, the model has 11,227,812 parameters. This time, we initialize the weights and biases of the batch normalization modules (as denoted by γ and β in [78]) to 0 and run the algorithms for 50 epochs. Fig. 8 shows the train and test losses and accuracy results of different algorithms.

From the results obtained, it is evident that both SGD and the adaptive gradient methods exhibit poor performance across all experiments. Despite the presence of inherent noise in the mini-batch gradient, these methods fail to navigate away from saddle points effectively. Conversely, the new algorithms demonstrate the ability to successfully escape saddle points and achieve significant reductions in loss. Moreover, when comparing the new algorithms with their counterparts, it becomes apparent that the newly introduced perturbation mechanism holds a distinct advantage over the uniform perturbation utilized in PGD and PAGD. It is

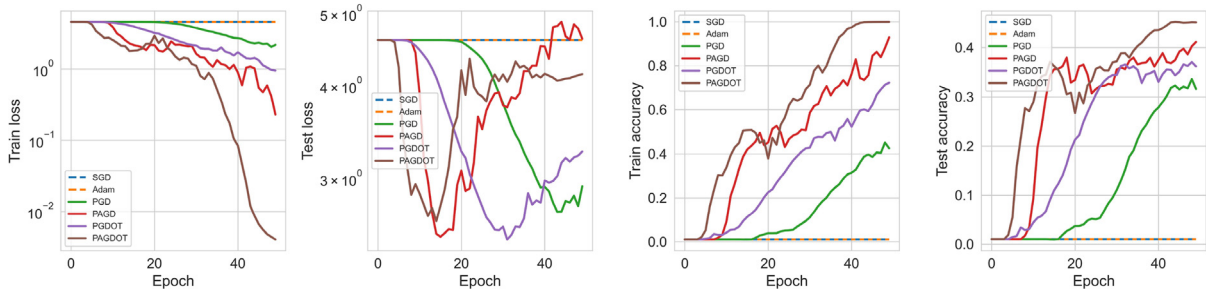

Fig. 8. Experimental results of training ResNet18 on CIFAR-100 using various algorithms.

important to highlight that the observed generalization gap in CIFAR-10 and CIFAR-100 can be mainly attributed to the specific initialization scheme employed.

5. Conclusion

In this paper, we develop a new perturbation mechanism in which the perturbations are adapted to the history of states via the notion of occupation time. This mechanism is integrated into the framework of PGD and PAGD resulting in two new algorithms: PGDOT and PAGDOT. We prove that PGDOT and PAGDOT converge rapidly to second-order stationary points, which is corroborated by empirical studies ranging from time series analysis and the phase retrieval problem to deep learning.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Monotone convergence of gradient descent

Here we prove a property of gradient descent applied to a function $f : \mathbb{R} \to \mathbb{R}$, as mentioned in the introduction. This property of gradient descent supports the use of our proposed perturbation mechanism.

Proposition A.1. *Let $f \in C^2(\mathbb{R})$. Assume that we start gradient descent at some arbitrary point x_0, and the corresponding iterates $\{x_n\}_{n\geq 0}$ converge to the point x_s with $f''(x_s) \neq 0$. Then, if f is ℓ-gradient Lipschitz and the step size is less than $\frac{1}{\ell}$, the sequence $\{x_n\}_{n\geq 0}$ converges monotonically to x_s.*

In order to prove this proposition, we break it down into two lemmas.

Lemma A.2. *Let $f \in C^2(\mathbb{R})$. Assume that we start gradient descent at some arbitrary point x_0, and the corresponding iterates $\{x_n\}_{n\geq 0}$ converge to the point x_s with $f''(x_s) \neq 0$. Then, if f is ℓ-gradient Lipschitz and the step size is less than $\frac{1}{\ell}$, there exists $M > 0$ such that the sequence $\{x_n\}_{n\geq M}$ converges monotonically to x_s.*

Proof. Note that for $n \geq 0$, $x_{n+1} = x_n - \eta f'(x_n)$, where $0 < \eta < \frac{1}{\ell}$ is the step size. Also, it is easy to show that $f'(x_s) = 0$. Assume that at some point $x_n \geq x_s$. Then, since $|f'(x_n) - f'(x_s)| = |f'(x_n)| \leq \ell |x_n - x_s|$, we have

$$\begin{aligned} x_s \leq x_n - \frac{1}{\ell}|f'(x_n)| &\leq x_n - \eta |f'(x_n)| \\ &\leq x_n - \eta f'(x_n) = x_{n+1}. \end{aligned}$$

Similarly, if $x_n \leq x_s$, then we get $x_{n+1} \leq x_s$. This implies that the sequence $\{x_n\}_{n\geq 0}$ is entirely either on the left hand side of x_s or on its right hand side (including x_s).

Without loss of generality, assume that the entire sequence of iterations lies on the right hand side of x_s. If at some iteration, $x_m = x_s$, then since $f'(x_s) = 0$, $x_n = x_s$ for $n \geq m$, which yields the desired result. So we can assume that $x_n \neq x_s$ for all $n \geq 0$. Using a similar argument, we can also assume that $f'(x_n) \neq 0$ for all $n \geq 0$. Suppose by contradiction that there is no such M as described in the lemma. Then there exist infinitely many n such that $x_n < x_{n+1}$ implying that for infinitely many n, $f'(x_n) < 0$. Since $\lim_{n\to\infty} x_n = x_s$ and the entire sequence is on the right hands side of x_s, we also have infinitely many n such that $f'(x_n) > 0$. Combining these results, one can construct a strictly decreasing sub-sequence $\{y_n\}_{n\geq 0}$ of the iterations such that $\lim_{n\to\infty} y_n = x_s$, $f'(y_{2m}) > 0$, and $f'(y_{2m+1}) < 0$ for all $m \geq 0$. Since f' is continuous, there exists $y_{2m+1} < z_m < y_{2m}$ such that $f'(z_m) = 0$, for each $m \geq 0$. It is easy to see that $\{z_n\}_{n\geq 0}$ is also strictly decreasing and $\lim_{n\to\infty} z_n = x_s$. Note that since f'' is continuous, by the mean value theorem, one can find a sequence $\{t_n\}_{n\geq 0}$ such that for each $n \geq 0$, $z_{n+1} < t_n < z_n$ and $f''(t_n) = 0$. Since $\{z_n\}_{n\geq 0}$ converges to x_s, then so does $\{t_n\}_{n\geq 0}$. But this implies that $f''(x_s) = \lim_{n\to\infty} f''(t_n) = 0$ contradicting with the fact that $f''(x_s) \neq 0$. □

Lemma A.3. *Given the setting in Lemma A.2, $\{x_n\}_{n\geq 0}$ converges monotonically to x_s.*

Proof. Without loss of generality, assume that $x_0 \geq x_s$, then using what we obtained during the proof of Lemma A.2, we know that the entire sequence $\{x_n\}_{n\geq 0}$ lies on the right hand side of x_s ((including x_s). Let M be the minimum index that satisfies the condition in Lemma A.2. Suppose by contradiction that $M > 0$. So $x_s < x_{M-1} < x_M$, which implies $f'(x_{M-1}) < 0$ considering $x_M = x_{M-1} - \eta f'(x_{M-1})$. Since the sequence converges to x_s, there should be a $k \geq 0$ such that $x_{M+k+1} < x_{M-1} < x_{M+k}$. Note that $x_{M+k+1} = x_{M+k} - \eta f'(x_{M+k})$, so

$$\eta f'(x_{M+k}) = x_{M+k} - x_{M+k+1} > x_{M+k} - x_{M-1}.$$

Since $f'(x_{M-1}) < 0$, we have $\eta\big(f'(x_{M+k}) - f'(x_{M-1})\big) > \eta f'(x_{M+k}) > x_{M+k} - x_{M-1}$. This contradicts the fact that $\eta(f'(x_{M+k}) - f'(x_{M-1})) \leq \eta\ell(x_{M+k} - x_{M-1}) < x_{M+k} - x_{M-1}$. □

Appendix B. Background on convex optimization

We provide some context of gradient descent applied to convex functions.

Definition B.1.

(1) A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz if $\|\nabla f(\boldsymbol{x}_1) - \nabla f(\boldsymbol{x}_2)\| \leq \ell\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$.
(2) A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is α-strongly convex if $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \geq \alpha$ for all $\boldsymbol{x} \in \mathbb{R}^d$.

The gradient Lipschitz condition controls the amount of decay in each iteration, and the strong convexity condition guarantees that the unique stationary point is the global minimum. The ratio ℓ/α is often called the condition number of the function f. The following theorem shows the linear convergence of gradient descent to the global minimum $\boldsymbol{x}^\star$, see [79][Theorem 3.10] and [5][Theorem 2.1.15].

Theorem B.2 (*[5,79]*). *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz and α-strongly convex. For any $\epsilon > 0$, if we run gradient descent with step size $\eta = \ell^{-1}$, then the number of iterations to be ϵ-close to $\boldsymbol{x}^\star$ is $\frac{2\ell}{\alpha}\log\left(\frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|}{\epsilon}\right)$.*

Appendix C. Proof of Theorem 3.1

Suppose by contradiction that with positive probability, the walk is localized at some points $\{k, \dots, \ell\}$. We focus on the left end k. Let τ_n^k be the time at which the point k is visited n times. For n sufficiently large, the point $k+1$ is visited approximately at least n times by τ_n^k. So at time τ_n^k, the walk moves from k to $k+1$ with probability bounded from above by $C/w(n)$ for some constant $C > 0$. Consequently, the probability that the walk is localized at $\{k, \dots, \ell\}$ is less than $\prod_{n>0}\frac{C}{w(n)}$. By standard analysis, $\prod_{n>0}\frac{C}{w(n)} = 0$ if $w(n) \to \infty$ as $n \to \infty$. This leads to the desired result.

Appendix D. Proof of Theorem 3.2

We show how the thin-pancake property of saddle points is used to prove Theorem 3.2. Recall that an ϵ-second-order stationary point is a point with a small gradient, and where the Hessian does not have a large negative eigenvalue. Let us put down the basic idea in Section 2.2 with the parameters in Algorithm 3 (PGDOT). If we are currently at an iterate $\boldsymbol{x}_t$ which is not an ϵ-second-order stationary point, there are two cases: (1) The gradient is large: $\|\nabla f(\boldsymbol{x}_t)\| \geq g_{\text{thres}}$; (2) $\boldsymbol{x}_t$ is close to a saddle point: $\|\nabla f(\boldsymbol{x}_t)\| \leq g_{\text{thres}}$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}_t)) \leq -\sqrt{\rho\epsilon}$. The case (1) is easy to deal with by the following elementary lemma.

Lemma D.1. *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz. Then for GD with step size $\eta < \ell^{-1}$, we have $f(\boldsymbol{x}_{t+1}) - f(\boldsymbol{x}_t) \leq -\frac{\eta}{2}\|\nabla f(\boldsymbol{x}_t)\|^2$.*

The case (2) is more subtle, and the following lemma gives the decay of the function value after a random perturbation described in Algorithm 3 (PGDOT).

Lemma D.2. *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is ℓ-gradient Lipschitz and ρ-Hessian Lipschitz. If $\|\nabla f(\boldsymbol{x}_t)\| \leq g_{thres}$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}_t)) \leq -\sqrt{\rho\epsilon}$, then adding one perturbation step as in Algorithm 3 followed by t_{thres} steps of GD with step size η, we have $f(\boldsymbol{x}_{t+t_{thres}}) - f(\boldsymbol{x}_t) \leq -f_{thres}$ with probability at least $1 - \frac{d\ell}{\sqrt{\rho\epsilon}}e^{-\chi}$.*

[52] proved Lemma D.2 for PGD, and used it together with Lemma D.1 to prove Theorem 2.5. We will use the same argument, with Lemmas D.1 and D.2, leading to Theorem 3.2 for PGDOT.

Now, let us explain how to prove Lemma D.2 via a purely geometric property of saddle points. Consider a point $\widetilde{\boldsymbol{x}}$ satisfying the condition $\|\nabla f(\widetilde{\boldsymbol{x}})\| \leq g_{\text{thres}}$ and $\lambda_{\min}(\nabla^2 f(\widetilde{\boldsymbol{x}})) \leq -\sqrt{\rho\epsilon}$. After adding the perturbation in Algorithm 3, the resulting vector can be viewed as a distribution over the cube $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$. Similar as in [52], we call $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$ the perturbation cube which is divided into two regions: (1) escape region χ_{escape} which consists of all points $\boldsymbol{x} \in C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$ whose function value decreases by at least f_{thres} after t_{thres} steps; (2) stuck region χ_{stuck} which is the complement of χ_{escape} in $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$. The key idea is that the stuck region χ_{stuck} looks like a non-flat thin pancake, which has a very small volume compared to that of $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$. This claim can be formalized by the following lemma, which is a direct corollary of [52][Lemma 11] as $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \subseteq B^d(\widetilde{\boldsymbol{x}}, r)$:

Table 1
Hyperparameters utilized in the small-scale problems.

	d	h	t_{count}	η	t_{thres}	g_{thres}	r	Momentum	# of steps
Stair function	4	0.04	200	0.1	10	0.01	0.04	0.5	2000
Nonlinear regression	16	0.04	200	0.1	50	0.1	0.1	0.5	14 000
Regularized quadratic	2	1	200	0.01	50	0.01	0.01	0.5	3000
Phase retrieval	10	1	200	0.001	50	1	0.01	0.5	1200

Table 2
Hyperparameters utilized by SGD and the adaptive gradient methods in the image classification tasks.

	SGD	Adam	AMSGrad	AdaBelief	STORM
MLP on MNIST	lr: 0.01	lr: 0.001	lr: 0.001	lr: 0.001	k: 0.01, c: 100
SqueezeNet on CIFAR-10	lr: 1.0	lr: 0.0001	lr: 1.0	lr: 0.1	k: 0.01, c: 10
ResNet18 on CIFAR-100	lr: 0.1	lr: 0.001	lr: 0.001	lr: 0.001	k: 0.1, c: 100

Table 3
Hyperparameters utilized by PGD, PAGD, PGDOT, and PAGDOT in the image classification tasks.

	η	t_{thres}	g_{thres}	$r_{\text{PGD/PAGD}}$	$r_{\text{PGDOT/PAGDOT}}$	Momentum
MLP on MNIST	0.01	10	0.1	$1/\sqrt{79,510}$	1	0.9
SqueezeNet on CIFAR-10	0.001	10	1	$10/\sqrt{740,554}$	10	0.1
ResNet18 on CIFAR-100	0.01	200	1	$1/\sqrt{11,227,812}$	1	0.9

Lemma D.3. *Assume that $\widetilde{\boldsymbol{x}}$ satisfies $\|\nabla f(\widetilde{\boldsymbol{x}})\| \le g_{thres}$ and $\lambda_{\min}(\nabla^2 f(\widetilde{\boldsymbol{x}})) \le -\sqrt{\rho\epsilon}$. Let $\boldsymbol{e}_1$ be the smallest eigendirection of $\nabla^2 f(\widetilde{\boldsymbol{x}})$. For any $\delta < 1/3$ and any $\boldsymbol{u}, \boldsymbol{v} \in C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$, if $\boldsymbol{u} - \boldsymbol{v} = \mu r \boldsymbol{e}_1$ and $\mu \ge \delta/(2\sqrt{d})$, then at least one of $\boldsymbol{u}$ and $\boldsymbol{v}$ is not in the stuck region χ_{stuck}.*

To prove Lemma D.2, it suffices to check that $\mathbb{P}(\chi_{\text{stuck}}) \le C\delta$ for some $C > 0$. This criterion is general for any (random) perturbation. Let $\mathcal{O}_1, \dots, \mathcal{O}_{2^d}$ be the orthants centered at $\widetilde{\boldsymbol{x}}$; that is, the space $\mathbb{R}^d$ is divided into 2^d subspaces according to the coordinate signs of $\cdot - \widetilde{\boldsymbol{x}}$. The symbol $\text{sgn}(\mathcal{O}_i) \in \{-1, 1\}^d$ denotes the coordinate signs of $\boldsymbol{y} - \widetilde{\boldsymbol{x}}$ for any $\boldsymbol{y} \in \mathcal{O}_i$. For $1 \le i \le 2^d$, let

$$p_i = \prod_{\text{sgn}(\mathcal{O}_i)_k = -1} \frac{w(R_t^k)}{w(L_t^k) + w(R_t^k)} \prod_{\text{sgn}(\mathcal{O}_i)_k = 1} \frac{w(L_t^k)}{w(L_t^k) + w(R_t^k)}$$

be the probability that the random perturbation drives $\widetilde{\boldsymbol{x}}$ into $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i$. Consequently, $\mathbb{P}(\chi_{\text{stuck}}) = \sum_{i=1}^{2^d} p_i \frac{\text{Vol}(\chi_{\text{stuck}} \cap \mathcal{O}_i)}{\text{Vol}(C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i)}$, where $\text{Vol}(\cdot)$ denotes the volume of a domain. It is easy to see that $\text{Vol}(C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i) = (r/\sqrt{d})^d$. By Lemma D.3 and the slicing volume bound [80], $\text{Vol}(\chi_{\text{stuck}} \cap \mathcal{O}_i) \le \sqrt{2}(r/\sqrt{d})^{d-1} \frac{\delta r}{\sqrt{d}}$. Therefore, $\frac{\text{Vol}(\chi_{\text{stuck}} \cap \mathcal{O}_i)}{\text{Vol}(C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i)} \le \sqrt{2}\delta$ implying that $\mathbb{P}(\chi_{\text{stuck}}) \le \sqrt{2}\delta$.

Note that this proof does not rely on the full history of states for L_t and R_t. Thus, one can restrict the number of previous iterations as is done in Section 4 using the hyperparameter t_{count}.

Appendix E. Hyperparameter settings in the numerical examples

To ensure a fair comparison between algorithms in the small-scale problems, we opted to select a consistent set of hyperparameters. Specifically, we employed the same learning rate as GD for all other algorithms. Additionally, we set the common hyperparameters between the new algorithms and their counterparts to be equal. Table 1 provides an overview of the hyperparameters utilized in the small-scale problems.

For the image classification tasks (the large-scale problems), we do a grid search and report the best hyperparameters for SGD and the adaptive gradient methods. For SGD, Adam, AMSGrad, and AdaBelief, the only hyperparameter is the learning rate. For STORM, we tune both k and c while setting $w = 0.1$. Table 2 provides an overview of the hyperparameters utilized by SGD and the adaptive gradient methods in the image classification tasks. It is important to note that the significant disparity observed in the learning rates employed by various algorithms in the second image classification task stems from the fact that regardless of the learning rate value, none of the algorithms manage to escape the saddle points, leading to no improvement in the training loss.

To ensure fair comparison between PGDOT and PAGDOT and their counterparts, we select a consistent set of hyperparameters for them. The only difference is in the perturbation norm r. Note that norm of the actual perturbation applied in PGDOT and PAGDOT is $r/\sqrt{d}$. Since d is a large number in the large-scale problems, $r/\sqrt{d}$ would be significantly different than r. Therefore, we adjust the perturbation norm in PGD and PAGD accordingly. Additionally, in all the image classification tasks, we set $t_{\text{count}} = 100$ and $h = 10^{12}$. Table 3 provides an overview of the hyperparameters utilized by PGD, PAGD, PGDOT, and PAGDOT.

We also remark that the weight function in Algorithms 3 and 4 is set as $w(k) = 1 + k^5$ for all the small-scale and large-scale problems.

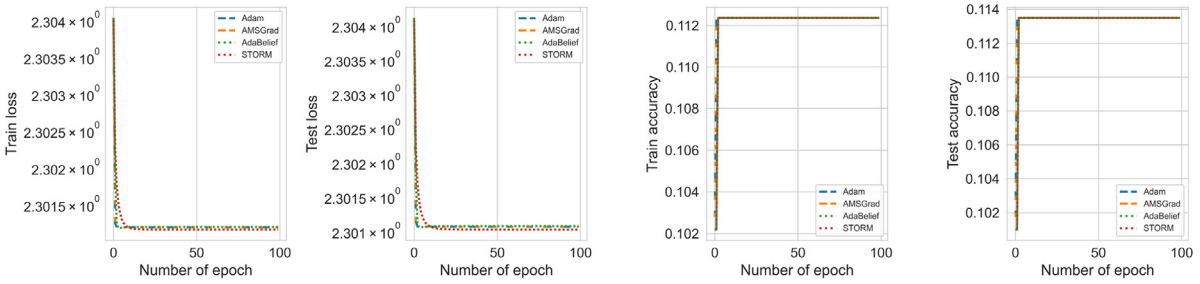

Fig. 9. Experimental results of training the MLP model on MNIST using several adaptive gradient methods.

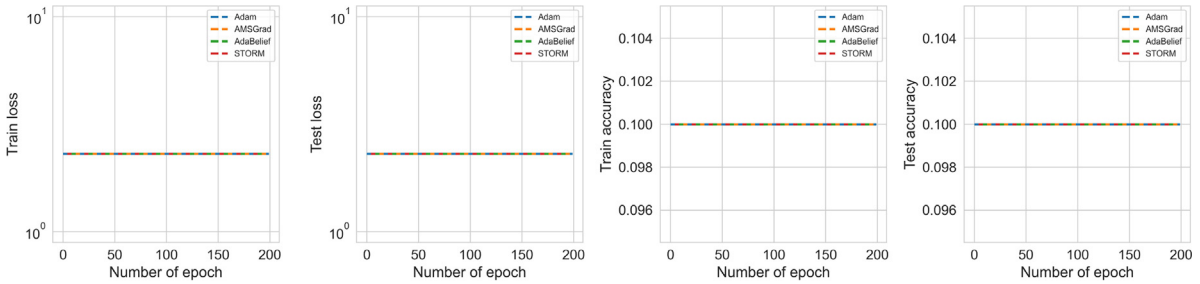

Fig. 10. Experimental results of training SqueezeNet on CIFAR-10 using several adaptive gradient methods.

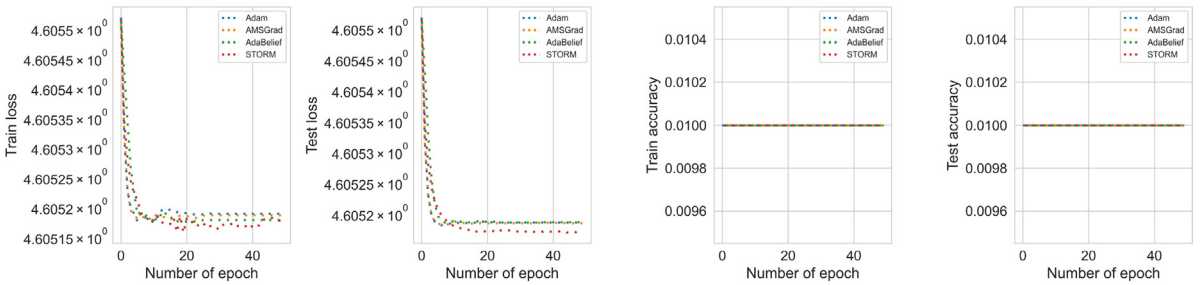

Fig. 11. Experimental results of training ResNet18 on CIFAR-100 using several adaptive gradient methods.

Appendix F. Image classification task results of adaptive gradient methods

Here, we present the training outcomes of AMSGrad, AdaBelief, and STORM, in addition to Adam, across the three distinct image classification tasks. See Figs. 9, 10, and 11. It is worth noting that as Adam exhibited unsatisfactory performance in all experiments, any training curves that closely resemble or are identical to Adam's indicate poor performance for the other algorithms as well.

Appendix G. Runtime comparison

In addition to the per-iteration basis comparison, in Figs. 12–16, we compare the algorithms runtime for each one of the examples considered. Note that for each example, the number of iterations/epochs is kept the same and the losses are plotted against the runtime.

It is worth mentioning that the per-iteration time complexity of the new algorithms is the same (up to a constant factor) as that of vanilla GD. To see this, note the followings:

1. The per-iteration time complexity is mainly determined by the gradient computation. In the new algorithms, similar to vanilla GD, gradient computation occurs at each iteration — we always compute the gradient to see if its norm is small enough to trigger the perturbation mechanism.
2. The number of the rest of the per-iteration operations (e.g. comparing the current iterate with previous iterates) is fixed due to considering only t_{count} previous iterations while computing the occupation time.

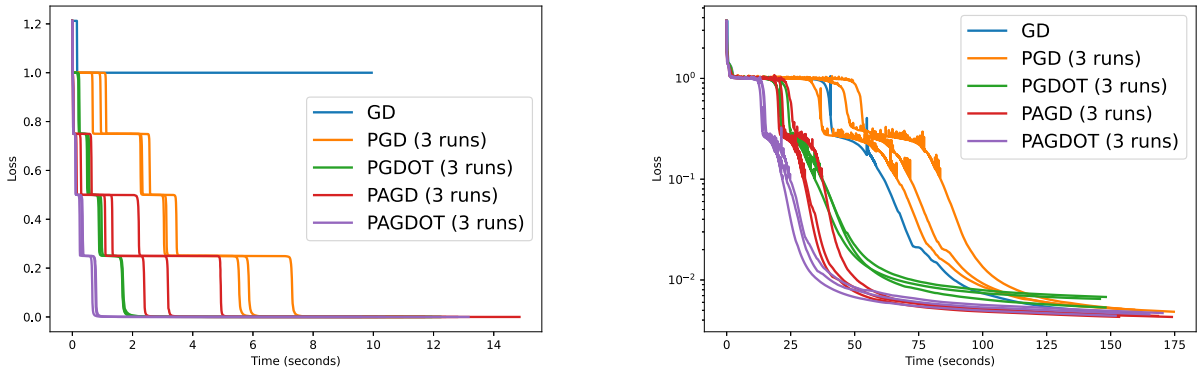

Fig. 12. Runtime comparison: stair function problem (left) and nonlinear regression problem (right).

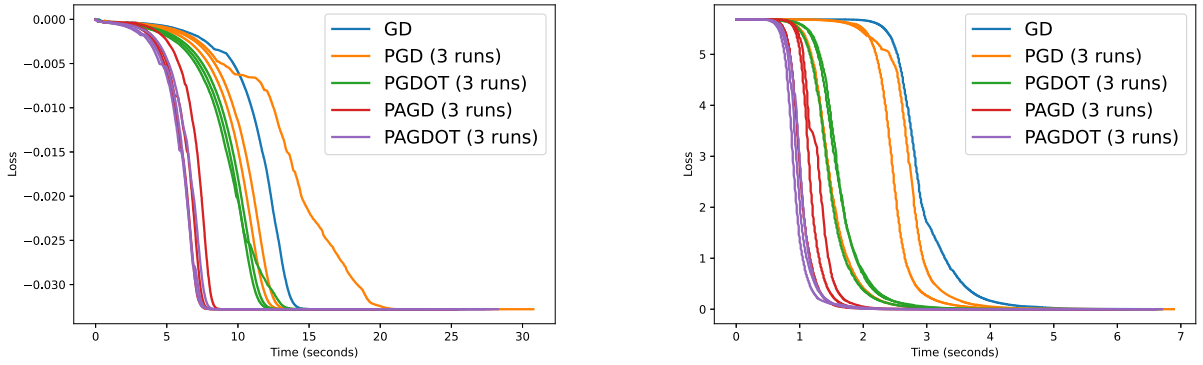

Fig. 13. Runtime comparison: regularized quadratic problem (left) and phase retrieval (right).

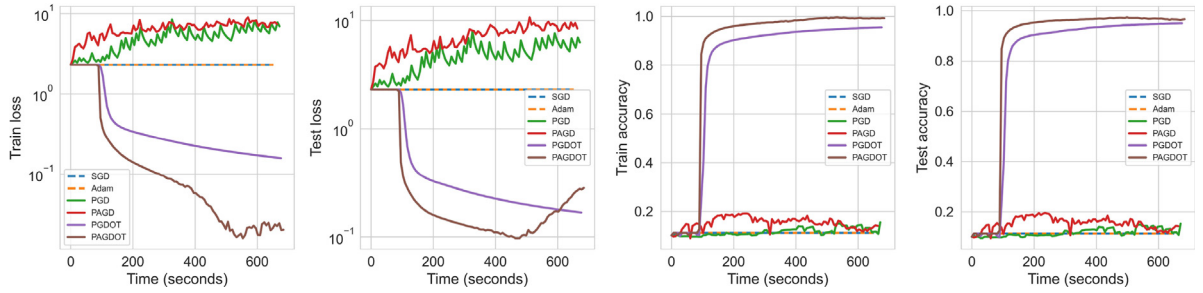

Fig. 14. Runtime comparison: training the MLP model on MNIST.

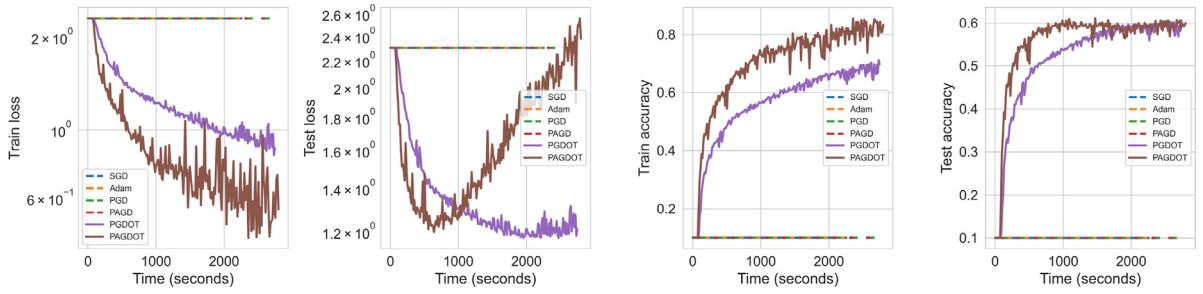

Fig. 15. Runtime comparison: training SqueezeNet on CIFAR-10.

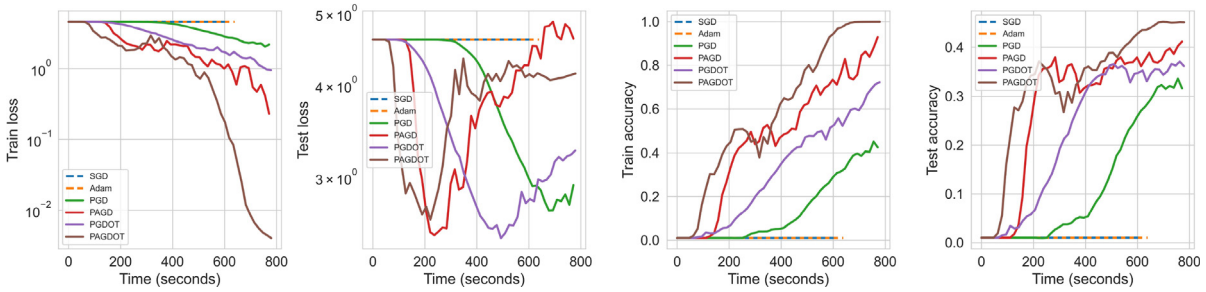

Fig. 16. Runtime comparison: training ResNet18 on CIFAR-100.

Appendix H. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jcmds.2024.100090.

References

[1] Cauchy A. Méthode générale pour la résolution des systemes d'équations simultanées. C R Math Sci Paris 1847;25(1847):536–8.
[2] Candès EJ, Li X, Ma Y, Wright J. Robust principal component analysis? J ACM 2011;58(3):37, Art. 11.
[3] Candès EJ, Li X, Soltanolkotabi M. Phase retrieval via wirtinger flow: theory and algorithms. IEEE Trans Inform Theory 2015;61(4):1985–2007.
[4] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature 1986;323(6088):533–6.
[5] Nesterov Y. Introductory lectures on convex optimization: A basic course. Applied optimization, vol. 87, Boston, MA: Kluwer Academic Publishers; 2004, p. xviii+236.
[6] Kirkpatrick S, Gelatt Jr CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220(4598):671–80.
[7] Geman S, Hwang CR. Diffusions for global optimization. SIAM J Control Optim 1986;24(5):1031–43.
[8] Holley RA, Kusuoka S, Stroock DW. Asymptotics of the spectral gap with applications to the theory of simulated annealing. J Funct Anal 1989;83(2):333–47.
[9] Menz G, Schlichting A, Tang W, Wu T. Ergodicity of the infinite swapping algorithm at low temperature. 2018, arXiv:1811.10174.
[10] Miclo L. Recuit simulé sur $\mathbf{R}^n$. Étude de l'évolution de l'énergie libre. Ann Inst Henri Poincare 1992;28(2):235–66.
[11] Baudoin F, Hairer M, Teichmann J. Ornstein–Uhlenbeck processes on Lie groups. J Funct Anal 2008;255(4):877–90.
[12] Monmarché P. Hypocoercivity in metastable settings and kinetic simulated annealing. Probab Theory Related Fields 2018;172(3–4):1215–48.
[13] Tang W, Zhou XY. Tail probability estimates of continuous-time simulated annealing processes. Numer Algebra Control Optim 2023;13(3–4):473–85.
[14] Tang W, Zhou XY. Simulated annealing from continuum to discretization: a convergence analysis via the Eyring–Kramers law. 2021, arXiv:2102.02339.
[15] Pavlyukevich I. Lévy flights, non-local search and simulated annealing. J Comput Phys 2007;226(2):1830–44.
[16] Yang XS, Deb S. Cuckoo search via Lévy flights. In: 2009 World congress on nature & biologically inspired computing. 2009, p. 210–4.
[17] Raginsky M, Rakhlin A, Telgarsky M. Non-convex learning via stochastic gradient langevin dynamics: a nonasymptotic analysis. In: Conference on learning theory. 2017, p. 1674–703.
[18] Chen X, Du SS, Tong XT. On stationary-point hitting time and ergodicity of stochastic gradient langevin dynamics. J Mach Learn Res 2020;21. Paper No. 68, 41.
[19] Hu Y, Wang X, Gao X, Gürbüzbalaban M, Zhu L. Non-convex optimization via non-reversible stochastic gradient langevin dynamics. 2020, arXiv:2004.02823.
[20] Chen Y, Chen J, Dong J, Peng J, Wang Z. Accelerating nonconvex learning via replica exchange langevin diffusion. In: International conference on learning representations. 2019.
[21] Dong J, Tong XT. Replica exchange for non-convex optimization. J Mach Learn Res 2021;22. Paper No. 173, 59.
[22] Ge R, Huang F, Jin C, Yuan Y. Escaping from saddle points – online stochastic gradient for tensor decomposition. In: Conference on learning theory. 2015, p. 797–842.
[23] Ge R, Lee JD, Ma T. Learning one-hidden-layer neural networks with landscape design. In: International conference on learning representations. 2018.
[24] Ge R, Ma T. On the optimization landscape of tensor decompositions. In: Advances in neural information processing systems. 2017, p. 3653–63.
[25] Sanjabi M, Baharlouei S, Razaviyayn M, Lee JD. When does non-orthogonal tensor decomposition have no spurious local minima? 2019, arXiv:1911.09815.
[26] Bandeira AS, Boumal N, Voroninski V. On the low-rank approach for semidefinite programs arising in synchronization and community detection. In: Conference on learning theory. 2016, p. 361–82.
[27] Mei S, Misiakiewicz T, Montanari A, Oliveira RI. Solving SDPs for synchronization and MaxCut problems via the grothendieck inequality. In: Conference on learning theory. 2017, p. 1476–515.
[28] Sun J, Qu Q, Wright J. Complete dictionary recovery over the sphere I: Overview and the geometric picture. IEEE Trans Inform Theory 2017;63(2):853–84.
[29] Sun J, Qu Q, Wright J. A geometric analysis of phase retrieval. Found Comput Math 2018;18(5):1131–98.
[30] Mei S, Bai Y, Montanari A. The landscape of empirical risk for nonconvex losses. Ann Statist 2018;46(6A):2747–74.
[31] Bhojanapalli S, Neyshabur B, Srebro N. Global optimality of local search for low rank matrix recovery. In: Advances in neural information processing systems. 2016, p. 3873–81.
[32] Ge R, Jin C, Zheng Y. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. In: International conference on machine learning. 2017, p. 1233–42.
[33] Ge R, Lee JD, Ma T. Matrix completion has no spurious local minimum. In: Advances in neural information processing systems. 2016, p. 2973–81.
[34] Park D, Kyrillidis A, Carmanis C, Sanghavi S. Non-square matrix sensing without spurious local minima via the Burer-Monteiro approach. In: Artificial intelligence and statistics. 2017, p. 65–74.
[35] Choromanska A, Henaff M, Mathieu M, Ben Arous G, LeCun Y. The loss surfaces of multilayer networks. In: Artificial intelligence and statistics. 2015, p. 192–204.
[36] Draxler F, Veschgini K, Salmhofer M, Hamprecht F. Essentially no barriers in neural network energy landscape. In: International conference on machine learning. 2018, p. 1309–18.
[37] Kawaguchi K. Deep learning without poor local minima. In: Advances in neural information processing systems. 2016, p. 586–94.
[38] Kazemipour A, Larsen B, Druckmann S. Avoiding spurious local minima in deep quadratic networks. 2019, arXiv:2001.00098.

[39] Liang S, Sun R, Lee JD, Srikant R. Adding one neuron can eliminate all bad local minima. In: Advances in neural information processing systems. 2018, p. 4350–60.
[40] Nguyen Q, Hein M. The loss surface of deep and wide neural networks. In: International conference on machine learning. 2017, p. 2603–12.
[41] Venturi L, Bandeira AS, Bruna J. Spurious valleys in one-hidden-layer neural network optimization landscapes. J Mach Learn Res 2019;20. Paper No. 133, 34.
[42] Wu C, Luo J, Lee JD. No spurious local minima in a two hidden unit ReLU network. 2018, Available at https://openreview.net/forum?id=B14uJzW0b.
[43] Dauphin YN, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: Advances in neural information processing systems. 2014, p. 2933–41.
[44] Du SS, Jin C, Lee JD, Jordan MI, Singh A, Poczos B. Gradient descent can take exponential time to escape saddle points. In: Advances in neural information processing systems. 2017, p. 1067–77.
[45] Jain P, Jin C, Kakade S, Netrapalli P. Global convergence of non-convex gradient descent for computing matrix squareroot. In: Artificial intelligence and statistics. 2017, p. 479–88.
[46] Dauphin Y, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. 2014, arXiv:1406.2572.
[47] Swirszcz G, Czarnecki WM, Pascanu R. Local minima in training of deep networks. 2016, Available at https://openreview.net/pdf?id=Syoiqwcxx.
[48] Ghadimi S, Lan G. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. SIAM J Optim 2013;23(4):2341–68.
[49] Kingma DP, Ba J. Adam: A method for stochastic optimization. In: International conference on learning representations. 2015.
[50] Du SS, Lee JD, Tian Y, Singh A, Poczos B. Gradient descent learns one-hidden-layer CNN: Don't be afraid of spurious local minima. In: International conference on machine learning. 2018, p. 1339–48.
[51] Lee JD, Simchowitz M, Jordan MI, Recht B. Gradient descent only converges to minimizers. In: Conference on learning theory. 2016, p. 1246–57.
[52] Jin C, Ge R, Netrapalli P, Kakade S, Jordan MI. How to escape saddle points efficiently. In: International conference on machine learning. 2017, p. 1724–32.
[53] Jin C, Netrapalli P, Jordan MI. Accelerated gradient descent escapes saddle points faster than gradient descent. In: Conference on learning theory. 2018, p. 1042–85.
[54] Jin C, Netrapalli P, Ge R, Kakade SM, Jordan MI. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. J ACM 2021;68(2):1–29.
[55] Neelakantan A, Vilnis L, Le QV, Sutskever I, Kaiser L, Kurach K, et al. Adding gradient noise improves learning for very deep networks. 2015, arXiv:1511.06807.
[56] Zhou M, Liu T, Li Y, Lin D, Zhou E, Zhao T. Toward understanding the importance of noise in training neural networks. In: International conference on machine learning. 2019.
[57] Peliti L, Pietronero L. Random walks with memory. Rivista Nuovo Cimento 1987;10(6):1–33.
[58] Davis B. Reinforced random walk. Probab Theory Related Fields 1990;84(2):203–29.
[59] Tóth B. "True" self-avoiding walks with generalized bond repulsion on **Z**. J Stat Phys 1994;77(1–2):17–33.
[60] Tóth B. The "true" self-avoiding walk with bond repulsion on **Z**: limit theorems. Ann Probab 1995;23(4):1523–56.
[61] Tóth B. Generalized ray-knight theory and limit theorems for self-interacting random walks on $\mathbf{Z}^1$. Ann Probab 1996;24(3):1324–67.
[62] Pemantle R. Vertex-reinforced random walk. Probab Theory Related Fields 1992;92(1):117–36.
[63] Volkov S. Phase transition in vertex-reinforced random walks on $\mathbb{Z}$ with non-linear reinforcement. J Theoret Probab 2006;19(3):691–700.
[64] Tarrès P. Vertex-reinforced random walk on $\mathbb{Z}$ eventually gets stuck on five points. Ann Probab 2004;32(3B):2650–701.
[65] Nesterov YE. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. In: Dokl. akad. nauk SSSR, vol. 269. 1983, p. 543–7.
[66] Reddi SJ, Kale S, Kumar S. On the convergence of adam and beyond. In: International conference on learning representations. 2018.
[67] Zhuang J, Tang T, Ding Y, Tatikonda SC, Dvornek N, Papademetris X, et al. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. Adv Neural Inf Process Syst 2020;33:18795–806.
[68] Cutkosky A, Orabona F. Momentum-based variance reduction in non-convex sgd. Adv Neural Inf Process Syst 2019;32.
[69] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE 1998;86(11):2278–324.
[70] Krizhevsky A, Hinton G, et al. Learning multiple layers of features from tiny images. Citeseer; 2009.
[71] Polyak BT. Some methods of speeding up the convergence of iteration methods. USSR Comput Math Math Phys 1964;4(5):1–17.
[72] Li Z, Han J, E W, Li Q. On the curse of memory in recurrent neural networks: Approximation and optimization analysis. In: International conference on learning representations. 2021.
[73] Wang JK, Lin CH, Abernethy J. Escaping saddle points faster with stochastic momentum. In: International conference on learning representations. 2019.
[74] Reddi S, Zaheer M, Sra S, Poczos B, Bach F, Salakhutdinov R, et al. A generic approach for escaping saddle points. In: International conference on artificial intelligence and statistics. 2018, p. 1233–42.
[75] Candès EJ, Eldar YC, Strohmer T, Voroninski V. Phase retrieval via matrix completion. SIAM J Imaging Sci 2013;6(1):199–225.
[76] Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. 2016, arXiv preprint arXiv:1602.07360.
[77] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 770–8.
[78] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. PMLR; 2015, p. 448–56.
[79] Bubeck S. Convex optimization: Algorithms and complexity. Found Trends Mach Learn 2015;8(3–4):231–357.
[80] Ball K. Cube slicing in $\mathbf{R}^n$. Proc Amer Math Soc 1986;97(3):465–73.