PGDOT – Perturbed Gradient Descent Adapted with Occupation Time

Xin Guo *
xinguo@berkeley.edu

Jiequn Han †
jiequnh@princeton.edu

Mahan Tajrobehkar ‡
mahan_tajrobehkar@berkeley.edu

Wenpin Tang §
wt2319@columbia.edu

Abstract

This paper develops further the idea of perturbed gradient descent (PGD), by adapting perturbation with the history of states via the notion of occupation time. The proposed algorithm, perturbed gradient descent adapted with occupation time (PGDOT), is shown to converge at least as fast as the PGD algorithm and is guaranteed to avoid getting stuck at saddle points. The analysis is corroborated by empirical studies, in which a mini-batch version of PGDOT is shown to outperform alternatives such as mini-batch gradient descent, Adam, AMSGrad, and RMSProp in training multilayer perceptrons (MLPs). In particular, the mini-batch PGDOT manages to escape saddle points whereas these alternatives fail.

1 Introduction

Gradient descent (GD), which dates back to Cauchy [8], aims to minimize a function \( f : \mathbb{R}^d \to \mathbb{R} \) via the iteration: 
\[
x_{t+1} = x_t - \eta \nabla f(x_t), \quad t = 0, 1, 2, \ldots,
\]
where \( \eta > 0 \) is the step size and \( \nabla f \) 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 [6], phase retrieval [7], and deep neural network [49], just to name a few. In the era of data deluge, many problems are concerned with large-scale optimization, i.e., the intrinsic dimension \( d \) is large. GD turns out to be efficient in dealing with convex optimization, where the first-order stationary point \( \nabla f(x) = 0 \) is necessarily the global minimum point. Algorithmically, it involves finding a point with small gradient \( ||\nabla f(x)|| < \epsilon \). A classical result of Nesterov [40] showed that the time required by GD to find such a point is of order \( \epsilon^{-2} \), independent of the dimension \( d \).

In non-convex settings, applying GD will still lead to an approximate first-order stationary point. However, this is not sufficient since for non-convex functions, first-order stationary points can be global minimum points, or local minimum points, or saddle points, or local maximum points. Inspired by annealing in metallurgy, [28] developed simulated annealing to approximate the global minimum of a given function. [20] proposed a diffusion simulated annealing and proved that it converges to the set of global minimum points. But subsequent works [21, 36, 37, 38, 54] revealed that it may take an exponentially long time (of order \( \exp(1/\epsilon) \)) for diffusion simulated annealing to be \( \epsilon \)-close to the global minimum. Some recent efforts, e.g., methods based on Lévy flights [44] or Cuckoo’s search [60] showed empirically faster convergence to the global minimum. Yet the theory of these algorithms is far-fetched, and finding the global minimum remains hard in general.

*Department of Industrial Engineer and Operations Research, UC Berkeley.
†Department of Mathematics, Princeton University.
‡Department of Industrial Engineer and Operations Research, UC Berkeley.
§Department of Industrial Engineer and Operations Research, Columbia University.

Preprint. Under review.
Algorithm 1 is a state-dependent adaptive algorithm, which perturbs GD in a non-uniform direction with "occupation times" that we will explain. The intuition is demonstrated by the following one-dimensional picture, see Figure 1 below. Imagine that there is a saddle point at 0, however to the left there is a possibility to escape from 0 and find a local minimum. 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 more.

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 [15, 18, 19, 50], semidefinite programming [2, 32, 34], dictionary learning [51], phase retrieval [52], robust regression [34], low-rank matrix factorization [3, 16, 17, 43], and certain classes of deep neural networks [9, 12, 25, 26, 33, 42, 50, 59]. Nevertheless, as shown in [11, 13, 22], saddle points may correspond to suboptimal solutions, and it may take a considerably long time to move from saddle points to a local minimum point. It has also been observed in empirical studies [10, 53] that GD and its variants with momentum such as Adam [27] may be trapped in saddle points. Thus, saddle points are the main bottleneck for GD in many non-convex optimization problems.

Ge et al. [15] took the first step to show that by adding noise at each iteration, GD can escape all saddle points in polynomial time. Additionally, [14, 51] proved that with random initialization, GD converges to the global minimum. Moreover, Jin et al. [23, 24] proposed the perturbed gradient descent (PGD) algorithm, and showed that it is efficient – the time complexity is almost independent of the dimension $d$, and hence overcomes the curse of dimensionality.

In this paper, we develop further the idea of PGD by adapting perturbation with the history of states. Recall that [23, 24] used the following perturbation update when perturbation conditions hold:

$$x_t' = x_t + \text{Unif}(B^d(0, r)), \quad x_{t+1} = x_t' - \eta \nabla f(x_t')$$

where $\text{Unif}(B^d(0, r))$ is a point picked uniformly/randomly in the ball of radius $r$. On the empirical side, [39, 61] applied this idea of GD with noise to train deep neural networks. Our key idea is to replace $\text{Unif}(B^d(0, r))$ with non-uniform perturbations, whose mechanism depends on the current state $x_t$ and the history of states $\{x_s; s \leq t\}$. There are conceivably many ways to add non-uniform perturbation based on the current and previous states, here we choose to adapt perturbation with “occupation times” that we will explain. The intuition is demonstrated by the following one-dimensional picture, see Figure 1 below. Imagine that there is a saddle point at 0, and GD approaches 0 from the right. It can be shown that GD converges monotonically to a stationary point (see Appendix B).

PGD will add noise with probability $1/2$ both to the right and to the left. To the right GD is again stuck at the saddle point 0, however to the left there is a possibility to escape from 0 and find a local minimum. 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 more.

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

$$x_{t+1} = \begin{cases} x_t - r \text{Unif}(0, 1) & \text{with probability } \frac{w(D_t)}{w(G_t) + w(D_t)}, \\ x_t + r \text{Unif}(0, 1) & \text{with probability } \frac{w(G_t)}{w(G_t) + w(D_t)}, \end{cases} \quad (1.1)$$

where $w : \{0, 1, \ldots\} \to (0, \infty)$ is an increasing weight function on the nonnegative integers (e.g., $w(n) = 1 + n^\alpha$ for $\alpha > 0$). By applying this one-dimensional dynamics in each direction, we propose a perturbed gradient descent algorithm adapted with occupation time (Algorithm 1).

Algorithm 1 is a state-dependent adaptive algorithm, which perturbs GD in a non-uniform direction according to the history of states. The aforementioned intuition suggests that it has some advantages over PGD. Indeed, we will prove that it converges at least as fast as PGD (see Theorem 3.1), which

\[ x_{t+1} = \begin{cases} x_t - r \text{Unif}(0, 1) & \text{with probability } \frac{w(D_t)}{w(G_t) + w(D_t)}, \\ x_t + r \text{Unif}(0, 1) & \text{with probability } \frac{w(G_t)}{w(G_t) + w(D_t)}, \end{cases} \quad (1.1) \]
Algorithm 1 Perturbed Gradient Descent Adapted with Occupation Time (Meta Algorithm)

for $t = 0, 1, \ldots$ do
  if perturbation condition holds then
    for $i = 1, \ldots, d$ do
      $G_i^t \leftarrow \#\{s < t : x_i^s < x_i^t\}$ and $D_i^t \leftarrow \#\{s < t : x_i^s \geq x_i^t\}$
      $x_i^t \leftarrow \begin{cases} x_i^t - r \text{ Unif}(0, 1) & \text{with probability } \frac{w(D_i^t)}{w(D_i^t) + w(D_i^t)}, \\ x_i^t + r \text{ Unif}(0, 1) & \text{with probability } \frac{w(G_i^t)}{w(G_i^t) + w(D_i^t)}, \end{cases}$
    $x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$
  endif
endfor

is further corroborated by experimental results. We will also show that a mini-batch version of Algorithm 1 outperforms momentum methods such as Adam, AMSGrad, and RMSProp in training multilayer perceptrons (MLPs) on some well-studied datasets such as MNIST [30] and CIFAR-10 [29]. The comparison is conducted using different initializations. While these momentum methods fail in most of the cases, the mini-batch PGDOT manages to escape saddle points in all cases.

Moreover, Algorithm 1 is closely related to several existing optimization and probabilistic tools:

(i) Algorithm 1 shares some spirit with simulated annealing and GD with momentum methods such as the heavy ball method [46] and Nesterov’s accelerated gradient descent method [41]. In simulated annealing, the perturbation is only time adapted while perturbation in Algorithm 1 is state adapted (with the history of states). In GD with momentum methods, a “momentum” term is added to control oscillations and to accelerate in low curvatures along the direction close to momentum. In Algorithm 1 however, no explicit momentum term is added. Instead, the momentum direction is “learned” using the history of states. Therefore, this algorithm may be viewed as PGD with “learned” momentum.

(ii) In every dimension of the algorithm, the dynamics (1.1) can be viewed as vertex-diminishing random walk, which is a counterpart to vertex reinforced random walk [45, 57] defined by $Z_{t+1} = Z_t - 1$ with probability $\frac{w(G_t)}{w(G_t) + w(D_t)}$, and $Z_{t+1} = Z_t + 1$ with probability $\frac{w(D_t)}{w(G_t) + w(D_t)}$. It is well known [55, 57] that vertex-reinforced random walk exhibits localization at a finite number of points for some choices of $w(\cdot)$, e.g., $w(n) \sim n^\alpha$ with $\alpha \geq 1$. In contrast to vertex-reinforced walk, vertex-diminishing walk will never be localized or stuck at some points (see Theorem 3.2 below). Though it is unclear whether the vertex-diminishing walk is recurrent, or transient, or neither, this non-localization property facilitates the process to escape from saddle points.

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 Unif($D$) be the uniform distribution on $D$, e.g., Unif(0, 1) is the uniform distribution on [0, 1]. For a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, let $\nabla f$ and $\nabla^2 f$ denote its gradient and Hessian, and $f^* := \min_{x \in \mathbb{R}^d} f(x)$ denote its global minimum. For $A$ a square matrix, let $\lambda_{\min}(A)$ 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 $x = (x^1, \ldots, x^d)$ and $r > 0$, let $B^d(x, r) := \{y : ||y - x|| \leq r\}$ be the $d$-dimensional ball centered at $x$ with radius $r$, and $C^d(x, r) := \{y : ||y^i - x^i|| \leq r \text{ for } 1 \leq i \leq d\}$ be the $d$-dimensional hypercube centered at $x$ 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 result and its proof. 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 A). In this case, it is generally difficult to find the global minima. A popular approach is to consider the first-order stationary points instead.
We say that a differentiable function \( f : \mathbb{R}^d \to \mathbb{R} \) is a first-order stationary point of \( f \) if \( \nabla f(x) = 0 \); (ii) \( x \) is an \( \epsilon \)-first-order stationary point of \( f \) if \( \| \nabla f(x) \| \leq \epsilon \).

We say that a differentiable function \( f : \mathbb{R}^d \to \mathbb{R} \) is \( \ell \)-gradient Lipschitz if \( \| \nabla^2 f(x_1) - \nabla^2 f(x_2) \| \leq \ell \| x_1 - x_2 \| \) for all \( x_1, x_2 \in \mathbb{R}^d \). For gradient Lipschitz functions, GD converges to the first-order stationary points, which is quantified by the following theorem from [40, Section 1.2.3].

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

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, or a saddle point, or a local maximum. The following definition is taken from [23, Definition 4].

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

For a twice differentiable function \( f \), note that \( \lambda_{\min}(\nabla^2 f(x)) \leq 0 \) for any saddle point \( x \). So by assuming a saddle point \( x \) to be strict, we rule out the case \( \lambda_{\min}(\nabla^2 f(x)) = 0 \). In the next subsection, we will review a perturbed gradient method that allows jumping out of strict saddle points.

### 2.2 Results of PGD

One drawback of GD in non-convex optimization is that it may get stuck at saddle points. [23] proposed the PGD algorithm to escape saddle points, which we review here. To proceed further, we need some vocabularies regarding the Hessian of the function \( f \).

**Definition 2.4.** A twice differentiable function \( f : \mathbb{R}^d \to \mathbb{R} \) is \( \rho \)-Hessian Lipschitz if \( \| \nabla^2 f(x_1) - \nabla^2 f(x_2) \| \leq \rho \| x_1 - x_2 \| \) for all \( x_1, x_2 \in \mathbb{R}^d \). Furthermore, we say that (i) \( x \) is a second-order stationary point of \( f \) if \( \nabla f(x) = 0 \) and \( \lambda_{\min}(\nabla^2 f(x)) \geq 0 \); (ii) \( x \) is a \( \epsilon \)-second-order stationary point of \( f \) if \( \| \nabla f(x) \| \leq \epsilon \) and \( \lambda_{\min}(\nabla^2 f(x)) \geq -1/\rho \).

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 PGD is as follows. Imagine that we are currently at an iterate \( x_t \) which is not an \( \epsilon \)-second-order stationary point. There are two scenarios: (i) The gradient \( \| \nabla f(x_t) \| \) is large, and a usual iteration of GD \( x_{t+1} = x_t - \eta \nabla f(x_t) \) is enough; (ii) The gradient \( \| \nabla f(x_t) \| \) is small but \( \lambda_{\min}(\nabla^2 f(x_t)) \leq -1/\rho \) (large negative). So \( x_t \) is around a saddle point, and a perturbation \( \xi \) is needed to escape from the saddle region: \( x_{t+1} = x_t + \xi \).

[23] proposed the PGD algorithm for an \( \ell \)-gradient Lipschitz and \( \rho \)-Hessian Lipschitz function \( f \). Their main result, Theorem 3 in [23] stated below, shows that the time complexity of PGD is almost dimension-free (with a log factor).

**Theorem 2.5.** Assume that \( f : \mathbb{R}^d \to \mathbb{R} \) is \( \ell \)-gradient Lipschitz and \( \rho \)-Hessian Lipschitz. Then there exists \( c_{\max} > 0 \) such that for any \( \delta \geq 0 \), \( \epsilon \leq \ell^2/\rho \), \( \Delta f \geq f(x_0) - f^{*} \), and \( c \leq c_{\max} \) PGD algorithm outputs an \( \epsilon \)-second-order stationary point with probability \( 1 - \delta \), terminating within the number of iterations

\[
O \left( \frac{\ell(f(x_0) - f^{*})}{\epsilon^2} \log^4 \left( \frac{d\Delta f}{\epsilon^2 \delta} \right) \right).
\]

Comparing 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.

### 3 Main Results

In this section, we formalize the idea of perturbations adapted with occupation time. Our main result shows that the resulting algorithm converges rapidly to a second-order stationary point.
3.1 Perturbed Gradient Descent Adapted with 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 with random perturbation depending on the occupation dynamics. We follow the parameter setting as in [23]. Our algorithm performs GD with step size \( \eta \), and get a perturbation of amplitude \( r \) near saddle points at most once every \( t_{\text{thres}} \) iterations. The threshold \( t_{\text{thres}} \) ensures that the dynamics of the algorithm is mostly GD. The threshold \( g_{\text{thres}} \) determines if a perturbation is needed, and the threshold \( f_{\text{thres}} \) decides when the algorithm terminates.

**Algorithm 2** Perturbed Gradient Descent Adapted with Occupation Times (PGDOT)

\[
\begin{align*}
\chi &\leftarrow 3 \max \left\{ \log \left( \frac{d \Delta_f}{c \epsilon^2} \right), 4 \right\}, \quad \eta \leftarrow \frac{\epsilon \sqrt{d}}{\chi^2}, \quad r \leftarrow \frac{\epsilon \sqrt{d}}{\chi^2} \\
g_{\text{thres}} &\leftarrow \frac{\chi}{\sqrt{c}}, \quad f_{\text{thres}} \leftarrow \frac{\chi}{\sqrt{c}}, \quad t_{\text{thres}} \leftarrow \frac{\chi}{\sqrt{c}} \\
t_{\text{noise}} &\leftarrow t_{\text{thres}} - 1 \\
\text{for } t = 0, 1, \ldots \text{ do} \\
&\quad \text{if } |\nabla f(x_t)| \leq g_{\text{thres}} \text{ and } t - t_{\text{noise}} > t_{\text{thres}} \text{ then} \\
&\quad \quad \tilde{x}_t \leftarrow x_t, \quad t_{\text{noise}} \leftarrow t \\
&\quad \text{for } i = 1, \ldots, d \text{ do} \\
&\quad \quad G_t^i \leftarrow \# \{ s < t : x^i_s < x^i_t \} \text{ and } D_t^i \leftarrow \# \{ s < t : x^i_s \geq x^i_t \} \\
&\quad \quad x^i_{t+1} \leftarrow \begin{cases} 
\tilde{x}^i_t - \frac{r}{\sqrt{d}} \text{Unif}(0, 1) & \text{with probability } \frac{w(D_t^i)}{w(G_t^i) + w(D_t^i)} \\
\tilde{x}^i_t + \frac{r}{\sqrt{d}} \text{Unif}(0, 1) & \text{with probability } \frac{w(G_t^i)}{w(G_t^i) + w(D_t^i)} 
\end{cases} \\
&\quad \text{if } t - t_{\text{noise}} = t_{\text{thres}} \text{ and } f(x_t) - f(x_{\text{noise}}) > -f_{\text{thres}} \text{ then} \\
&\quad \quad \text{return } x_{t_{\text{noise}}} \\
&\quad x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)
\end{align*}
\]

The next theorem gives the convergence rate of Algorithm 2. PGDOT finds a second-order stationary point in the same number of iterations (up to a constant factor) as PGD does.

**Theorem 3.1.** Assume that \( f : \mathbb{R}^d \to \mathbb{R} \) is \( \ell \)-gradient Lipschitz and \( p \)-Hessian Lipschitz. Then there exists \( c_{\text{max}} > 0 \) such that for any \( \delta > 0, \epsilon \leq \ell^2 / \rho, \Delta_f \geq f(x_0) - f^* \), and \( c \leq c_{\text{max}} \). Algorithm 2 (PGDOT) outputs an \( \epsilon \)-second-order stationary point with probability \( 1 - \delta \), terminating within the number of iterations

\[
O \left( \frac{(f(x_0) - f^*)}{\epsilon^2} \log^4 \left( \frac{d \Delta_f}{\epsilon^2 \delta} \right) \right).
\]

The proof of Theorem 3.1 is based on a geometric characterization of saddle points – thin pancake property [23]. In Section 3.2, we will discuss this property, and show how it is used to prove Theorem 3.1. Moreover, the following theorem suggests that PGDOT avoids getting stuck at saddle point, as the dynamics of vertex-diminishing random walk prescribed in (14) does not localize.

**Theorem 3.2.** Let \( \{Z_t, t = 0, 1, \ldots\} \) be vertex-diminishing random walk defined by

\[
Z_{t+1} = \begin{cases} 
Z_t - 1 & \text{with probability } \frac{w(D_t)}{w(G_t) + w(D_t)}, \\
Z_t + 1 & \text{with probability } \frac{w(G_t)}{w(G_t) + w(D_t)}
\end{cases}
\]

where \( w : \{0, 1, \ldots \} \to (0, \infty) \) is an increasing function such that \( w(n) \to \infty \) as \( n \to \infty \). Then \( \mathbb{P} (\exists t_0 > 0, k \leq \ell : Z_t \in \{k, \ldots, \ell\} \text{ for all } t \geq t_0) = 0 \).

**Proof.** Suppose by contradiction that with positive probability, the walk is localized at some points \( \{k, \ldots, \ell\} \). We focus on the left end \( k \). Let \( \tau^n_k \) 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 \( \tau^n_k \). So at time \( \tau^n_k \), 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 \( \{k, \ldots, \ell\} \) 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. \( \square \)
### 3.2 Proof of Theorem 3.1

We show how the thin-pancake property of saddle points is used to prove Theorem 3.1. Recall that an $\epsilon$-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 2 (PGDOT). If we are currently at an iterate $x_t$ which is not an $\epsilon$-second-order stationary point, there are two cases: (1) The gradient is large: $\|\nabla f(x_t)\| \geq \delta_{\text{thres}}$; (2) $x_t$ is close to a saddle point: $\|\nabla f(x_t)\| \leq \delta_{\text{thres}}$ and $\lambda_{\text{min}}(\nabla^2 f(x_t)) \leq -\sqrt{\epsilon}$. The case (1) is easy to deal with by the following elementary lemma.

**Lemma 3.3.** Assume that $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is $\ell$-gradient Lipschitz. Then for GD with step size $\eta < \ell^{-1}$, we have $f(x_{t+1}) - f(x_t) \leq -\frac{\eta}{2} \|\nabla f(x_t)\|^2$.

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

**Lemma 3.4.** Assume that $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is $\ell$-gradient Lipschitz and $\rho$-Hessian Lipschitz. If $\|\nabla f(x_t)\| \leq \delta_{\text{thres}}$ and $\lambda_{\text{min}}(\nabla^2 f(x_t)) \leq -\sqrt{\epsilon}$, then adding one perturbation step as in Algorithm 2 followed by $\ell_{\text{thres}}$ steps of GD with step size $\eta$, we have $f(x_{t+1} + \ell_{\text{thres}}) - f(x_t) \leq -f_{\text{thres}}$ with probability at least $1 - \frac{\ell_{\text{thres}}}{\sqrt{\epsilon}} e^{-\lambda}$.

[23] proved Lemma 3.4 for PGD, and used it together with Lemma 3.3 to prove Theorem 2.5. We will use the same argument, with Lemmas 3.3 and 3.4, leading to Theorem 3.1 for PGDOT.

Now, let us explain how to prove Lemma 3.4 via a purely geometric property of saddle points. Consider a point $\bar{x}$ satisfying the condition $\|\nabla f(\bar{x})\| \leq \delta_{\text{thres}}$ and $\lambda_{\text{min}}(\nabla^2 f(\bar{x})) \leq -\sqrt{\epsilon}$. After adding the perturbation in Algorithm 2, the resulting vector can be viewed as a distribution over the cube $C^d(\bar{x}, r/\sqrt{d})$. Similar as in [23], we call $C^d(\bar{x}, r/\sqrt{d})$ the perturbation cube which is divided into two regions: (1) escape region $\chi_{\text{escape}}$, which consists of all points $x \in C^d(\bar{x}, r/\sqrt{d})$ whose function value decreases by at least $\delta_{\text{thres}}$ after $\ell_{\text{thres}}$ steps; (2) stuck region $\chi_{\text{stuck}}$ which is the complement of $\chi_{\text{escape}}$ in $C^d(\bar{x}, r/\sqrt{d})$. The key idea is that the stuck region $\chi_{\text{stuck}}$ looks like a non-flat thin pancake, which has a very small volume compared to that of $\chi_{\text{escape}}$.

**Lemma 3.5.** [23] Assume that $\bar{x}$ satisfies $\|\nabla f(\bar{x})\| \leq \delta_{\text{thres}}$ and $\lambda_{\text{min}}(\nabla^2 f(\bar{x})) \leq -\sqrt{\epsilon}$. Let $e_1$ be the smallest eigendirection of $\nabla^2 f(\bar{x})$. For any $\delta < 1/3$ and any $u, v \in C^d(\bar{x}, r/\sqrt{d})$, if $u - v = \mu e_1$ and $\mu \geq \delta/(2\sqrt{d})$, then at least one of $u$ and $v$ is not in the stuck region $\chi_{\text{stuck}}$.

To prove Lemma 3.4, it suffices to check that $\mathbb{P}(\chi_{\text{stuck}}) \leq C\delta$ for some $C > 0$. This criterion is general for any (random) perturbation. Let $O_1, \ldots, O_{2d}$ be the orthants centered at $\bar{x}$; that is, the space $\mathbb{R}^d$ is divided into $2^d$ subspaces according to the coordinate signs of $\cdot - \bar{x}$. The symbol $\text{sgn}(O_i) \in \{-1, 1\}^d$ denotes the coordinate signs of $y - \bar{x}$ for any $y \in O_i$. For $1 \leq i \leq 2^d$, let

$$p_i := \prod_{\text{sgn}(O_i)_k = -1} w(D^k_i^-) \prod_{\text{sgn}(O_i)_k = +1} w(D^k_i^+) \prod_{\text{sgn}(O_i)_k = 0} w(G^k_i)$$

be the probability that the random perturbation drives $\bar{x}$ into $C^d(\bar{x}, r/\sqrt{d}) \cap O_i$. Consequently, $\mathbb{P}(\chi_{\text{stuck}}) = \sum_{i=1}^{2^d} p_i \frac{\text{Vol}(\chi_{\text{stuck}} \cap O_i)}{\text{Vol}(C^d(\bar{x}, r/\sqrt{d}) \cap O_i)}$, where $\text{Vol}(\cdot)$ denotes the volume of a domain. It is easy to see that $\text{Vol}(C^d(\bar{x}, r/\sqrt{d}) \cap O_i) = (r/\sqrt{d})^d$. By Lemma 3.5 and the slicing volume bound $\text{Vol}(\chi_{\text{stuck}} \cap O_i) \leq \sqrt{2}(r/\sqrt{d})^{d-1} \frac{\delta r}{\sqrt{d}}$. Therefore, $\frac{\text{Vol}(\chi_{\text{stuck}} \cap O_i)}{\text{Vol}(C^d(\bar{x}, r/\sqrt{d}) \cap O_i)} \leq \sqrt{2}\delta$ implying that $\mathbb{P}(\chi_{\text{stuck}}) \leq \sqrt{2}\delta$.

Note that this proof does not rely on the full history of states for $G_t$ and $D_t$. Thus, one can restrict the number of previous iterations as done in the next section using the hyperparameter $t_{\text{count}}$.

### 4 Empirical Results

This section presents empirical results to corroborate theoretical analysis presented in the previous section. In examples 1 through 3, synthetic objective functions are used to demonstrate the advantage...
of PGDOT over PGD in cases where GD gets trapped near saddle points. In example 4, a mini-batch version of PGDOT is compared to mini-batch GD and Adam in training MLPs. The results show that mini-batch PGDOT outperforms those alternatives and is robust with different initializations.

In these experiments, we use $G_i'(h) := \# \{ t : t_{\text{count}} \leq s < t : x_i^t - h \leq x_i^t < x_i^t + h \}$ and $D_i'(h) := \# \{ t : t_{\text{count}} \leq s < t : x_i^t \leq x_i^t < x_i^t + h \}$ instead of $G_i'$ and $D_i'$ in Algorithm 2. Here $h$ is a hyperparameter characterizing the occupation time over a small interval. $t_{\text{count}}$ is another hyperparameter prescribing how long one should keep track of the history of $x_i$ in order to approximate the occupation time with a constant memory cost. We choose the weight function in Algorithm 2 as $w(n) = 1 + n^{0.3}$. All the hyperparameters used in the numerical examples as well as the total amount of compute will be reported in Appendix C.

**Example 1** Given $N \in \mathbb{Z}^+, L \in \mathbb{R}^+$, define a function $\tilde{f} : \mathbb{R}^+ \to \mathbb{R}^+$ as

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

For $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, we define $f(\mathbf{x}) = \tilde{f} \left( \frac{1}{d} \sum_{i=1}^{d} x_i^2 \right)$. Figure 2(a)(b) gives the visualization of the case $N = 4, L = 1$. Figure 2(c) presents the training curves of $f$ given by GD, PGD, and PGDOT. The initial values are all the same, and PGD and PGDOT are run 5 times considering the randomness of perturbation. We can see that PGDOT performs better and is more robust than PGD.

(a) staircase-like function $\tilde{f}$  
(b) Landscape of $f(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^2$  
(c) Learning curves when $d = 4$

Figure 2: Example 1 in the case of $N = 4, L = 1$.

**Example 2** We consider a nonlinear regression problem, adapted from learning time series data with a continuous dynamical system [32]. The loss function is defined as $f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} (\tilde{y}(s_i; \mathbf{x}) - y^*(s_i))^2$, where $\{s_i\}_{i=1}^{N}$ are $N$ sample points, $y^*(s)$ is the target function, and $\tilde{y}(s)$ is the function to fit with the form $\tilde{y}(s; \mathbf{x}) = \sum_{m=1}^{M} (a_m \cos(\lambda_ms) + b_m \sin(\lambda_ms))e^{\omega m s}$.

Here $\mathbf{x} = \{a_m, b_m, \lambda_m, \omega m \}_{m=1}^{M}$ and the optimization problem is non-convex. We assume $y^*(s) = \text{Ai}(\omega |s - s_0|)$, where $\omega = 3.2$, $s_0 = 3.0$, and $\text{Ai}(s)$ is the Airy function of the first kind, given by the improper integral $\text{Ai}(s) = \frac{1}{\pi} \int_{-\infty}^{\infty} \cos \left( \frac{u^3}{3} + su \right) du$.

For the specific regression model, we assume $M = 4$ and use $N = 50$ data points with $s_i = i/10, i = 0, \ldots, 49$. Figure 3(a) shows the target function and the fitted function obtained by PGDOT. We use three algorithms to optimize the parameters and present the learning curves in Figure 3(b). It is seen that there are saddle points when the loss is around 1, and the proposed PGDOT helps to escape those saddle points efficiently.

**Example 3** The next two non-convex optimization problems are taken from [58]. The first problem is a regularized linear-quadratic problem [47], whose loss function is

$$
f_1(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{1}{2} \mathbf{x}^T H \mathbf{x} + \mathbf{b}_i^T \mathbf{x} + ||\mathbf{x}||_1^2 / 10 \right),
$$
where we take $N = 10$, $H = \text{diag}([1, -0.1])$ and $b_i$'s instances of $\mathcal{N}(0, \text{diag}([0.1, 0.001]))$. The second problem is the phase retrieval problem [5] with loss function

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

where we choose $N = 200$, $x^*$ an instance of $\mathcal{N}(0, I_d/d)$ and $a_i$'s instances of $\mathcal{N}(0, I_d)$ with $d = 10$.

We initialize the regularized linear-quadratic problem with $x_0 = 0$, and the phase retrieval problem with $x_0$ an instance of $\mathcal{N}(0, I_d/(10000d))$. Figure 4 presents the learning curves of the two problems trained by three different algorithms. For the regularized linear-quadratic problem, note that GD gets stuck around a saddle point and it takes some time before it escapes that region, while both PGD and PGDOT manage to escape the saddle point more efficiently. For the phase retrieval problem, we see that GD is trapped near a saddle point, and PGDOT escapes the saddle point faster than PGD.

Example 4 Dauphin et al. [10] observed that in training simple MLPs (with only one hidden layer) on the MNIST and CIFAR-10 datasets, mini-batch GD may get stuck at saddle points. Moreover, as demonstrated in [53], Adam also gets stuck and performs poorly when a simple MLP is trained on the MNIST dataset with “bad” initialization. Inspired by these observations, we use the mini-batch version of PGDOT to train several simple MLPs on the MNIST and CIFAR-10 datasets, and compare with other algorithms including Adam.

We start experiments with “good” initialization so that the weights and biases of the simple MLPs are initialized by $\mathcal{N}(0, 0.01)$. $n_{\text{hidden}}$ is the number of neurons in the hidden layer. Figure 5 shows the training curves of different optimizers: mini-batch GD, Adam, PGD, and PGDOT. Observe that mini-batch Adam slightly outperforms the rest for the MNIST dataset, while mini-batch GD, PGD, and PGDOT escape the saddle point more efficiently for the CIFAR-10 dataset.

Now, we consider the simple MLPs with “bad” initialization so that the weights and biases are initialized with $\mathcal{N}(-1, 0.01)$. Figure 6 shows the training curves of four different mini-batch optimizers. (a) The target function $y^*(t)$ and fitted function $\hat{y}(t)$ obtained by PGDOT.

Figure 3: Numerical experiments of Example 2.

Figure 4: Numerical experiments of Example 3.
algorithms. Observe that mini-batch PGDOT outperforms the alternatives for both datasets. This result also implies that mini-batch PGDOT, in contrast to mini-batch GD and Adam, is robust with different initializations. We also mention that other variants of Adam such as AMSGrad [48] and RMSProp fail to escape the saddle points in the training process as well (see Appendix D).

5 Conclusion

In this paper, we develop an algorithm, PGDOT, to escape saddle points efficiently in the context of non-convex optimization. The main idea is to perturb GD in a non-uniform direction according to the history of states. We prove that PGDOT converges rapidly to a second-order stationary point,
and this is corroborated by empirical studies ranging from time series analysis, the phase retrieval problem to neural networks.

There are a few directions to extend this work. For instance, providing a theoretical guarantee of the mini-batch PGDOT is important. We also remark that mini-batch PGDOT escapes saddle points efficiently but at the expense of more computational efforts. It is of interest to study efficient ways to reduce the computational burden of the algorithm (possibly with parallel computing).

References

[1] K. Ball. Cube slicing in \( \mathbb{R}^n \). *Proc. Amer. Math. Soc.*, 97(3):465–473, 1986.
[2] A. S. Bandeira, N. Boumal, and V. Voroninski. On the low-rank approach for semidefinite programs arising in synchronization and community detection. In *Conference on Learning Theory*, pages 361–382, 2016.
[3] S. Bhojanapalli, B. Neyshabur, and N. Srebro. Global optimality of local search for low rank matrix recovery. In *Advances in Neural Information Processing Systems*, pages 3873–3881, 2016.
[4] S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.
[5] E. J. Candès, Y. C. Eldar, T. Strohmer, and V. Voroninski. Phase retrieval via matrix completion. *SIAM J. Imaging Sci.*, 6(1):199–225, 2013.
[6] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *J. ACM*, 58(3):Art. 11, 37, 2011.
[7] E. J. Candès, X. Li, and M. Soltanolkotabi. Phase retrieval via Wirtinger flow: theory and algorithms. *IEEE Trans. Inform. Theory*, 61(4):1985–2007, 2015.
[8] A. Cauchy. Méthode générale pour la résolution des systemes d’équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.
[9] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.
[10] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. 2014. arXiv:1406.2572.
[11] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.
[12] F. Draxler, K. Veschgini, M. Salmhofer, and F. Hamprecht. Essentially no barriers in neural network energy landscape. In *International Conference on Machine Learning*, pages 1309–1318, 2018.
[13] S. S. Du, C. Jin, J. D. Lee, M. I. Jordan, A. Singh, and B. Poczos. Gradient descent can take exponential time to escape saddle points. In *Advances in Neural Information Processing Systems*, pages 1067–1077, 2017.
[14] S. S. Du, J. D. Lee, Y. Tian, A. Singh, and B. Poczos. Gradient descent learns one-hidden-layer CNN: Don’t be afraid of spurious local minima. In *International Conference on Machine Learning*, pages 1339–1348, 2018.
[15] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points – online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015.
[16] R. Ge, C. Jin, and Y. Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. In *International Conference on Machine Learning*, pages 1233–1242, 2017.
[17] R. Ge, J. D. Lee, and T. Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems*, pages 2973–2981, 2016.
[18] R. Ge, J. D. Lee, and T. Ma. Learning one-hidden-layer neural networks with landscape design. In *International Conference on Learning Representations*, 2018.
[19] R. Ge and T. Ma. On the optimization landscape of tensor decompositions. In Advances in Neural Information Processing Systems, pages 3653–3663, 2017.

[20] S. Geman and C.-R. Hwang. Diffusions for global optimization. SIAM J. Control Optim., 24(5):1031–1043, 1986.

[21] R. A. Holley, S. Kusuoka, and D. W. Stroock. Asymptotics of the spectral gap with applications to the theory of simulated annealing. J. Funct. Anal., 83(2):333–347, 1989.

[22] P. Jain, C. Jin, S. Kakade, and P. Netrapalli. Global convergence of non-convex gradient descent for computing matrix squareroot. In Artificial Intelligence and Statistics, pages 479–488, 2017.

[23] C. Jin, R. Ge, P. Netrapalli, S. Kakade, and M. I. Jordan. How to escape saddle points efficiently. In International Conference on Machine Learning, pages 1724–1732, 2017.

[24] C. Jin, P. Netrapalli, R. Ge, S. M. Kakade, and M. I. Jordan. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. Journal of the ACM, 68(2):1–29, 2021.

[25] K. Kawaguchi. Deep learning without poor local minima. In Advances in Neural Information Processing Systems, pages 586–594, 2016.

[26] A. Kazemipour, B. Larsen, and S. Druckmann. Avoiding spurious local minima in deep quadratic networks. 2019. arXiv:2001.00098.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In International Conference on Learning Representations, 2015.

[28] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.

[29] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[31] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht. Gradient descent only converges to minimizers. In Conference on Learning Theory, pages 1246–1257, 2016.

[32] Z. Li, J. Han, W. E, and Q. Li. On the curse of memory in recurrent neural networks: Approximation and optimization analysis. In International Conference on Learning Representations, 2021.

[33] S. Liang, R. Sun, J. D. Lee, and R. Srikant. Adding one neuron can eliminate all bad local minima. In Advances in Neural Information Processing Systems, pages 4350–4360, 2018.

[34] S. Mei, Y. Bai, and A. Montanari. The landscape of empirical risk for nonconvex losses. Ann. Statist., 46(6A):2747–2774, 2018.

[35] S. Mei, T. Misiakiewicz, A. Montanari, and R. I. Oliveira. Solving SDPs for synchronization and MaxCut problems via the Grothendieck inequality. In Conference on Learning Theory, pages 1476–1515, 2017.

[36] G. Menz, A. Schlichting, W. Tang, and T. Wu. Ergodicity of the infinite swapping algorithm at low temperature. 2018. arXiv:1811.10174.

[37] L. Miclo. Recuit simulé sur $\mathbb{R}^n$. Étude de l’évolution de l’énergie libre. Ann. Inst. H. Poincaré Probab. Statist., 28(2):235–266, 1992.

[38] P. Monmarché. Hypocoercivity in metastable settings and kinetic simulated annealing. Probab. Theory Related Fields, 172(3-4):1215–1248, 2018.

[39] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. 2015. arXiv:1511.06807.

[40] Y. Nesterov. Introductory lectures on convex optimization: A basic course, volume 87 of Applied Optimization. Kluwer Academic Publishers, Boston, MA, 2004.

[41] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. In Dokl. Akad. Nauk SSSR, volume 269, pages 543–547, 1983.

[42] Q. Nguyen and M. Hein. The loss surface of deep and wide neural networks. In International Conference on Machine Learning, pages 2603–2612, 2017.
[43] D. Park, A. Kyrillidis, C. Carmanis, and S. Sanghavi. Non-square matrix sensing without spurious local minima via the burer-monteiro approach. In Artificial Intelligence and Statistics, pages 65–74, 2017.

[44] I. Pavlyukevich. Lévy flights, non-local search and simulated annealing. Journal of Computational Physics, 226(2):1830–1844, 2007.

[45] R. Pemantle. Vertex-reinforced random walk. Probab. Theory Related Fields, 92(1):117–136, 1992.

[46] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.

[47] S. Reddi, M. Zaheer, S. Sra, B. Poczos, F. Bach, R. Salakhutdinov, and A. Smola. A generic approach for escaping saddle points. In International Conference on Artificial Intelligence and Statistics, pages 1233–1242, 2018.

[48] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. In International Conference on Learning Representations, 2018.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. Nature, 323(6088):533–536, 1986.

[50] M. Sanjabi, S. Baharlouei, M. Razaviyayn, and J. D. Lee. When does non-orthogonal tensor decomposition have no spurious local minima? 2019. arXiv:1911.09815.

[51] J. Sun, Q. Qu, and J. Wright. Complete dictionary recovery over the sphere I: Overview and the geometric picture. IEEE Trans. Inform. Theory, 63(2):853–884, 2017.

[52] J. Sun, Q. Qu, and J. Wright. A geometric analysis of phase retrieval. Found. Comput. Math., 18(5):1131–1198, 2018.

[53] G. Swirszcz, W. M. Czarnecki, and R. Pascanu. Local minima in training of deep networks. 2016. Available at https://openreview.net/pdf?id=Syoiqwcxx.

[54] W. Tang and X. Y. Zhou. Simulated annealing from continuum to discretization: a convergence analysis via the Eyring–Kramers law. 2021. arXiv:2102.02339.

[55] P. Tarrès. Vertex-reinforced random walk on Z eventually gets stuck on five points. Ann. Probab., 32(3B):2650–2701, 2004.

[56] L. Venturi, A. S. Bandeira, and J. Bruna. Spurious valleys in one-hidden-layer neural network optimization landscapes. J. Mach. Learn. Res., 20:Paper No. 133, 34, 2019.

[57] S. Volkov. Phase transition in vertex-reinforced random walks on Z with non-linear reinforcement. J. Theoret. Probab., 19(3):691–700, 2006.

[58] J.-K. Wang, C.-H. Lin, and J. Abernethy. Escaping saddle points faster with stochastic momentum. In International Conference on Learning Representations, 2019.

[59] C. Wu, J. Luo, and J. D. Lee. No spurious local minima in a two hidden unit ReLU network. 2018. Available at https://openreview.net/forum?id=B14uJzW0b.

[60] X.-S. Yang and S. Deb. Cuckoo search via Lévy flights. In 2009 World congress on nature & biologically inspired computing (NaBIC), pages 210–214, 2009.

[61] M. Zhou, T. Liu, Y. Li, D. Lin, E. Zhou, and T. Zhao. Toward understanding the importance of noise in training neural networks. In International Conference on Machine Learning, 2019.
A Background on convex optimization

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

Definition A.1.

1. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz if $\|\nabla f(x_1) - \nabla f(x_2)\| \leq \ell \|x_1 - x_2\|$ for all $x_1, x_2 \in \mathbb{R}^d$.

2. A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is $\alpha$-strongly convex if $\lambda_{\text{min}}(\nabla^2 f(x)) \geq \alpha$ for all $x \in \mathbb{R}^d$.

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 $\ell/\alpha$ is often called the condition number of the function $f$. The following theorem shows the linear convergence of gradient descent to the global minimum $x^*$, see [4, Theorem 3.10] and [40, Theorem 2.1.15].

Theorem A.2. [4, 40] Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz and $\alpha$-strongly convex. For any $\epsilon > 0$, if we run gradient descent with step size $\eta = \ell^{-1}$, then the number of iterations to be $\epsilon$-close to $x^*$ is $\frac{\alpha}{\epsilon} \log \left( \frac{\|x_0 - x^*\|}{\epsilon} \right)$.

B 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 perturbed gradient descent.

Proposition B.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 $\ell$-smooth and the step size is less than $\frac{\ell}{2}$, 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 B.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 $\ell$-smooth and the step size is less than $\frac{\ell}{2}$, 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

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

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 hand 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
Lemma B.2. The sequence \( \{x_n\}_{n \geq 0} \) converges monotonically to an upper bound. Let \( M \) be the minimum index that satisfies the condition in Lemma B.2. Suppose by contradiction that \( M > 0 \). So \( x_M < x_{M-1} < x_M \), which implies \( f'(x_{M-1}) < 0 \) considering \( x_{M-1} = x_M - \eta f'(x_{M-1}) \). Since the sequence converges to \( x_s \), there should be a \( k \geq 0 \) such that \( x_{M+k} < x_{M-1} < x_{M+k} \). Note that \( x_{M+k} = x_M + k - \eta f'(x_{M+k}) \), 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 \( \eta f'(x_{M+k}) - f'(x_{M-1}) > \eta f'(x_{M+k}) > x_{M+k} - x_{M-1} \). This contradicts the fact that \( \eta f'(x_{M+k}) - f'(x_{M-1}) \leq \eta \ell(x_{M+k} - x_{M-1}) < x_{M+k} - x_{M-1} \). □

C Hyperparameters in the numerical examples

Table 1 summarizes the hyperparameters used for empirical studies in Section 4. In example 4, batch size is set to 128 for both the MNIST and CIFAR-10 datasets.

| Example | \( d \) | # of steps | \( h \) | \( t_{\text{count}} \) | \( \eta \) | \( t_{\text{thres}} \) | \( g_{\text{thres}} \) | \( r \) |
|---------|--------|------------|--------|--------------------|-------|----------------|-------------|-------|
| Example 1 | 4 | 2500 | 0.04 | 200 | 0.04 | 4 | 0.01 | 0.04 |
| Example 2 | 16 | 35000 | 0.04 | 200 | 0.04 | 4 | 0.02 | 0.04 |
| Example 3 (1) | 2 | 4000 | 0.04 | 200 | 0.008 | 200 | 0.01 | 0.1 |
| Example 3 (2) | 10 | 900 | 0.04 | 200 | 0.001 | 50 | 1 | 0.01 |
| Example 4 (MNIST) | - | 150 (epochs) | 0.04 | 200 | 0.01 | 10 | 0.1 | 0.5 |
| Example 4 (CIFAR-10) | - | 150 (epochs) | 0.04 | 200 | 0.01 | 10 | 0.1 | 0.5 |

All the experiments are conducted on either a local machine or Google Colab using a CPU. Each of the first three examples takes a few minutes to run, and each of the experiments in example 4 takes a few hours.

D AMSGrad and RMSProp also fail

Here, we consider training simple MLPs on the MNIST and CIFAR-10 datasets using three minibatch algorithms: Adam, AMSGrad, and RMSProp. The weights and biases of the simple MLPs are initialized with \( \mathcal{N}(-1, 0.01) \). Note that we have used the PyTorch implementation of these algorithms with learning rate 0.01 (default values are used for the other parameters) and the batch size is set to 128.

Figure 7 shows the training curves of these algorithms. Observe that like Adam, both AMSGrad and RMSProp fail in escaping the saddle points (compare to Figure 6).
Figure 7: Training simple MLPs with “bad” initialization. Top: MNIST; Bottom: CIFAR 10.