A Convergent and Dimension-Independent First-Order Algorithm for Min-Max Optimization

Vijay Keswani  
Yale University  

Oren Mangoubi  
WPI  

Sushant Sachdeva  
University of Toronto  

Nisheeth K. Vishnoi  
Yale University  

June 7, 2021

Abstract

Motivated by the recent work of [33], we propose a variant of the min-max optimization framework where the max-player is constrained to update the maximization variable in a greedy manner until it reaches a first-order stationary point. We present an algorithm that provably converges to an approximate local equilibrium for our framework from any initialization and for nonconvex-nonconcave loss functions. Compared to the second-order algorithm of [33], whose iteration bound is polynomial in the dimension, our algorithm is first-order and its iteration bound is independent of dimension. We empirically evaluate our algorithm on challenging nonconvex-nonconcave test-functions and loss functions that arise in GAN training. Our algorithm converges on these test functions and, when used to train GANs, trains stably on synthetic and real-world datasets and avoids mode collapse.
1 Introduction

For a loss function \( f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \) on some (convex) domain \( \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^d \times \mathbb{R}^d \), we consider:

\[
\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} f(x, y).
\]

This min-max optimization problem has several applications to machine learning, including GANs \([20]\) and adversarial training \([32]\). In many of these applications, only first-order access to \( f \) is available efficiently, and first-order algorithms are widely used in practice.

Unfortunately, there is a lack of first-order algorithms with convergence guarantees for this min-max framework if one allows for loss functions \( f(x, y) \) which are nonconvex/nonconcave in \( x \) and \( y \). This lack of convergence guarantees can be a serious problem in practice, since popular algorithms such as gradient descent-ascent (GDA) oftentimes fail to converge, and GANs trained with these algorithms can suffer from issues such as cycling \([2]\) and “mode collapse” \([14, 9, 47]\).

Since min-max optimization includes minimization (and maximization) as special cases, it is intractable for general nonconvex/nonconcave functions. Motivated by the success of a long line of results which show efficient convergence of minimization algorithms to various (approximate) local minimum notions, e.g., \([43, 18, 1]\), previous works have sought to extend these ideas of local minima to various (approximate) notions of local min-max point—that is, a point \((x^*, y^*)\) where \( x^* \) is a local minimum of \( f(\cdot, y^*) \) and \( y^* \) is a local maximum of \( f(x^*, \cdot) \)—in the hope that this will allow for algorithms with convergence guarantees to such points. Unfortunately, in order to prove convergence, these works (e.g., \([41, 31]\)) make strong assumptions on \( f \), for instance assume \( f(x, y) \) is concave in \( y \), or that their algorithm is given a starting point such that its underlying dynamical system converges \([21, 36]\) (see also Related works). It is a challenge to develop first-order algorithms which converge efficiently to an equilibrium for even a local variant of the min-max framework under less restrictive assumptions comparable to those required for convergence of algorithms to local minima.

Our contributions. We propose a variant of the min-max framework which allows the max-player to update \( y \) in a “greedy” manner (Section 3.1). This greedy restriction, motivated by the recent work of \([33]\), models first-order maximization algorithms such as gradient ascent, popular in machine learning applications, which can make updates far from the current value of \( y \) when run for multiple steps. Roughly speaking, from the current point \((x, y)\), our framework allows the max-player to update \( y \) along any continuous path, along which the loss \( f(x, \cdot) \) is nondecreasing.

Our main contribution is a new first-order algorithm (Algorithm 1) that provably converges from any initial point to an approximate local equilibrium for this framework (Definition 3.2). In particular, for a \( b \)-bounded function \( f \) with \( L \)-Lipschitz gradient, and \( \varepsilon \geq 0 \), our algorithm requires \( \text{poly}(b, L, 1/\varepsilon) \) gradient and function oracle calls to converge to an \( \varepsilon \)-approximate local equilibrium (Theorem 3.4). The number of oracle calls required by our algorithm is independent of the dimension \( d \). First-order algorithms that converge in (almost) dimension independent number of iterations are required for practical machine-learning applications where the dimension \( d \), equal to the number of trainable parameters, can be very large.

In the special case when \( f \) is convex-concave and \( \mathcal{X}, \mathcal{Y} \) are compact, we further show that our framework is equivalent to the (global) min-max framework (Appendix A). While we state our algorithm for unconstrained domains, we show how to extend it to loss functions with compact convex domains (Appendix A). In particular, when applied to compactly supported bilinear functions, our algorithm can be shown to converge to a point near the global min-max point (Appendix A.2).
Our algorithm applied to the function $f(x, y) = (4x^2 - (y - 3x + 0.05x^3)^2 - 0.1y^4)e^{-0.01(x^2+y^2)}$ with global min-max point $(x, y) = (0, 0)$ (yellow star). Our algorithm’s max-player (red segments) first finds a point where $f(x, \cdot)$ is maximized. The min-player then proposes random updates to $x$, and only accepts those updates which lead to a decrease in the value of $f(x, y)$ after the response of the max-player is taken into account (blue segments). This allows our algorithm to converge to $(0, 0)$. This function is considered as a challenging test function in [51], who show that several first-order algorithms, namely GDA and OMD, as well as extra-gradient method [25], fail to converge on this function and instead cycle forever (see Figure 2).

Empirically, we show that our algorithm converges on test functions [51] on which other popular first-order min-max optimization algorithms such as GDA and optimistic mirror descent (OMD) [11] are known to either diverge or cycle (Figure 1, see also Figure 2 in Section 5.1). We also show that a practical variant of our algorithm can be employed for training GANs, with a per-step complexity and memory requirement similar to GDA. We observe that our algorithm consistently learns a greater number of modes than GDA, OMD, and unrolled GANs (Table 1 in Section 5), when applied to training GANs on a Gaussian mixture dataset. While not the focus of this paper, we also provide results for our algorithm on the real-world datasets in the Appendix.

**Discussion of equilibrium.** The equilibrium points $(x^*, y^*)$ our algorithm converges to can be viewed as local equilibria for a game where the maximizing player is restricted to making greedy updates to the value of $y$. Namely, the point $x^*$ is an approximate local minimum of an alternative to the function $\max_y f(\cdot, y)$ where, rather than maximizing over all $y \in \mathbb{R}^d$, the maximum is instead taken over all “greedy” paths—that is, paths along which $f(x^*, \cdot)$ is increasing—initialized at the current value $y^*$. Additionally, $y^*$ is an approximate local maximum of $f(x^*, \cdot)$ (see Section 3.1 for more details). In particular, we show that any point $(x^*, y^*)$ which is a local min-max point is also an equilibrium point for our algorithm (Corollary B.2 in the Appendix).

**Discussion of assumptions.** For our main result, we assume $f$ is bounded above and below. The assumption that the loss function is bounded below is standard in the minimization literature (see e.g., [43]), as an unbounded function need not achieve its minimum and a minimization algorithm could diverge in a manner such that the loss function value tends to $-\infty$. Thus, in min-max optimization, both the upper and lower bound assumptions are necessary to ensure the existence of
even an (approximate) global min-max point. If we drop the lower bound assumption, and only assume \( f(x, \cdot) \) is bounded above for \( x \in \mathbb{R}^d \), our algorithm still does not cycle: instead it either converges to a local equilibrium \((x^*, y^*)\), or the value of \( f \) diverges monotonically to \(-\infty\). Such functions include popular losses, including cross entropy [20], which is bounded above by zero, making our algorithm applicable to training GANs.

2 Related work

Local frameworks. In addition to the local min-max framework (e.g., \([41, 31, 21, 36]\)), previous works have proposed local frameworks where the max-player is able to choose its move after the min-player reveals its move. These frameworks include the local stackleberg equilibrium proposed in [15] and the closely related local minimax point proposed in [22]. In both the local min-max, as well as the local stackleberg and local minimax frameworks, the max-player is restricted to move in a small ball around the current point \( y \). In contrast, in our framework, the max-player can move much farther, as long as it is following a path along which the loss function is continuously increasing.

Convergence guarantees. Several works have studied the convergence properties of GDA dynamics \([10, 36, 5, 12, 22, 27]\), and established that GDA suffers from severe limitations: GDA can exhibit rotation around some points, or otherwise fail to converge. To address convergence issues for GDA, multiple works have analyzed algorithms based on Optimistic Mirror Descent (OMD), Extra-gradient methods, or similar approaches \([19, 12, 29, 13, 38]\). For instance, [11] guarantee convergence for OMD to a global min-max point on a bilinear loss \( f(x,y) = x^\top A y \), and [39] prove convergence of Extra-gradient (EG) methods for strongly convex-strongly concave losses. However, as observed in [51], GDA, OMD, and EG fail to converge on some simple nonconvex-nonconcave test functions; in comparison, we observe that our algorithm converges for these functions (Figure 2). Other works make additional assumptions to prove convergence – [35] show asymptotic convergence of OMD under a “coherence” assumption, and [5] show convergence of a second-order algorithm assuming that \( f \) corresponds to a Hamiltonian game. However, many simple functions, e.g. \( f(x,y) = \sin(x) \), do not satisfy these coherence or Hamiltonian game assumptions. For this reason, multiple works show convergence to an (approximate) local min-max point. For instance [21] prove convergence of finite-step GDA to a local min-max point, under the assumption that their algorithm is initialized such that the underlying continuous dynamics converge to a local min-max point. And [30] show convergence if their algorithm is initialized in a small enough neighborhood containing a local min-max point. In addition, many works provide convergence guarantees to a local stackleberg or local minimax point, under the assumption that their algorithm is provided with a starting point in the region of attraction \([16, 15]\), or a small enough neighborhood \([31]\), of such an equilibrium. And other works \([12, 24, 11, 45, 31, 80, 44, 49]\) provide convergence guarantees to an approximate local min-max point when \( f \) may be nonconvex in \( x \), but is concave in \( y \). In contrast to the above works, our algorithm is guaranteed to converge for any nonconvex-nonconcave loss, from any starting point, in a number of gradient evaluations that is independent of the dimension \( d \) and polynomial in \( L \) and \( b \) if \( f \) is \( b \)-bounded with \( L \)-Lipschitz gradient. Such smoothness/Lipschitz bounds are standard in convergence guarantees for optimization algorithms \([8, 43, 18, 50]\).

Greedy paths. The paths along which the max-player is allowed to make updates in our framework are inspired from the work of [33], which gives a second-order algorithm for min-max optimization.
The “greedy paths” considered in their work are defined such that at every point along these paths, \( f \) is non-decreasing, and the first derivative of \( f \) is at least \( \varepsilon \) or the 2nd derivative is at least \( \sqrt{\varepsilon} \). In contrast, we just require a condition on the first derivative of \( f \) along the path. This distinction gives rise to a different framework and equilibrium than the one presented in their work. Secondly, [33] is a second-order method that converges to an \( \varepsilon \)-approximate local equilibrium in \( \text{poly}(d, b, L, 1/\varepsilon) \) Hessian evaluations. On the other hand, the convergence bound of our algorithm is independent of \( d \); it requires \( \text{poly}(b, L, 1/\varepsilon) \) gradient evaluations for convergence.

**Training GANs.** An important line of work focuses on designing min-max optimization algorithms that mitigate the non-convergence behavior such as cycling when training GANs using GDA [20]. [11] show that OMD can mitigate cycling when training GANs with Wasserstein loss. In contrast to both GDA and OMD, where at each iteration the min-player and max-player are allowed only to make small updates roughly proportional to their respective gradients, our algorithm empowers the max-player to make large updates at each iteration. [37] introduced Unrolled GANs, where the min-player optimizes an “unrolled” loss function that allows the min-player to simulate a fixed number of max-player updates. While this has some similarity to our algorithm the main distinction is that the min-player in Unrolled GANs may not reach a first-order stationary point, and hence their algorithm does not have any convergence guarantees. We observe that our algorithm, applied to training GANs, trains stably and avoids mode collapse.

### 3 Theoretical results

#### 3.1 Framework and equilibrium

We consider a variant of the min-max framework, which empowers the max-player to update \( y \) in a “greedy” manner. More specifically, we restrict the max-player to update the current point \((x, y)\) to any point in a set \( P(x, y) \) consisting of the endpoints of paths in \( \mathcal{Y} \) initialized at \( y \) along which \( f(x, \cdot) \) is nondecreasing. These paths model the paths taken by a class of first-order algorithms, which includes popular algorithms such as gradient ascent. Our framework therefore allows the min-player to learn from max-players which are computationally tractable and yet (in contrast to the local min-max framework) are still empowered to make updates to the value of \( y \) which may lead to large increases in \( f(x, y) \). Given a bounded loss \( f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \), where \( \mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^d \) are convex, an equilibrium for our framework is a point \((x^*, y^*) \in \mathcal{X} \times \mathcal{Y} \) such that

\[
\begin{align*}
    x^* &\in \arg\min_{x \in \mathcal{X}} \left( \max_{y \in P(x, y^*)} f(x, y) \right), \\
    y^* &\in \arg\max_{y \in P(x^*, y^*)} f(x^*, y).
\end{align*}
\]

This is in contrast to the (global) min-max framework of (1) where the maximum is taken over all \( y \in \mathcal{Y} \). However, solutions to (2) and (3) may not exist, and even when they do exist, finding such a solution is intractable in general since (2) generalizes nonconvex minimization.

**Local equilibrium.** Replacing the global minimum in (2) with a local minimum leads to the following local version of our framework’s equilibrium. A point \((x^*, y^*) \in \mathcal{X} \times \mathcal{Y} \) is a local equilibrium if, for some \( \nu > 0 \) (and denoting the ball of radius \( \nu \) centered at \( x^* \) by \( B(x^*, \nu) \)),

\[
\begin{align*}
    x^* &\in \arg\min_{x \in \mathcal{X}} \left( \max_{y \in P(x, y^*)} f(x, y) \right), \\
    y^* &\in \arg\max_{y \in P(x^*, y^*)} f(x^*, y).
\end{align*}
\]
\[ x^* \in \arg\min_{x \in B(x^*, \nu) \cap \mathcal{X}} \left( \max_{y \in \mathcal{P}(x, y^*)} f(x, y) \right) \quad \text{(4)} \\
y^* \in \arg\max_{y \in \mathcal{P}(x^*, y^*)} f(x^*, y) \quad \text{(5)} \\

**Approximate local equilibrium.** Similar to previous work on local minimization for smooth non-convex objectives, we would like to solve (4) and (5) to converge to approximate first-order stationary points (points where, roughly speaking, the magnitude of the gradient is bounded by some positive number \( \| \nabla f(x) \| \)). Towards this end, along similar lines as the framework of [33], we can replace \( P(x, y^*) \) in Equations (4) and (5) with the set \( P_e(x, y^*) \) of endpoints of paths starting at \( y^* \) along which \( f(x, \cdot) \) increases at the “rate” of \( \varepsilon \), for a given \( \varepsilon > 0 \).

**Definition 3.1.** For any \( x \in \mathcal{X} \), \( y \in \mathcal{Y} \), and \( \varepsilon \geq 0 \), define \( P_e(x, y) \subseteq \mathcal{Y} \) to be points \( w \in \mathcal{Y} \) s.t. there is a continuous and (except at finitely many points) differentiable path \( \gamma(t) \) starting at \( y \), ending at \( w \), and unit-speed, i.e., \( \frac{d}{dt} \gamma(t) \parallel \leq 1 \), such that at any point on \( \gamma \), we have \( \frac{d}{dt} f(x, \gamma(t)) \geq \varepsilon \).

The above definition restricts the max-player to updating \( y \) via any “greedy” algorithm, e.g. gradient ascent. While this definition is similar to the “greedy paths” definition of [33], the crucial difference is that the definition in [33] also requires \( \frac{d^2}{dt^2} \) \( f(x, \gamma(t)) \geq \varepsilon \) so as to achieve the goal of converging to an approximate second-order local equilibrium. Our goal, on the other hand, is to approximate first-order equilibrium points and, hence, the condition on first derivative in Definition 3.1 suffices.

While we would also like to replace the local minimum in (4) with an approximate stationary point, the **min-player’s objective**, \( \mathcal{L}_e(x, y) := \max_{z \in \mathcal{P}_e(x, y)} f(x, z) \), may not be continuous in \( x \), and thus, gradient-based notions of approximate local minimum do not apply. To bypass this difficulty and to define a notion of approximate local minimum which applies to discontinuous functions, we sample updates to \( x \), and test whether \( \mathcal{L}_e(\cdot, y) \) has decreased. Formally, given a choice of sampling distribution \( Q_x \) (which may depend on \( x \)), and \( \delta, \omega > 0 \), \( x^* \) is said to be an approximate local minimum of a (possibly discontinuous) function \( g : \mathcal{X} \rightarrow \mathbb{R} \) if \( \Pr_{\Delta \sim Q_x, y^*} \left[ g(x^* + \Delta) < g(x^*) - \delta \right] < \omega \).

Thus, replacing the set \( P \) with \( P_e \) in equations (4) and the “exact” local minimum in (4) with an approximate local minimum, we arrive at our equilibrium definition:

**Definition 3.2.** Given \( \varepsilon, \delta, \omega > 0 \) and a distribution \( Q_{x,y} \), we say that a point \( (x^*, y^*) \in \mathcal{X} \times \mathcal{Y} \) is an approximate local equilibrium for our framework if

\[
\Pr_{\Delta \sim Q_{x^*, y^*}} \left[ \max_{y \in \mathcal{P}_e(x^* + \Delta, y^*)} f(x^* + \Delta, y) < \max_{y \in \mathcal{P}_e(x^*, y^*)} f(x^*, y) - \delta \right] < \omega, \quad \text{(6)}
\]

\[
y^* \in \arg\max_{y \in \mathcal{P}_e(x^*, y^*)} f(x^*, y) \quad \text{(7)}
\]

One can set \( Q_{x,y} \) to be the standard normal distribution, or, as we do to train GANs, to be the distribution of stochastic gradients with mean \(-\nabla_x f(x, y)\) (see Remark 3.3).

**3.2 Algorithm**

We present an algorithm for our framework (Algorithm 1), along with the gradient ascent subroutine which it uses to compute max-player updates (Algorithm 2). In Theorem 3.4, we show that it is

---

¹Consider the example \( f(x, y) = \min(x^3 y^2, 1) \). The min-player’s objective for \( \varepsilon > 0 \) is \( \mathcal{L}_e(x, y) = f(x, y) \) if \( 2x^2y < \varepsilon \), and 1 otherwise. Thus \( \mathcal{L}_{1/2} \) is discontinuous at \( (1/2, 1) \).
able to efficiently find an approximate local equilibrium (Definition 3.2).

We consider bounded loss functions \( f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \), where \( f \) is an empirical risk loss over \( m \) training examples, i.e., \( f := \frac{1}{m} \sum_{i \in [m]} f_i \). We assume we are given access to \( f \) via a randomized oracle \( \mathcal{G} \) such that \( \mathbb{E}[\mathcal{G}] = f \). We call such an oracle a stochastic zeroth-order oracle for \( f \). We are also given randomized oracles \( G_x, G_y \) for \( \nabla_x f, \nabla_y f \), such that \( \mathbb{E}[G_x] = \nabla_x f \), and \( \mathbb{E}[G_y] = \nabla_y f \), and call such oracles stochastic gradient oracles for \( f \). These oracles are computed by randomly sampling “batches” \( B, B_x, B_y \subseteq [m] \) (iid, with replacement) and returning \( \mathcal{F} = \frac{1}{|B|} \sum_{i \in B} f_i, G_x = \frac{1}{|B_x|} \sum_{i \in B_x} \nabla x f_i \), and \( G_y = \frac{1}{|B_y|} \sum_{i \in B_y} \nabla y f_i \). For our convergence guarantees, we require the following bounds on standard smoothness parameters for functions \( f_i \) such that for all \( i \) and all \( x, y \), we have \(|f_i(x, y)| \leq b \), and \( L \) such that \( \|\nabla f_i(x, y) - \nabla f_i(x', y')\| \leq L\|x-x'\| + L\|y-y'\| \). These bounds imply \( f \) is also continuous, \( b \)-bounded, and \( L \)-gradient-Lipschitz.

Algorithm 1: Our algorithm for min-max optimization

**input:** Stochastic zeroth-order oracle \( \mathcal{F} \) for bounded loss function \( f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \) with \( L \)-Lipschitz gradient, stochastic gradient oracle \( G_y \) with mean \( \nabla y f \), Initial point \((x_0, y_0)\), Error parameters \( \varepsilon, \delta > 0 \)

**hyperparameters:** \( \eta > 0 \) (learning rate), \( r_{\text{max}} \) (maximum number of rejections); \( \tau_1 \) (for annealing);

1. Set \( i \leftarrow 0 \), \( r \leftarrow 0 \), \( \varepsilon_0 = \frac{\varepsilon}{2} \), \( f_{\text{old}} \leftarrow \infty \)
2. while \( r \leq r_{\text{max}} \) do
3. Sample \( \Delta_i \) from the distribution \( Q_{x,y} \),
4. Set \( X_{i+1} \leftarrow x_i + \Delta_i \) \{Compute the proposed update for the min-player\}
5. Run Algorithm 2 with inputs \( x \leftarrow X_{i+1}, y_0 \leftarrow y_i \), and \( \varepsilon' \leftarrow \frac{\varepsilon}{r + 1} \) \{max-player update\}
6. Set \( y_{i+1} \leftarrow y_{\text{stationary}} \) to be the output of Algorithm 2
7. Set \( f_{\text{new}} \leftarrow \mathcal{F}(X_{i+1}, Y_{i+1}) \) \{Compute the new loss value\}
8. Set \( \text{Accept}_i \leftarrow \text{True} \)
9. if \( f_{\text{new}} > f_{\text{old}} \leftarrow \frac{d}{\delta} \), then
10. Set \( \text{Accept}_i \leftarrow \text{False} \) with probability \( \max(0, 1 - e^{-\frac{\pi}{2}}) \) \{Decide to accept or reject\}
11. if \( \text{Accept}_i = \text{True} \) then
12. Set \( x_{i+1} \leftarrow X_{i+1}, y_{i+1} \leftarrow Y_{i+1} \) \{accept the proposed \( x \) and \( y \) updates\}
13. Set \( f_{\text{old}} \leftarrow f_{\text{new}}, r \leftarrow r + 1 \), \( \varepsilon_{i+1} \leftarrow \varepsilon_i \times \frac{1}{1-2\eta L} \)
14. else
15. Set \( x_{i+1} \leftarrow x_i, y_{i+1} \leftarrow y_i \), \( r \leftarrow r + 1 \), \( \varepsilon_{i+1} \leftarrow \varepsilon_i \) \{Reject the proposed updates\}
16. Set \( i \leftarrow i + 1 \)
17. return \((x^*, y^*) \leftarrow (x_i, y_i)\)

Overview and intuition of our algorithm. From the current point \((x, y)\), Algorithm 1 first proposes a random update \( \Delta \) from the given distribution \( Q_{x,y} \) to update the min-player’s parameters to \( x + \Delta \). In practice, we oftentimes choose \( Q_{x,y} \) to be the distribution of the (scaled) stochastic gradient \( -G_x \), although one may implement Algorithm 1 with any choice of distribution \( Q_{x,y} \).

Then, it updates the max-player’s parameters greedily by running gradient ascent using the stochastic gradients \( G_y \) until it reaches a first-order \( \varepsilon \)-stationary point \( y' \), that is, a point where \( \|\nabla_y f(x + \Delta, y')\| \leq \varepsilon \). Thus, the point \( y' \) satisfies (7). However, Algorithm 1 still needs to eventually find a pair of points \((x^*, y^*)\) where \( x^* \) is an approximate local minimum of the min-player’s objective \( L_\varepsilon(\cdot, y) \) in order to satisfy (6). Moreover, it must also ensure that this new point \( y^* \) satisfies (7). Towards this, Algorithm 1 does the following:
1) The algorithm re-uses this same point $y'$ to compute an approximation $f(x+\Delta, y')$ for $L_c(x+\Delta, y)$ in order to have access to the value of the min-player’s objective $L_c$ to be able to minimize it.
2) If $f(x+\Delta, y')$ is less than $f(x, y)$ the algorithm concludes that $L_c(x+\Delta, y)$ has decreased and, consequently, accepts the updates $x+\Delta$ and $y'$; otherwise it rejects both updates. We show that after accepting both $x$ and $y'$, $L_c(x+\Delta, y') < L_c(x, y)$, implying that the algorithm does not cycle.
3) It then starts the next iteration and proposes a new random update which again depends on its current position.
4) While Algorithm 1 does not cycle, to avoid getting stuck, if it is unable to decrease $L_c$ after roughly $\frac{1}{\varepsilon}$ attempts, it concludes w.h.p. that the current $x$ is an approximate local minimum for $L_c(\cdot, y)$ with respect to the given distribution. This is because, by definition, at an approximate local minimum, a random update from the given distribution has probability at most $\omega$ of decreasing the min-player’s objective $L_c$. We also show that the current $y$ is an $\varepsilon$-stationary point for $f(x, \cdot)$.

**Remark 3.3 (Proposal distribution).** In practice, we oftentimes choose $Q_{x,y}$ to be the stochastic (batch) gradient distribution with mean $-\nabla_x f$. This is motivated by the fact that adding stochastic gradient noise to steps taken by deep learning algorithms is known empirically to lead to better learning outcomes than, e.g., standard Gaussian noise (see e.g., [53]). Empirically, we observe this choice of $Q_{x,y}$ leads to GANs that are able to successfully learn the dataset’s distribution (Section 5).

Moreover, it can be shown that any update $\Delta$ which decreases $L_c$ must have a component in the direction of $-\nabla_x f$. To see why this is, note that once the min-player proposes an update $\Delta$ to $x$, the max-player’s updates will only increase $f$, i.e., $L_c(x+\Delta, y) \geq f(x+\Delta, y)$. Moreover, since $y$ is a first-order stationary point of $f(x, \cdot)$ (because $y$ was computed using gradient ascent in the previous iteration), we also have $L_c(x, y) = f(x, y)$. Therefore, we want an update $\Delta$ such that $f(x+\Delta, y) \leq L_c(x+\Delta, y) \leq L_c(x, y) = f(x, y)$, which implies that any proposed step which decreases $L_c$ must also decrease $f$ (converse is not true).

**Algorithm 2** Stochastic gradient ascent (for max-player updates)

**input:** Stochastic gradient oracle $G_y$ for $\nabla_y f$; initial points $x, y_0$; error parameter $\varepsilon'$

**hyperparameters:** $\eta > 0$

1: Set $j \leftarrow 0$, Stop $\leftarrow$ False
2: **while** Stop = False **do**
3: Set $g_{y,j} \leftarrow G_y(x, y_j)$
4: **if** $\|g_{y,j}\| > \varepsilon'$ **then**
5: Set $y_{j+1} \leftarrow y_j + \eta g_{y,j}$
6: Set $j \leftarrow j + 1$
7: **else**
8: Set Stop $\leftarrow$ True
9: **return** $y_{\text{stationary}} \leftarrow y_j$

We conclude this section with a few remarks: 1) In practice our algorithm can be implemented just as easily with ADAM instead of SGD, as in some of our experiments. 2) Algorithm 1 uses a randomized accept-reject rule inspired by simulated annealing – if the resulting loss has decreased, the updates for $x$ and $y$ are accepted; otherwise they are only accepted with a small probability $e^{-\varepsilon'}/\tau_1$ at each iteration $i$, where $\tau_1$ is a “temperature” parameter. 3) While our main result still holds if one replaces simulated annealing with a deterministic acceptance rule, the annealing step seems to be beneficial in practice in the early period of training when our algorithm is implemented.
with ADAM gradients. 4) Finally, in simulations, we find that Algorithm 4's implementation can be simplified by only taking a small fixed number of max-player updates at each iteration.

3.3 Convergence guarantee

Theorem 3.4 (Main result). Algorithm \( \text{Algorithm 4} \) with hyperparameters \( \eta > 0, \tau_1 > 0 \), given access to stochastic zeroth-order and gradient oracles for a function \( f = \sum_{i \in [m]} f_i \), where each \( f_i \) is \( b \)-bounded with \( L \)-Lipschitz gradient for some \( b, L > 0 \), and \( \varepsilon, \delta, \omega > 0 \), and an oracle for sampling from a distribution \( Q_{x,y} \), with probability at least \( \frac{9}{10} \) returns \( (x^*, y^*) \) such that, for some \( \varepsilon^* \in \left[ \frac{1}{2} \varepsilon, \varepsilon \right] \), \( (x^*, y^*) \) is an approximate local equilibrium (Definition 3.2) w.r.t. \( (\varepsilon^*, \delta, \omega) \) and the distribution \( Q_{x,y} \). The number of stochastic gradient, function, and sampling oracle calls required by the algorithm is \( \text{poly}(b, L, \frac{1}{\varepsilon}, \frac{1}{\delta}, \frac{1}{\omega}) \) and does not depend on the dimension \( d \).

Theorem 3.4 says that our algorithm is guaranteed to converge to an approximate local equilibrium for our framework from any starting point, for any \( f \) which is bounded with Lipschitz gradients including nonconvex-nonconcave \( f \). As discussed in related work this is in contrast to prior works which assume e.g., that \( f(x,y) \) is concave in \( y \) or that the algorithm is provided with an initial point such that the underlying continuous dynamics converge to a local min-max point. We present a proof overview for Theorem 3.4 and the full proof in Appendix 6.

4 Proof overview for Theorem 3.4

For simplicity, assume \( b = L = \tau_1 = 1 \) and \( \varepsilon = \delta = \omega \). There are two key pieces to proving Theorem 3.4. The first is to show that our algorithm converges to some point \( (x^*, y^*) \) in a number of gradient, function, and sampling oracle calls that is \( \text{poly}(1/\varepsilon) \) and independent of the dimension \( d \) (Lemma 4.1). Secondly, we show that \( y^* \) is a first-order \( \varepsilon \)-stationary point for \( f(x^*, \cdot) \), and \( x^* \) is an approximate local minimum of \( \mathcal{L}_{\varepsilon}(\cdot, y^*) \) (Lemma 4.2).

Step 1: Bounding the number of oracle evaluations.

Lemma 4.1 (Informal, see Lemma 6.5). Algorithm 4 terminates after at most \( \text{poly}(b, L, 1/\varepsilon, 1/\delta, 1/\omega) \) gradient, function, and sampling oracle evaluations.

Proof outline of Lemma 4.1. After \( \Theta(\log(\frac{1}{\varepsilon})) \) iterations of Algorithm 4, the decaying acceptance rate (Line 10 of Algorithm 1) ensures that, with probability at least \( 1 - O(\varepsilon) \), at any iteration \( i \) for which Algorithm 1 accepts a proposed update to \( (x_i, y_i) \), we have that

\[
f(x_{i+1}, y_{i+1}) \leq f(x_i, y_i) - \varepsilon. \tag{8}
\]

Next, we note that the stopping condition in Line 2 of Algorithm 1 implies that our algorithm stops whenever \( r_{\max} = \Theta(1/\varepsilon) \) proposed steps are rejected in a row. Thus, Inequality (8) implies that for every \( \Theta(r_{\max}) \) iterations where the algorithm does not terminate, with probability at least \( 1 - O(\varepsilon) \) the value of the loss decreases by at least \( \Omega(\varepsilon) \). Since \( f \) is \( 1 \)-bounded, this implies our algorithm terminates after roughly \( O(r_{\max}/\varepsilon) \) iterations of the minimization routine w.h.p. (Proposition 6.4).

Next, we use the fact that \( G_y(x,y) \) is a batch gradient, \( G_y(x,y) = \frac{1}{|B_y|} \sum_{i \in B_y} \nabla_y f_i(x,y) \), with batch size \( |B_y| = O(\varepsilon^{-2} \log(\frac{1}{\varepsilon})) \), together with the Azuma-Hoeffding concentration inequality, to show that with high probability (Proposition 6.1),

\[
\|G_y(x,y) - \nabla_y f(x,y)\| \leq O(\varepsilon). \tag{9}
\]
We then use Inequality (9) together with the fact that \( f \) is 1-bounded with 1-Lipschitz gradient, to show that, with high probability, the maximization subroutine (Algorithm 2) requires at most \( \text{poly}(\frac{1}{\epsilon}) \) stochastic gradient ascent steps to reach an \( \epsilon \)-stationary point (Proposition 6.3).

Since each step of the maximization subroutine requires one (stochastic) gradient evaluation, and each iteration of the minimization routine calls the maximization routine exactly once (and makes \( O(1) \) oracle calls), the total number of oracle calls is \( \text{poly}(\frac{1}{\epsilon}) \).

\[ \square \]

Step 2: Show \( x^* \) is approximate local minimum for \( L_{\epsilon}(\cdot, y^*) \), and \( y^* \) is \( \epsilon \)-stationary point.

Lemma 4.2 (Informal, see Lemma 6.7). W.h.p., the output \((x^*, y^*)\) of Algorithm 1 is an approximate local equilibrium for our framework, for parameters \((\epsilon, \delta, \omega)\) and proposal distribution \( Q \).

Proof outline of Lemma 4.2. Since we have already shown that Algorithm 2 runs stochastic gradient ascent until it reaches a \( \epsilon \)-stationary point, \( \|\nabla_y f(x^*, y^*)\| \leq \epsilon \).

The accept/reject rule (Line 10 of Algorithm 1) says that the proposed update \( x^* + \Delta \) is rejected with probability at least \( 1 - O(\epsilon) \) whenever

\[ f(x^* + \Delta, y') \geq f(x^*, y^*) - \epsilon, \tag{10} \]

where the maximization subroutine computes \( y' \) by gradient ascent on \( f(x^*, \cdot) \) initialized at \( y^* \). And the stopping condition in Line 2 of Algorithm 1 implies that the last \( r_{\text{max}} \) updates \( x^* + \Delta \) proposed by the min-player were all rejected, and hence were sampled from the distribution \( Q_{x^*, y^*} \). Roughly speaking, this fact together with Inequality (10) implies that, with high probability, the proposal distribution \( Q_{x^*, y^*} \) at the point \((x^*, y^*)\) satisfies

\[ \Pr_{\Delta \sim Q_{x^*, y^*}} \left[ f(x^* + \Delta, y') \geq f(x^*, y^*) - \epsilon \right] \geq 1 - O \left( \frac{1}{r_{\text{max}}} \right) = 1 - O(\epsilon). \tag{11} \]

To show (6) holds, we need to replace \( f \) in the above equation with the min-player’s objective \( L_{\epsilon} \). Towards this end, we first use the fact that \( f \) has \( O(1) \)-Lipschitz gradient, together with (10), to show that, with high probability, the stochastic gradient ascent steps of Algorithm 2 form an \( \epsilon \)-increasing” path, starting at \( y^* \) with endpoint \( y' \), along which \( f \) increases at rate at least \( \epsilon \) (Prop. 6.6). Since \( L_{\epsilon} \) is the supremum of \( f \) at the endpoints of all such \( \epsilon \)-increasing paths starting at \( y^* \),

\[ f(x^* + \Delta, y') \leq L_{\epsilon}(x^* + \Delta, y^*). \tag{12} \]

Finally, recall (Section 3) that \( \|\nabla_y f(x^*, y^*)\| \leq \epsilon^* \) implies that \( L_{\epsilon}(x^*, y^*) = f(x^*, y^*) \), and hence (7) holds. Plugging this and (12) into (11) implies that

\[ \Pr_{\Delta \sim Q_{x^*, y^*}} \left[ L_{\epsilon}(x^* + \Delta, y') \geq L_{\epsilon}(x^*, y^*) - \epsilon \right] \geq 1 - O(\epsilon), \tag{13} \]

and hence that (6) holds. \[ \square \]
5 Empirical results

5.1 Performance on low-dimensional test functions

We apply our algorithm to three test loss functions previously considered in [51]:

\[ F_1(x, y) = -3x^2 - y^2 + 4xy, \]
\[ F_2(x, y) = 3x^2 + y^2 + 4xy, \]
\[ F_3(x, y) = (4x^2 - (y - 3x + 0.05x^3)^2 - 0.1y^4)e^{-0.01(x^2+y^2)}. \]

We choose these functions because they are known to be challenging for first-order algorithms.

Both \( F_1 \) and \( F_3 \) have global min-max at \((0, 0)\), yet popular first-order algorithms including GDA, OMD, and extra-gradient (EG) algorithm were shown in [51] not to converge on these functions. In contrast, we observe that our algorithm finds the global min-max points of both \( F_1 \) and \( F_3 \). To see why our algorithm converges, note that it uses the maximization subroutine (Algorithm 2) to first find the “ridge” along which \( f(x, y) \) is a local maximum in the \( y \) variable, and to then return to a point on the ridge every time the min-player proposes an update. Since the min-player in our algorithm only accepts updates which lead to a net decrease in \( f \), our algorithm eventually finds the point \((0, 0)\) on this ridge where \( f \) is minimized. In comparison, for GDA and OMD, the max-player’s gradient \( \nabla_y f \) is zero along the ridge where \( f(x, y) \) is a local maximum in the \( y \) variable, while the min-player’s gradient \( -\nabla_x f \) can be large; on \( F_1 \) and \( F_3 \) \( -\nabla_x f \) points away from this ridge, and this can prevent GDA and OMD from converging to the point \((0, 0)\).

In case of \( F_2 \), there is no global min-max at any \((x, y) \in \mathbb{R} \times \mathbb{R}\), and, instead, \( \min_{x \in \mathbb{R}} \max_{y \in \mathbb{R}} f(x, y) = +\infty \). On \( F_2 \), GDA, OMD, and EG all converge to \((0, 0)\) which is neither a global min-max nor a local min-max point. In contrast, our algorithm, following the max-player updates, diverges to infinity.

When applying our algorithm on the above test functions we use \( \eta = 0.05 \) and \( Q_{x,y} \sim \mathcal{N}(0, 0.25) \). For GDA and OMD we use learning rate 0.05 (see Appendix C.1).
5.2 Performance when training GANs

We apply our algorithm to train GANs to learn from both synthetic and real-world datasets. We formulate the GAN using our framework with the cross entropy loss,

\[
f(x, y) = - \log(D_y(\zeta)) + \log(1 - D_y(G(x(\xi)))),
\]

where \(x, y\) are the parameters of the generator and discriminator networks \(G\) and \(D\) respectively, \(\zeta\) is sampled from the data, and \(\xi \sim N(0, I_d)\). To adapt Algorithm 1 to training GANs, we make the following simplifications in our simulations:

1) Temperature schedule: we use a fixed temperature \(\tau\), constant with iteration \(i\), making it simpler to choose a good temperature value rather than a temperature schedule;

2) Accept/reject rule: we replace the randomized acceptance rule with a deterministic rule: If \(f_{\text{new}} \leq f_{\text{old}}\) we accept the proposed step, and if \(f_{\text{new}} > f_{\text{old}}\) we only accept if \(i\) is a multiple of \(e^{-1/\tau}\) (i.e., an average acceptance rate of \(e^{-1/\tau}\));

3) Max-player steps: we take a fixed number of max-player steps at each iteration, instead of taking as many steps as needed to achieve a small gradient.

These simplifications do not significantly affect our algorithm’s performance (see Appendix E.1).

Gaussian mixture dataset. This synthetic dataset consists of 512 points sampled from a mixture of four equally weighted Gaussians in two dimensions with standard deviation 0.01 and means at \((0, 1), (1, 0), (-1, 0)\) and \((0, -1)\). Since the modes in this dataset are well-separated, mode collapse, when it occurs, can be clearly detected. We report the number of modes learned by the GAN from each training algorithm across iterations.

Baselines. We compare our algorithm’s performance to GDA, Optimistic Mirror Descent (OMD) [11] (github.com/vsyrgkanis/optimistic_GAN_training/), and unrolled GAN [37] (github...
Table 1: Gaussian mixture dataset: The fraction of times (out of 20 runs) that each method generates $m$ modes, for $m \in \{1, 2, 3, 4\}$. $k$ is the number of max-player steps, per min-player step. Our algorithm learns four modes in more runs than the other algorithms.

| Method           | Number of modes learnt |
|------------------|------------------------|
| This paper       | 0 0.15 0.15 0.70       |
| GDA ($k = 1$)    | 0.95 0.05 0 0          |
| GDA ($k = 6$)    | 0.05 0.75 0 0.20       |
| OMD              | 0.80 0.20 0 0          |
| Unrolled-GAN     | 0.75 0.15 0.10 0       |

Results. We trained GANs on the Gaussian mixture dataset for 1500 iterations using our algorithm, unrolled GANs with 6 unrolling steps, GDA with $k = 1$ and $k = 6$ max-player steps, and OMD with $k = 6$ max-player steps. We repeated each simulation 20 times. The performance of the output GAN learned by all algorithms is presented in Table 1, while Figure 3 also shows the samples from generators of the different training algorithms at various iterations. The GAN returned by our algorithm learns all four modes in 14 (out of 20) repetitions. This is significantly better than the other training algorithms, amongst which only GDA with $k = 6$ max-player steps sometimes returns GANs that learn four modes, but only for 4 (out of 20) repetitions. In particular, OMD and unrolled GANs learn at most 2 modes of the dataset, while GDA with $k = 1$ seems to exhibit cycling behaviour in most cases (see Appendix C.3 for images from all runs). Thus, for this synthetic dataset, our algorithm is the most effective in avoiding mode collapse and cycling in comparison to baselines. We also present preliminary results for the performance of our algorithm on real-world datasets (CIFAR-10 and MNIST), please see Appendix D, E for more details.

6 Proof of Theorem 3.4

In this section we give the proof of Theorem 3.4.

Setting parameters: We start by setting parameters which will be used in the proof. Let $b_0 = |B|, b_y = |B_y|$ denote the batch sizes. And note that the fact that each $f_i$ has $L$-Lipschitz gradient for all $i \in [m]$, implies that each $f_i$ is also $L_1$-Lipschitz, where $L_1 = \sqrt{2Lb}$.

For the theoretical analysis, we assume $0 < \varepsilon \leq 1$, and set the following parameters:

1. $\nu = \begin{aligned} &\frac{1}{10} \left[ \frac{320b(L+1)}{\varepsilon^2} \left( \tau_1 \log(\frac{128}{\omega^2}) + \frac{2048}{\omega^2} \log^2(\frac{100}{\omega} (\tau_1 + 1)(8 \frac{b}{\omega} + 1)) + 1 \right)^2 \right] \\
\end{aligned}$
2. $\tau_{\max} = \frac{128}{\omega} \log^2 \left( \frac{100}{\omega} (\tau_1 + 1)(8 \frac{b}{\omega} + 1) + \log(\frac{1}{\varepsilon}) \right)$
3. Define $\mathcal{I} := \tau_1 \log(\frac{\tau_{\max}}{\varepsilon^2}) + 8 \tau_{\max} \frac{b}{\omega} + 1$
4. \( \eta = \min(\frac{1}{10L}, \frac{1}{8L^2}) \)

5. Define \( \mathcal{J} := \frac{16b}{\eta^2} \)

6. \( \hat{\varepsilon}_1 = \min(\varepsilon, \eta L, \frac{\delta}{8}) \)

7. \( b_0 = \hat{\varepsilon}_1^{-2}140^2b^2 \log(1/\nu) \)

8. \( b_y = \hat{\varepsilon}_1^{-2}140^2L_1^2 \log(1/\nu) \)

In particular, we note that \( \nu \leq \frac{1}{10}(2\mathcal{J}^2 + 2 \times (r_{\max} \frac{8b}{7} + 1))^{-1} \), and \( r_{\max} \geq \frac{4}{\omega} \log(\frac{100L}{\omega}) \). At every iteration \( i \leq I \), where we set \( \varepsilon' = \varepsilon_i \). We also have \( \varepsilon' \leq \varepsilon_0 \left( \frac{1}{1-2\eta L} \right)^{2i} \leq \varepsilon_i \).

6.1 Step 1: Bounding the number of gradient, function, and sampling oracle evaluations

The first step in our proof is to bound the number of gradient, function, and sampling oracle evaluations required by our algorithm. Towards this end, we begin by showing a concentration bound (Proposition 6.1) for the value of the stochastic gradient and function oracles used by our algorithm. Next, we bound the number of iterations of its discriminator update subroutine Algorithm 2 (Proposition 17), and the number of iterations in Algorithm 1 (Proposition 6.4); together, these two bounds imply a poly\((b, L, 1/\varepsilon, 1/\delta, 1/\omega)\) bound on the number of gradient, function, and sampling oracle evaluations (Lemma 6.5).

Proposition 6.1. For any \( \hat{\varepsilon}_1, \nu > 0 \), if we use batch sizes \( b_y = \hat{\varepsilon}_1^{-2}140^2L_1^2 \log(1/\nu) \) and \( b_0 = \hat{\varepsilon}_1^{-2}140^2b^2 \log(1/\nu) \), we have that

\[
P\left( \|G_y(x, y) - \nabla_y f(x, y)\| \geq \frac{\hat{\varepsilon}_1}{10} \right) < \nu, \tag{14}\]

and

\[
P\left( \|F(x, y) - f(x, y)\| \geq \frac{\hat{\varepsilon}_1}{10} \right) < \nu. \tag{15}\]

Proof. From Section 3 we have that

\[
G_y(x, y) - \nabla_y f(x, y) = \frac{1}{b_y} \sum_{i \in B_y} [\nabla_y f_i(x, y) - \nabla_y f(x, y)],
\]

where the batch \( B_y \subseteq [m] \) is sampled iid with replacement from \([m]\). But since each \( f_i \) has \( L \)-Lipschitz gradient, we have (with probability 1) that

\[
\|\nabla_y f_i(x, y) - \nabla_y f(x, y)\| \leq \|\nabla_y f_i(x, y)\| + \|\nabla_y f(x, y)\| \leq 2L_1.
\]

Now,

\[
\mathbb{E}[\nabla_y f_i(x, y) - \nabla_y f(x, y)] = \mathbb{E}[\nabla_y f_i(x, y) - \mathbb{E}[\nabla_y f_i(x, y)]] = 0.
\]
Therefore, by the Azuma-Hoeffding inequality for mean-zero bounded vectors, we have

\[ P \left( \left\| \frac{1}{b_y} \sum_{i \in B_y} \left[ \nabla_y f_i(x, y) - \nabla_y f(x, y) \right] \right\| \geq \frac{s \sqrt{b_y} + 1}{b_y} \frac{2L_1}{b_y} \right) < 2e^{1 - \frac{1}{2}s^2} \quad \forall s > 0. \]

Hence, if we set \( s = 6 \log^{1/2}(\frac{2}{\nu}) \), we have that \( 7 \log^{1/2}(\frac{2}{\nu}) \sqrt{b_y} + 1 \geq s \sqrt{b_y} + 1 \) and hence that

\[ P \left( \left\| \frac{1}{b_y} \sum_{i \in B_y} \left[ \nabla_y f_i(x, y) - \nabla_y f(x, y) \right] \right\| \geq \frac{7 \log^{1/2}(\frac{2}{\nu}) \sqrt{b_y}}{b_y} \frac{2L_1}{b_y} \right) < \nu. \]

Therefore,

\[ P \left( \left\| \frac{1}{b_y} \sum_{i \in B_y} \left[ \nabla_y f_i(x, y) - \nabla_y f(x, y) \right] \right\| \geq \frac{7 \log^{1/2}(\frac{2}{\nu}) \sqrt{b_y}}{b_y} \frac{2L_1}{b_y} \right) < \nu \]

which completes the proof of Inequality (14). Inequality (15) follows from the exact same steps as the proof of Inequality (14), if we replace the bound \( L_1 \) for \( \| \nabla_y f_i(x, y) \| \) with the bound \( b \) on \( |f_i(x, y)| \).

\[
\text{Proposition 6.2. For every } j, \text{ with probability at least } 1 - \nu \text{ we have that either } \|G_y(x, y_j)\| < \varepsilon, \text{ or that } \\
\|\nabla_y f(x, y_j) - G_y(x, y_j)\| \leq \frac{1}{10} \eta L \times \min \left( \|G_y(x, y_j)\|, \|\nabla_y f(x, y_j)\| + \frac{\hat{\varepsilon}_1}{10} \right) \quad (16) \\
\text{and } \\
\|y_{j+1} - y_j\| \leq \eta \|G_y(x, y_j)\| \leq 2\eta \|\nabla_y f(x, y_j)\| \quad (17) 
\]

**Proof.** By proposition 6.1, we have that, with probability at least \( 1 - \nu \),

\[ \|\nabla_y f(x, y_j) - G_y(x, y_j)\| \leq \frac{\hat{\varepsilon}_1}{10} \leq \frac{1}{10} \varepsilon \eta L \leq \frac{1}{10} \eta L \times \min \left( \|G_y(x, y_j)\|, \|\nabla_y f(x, y_j)\| + \frac{\hat{\varepsilon}_1}{10} \right), \]

since \( \hat{\varepsilon}_1 \leq \min(\varepsilon, \eta L) \). This proves Inequality (16). Moreover, we have that

\[ \|y_{j+1} - y_j\| = \|\eta G_y(x, y_j)\| \leq \eta \left( \|\nabla_y f(x, y_j)\| + \frac{\hat{\varepsilon}_1}{10} \right) \]

since \( \hat{\varepsilon}_1 \leq \varepsilon \) and \( \|G_y(x, y_j)\| \geq \varepsilon \). This proves Inequality (17). \[ \square \]

**Proposition 6.3.** Algorithm 2 terminates in at most \( J := \frac{16b}{\eta \varepsilon^2} \) iterations of its “While” loop, with probability at least \( 1 - \nu \times J \).
Proof. Let \( j_{\text{max}} \in \mathbb{N} \cup \{\infty\} \) be the number of iterations of the “While” loop in Algorithm 2. First, we note that the stopping condition for Algorithm 2 implies that

\[
\|G_y(x, y_j)\| \geq \frac{1}{2}\epsilon
\]

for all \( j \leq j_{\text{max}} - 1 \). Since \( f \) has \( L \)-Lipschitz gradient, there exists a vector \( u \), with \( \|u\| \leq L\|y_{j+1} - y_j\| \), such that, for all \( j \leq j_{\text{max}} - 1 \)

\[
f(y_{j+1}) - f(y_j) = \langle y_{j+1} - y_j, \nabla_y f(x, y_j) + u \rangle
\]

\[
= \langle \eta G_y(x, y_j), G_y(x, y_j) \rangle - \langle \eta G_y(x, y_j), G_y(x, y_j) - \nabla_y f(x, y_j) \rangle + \langle \eta G_y(x, y_j), u \rangle
\]

\[
\geq \eta \|G_y(x, y_j)\|^2 - \eta \|G_y(x, y_j)\| \times \|G_y(x, y_j) - \nabla_y f(x, y_j)\| - \eta \|G_y(x, y_j)\| \times \|u\|
\]

by Proposition 6.2.

\[
\geq \frac{1}{8} \eta \|G_y(x, y_j)\|^2
\]

with probability at least \( 1 - \nu \), where the second-to-last inequality holds since \( \eta \leq \frac{1}{10L} \). Since \( f \) takes values in \([-b, b]\), Inequality (19) implies that Algorithm 2 terminates in at most \( J := \frac{16b}{\eta\epsilon^2} \) iterations of its “While” loop, with probability at least \( 1 - \nu \times J \).

Proposition 6.4. Algorithm 2 terminates in at most \( I := \tau_1 \log(\frac{J_{\text{max}}}{\nu}) + 8r_{\text{max}} \frac{b}{\tau} + 1 \) iterations of its “While” loop, with probability at least \( 1 - 2\nu \times (r_{\text{max}} \frac{b}{\tau} + 1) \).

Proof. For any \( i > 0 \), let \( E_i \) be the “bad” event that both \( f(x_{i+1}, y_{i+1}) - f(x_i, y_i) > -\frac{\epsilon}{3} \) and \( \text{Accept}_i = \text{True} \). Then by Proposition 6.1 since \( \frac{b}{10} \leq \frac{b}{8} \), we have that

\[
\mathbb{P}(E_i) \leq e^{-\frac{i}{10}} + \nu
\]

Define \( \hat{I} := \tau_1 \log(\frac{J_{\text{max}}}{\nu}) \). Then for \( i \geq \hat{I} \), from Line 10 of Algorithm 1 we have by Inequality (20) that

\[
\mathbb{P}(E_i) \leq 2\nu.
\]

Define \( h := r_{\text{max}} \frac{b}{\tau} + 1 \). Then

\[
\mathbb{P}\left(\bigcup_{i=\hat{I}}^{\hat{I}+h} E_i\right) \leq 2\nu \times h.
\]

(21)
Since \( f \) takes values in \([-b, b]\), if \( \bigcup_{i=\hat{I}}^{\hat{I}+h} E_i \) does not occur, the number of accepted steps over the iterations \( \hat{I} \leq i \leq \hat{I} + h \) (that is, the size of the set \( \{i : \hat{I} \leq i \leq \hat{I} + h, \text{Accept}_i = \text{True}\} \)) is at most \( \frac{8b}{\delta} \). Therefore, since \( h = r_{\max} \frac{8b}{\delta} + 1 \), we must have that there exists a number \( i \), with \( \hat{I} \leq i \leq i + r_{\max} \leq \hat{I} + h \), such that \( \text{Accept}_i = \text{False} \) for all \( i \in [i, i + r_{\max}] \).

Hence the condition in the While loop (Line 2) of Algorithm 1 implies that Algorithm 1 terminates after at most \( i + r_{\max} \leq \hat{I} + h \) iterations of its While loop, as long as \( \bigcup_{i=\hat{I}}^{\hat{I}+h} E_i \) does not occur. Therefore, Inequality (21) implies that, with probability at least \( 1 - 2\nu \times (r_{\max} \frac{8b}{\delta} + 1) \), Algorithm 1 terminates after at most

\[
\hat{I} + h = \tau_1 \log \left( \frac{r_{\max}}{\nu} \right) + 8r_{\max} \frac{b}{\delta} + 1
\]

iterations of its “While” loop.

\[\square\]

**Lemma 6.5.** Algorithm 1 terminates after at most \((\tau_1 \log(r_{\max} / \nu) + 4r_{\max} \frac{b}{\delta} + 1) \times (J \times b_y + b_0 + b_x)\) gradient, function, and sampling oracle evaluations.

**Proof.** Each iteration of the While loop in Algorithm 1 computes one batch gradient with batch size \( b_x \), one stochastic function evaluation of batch size \( b_0 \), generates one sample from the proposal distribution \( Q \), and calls Algorithm 2 exactly once.

Each iteration of the While loop in Algorithm 2 computes one batch gradient with batch size \( b_y \). The result then follows directly from Propositions 6.4 and 6.3 \(\square\)

### 6.2 Step 2: Proving the output \((x^*, y^*)\) of Algorithm 1 is an approximate local equilibrium

The second step in our proof is to show that the output of Algorithm 1 is an approximate local equilibrium (Definition 3.2) for our framework with respect to \( \varepsilon, \delta, \omega > 0 \) and the distribution \( Q_{x,y} \) (Lemma 6.7). Towards this end, we first show that the steps taken by the discriminator update subroutine (Algorithm 2) form a path along which the loss \( f \) is increasing (Proposition 6.6).

Recall the paths \( \gamma(t) \) from Definition 3.1. From now on we will refer to such paths as “\( \varepsilon \)-increasing paths”. That is, for any \( \varepsilon > 0 \), we say that a path \( \gamma(t) \) is an “\( \varepsilon \)-increasing path” if at every point along this path we have that \( \left\| \frac{d}{dt} \gamma(t) \right\| = 1 \) and that \( \frac{d}{dt} f(x, \gamma(t)) \geq \varepsilon \).

**Proposition 6.6.** Every time Algorithm 2 is called we have that, with probability at least \( 1 - 2\nu J \), the path consisting of the line segments \([y_j, y_{j+1}] \) formed by the points \( y_j \) computed by Algorithm 2 has a parametrization \( \gamma(t) \) which is an \((1 - 2\eta L)\varepsilon'\)-increasing path.

**Proof.** We consider the following continuous unit-speed parametrized path \( \gamma(t) \):

\[
\gamma(t) = y_j + \left( t - \sum_{k=1}^{j-1} \|v_k\| \right) \frac{v_j}{\|v_j\|}, \quad \forall \ t \in \left[ \sum_{k=1}^{j-1} \|v_k\|, \sum_{k=1}^{j} \|v_k\| \right], \quad j \in [j_{\max}],
\]

where \( v_j := \eta G_y(x, y_j) \) and \( j_{\max} \) is the number of iterations of the While loop of Algorithm 2.

Next, we show that \( \frac{d}{dt} f(x, \gamma(t)) \geq \varepsilon' \). For each \( j \in [j_{\max}] \) we have that

\[
\frac{d}{dt} f(x, \gamma(t)) \geq [\nabla_y f(x, y_j) - L\|y_{j+1} - y_j\|u^\top v_j] \frac{v_j}{\|v_j\|} \tag{22}
\]
While loop in Algorithm 1, the path traced by Algorithm 2 is not an $\varepsilon$-oracles. Let $i$ be the "bad" event that, when Algorithm 2 is called during the $i$th iteration of the "While" loop in Algorithm 1 for all $i \in \{i^*, \ldots, i^* + r_{\text{max}}\}$, and that Algorithm 1 stops after exactly $i^* + r_{\text{max}}$ iterations of the "While" loop in Algorithm 1.

Let $H_i$ be the "bad" event that, when Algorithm 2 is called during the $i$th iteration of the "While" loop in Algorithm 1 the path traced by Algorithm 2 is not an $\varepsilon_i$-increasing path. Then, by Proposition 6.6 we have that $j_{\text{max}} \leq J$ with probability at least 1 $\nu \times J$. Therefore inequality 22 implies that

$$
\frac{d}{dt} f(x, \gamma(t)) \geq (1 - 2\eta L)\varepsilon' \quad \forall t \in \left[0, \sum_{k=1}^{J_{\text{max}}} \|v_k\|\right],
$$

with probability at least 1 $\nu$ for some unit vectors $u, w \in \mathbb{R}^d$. But by Proposition 6.3 we have that $j_{\text{max}} \leq J$ with probability at least 1 $\nu \times J$. Therefore inequality 22 implies that

$$
\frac{d}{dt} f(x, \gamma(t)) \geq (1 - 2\eta L)\varepsilon' \quad \forall t \in \left[0, \sum_{k=1}^{J_{\text{max}}} \|v_k\|\right],
$$

with probability at least 1 $\nu J$.

\[\square\]

Lemma 6.7. Let $i^*$ be such that $i^* - 1$ is the last iteration $i$ of the "While" loop in Algorithm 2 for which $\text{Accept}_i = \text{True}$. Then with probability at least 1 $2\nu J 1 - \frac{\omega}{100} - 2\nu \times (r_{\text{max}} \frac{8b}{\varepsilon} + 1)$ we have that

$$
\|\nabla_y f(x^*, y^*)\| \leq (1 - \eta L)\varepsilon_{i^*}.
$$

Moreover, with probability at least 1 $2\nu J 1 - \frac{\omega}{100} - 2\nu \times (r_{\text{max}} \frac{8b}{\varepsilon} + 1)$ we have that

$$
\mathbb{P}_{\Delta \sim Q_{x^*, y^*}} \left( \mathcal{L}_{\varepsilon_{i^*}} (x^* + \Delta, y^*) \leq \mathcal{L}_{\varepsilon_{i^*}} (x^*, y^*) - \frac{1}{2\delta} \|x^* - y^*\| \right) \leq \frac{1}{2^2}. \quad (24)
$$

and that

$$
\frac{\varepsilon}{2} \leq \varepsilon_{i^*} \leq \varepsilon. \quad (25)
$$

Proof. First, we note that $(x^*, y^*) = (x_i, y_i)$ for all $i \in \{i^*, \ldots, i^* + r_{\text{max}}\}$, and that Algorithm 1 stops after exactly $i^* + r_{\text{max}}$ iterations of the "While" loop in Algorithm 1.

Let $H_i$ be the "bad" event that, when Algorithm 2 is called during the $i$th iteration of the "While" loop in Algorithm 1 the path traced by Algorithm 2 is not an $\varepsilon_i$-increasing path. Then, by Proposition 6.6 we have that

$$
\mathbb{P}(H_i) \leq 2\nu J. \quad (26)
$$

Let $K_i$ be the "bad" event that $\|G_y(x_i, y_i) - \nabla_y f(x_i, y_i)\| \geq \frac{\varepsilon_i}{10}$. Then by Propositions 6.1 and 6.3 we have that

$$
\mathbb{P}(K_i) \leq 2\nu J. \quad (27)
$$

Whenever $K_i$ occurs we have that

$$
\|\nabla_y f(x_i, y_i)\| \leq \|G_y(x_i, y_i)\| + \|G_y(x_i, y_i) - \nabla_y f(x_i, y_i)\| \quad (28)
$$

19
\[
\leq (1 - 2\eta L)\epsilon_i + \|G(y(x_i, y_i) - \nabla_y f(x_i, y_i)\|
\leq (1 - 2\eta L)\epsilon_i + \frac{\hat{\epsilon}_1}{10}
\leq (1 - \eta L)\epsilon_i,
\]
where the second Inequality holds by Line 4 of Algorithm 2, and the last inequality holds since \(\frac{\hat{\epsilon}_1}{10} \leq \eta L\). Therefore, Inequalities (27) and (28) together with Proposition 6.4 imply that
\[
\|\nabla_y f(x^*, y^*)\| \leq (1 - \eta L)\epsilon_i^*
\]
with probability at least \(1 - 2\nu \mathcal{J} - 2\nu \times (r_{\max} \frac{8\eta}{\sqrt{2}} + 1)\). This proves Inequality (23).

Inequality (28) also implies that, whenever \(K_i^c\) occurs, the set \(P_{\epsilon_i}(x_i, y_i)\) of endpoints of \(\epsilon_i\)-increasing paths with initial point \(y_i\) (and \(x\)-value \(x_i\)) consists only of the single point \(y_i\). Therefore, we have that
\[
\mathcal{L}_{\epsilon_i}(x_i, y_i) = f(x_i, y_i)
\tag{29}
\]
whenever \(K_i^c\) occurs.

Moreover, whenever \(H_i^c\) occurs we have that \(\mathcal{Y}_{i+1}^c\) is the endpoint of an \(\epsilon_i\)-increasing path with starting point \((x_i + \Delta_i, y_i)\). Now, \(\mathcal{L}_{\epsilon_i}(x_i + \Delta_i, y_i)\) is the supremum of the value of \(f\) at the endpoints of all \(\epsilon_i\)-increasing paths with starting point \((x_i + \Delta_i, y_i)\). Therefore, we must have that
\[
\mathcal{L}_{\epsilon_i}(x_i + \Delta_i, y_i) \geq f(x_i + \Delta_i, \mathcal{Y}_{i+1}^c)
\tag{30}
\]
whenever \(H_i^c\) occurs. Therefore,
\[
\mathbb{P}_{\Delta \sim Q_{x_i, y_i}} \left( \mathcal{L}_{\epsilon_i}(x_i + \Delta_i, y_i) > \mathcal{L}_{\epsilon_i}(x_i, y_i) - \frac{1}{2} \delta_{x_i, y_i} \right)
\tag{31}
\]
and
\[
\geq \mathbb{P}_{\Delta \sim Q_{x_i, y_i}} \left( f(x_i + \Delta, \mathcal{Y}_{i+1}^c) > f(x_i, y_i) - \frac{1}{2} \delta_{x_i, y_i} \right) - \mathbb{P}(H_i) - \mathbb{P}(K_i)
\tag{29}\tag{30}
\]
Proposition 6.1 implies that
\[
\geq \mathbb{P}_{\Delta \sim Q_{x_i, y_i}} \left( F(x_i + \Delta, \mathcal{Y}_{i+1}^c) > F(x_i, y_i) - \frac{1}{4} \delta_{x_i, y_i} \right) - 2\nu - \mathbb{P}(H_i) - \mathbb{P}(K_i)
\]

\[
\geq \mathbb{P} \left( \text{Accept}_i = \text{False} | x_i, y_i \right) - 2\nu - \mathbb{P}(H_i) - \mathbb{P}(K_i)
\tag{29}\tag{30}
\]

\[
\geq \mathbb{P} \left( \text{Accept}_i = \text{False} | x_i, y_i \right) - 2\nu - 2\nu \mathcal{J} - 2\nu \mathcal{J}, \ \forall i \leq \mathcal{I},
\]
where the second inequality holds by Proposition 6.1 since \(\frac{\hat{\epsilon}_1}{10} \leq \frac{\delta}{8}\). Define
\[
p_i := \mathbb{P}_{\Delta \sim Q_{x_i, y_i}} \left( \mathcal{L}_{\epsilon_i}(x_i + \Delta, y_i) > \mathcal{L}_{\epsilon_i}(x_i, y_i) - \frac{1}{2} \delta_{x_i, y_i} \right)
\]
for every \(i \in \mathbb{N}\). Then Inequality (31) implies that
\[
\mathbb{P} \left( \text{Accept}_i = \text{False} | x_i, y_i \right) \leq p_i + \nu(4\mathcal{J} + 2) \leq p_i + \frac{1}{8} \omega \quad \forall i \leq \mathcal{I},
\tag{32}
\]
since $\nu \leq \frac{\omega}{32J^2+16}$. We now consider what happens for indices $i$ for which $p_i \leq 1 - \frac{1}{2}\omega$. Since $(x_{i+s}, y_{i+s}) = (x_i, y_i)$ whenever $\text{Accept}_{i+k} = \text{False}$ for all $0 \leq k \leq s$, we have by Inequality (32) that

$$\mathbb{P} \left( \cap_{s=0}^{\nu_{\text{max}}} \{ \text{Accept}_{i+s} = \text{False} \} \middle| p_i \leq 1 - \frac{1}{2}\omega \right) \leq \left(1 - \frac{1}{4}\omega\right)^{\nu_{\text{max}}} \leq \frac{\omega}{100J} \quad \forall i \leq I - \nu_{\text{max}}$$

since $\nu_{\text{max}} \geq \frac{4}{\omega} \log(\frac{100J}{\omega})$.

Therefore, with probability at least $1 - \frac{\omega}{100J} \times I = 1 - \frac{\omega}{100}$, we have that the event $\cap_{s=0}^{\nu_{\text{max}}} \{ \text{Accept}_{i+s} = \text{False} \}$ does not occur for any $i \leq I - \nu_{\text{max}}$ for which $p_i \leq 1 - \frac{1}{2}\omega$. Recall from Proposition 6.4 that Algorithm 1 terminates in at most $I$ iterations of its “While” loop, with probability at least $1 - 2\nu \times (\nu_{\text{max}} \frac{8b}{\omega} + 1)$.

Therefore,

$$\mathbb{P} \left( p_i > 1 - \frac{1}{2}\omega \right) \geq 1 - \frac{\omega}{100} - 2\nu \times \left( \nu_{\text{max}} \frac{8b}{\omega} + 1 \right).$$

(33)

In other words, by the definition of $p_i$, Inequality (33) implies that with probability at least $1 - \frac{\omega}{100} - 2\nu \times (\nu_{\text{max}} \frac{8b}{\omega} + 1)$, the point $(x^*, y^*)$ is such that

$$\mathbb{P}_{\Delta \sim Q_{x^*, y^*}} \left( \mathcal{L}_{\varepsilon, y^*}(x^* + \Delta, y^*) \leq \mathcal{L}_{\varepsilon, y^*}(x^*, y^*) - \frac{1}{2}\delta \bigg| x^*, y^* \right) \leq \frac{1}{2}\omega.$$ 

This completes the proof of inequality (24). Finally we note that when Algorithm 1 terminates in at most $I$ iterations of its “While” loop, we have

$$\varepsilon_{i^*} = \varepsilon_0 \left( \frac{1}{1 - 2\eta L} \right)^{2i^*} \leq \varepsilon_0 \left( \frac{1}{1 - 2\eta L} \right)^{2I} \leq \varepsilon,$$

(34)

since $\eta \leq \frac{1}{8L^2}$. This completes the proof of Inequality (25). \qed

We can now complete the proof of the main theorem:

**Proof of Theorem 3.4.** First, by Lemma 6.5 our algorithm converges to some point $(x^*, y^*)$ after at most $(\tau J \log(\frac{2\nu_{\text{max}}}{\nu}) + 4\nu_{\text{max}} \frac{b}{\omega} + 1) \times (J \times b_y + b_0 + b_x)$ gradient, function, and sampling oracle evaluations, which is polynomial in $b, L_1, L, \frac{1}{\varepsilon}, \frac{1}{\delta}, \frac{1}{\omega}$, and does not depend on the dimension $d$.

By Lemma 6.7 if we set $\varepsilon^* = \varepsilon_{i^*}$, we have that Inequalities (7) and (8) hold for parameters $\varepsilon^* \in [\frac{1}{8}\varepsilon, \varepsilon], \delta, \omega$ and distribution $Q$, with probability at least $1 - 2\nu J I - 2\nu \times (\nu_{\text{max}} \frac{8b}{\omega} + 1) \geq \frac{9}{10}$, since $\nu \leq \frac{1}{10}(2J I + 2 \times (\nu_{\text{max}} \frac{8b}{\omega} + 1))^{-1}$. \qed

7 Conclusion and future directions

In this paper we introduce a new variant of the min-max optimization framework, and provide a first-order algorithm with efficient convergence guarantees to an equilibrium for this framework. Our guarantees hold for nonconvex-nonconcave loss functions and from any initial point. Empirically, we observe our algorithm converges on many challenging test functions, and when used to train GANs, our algorithm showed improved stability in training.
While we show our algorithm runs in time polynomial in \(b, L\), and independent of dimension \(d\), we do not believe our bounds are tight and it would be interesting to show our algorithm runs in time linear in \(b, L\). Moreover, while our guarantees hold for any distribution \(Q\), we implement our algorithm when training GANs using the distribution of ADAM stochastic gradients rather than, e.g., a standard Gaussian as this leads to better performance in practice. It would be interesting to see if a specialized analysis geared towards adaptively preconditioned distributions can lead improved to bounds. Our framework can be also extended to more general settings; e.g., to multi-agent minimization problems such as those arising in model-agnostic meta-learning [17].

Finally, while the current paper is theoretical in nature, it also comes with application to improving the stability and mode collapse in training of GANs. There are many applications of GANs which promise to have positive impacts, including applications to medicine such as medical imaging [3] and drug discovery [52]. On the other hand, GANs can also be used to generate fake audio and video footage in order to spread disinformation over social media [10], and finding methods of identifying fake footage created by GANs and other generative methods is an important area of research [28, 34].

Acknowledgments

This research was supported in part by NSF CCF-1908347 grant and an AWS ML research award.

References

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

[2] Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[3] Karim Armanious, Chenming Jiang, Marc Fischer, Thomas Küstner, Tobias Hepp, Konstantin Nikolau, Sergios Gatidis, and Bin Yang. Medgan: Medical image translation using gans. *Computerized Medical Imaging and Graphics*, 79:101684, 2020.

[4] Rowel Atienza. GAN by example using keras on tensorflow backend. “https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0”, 2017.

[5] David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 354–363, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[6] Ali Borji. Pros and cons of GAN evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.
[7] Jason Brownlee. How to develop a GAN to generate CIFAR10 small color photographs. “https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/”, 2019.

[8] Sébastien Bubeck. Convex optimization: Algorithms and complexity. Foundations and Trends in Machine Learning, 8, 2017.

[9] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. In International Conference on Learning Representations, ICLR, 2017.

[10] Bobby Chesney and Danielle Citron. Deep fakes: A looming challenge for privacy, democracy, and national security. Calif. L. Rev., 107:1753, 2019.

[11] Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training GANs with optimism. In International Conference on Learning Representations, 2018.

[12] Constantinos Daskalakis and Ioannis Panageas. The limit points of (optimistic) gradient descent in min-max optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 9236–9246. Curran Associates, Inc., 2018.

[13] Constantinos Daskalakis and Ioannis Panageas. Last-iterate convergence: Zero-sum games and constrained min-max optimization. In 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA, pages 27:1–27:18, 2019.

[14] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martín Arjovsky, Olivier Mastropietro, and Aaron C. Courville. Adversarially learned inference. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[15] Tanner Fiez, Benjamin Chasnov, and Lillian Ratliff. Implicit learning dynamics in stackelberg games: Equilibria characterization, convergence analysis, and empirical study. In International Conference on Machine Learning, pages 3133–3144. PMLR, 2020.

[16] Tanner Fiez, Benjamin Chasnov, and Lillian J Ratliff. Convergence of learning dynamics in stackelberg games. arXiv preprint arXiv:1906.01217, 2019.

[17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International Conference on Machine Learning, pages 1126–1135. PMLR, 2017.

[18] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In Conference on Learning Theory, pages 797–842, 2015.

[19] Gauthier Gidel, Reyhane Askari Hemmat, Mohammad Pezeshki, Rémi Le Priol, Gabriel Huang, Simon Lacoste-Julien, and Ioannis Mitliagkas. Negative momentum for improved game dynamics. In Kamalika Chaudhuri and Masashi Sugiyama, editors, Proceedings of Machine
[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[21] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[22] Chi Jin, Praneeth Netrapalli, and Michael I. Jordan. What is local optimality in nonconvex-nonconcave minimax optimization? In *ICML 2020*. International Machine Learning Society (IMLS), 2020.

[23] Renu Khandelwal. Generative adversarial network (GAN) using keras. “https://medium.com/datadriveninvestor/generative-adversarial-network-gan-using-keras-ce1c05cf8fd3”, 2019.

[24] David Kinderlehrer and Guido Stampacchia. *An introduction to variational inequalities and their applications*, volume 31. Siam, 1980.

[25] GM Korpelevich. The extragradient method for finding saddle points and other problems. *Matekon: translations of Russian and East European mathematical economics*, 12:747–756, 1976.

[26] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[27] Jerry Li, Aleksander Madry, John Peebles, and Ludwig Schmidt. On the limitations of first-order approximation in GAN dynamics. In *International Conference on Machine Learning*, pages 3011–3019, 2018.

[28] Yuezun Li and Siwei Lyu. Exposing deepfake videos by detecting face warping artifacts. *arXiv preprint arXiv:1811.00656*, 2018.

[29] Tengyuan Liang and James Stokes. Interaction matters: A note on non-asymptotic local convergence of generative adversarial networks. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 907–915, 2019.

[30] Tianyi Lin, Chi Jin, and Michael I Jordan. On gradient descent ascent for nonconvex-concave minimax problems. In *37th International Conference on Machine Learning, ICML 2020*. International Machine Learning Society (IMLS), 2020.

[31] Songtao Lu, Ioannis Tsaknakis, Mingyi Hong, and Yongxin Chen. Hybrid block successive approximation for one-sided non-convex min-max problems: algorithms and applications. *IEEE Transactions on Signal Processing*, 2020.

[32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.
[33] Oren Mangoubi and Nisheeth K. Vishnoi. Greedy adversarial equilibrium: An efficient alternative to nonconvex-nonconcave min-max optimization. In ACM Symposium on Theory of Computing (STOC), 2021.

[34] Francesco Marra, Diego Gragnaniello, Luisa Verdoliva, and Giovanni Poggi. Do gans leave artificial fingerprints? In 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), pages 506–511. IEEE, 2019.

[35] Panayotis Mertikopoulos, Bruno Lecouat, Houssam Zenati, Chuan-Sheng Foo, Vijay Chandrasekhar, and Georgios Piliouras. Optimistic mirror descent in saddle-point problems: Going the extra (gradient) mile. In International Conference on Learning Representations (ICLR 2019), 2019.

[36] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of GANs. In Advances in Neural Information Processing Systems, pages 1825–1835, 2017.

[37] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[38] Aryan Mokhtari, Asuman Ozdaglar, and Sarath Pattathil. Proximal point approximations achieving a convergence rate of $O(1/k)$ for smooth convex-concave saddle point problems: Optimistic gradient and extra-gradient methods. arXiv preprint arXiv:1906.01115, 2019.

[39] Aryan Mokhtari, Asuman Ozdaglar, and Sarath Pattathil. A unified analysis of extra-gradient and optimistic gradient methods for saddle point problems: Proximal point approach. In International Conference on Artificial Intelligence and Statistics, pages 1497–1507. PMLR, 2020.

[40] Vaishnavh Nagarajan and J Zico Kolter. Gradient descent GAN optimization is locally stable. In Advances in Neural Information Processing Systems, pages 5585–5595, 2017.

[41] Arkadi Nemirovski. Prox-method with rate of convergence $O(1/T)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems. SIAM Journal on Optimization, 15(1):229–251, 2004.

[42] Arkadi S Nemirovski and David Berkovich Yudin. Cesari convergence of the gradient method of approximating saddle points of convex-concave functions. In Doklady Akademii Nauk, volume 239, pages 1056–1059. Russian Academy of Sciences, 1978.

[43] Yurii Nesterov and Boris T Polyak. Cubic regularization of Newton method and its global performance. Mathematical Programming, 108(1):177–205, 2006.

[44] Maher Nouiehed, Maziar Sanjabi, Tianjian Huang, Jason D Lee, and Meisam Razaviyayn. Solving a class of non-convex min-max games using iterative first order methods. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 14905–14916. Curran Associates, Inc., 2019.

[45] Hassan Rafique, Mingrui Liu, Qihang Lin, and Tianbao Yang. Non-convex min-max optimization: Provable algorithms and applications in machine learning. arXiv preprint arXiv:1810.02060, 2018.
[46] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving GANs using optimal transport. In *International Conference on Learning Representations*, 2018.

[47] Shibani Santurkar, Ludwig Schmidt, and Aleksander Madry. A classification-based study of covariate shift in GAN distributions. In *International Conference on Machine Learning*, pages 4480–4489. PMLR, 2018.

[48] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in GANs using implicit variational learning. In *Advances in Neural Information Processing Systems*, pages 3308–3318, 2017.

[49] Kiran Koshy Thekumparampil, Prateek Jain, Praneeth Netrapalli, and Sewoong Oh. Efficient algorithms for smooth minimax optimization. *NeurIPS*, 2019.

[50] Nisheeth K. Vishnoi. *Algorithms for Convex Optimization*. Cambridge University Press, 2021.

[51] Yuanhao Wang, Guodong Zhang, and Jimmy Ba. On solving minimax optimization locally: A follow-the-ridge approach. In *International Conference on Learning Representations*, 2019.

[52] Lingling Zhao, Junjie Wang, Long Pang, Yang Liu, and Jun Zhang. Gansdta: Gans aided drug–target binding affinity prediction. *Frontiers in Genetics*, 10:1243, 2019.

[53] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. In *ICML*, pages 7654–7663, 2019.
A Extension to functions with compact convex support

In this section we introduce a version of our algorithm (Section A.1) for loss functions with compact convex support, and run it on a simple bilinear loss function (Section A.2). We then show that, if \( f \) is also convex-concave, then, for \( \varepsilon = \delta = \omega = 0 \), an approximate local equilibrium for our framework is equivalent to a global min-max point (Section A.3).

A.1 Projected gradient algorithm for compactly supported loss

Our algorithm can be easily extended to the setting of functions \( f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \) where \( \mathcal{X} \subseteq \mathbb{R}^d \) and \( \mathcal{Y} \subseteq \mathbb{R}^d \) are compact convex sets, by replacing the stochastic gradient update \( y - \eta G_y(x, y) \) in Algorithm 1 with a projected gradient update

\[
P_y(y - \eta G_y(x, y)) \quad \text{(Algorithm 3), if one has access to a projection oracle } P_y \text{ for the set } \mathcal{Y}. \quad \text{(In other words, for any point } z \in \mathbb{R}^d, P_y(z) = \text{argmin}_{w \in \mathcal{Y}} \|w - z\| \text{ is the projection of } z \text{ onto the set } \mathcal{Y}.)
\]

When choosing a proposal distribution \( Q_{x,y} \) for the proposed min-player updates in Algorithm 3, one must be careful to make sure that each sample \( \Delta \sim Q_{x,y} \) from this distribution is such that \( x + \Delta \in \mathcal{X} \). For instance, given access to a projection oracle \( P_X \) for the compact convex set \( \mathcal{X} \), one may choose \( Q_{x,y} \) to be the distribution such that the proposed updates \( \Delta \sim Q_{x,y} \) are

\[
\Delta = P_X(x + \xi) - x, \quad \text{where } \xi \sim N(0, I_d).
\]

Alternatively, one may choose the distribution of proposed updates \( \Delta \sim Q_{x,y} \) to be such that

\[
\Delta = P_X(x - \eta G_x(x, y)) - x, \quad \text{where } G_x \text{ is a stochastic gradient with mean } \nabla_x f(x, y) \text{ and } \eta > 0.
\]

A.2 Application to bi-linear losses

As a simple application of Algorithm 3, consider the bilinear loss function \( f(x, y) = xy \) where \( x \) and \( y \) are constrained to the set \([-\frac{1}{2}, \frac{1}{2}]\). It is easy to see that the set of global min-max points consists of the points \((x, y)\) where \( x = 0 \) and \( y = 0 \) is any point in \([-\frac{1}{2}, \frac{1}{2}]\). The approximate local equilibria for our framework (with \( \varepsilon = \delta = 0, \omega = 0 \), and e.g. proposal distribution \( \Delta \sim Q_{x,y} \) with \( \Delta = P_X(x + \xi) - x, \quad \text{where } \xi \sim N(0, 1) \) are the set of points \((x, y)\) where \( x \) is any point in \([-\varepsilon, \varepsilon]\) and \( y \) is any point in \([-\frac{1}{2}, \frac{1}{2}]\).

This is because, when running Algorithm 3 on this example, if \( x \) is outside the set \([-\varepsilon, \varepsilon]\), the max-player will follow an increasing trajectory to always return a point \( y = -\frac{1}{2} \) or \( y = \frac{1}{2} \), which means that, roughly speaking, the min-player is attempting to minimize the function \( \frac{1}{2}|x| \). This means that the algorithm will accept all updates \( x + \Delta \) for which \( \frac{1}{2}|x + \Delta| < \frac{1}{2}|x| - \frac{\varepsilon}{2} \), implying that the algorithm converges towards a point with \( |x| \leq \varepsilon \).

Thus, as \( \varepsilon \) goes to zero, the set of approximate local equilibria for our framework coincides with the set of global min-max optima for the function \( f(x, y) = xy \).

This latter fact holds more generally for any convex-concave function with compact convex domain (see Theorem A.1).

Experiments on compactly supported bi-linear loss. Next, we discuss numerical simulations on the bilinear loss function \( f(x, y) = xy \) with compact support \((x, y) \in [-1, 1] \times [-1, 1]\). For this function, the gradient descent-ascent algorithm is known to diverge away from the global min-max point (see for instance [22]).
Algorithm 3 Our algorithm for compactly supported loss

\textbf{input:} A stochastic zeroth-order oracle $F$ for loss function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ where $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^d$ are compact convex sets. Stochastic gradient oracle $G_y$ with mean $\nabla_y f$, and sampling oracle for a distribution $Q_{x,y}$ where $\Delta \sim Q_{x,y}$ is such that $x + \Delta \in \mathcal{X}$. Projection oracle $P_Y : \mathbb{R}^d \rightarrow \mathcal{Y}$ for $\mathcal{Y}$. An initial point $(x, y) \in \mathcal{X} \times \mathcal{Y}$, and parameters $\varepsilon, \delta, \omega$.

\textbf{output:} A point $(x^*, y^*)$

\textbf{hyperparameters:} $r_{\text{max}}$ (maximum number of rejections), $\eta > 0$.

Set $r \leftarrow 0, i \leftarrow 0$

\textbf{while} $r \leq r_{\text{max}}$ \textbf{do}

\hspace{1em} $f_{\text{old}} \leftarrow F(x, y)$,

\hspace{1em} Sample $\Delta \sim Q_{x,y} \{\text{Sample proposed min-player update from the given distribution } Q\}$

\hspace{1em} Set $x' \leftarrow x + \Delta \{\text{Compute the proposed update for the min-player}\}$

\hspace{1em} Set $j \leftarrow 0$

\hspace{1em} Set $y \leftarrow y$ and Stop = False

\hspace{1em} \textbf{while} Stop = False \textbf{do}

\hspace{2em} Set $j \leftarrow j + 1$

\hspace{2em} Set $g_{y,j} \leftarrow G_y(x, y_j)$

\hspace{2em} if $\frac{1}{\eta} \|y - P_Y(y - \eta g_{y,j})\| > \varepsilon'$ \textbf{then}

\hspace{3em} Set $y \leftarrow P_Y(y - \eta g_{y,j}) \{\text{Compute max-player’s update via gradient ascent}\}$

\hspace{2em} \textbf{else}

\hspace{3em} Set Stop = True

\hspace{2em} Set $f_{\text{new}} \leftarrow F(x', y') \{\text{Compute the new loss value}\}$

\hspace{2em} Set Accept $\leftarrow$ True.

\hspace{2em} if $f_{\text{new}} > f_{\text{old}} - \delta/2$, set Accept $\leftarrow$ False \{accept or reject\}

\hspace{2em} if Accept = True \textbf{then} \hspace{1em} Set $x \leftarrow x'$, $y \leftarrow y'$, $r \leftarrow 0 \{\text{Accept the updates}\}$

\hspace{2em} \textbf{else} \hspace{1em} Set $r \leftarrow r + 1 \{\text{Reject the updates, and track how many successive steps were rejected.}\}$

\hspace{2em} Set $i \leftarrow i + 1$

\hspace{1em} \textbf{return} $(x, y)$
This function has global min-max point at every point in the set \( \{(x, y) : x = 0, y \in [-1, 1]\} \). We ran Algorithm 3 on this function with hyperparameters \( \eta = 0.2, \varepsilon = \delta = 0.06 \), and \( r_{\text{max}} = 5 \), value oracle \( F(x, y) = f(x, y) \), gradient oracle \( G_y(x, y) = \nabla_y f(x, y) \), and initial point \((x, y) = (0.4, 0.4)\).

Our choice of distribution \( Q_{x,y} \) for the proposed updates \( \Delta \sim Q_{x,y} \) is such that \( \Delta = \mathcal{P}_X(\nabla_x f(x, y) + \xi) - x \), where \( \xi \sim N(0, 1) \) and \( \mathcal{P}_X \) is a projection oracle for the set \( X = [-1, 1] \). After 341 iterations of the outer loop, our algorithm reached the point \((0.0279, -0.9944)\), which is very close to one of its true global min-max points, \((0, -1)\).

### A.3 Comparison of local equilibrium point and global min-max point in the compactly supported convex-concave setting

In the following we assume that \( f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \), where \( \mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^d \) are two compact convex sets, and that \( f \) is continuously differentiable on \( \mathcal{X} \times \mathcal{Y} \). We denote by \( \nabla^\mathcal{Y}_f \) the projected gradient in the \( y \) variable for the set \( \mathcal{Y} \).

The following theorem shows that, in the compactly supported convex-concave setting, a point \((x^*, y^*)\) is an approximate local equilibrium for our framework (for \( \varepsilon = \delta = \omega = 0 \)) if and only if it is a global min-max point:

**Theorem A.1.** Let \( f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \) be convex-concave, where \( \mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^d \) are compact convex sets.

And let \( Q_{x,y} \) be a continuous distribution with support on \( \mathcal{X} \) such that, for every \((x, y) \in \mathcal{X} \times \mathcal{Y} \), there is some open ball \( B \subseteq \mathbb{R}^d \) containing \( x \) such that \( Q_{x,y} \) has non-zero probability density at every point in \( B \cap \mathcal{Y} \). Then \((x^*, y^*)\) is an approximate local equilibrium for our framework, with respect to \( \varepsilon = \omega = \delta = 0 \) and the given distribution \( Q_{x,y} \), if and only if it is a global min-max point.

**Proof.** Define the “global max” function \( \psi(x) := \max_{y \in \mathcal{Y}} f(x, y) \) for all \( x \in \mathcal{X} \). We start by showing that the function \( \psi(x) \) is convex on the convex set \( \mathcal{X} \). Indeed, for any \( x_1, x_2 \in \mathcal{X} \) and any \( \lambda \in [0, 1] \) we have

\[
\lambda \psi(\lambda x_1 + (1 - \lambda) x_2) = \max_{y \in \mathcal{Y}} f(\lambda x_1 + (1 - \lambda) x_2, y) \\
\leq \max_{y \in \mathcal{Y}} [\lambda f(x_1, y) + (1 - \lambda) f(x_2, y)] \\
\leq \lambda \max_{y \in \mathcal{Y}} f(x_1, y) + (1 - \lambda) \max_{y \in \mathcal{Y}} f(x_2, y) \\
= \lambda \psi(x_1) + (1 - \lambda) \psi(x_2),
\]

where the second inequality holds by convexity of \( f(\cdot, y) \).

Moreover, we note that, since, for all \( x \in \mathcal{X} \), \( f(x, \cdot) \) is continuously differentiable on a compact convex set, every allowable path (with parameter \( \varepsilon = 0 \)) can be extended to an allowable path whose endpoint \( \hat{y} \) has projected gradient \( \nabla^\mathcal{Y}_f(x^*, \hat{y}) = 0 \).

Therefore, for every \((x, y) \in \mathcal{X} \times \mathcal{Y} \), there exists an allowable path with initial point \( y \) whose endpoint \( \hat{y} \) satisfies

\[
\nabla^\mathcal{Y}_f(x, \hat{y}) = 0.
\]

Since \( f(x, \cdot) \) is concave, (35) implies that

\[
f(x, \hat{y}) = \max_{y \in \mathcal{Y}} f(x, y),
\]

(36)
and hence that
\[ L_0(x, y) = f(x, \hat{y}). \] (37)

Thus, (36) and (37) imply that
\[ L_0(x, y) = \psi(x) \quad \forall (x, y) \in X \times Y \] (38)
since \( \psi(x) = \max_{y \in Y} f(x, y). \)

1. **First we prove the “only if” direction.** Suppose that \((x^\star, y^\star)\) is an approximate local equilibrium of \(f\) for our framework with respect to \(\varepsilon = \delta = \omega = 0\) and the distribution \(Q\). Let \(y^\dagger\) be a global maximizer of the function \(f(x^\star, \cdot)\) (the function achieves its global maximum since it is continuous and \(Y\) is compact). Then the projected gradient at this point is
\[ \nabla_Y y f(x^\star, y^\dagger) = 0. \] (39)

Since \(f(x, \cdot)\) is concave for all \(x\), and \(\nabla_Y y f(x^\star, y^\star) = 0\), at every point \(y\) along the line \([y^\dagger, y^\star]\) connecting the points \(y^\dagger\) and \(y^\star\), (39) implies that
\[ \nabla_Y y f(x^\star, y) = 0, \quad \forall y \in [y^\dagger, y^\star]. \] (40)

Therefore, (40) implies that
\[ f(x^\star, y^\dagger) = f(x^\star, y^\star), \]

and hence that
\[ f(x^\star, y^\star) = \max_{y \in Y} f(x^\star, y), \] (41)
since \(\max_{y \in Y} f(x^\star, y) = f(x^\star, y^\dagger)\). Now, since \((x^\star, y^\star)\) is an approximate local equilibrium with respect to \(\varepsilon = \delta = \omega = 0\) and the distribution \(Q\),
\[ \Pr_{\Delta \sim Q_{x^\star, y^\star}} [L_0(x^\star + \Delta, y^\star) < L_0(x^\star, y^\star)] = 0. \] (42)

Thus, (38) and (42) together imply that
\[ \Pr_{\Delta \sim Q_{x^\star, y^\star}} [\psi(x^\star + \Delta) < \psi(x^\star)] = 0. \] (43)

Since \(\psi\) is convex, and since there is an open ball \(B\) for which \(Q_{x^\star, y^\star}\) has non-zero probability density at every point in \(B \cap Y\), (43) implies that \(x^\star\) is a global minimizer for \(\psi\):
\[ \psi(x^\star) = \min_{x \in X} \psi(x). \] (44)

Therefore, (41) and (44) imply that \((x^\star, y^\star)\) is a global min-max point for \(f : X \times Y \to \mathbb{R}\) whenever \((x^\star, y^\star)\) is an approximate local equilibrium with respect to \(\varepsilon = \delta = \omega = 0\) and the distribution \(Q\).
2. Next, we prove the “if” direction. Conversely, suppose that \((x^*, y^*)\) is a global min-max point for \(f : X \times Y \rightarrow \mathbb{R}\). Then \(f(x^*, y^*) = \max_{y \in Y} f(x^*, y)\). Since \(f\) is differentiable on \(X \times Y\), this implies that
\[
\nabla_y f(x^*, y^*) = 0.
\]
Moreover, since \(f(x^*, y^*)\) is a global min-max point, we also have that
\[
f(x^*, y^*) = \min_{x \in X} \left( \max_{y \in Y} f(x, y) \right) = \min_{x \in X} \psi(x),
\]
and hence that
\[
\psi(x^*) = \min_{x \in X} \psi(x).
\]
Since we have already shown that \(\psi\) is convex, (46) implies that
\[
\Pr_{\Delta \sim Q_{x^*, y^*}} [\psi(x^* + \Delta) < \psi(x^*)] = 0.
\]
Since we have also shown in (38) that \(\psi(x) = \mathcal{L}_0(x, y)\) for all \((x, y) \in X \times Y\), (47) implies that
\[
\Pr_{\Delta \sim Q_{x^*, y^*}} [\mathcal{L}_0(x^* + \Delta, y^*) < \mathcal{L}_0(x^*, y^*)] = 0.
\]
Therefore, (45) and (48) imply that, \((x^*, y^*)\) is an approximate local equilibrium for \(f : X \times Y \rightarrow \mathbb{R}\) (with respect to \(\varepsilon = \delta = \omega = 0\) and the distribution \(Q\)) whenever \((x^*, y^*)\) is a global min-max point for \(f\).

\[
\square
\]

B Comparison of local equilibrium point and local min-max point

**Lemma B.1.** Suppose that \((x^*, y^*)\) is such that \(y^*\) is a local maximum point of \(f(x^*, \cdot)\) and \(x^*\) is a local minimum point of \(f(\cdot, y^*)\). Then \((x^*, y^*)\) is also a local equilibrium of \(f\).

**Proof.** Fix any \(\varepsilon \geq 0\) (the proof of this Lemma requires only \(\varepsilon = 0\), but we state the proof for any \(\varepsilon \geq 0\) since this will allow us to prove Corollary B.2).

Since \(y^*\) is a local maximum of \(f(x^*, \cdot)\), there is only one \(\varepsilon\)-greedy path with initial point \(y^*\), namely, the path \(\{y^*\}\) consisting of the single point \(y^*\) (since \(f\) must increase at rate at least \(\varepsilon\) at every point on an \(\varepsilon\)-greedy path). Thus,
\[
P_\varepsilon(x^*, y^*) = \{y^*\}
\]
Hence, (49) implies that
\[
y^* \in \arg\max_{y \in P_\varepsilon(x^*, y^*)} f(x^*, y)
\]
which proves Equation (5). Next, we will show that Equation (4) holds. Since \(x^*\) is a local minimum point of \(f(\cdot, y^*)\), there exists \(\nu > 0\) such that
\[
f(z, y^*) \geq f(x^*, y^*) \quad \forall z \in B(x^*, \nu)
\]
(51)
Since \( y^* \in P_t(x, y^*) \) for all \( x \in \mathcal{X} \), we have that
\[
\max_{y \in P_t(x, y^*)} f(x, y) \geq f(x, y^*) \quad \forall x \in \mathcal{X},
\]
and hence that
\[
\min_{x \in B(x^*, \nu) \cap \mathcal{X}} \max_{y \in P_t(x, y^*)} f(x, y) \geq \min_{x \in B(x^*, \nu)} f(x, y^*) \geq \max_{y \in P_t(x^*, y^*)} f(x^*, y),
\]
which proves Equation (4).

\[\square\]

**Corollary B.2.** Suppose that \((x^*, y^*)\) is such that \(y^*\) is a local maximum point of \(f(x^*, \cdot)\) and \(x^*\) is a local minimum point of \(f(\cdot, y^*)\). Then there exists \(\nu > 0\) such that, for any \(\varepsilon, \delta \geq 0\), and any proposal distribution \(Q\) with support on \(\mathcal{X}\) which satisfies
\[
Pr_{\Delta \sim Q_{x^*, y^*}} (\|\Delta\| \geq \nu) < \omega,
\]
for some \(\omega > 0\), \((x^*, y^*)\) is also an approximate local equilibrium of \(f\) for parameters \((\varepsilon, \delta, \omega)\) and proposal distribution \(Q\).

We note that many distributions satisfy (53), for instance the distribution \(Q_{x, y} \sim N(0, \sigma^2 I_d)\) for \(\sigma = O(\nu \log^{-1}(\frac{1}{\omega}))\).

**Proof.** By Inequality (54) in the proof of Lemma B.1 there exists \(\nu > 0\) such that
\[
\min_{x \in B(x^*, \nu) \cap \mathcal{X}} \max_{y \in P_t(x, y^*)} f(x, y) \geq \max_{y \in P_t(x^*, y^*)} f(x^*, y),
\]
Thus, for any proposal distribution \(Q\) which satisfies Inequality (54), Inequality (55) implies that, for any \(\delta \geq 0\),
\[
Pr_{\Delta \sim Q_{x^*, y^*}} \left[ \max_{y \in P_t(x^* + \Delta, y^*)} f(x^* + \Delta, y) < \max_{y \in P_t(x^*, y^*)} f(x^*, y) - \delta \right]
\leq Pr_{\Delta \sim Q_{x^*, y^*}} [x^* + \Delta \notin B(x^*, \nu) \cap \mathcal{X}]
= Pr_{\Delta \sim Q_{x^*, y^*}} (\|\Delta\| \geq \nu)
\leq \omega,
\]
This proves Inequality (6). Inequality (7) follows directly from Inequality (50) in the proof of Lemma B.1.

\[\square\]
C  Additional empirical details and results for test functions and Gaussian mixture dataset

C.1  Simulation setup for low-dimensional test functions

In this section we describe the setup for the simulations on the low-dimensional test functions presented in Figures 1 and 2. For our algorithm, we use a learning rate of $\eta = 0.05$ for the max-player, and a proposal distribution of $Q_{x,y} \sim N(0, 0.25)$ for the min-player. For GDA and OMD we use a learning rate of 0.05 for both the min-player and the max-player. When generating Figures 1 and 2 we used the initial point $(x_0, y_0) = (5.5, 5.5)$ for all three algorithms.

C.2  Simulation setup for Gaussian mixture dataset

In this section we discuss the neural network architectures, choice of hyperparameters, and hardware used for the Gaussian mixture dataset.

Hyperparameters for Gaussian mixture simulations. For the simulations on Gaussian mixture data, we have used the code provided by the authors of [37], which uses a batch size 512, Adam learning rates of $10^{-3}$ for the generator and $10^{-4}$ for the discriminator, and Adam parameter $\beta_1 = 0.5$ for both the generator and discriminator.

We use the same neural networks that were used in the code from [37]: The generator uses a fully connected neural network with 2 hidden layers of size 128 and RELU activation, followed by a linear projection to two dimensions. The discriminator uses a fully connected neural network with 2 hidden layers of size 128 and RELU activation, followed by a linear projection to 1 dimension (which is fed as input to the cross entropy loss function). As in the paper [37], we initialize all the neural network weights to be orthogonal with scaling 0.8.

For OMD, we once again use Wasserstein loss and clip parameter 0.01.

Setting hyperparameters. In our simulations, our goal was to be able to use the smallest number of discriminator or unrolled steps while still learning the distribution in a short amount of time, and we therefore decided to compare all algorithms using the same hyperparameter $k$. To choose this single value of $k$, we started by running each algorithm with $k = 1$ and increased the number of discriminator steps until one of the algorithms was able to learn the distribution consistently in the first 1500 iterations.

The experiments were performed on four 3.0 GHz Intel Scalable CPU Processors, provided by AWS.

C.3  Additional simulation results for Gaussian mixture dataset

In this section we show the results of all the runs of the simulation mentioned in Figure 3 where all the algorithms were trained on a 4-Gaussian mixture dataset for 1500 iterations. For each run, we

Note that the authors also mention using slightly different ADAM parameters and neural network architecture in their paper than in their code; we have used the Adam parameters and neural network architecture provided in their code.
plot points from the generated distribution at iteration 1,500. Figure 4 gives the results for GDA with $k = 1$ discriminator step. Figure 5 gives the results for GDA with $k = 6$ discriminator steps. Figure 6 gives the results for the Unrolled GANs algorithm. Figure 7 gives the results for the OMD algorithm. Figure 8 gives the results for our algorithm.

**Unrolled GANs with 6 unrolling steps**

Figure 6: The generated points at the 1500'th iteration for all runs of the Unrolled GAN algorithm for the example in Figure 3 with $k = 6$ unrolling steps.
Figure 7: The generated points at the 1500'th iteration for all runs of OMD algorithm.

Figure 8: The generated points at the 1500'th iteration for all runs of our algorithm, for the simulation mentioned in Figure 3. Our algorithm used $k = 6$ discriminator steps and an acceptance rate hyperparameter of $\tau = \frac{1}{3}$. By the 1500'th iteration, our algorithm seems to have learned all four modes 70% of the runs, three modes 15% of the runs, and two modes 15% of the runs.

Figure 9: Inception score average (and standard deviation in errorbars) of all methods across iterations. Note that mean inception score of our algorithm is higher than the mean inception score of OMD, while the standard deviation of inception score of our algorithm is lower than the standard deviation of inception score of GDA.
Figure 10: GAN trained using our algorithm (with $k = 1$ discriminator steps and acceptance rate $e^{-1/\tau} = 1/2$). We repeated this simulation multiple times; here we display images generated from some of the resulting generators for our algorithm.

Figure 11: GAN trained using GDA (with $k = 1$ discriminator steps). We repeated this simulation multiple times; here we display images generated from some of the resulting generators for GDA.
Figure 12: GAN trained using OMD. We repeated this simulation multiple times; here we display images generated from some of the resulting generators for OMD.

Table 2: CIFAR-10 dataset: The mean (and standard error) of Inception Scores of models from different training algorithms. Note that, GDA and our algorithm return generators with similar mean performance; however, the standard error of the Inception Score in case of GDA is relatively larger.

| Method | Iteration | 5000     | 10000    | 25000    | 50000    |
|--------|-----------|----------|----------|----------|----------|
| Ours   |           | 2.71 (0.28) | 3.57 (0.26) | 4.10 (0.35) | 4.68 (0.39) |
| GDA    |           | 2.80 (0.52) | 3.56 (0.64) | 4.28 (0.77) | 4.51 (0.86) |
| OMD    |           | 1.60 (0.18) | 1.80 (0.37) | 1.73 (0.25) | 1.96 (0.26) |

D Empirical results for CIFAR-10 dataset

This real-world dataset contains 60K color images from 10 classes. Previous works [6, 37, 48] have noted that it is challenging to detect mode collapse on CIFAR-10, visually or using standard metrics such as Inception Scores, because the modes are not well-separated. We use this dataset primarily to compare the scalability, quality, and stability of GANs in our framework obtained using our training algorithm.

For CIFAR-10, in addition to providing images generated by the GANs, we also report the Inception Scores [46] at different iterations. Inception Score is a standard heuristic measure for evaluating the quality of CIFAR-10 images and quantifies whether the generated images correspond to specific objects/classes, as well as, whether the GAN generates diverse images. A higher Inception Score is better, and the lowest possible Inception Score is 1.

Hyperparameters for CIFAR-10 simulations. For the CIFAR-10 simulations, we use a batch size of 128, with Adam learning rate of 0.0002 and hyperparameter $\beta_1 = 0.5$ for both the generator and discriminator gradients. Our code for the CIFAR-10 simulations is based on the code of Jason
Brownlee \cite{brownlee}, which originally used gradient descent ascent and ADAM gradients for training.

For the generator we use a neural network with input of size 100 and 4 hidden layers. The first hidden layer consists of a dense layer with 4,096 parameters, followed by a leaky RELU layer, whose activations are reshaped into 246 $4 \times 4$ feature maps. The feature maps are then upscalled to an output shape of 32 $\times$ 32 via three hidden layers of size 128 each consisting of a convolutional $\text{Conv2DTranspose}$ layer followed by a leaky RELU layer, until the output layer where three filter maps (channels) are created. Each leaky RELU layer has “alpha” parameter 0.2.

For the discriminator, we use a neural network with input of size 32 $\times$ 32 $\times$ 3 followed by 5 hidden layers. The first four hidden layers each consist of a convolutional $\text{Conv2DTranspose}$ layer followed by a leaky RELU layer with “alpha” parameter 0.2. The first layer has size 64, the next two layers each have size 128, and the fourth layer has size 256. The output layer consists of a projection to 1 dimension with dropout regularization of 0.4 and sigmoid activation function.

We ran our algorithm (with $k = 1$ discriminator steps and acceptance rate $e^{-1/\tau} = 1/2$) on CIFAR-10 for 20 repetitions and 50,000 iterations per repetition. We compare with GDA with $k = 1$ discriminator steps and OMD. For all algorithms, we compute the Inception Score every 500 iterations; Table 2 reports the Inception Scores at iteration 5000, 10000, 25000, and 50000, while Figure 9 in the appendix provides the complete plot for Inception Score vs. training iterations. Sample images from all three algorithms are also provided in Figures 11, 12, 10.

The average Inception Score of GANs from both GDA and our algorithm are fairly close to each other, with the final mean Inception Score of 4.68 for our algorithm being somewhat higher than the final mean of 4.51 for GDA. However, the standard error of Inception Scores of GDA is much larger than of our algorithm. The relatively larger standard deviation of GDA is because GDA, in certain runs, does not learn an appropriate distribution at all (Inception Score is close to 1 throughout training in this case), leading to a larger value of standard deviation. Visually, in these GDA runs, the GANs from GDA do not generate recognizable images (Figure 11, top-right image). For all other trials, the images generated by GDA have similar Inception Score (and similar quality) as the images generated by our algorithm. In other words, our algorithm seems to be more stable than GDA and returns GANs that generate high quality images in every repetition.

GANs trained using OMD attain much lower Inception Scores than our algorithm\footnote{We could not replicate the performance of OMD reported in \cite{11}, even with the implementation provided here -}. Moreover, the images generated by GANs trained using OMD have visually much lower quality than the images generated by GANs trained using our algorithm (Figure 12).

Evaluation on CIFAR-10 dataset shows that the GANs from our training algorithm can always generate good quality images; in comparison to OMD, the GANs trained using our algorithm generate higher quality images, while in comparison to GDA, it is relatively more stable.

**Hardware.** Our simulations on the CIFAR-10 dataset were performed on the above, and using one GPU with High frequency Intel Xeon E5-2686 v4 (Broadwell) processors, provided by AWS.

**Clock time per iteration.** When training on CIFAR-10, our algorithm and GDA both took the same amount of time per iteration, 0.08 seconds, on the AWS GPU server.

We evaluate our algorithm on MNIST dataset as well, where it also learns to generate from multiple modes; the results are presented in Appendix E.
E Empirical results for MNIST dataset

This dataset consists of 60k images of hand-written digits [26]. We use two versions of this dataset: the full dataset and the dataset restricted to 0-1 digits.

Hyperparameters for MNIST simulations. For the MNIST simulations, we use a batch size of 128, with Adam learning rate of 0.0002 and hyperparameter $\beta_1 = 0.5$ for both the generator and discriminator gradients. Our code for the MNIST simulations is based on the code of Renu Khandelwal [23] and Rowel Atienza [4], which originally used gradient descent ascent and ADAM gradients for training.

For the generator we use a neural network with input of size 256 and 3 hidden layers, with leaky RELUS each with “alpha” parameter 0.2 and dropout regularization of 0.2 at each layer. The first layer has size 256, the second layer has size 512, and the third layer has size 1024, followed by an output layer with hyperbolic tangent (“tanh”) activation.

For the discriminator we use a neural network with 3 hidden layers, and leaky RELUS each with “alpha” parameter 0.2, and dropout regularization of 0.3 (for the first two layers) and 0.2 (for the last layer). The first layer has size 1024, the second layer has size 512, the third layer has size 256, and the hidden layers are followed by a projection to 1 dimension with sigmoid activation (which is fed as input to the cross entropy loss function).

Results for 0-1 MNIST. We trained GANs using both GDA and our algorithm on the 0-1 MNIST dataset, and ran each algorithm for 3000 iterations (Figure 13). GDA seems to briefly generate shapes that look like a combination of 0’s and 1’s, then switches to generating only 1’s, and then re-learns how to generate 0’s. In contrast, our algorithm seems to learn how to generate both 0’s and 1’s early on and does not mode collapse to either digit. (See Figure 14 for images generated by all the runs of GDA, and [15] for images generated by the GAN for all the runs of our algorithm.)

Full MNIST. Next we evaluate the utility of our algorithm on the full MNIST dataset. We trained a GAN on the full MNIST dataset using our algorithm for 39,000 iterations (with $k = 1$

https://github.com/vsyrgkanis/optimistic_GAN_training
Figure 14: Images generated at the 1000'th iteration of the 13 runs of the GDA simulation mentioned in Figure 13. In 77% of the runs the generator seems to be generating only 1's at the 1000'th iteration.

Figure 15: Images generated at the 1000'th iteration of each of the 22 runs of our algorithm for the simulation mentioned in Figure 13.
Figure 16: We ran our algorithm (with $k = 1$ discriminator steps and acceptance rate $e^{-\frac{1}{\tau}} = \frac{1}{5}$) on the full MNIST dataset for 39,000 iterations, and then plotted images generated from the resulting generator. We repeated this simulation five times; the generated images from each of the five runs are shown here.

discriminator steps and acceptance rate $e^{-1/\tau} = 1/5$). We ran this simulation five times; each time the GAN learned to generate all ten digits (see Fig. 16 for generated images).

E.1 Randomized acceptance rule with decreasing temperature

In this section we give the simulations mentioned in the paragraph towards the beginning of Section 5, which discusses simplifications to our algorithm. We included these simulations to verify that our algorithm also works well when it is implemented using a randomized acceptance rule with a decreasing temperature schedule (Figure 17).
Figure 17: In this simulation we used a randomized accept/reject rule, with a decreasing temperature schedule. The algorithm was run for 39,000 iterations, with a temperature schedule of $e^{-\frac{1}{\tau_i}} = \frac{1}{4 + e^{i/20000}}$. Proposed steps which decreased the computed value of the loss function were accepted with probability 1, and proposed steps which increased the computed value of the loss function were rejected with probability $\max(0, 1 - e^{-\frac{1}{\tau_i}})$ at each iteration $i$. We ran the simulation 5 times, and obtained similar results each time, with the generator learning both modes. In this figure, we plotted the generated images from one of the runs at various iterations, with the iteration number specified at the bottom of each figure (see also Figure 18 for results from the other four runs).

Figure 18: Images generated at the 39,000'th iteration of each of the 5 runs of our algorithm for the simulation mentioned in Figure 17 with a randomized acceptance rule with a temperature schedule of $e^{-\frac{1}{\tau_i}} = \frac{1}{4 + e^{i/20000}}$. 