A Convergent and Dimension-Independent Min-Max Optimization Algorithm

Vijay Keswani 1  Oren Mangoubi 2  Sushant Sachdeva 3  Nisheeth K. Vishnoi 4

Abstract

We study a variant of a recently introduced min-max optimization framework where the max-player is constrained to update its parameters in a greedy manner until it reaches a first-order stationary point. Our equilibrium definition for this framework depends on a proposal distribution which the min-player uses to choose directions in which to update its parameters. We show that, given a smooth and bounded nonconvex-nonconcave objective function, access to any proposal distribution for the min-player’s updates, and stochastic gradient oracle for the max-player, our algorithm converges to the aforementioned approximate local equilibrium in a number of iterations that does not depend on the dimension. The equilibrium point found by our algorithm depends on the proposal distribution, and when applying our algorithm to train GANs we choose the proposal distribution to be a distribution of stochastic gradients. We empirically evaluate our algorithm on challenging nonconvex-nonconcave test-functions and loss functions arising 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:

$$
\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} f(x, y). \tag{1}
$$

This min-max optimization problem has several applications to machine learning, including GANs (Goodfellow et al., 2014) and adversarial training (Madry et al., 2018). In many of these applications, only first-order access to $f$ is available efficiently, and gradient-based algorithms are widely used.

Unfortunately, there is a lack of gradient-based 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 (Arjovsky & Bottou, 2017) and “mode collapse” (Dumoulin et al., 2017; Che et al., 2017; Santurkar et al., 2018).

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., (Nesterov & Polyak, 2006; Ge et al., 2015; Agarwal et al., 2017)), 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, to prove convergence, these works (e.g., Nemirovski (2004); Lu et al. (2020)) make strong assumptions on $f$, e.g. assume $f(x, y)$ is concave in $y$, or that their algorithm is given a starting point such that its underlying dynamical system converges (Heusel et al., 2017; Mescheder et al., 2017). It is a challenge to develop gradient-based 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 study a variant of the min-max framework which allows the max-player to update $y$ in a “greedy” manner (Section 3). This greedy restriction 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, 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 gradient-based algorithm (Algorithm 1) that provably converges from any initial point to an approximate local equilibrium for this framework (Definition 3.2). Our approximate equilibrium definition depends on the choice of proposal distribution \(Q_{x, y}\), parametrized by \((x, y)\), which the min-player uses to update its parameters at any given point \((x, y)\). In particular, for a \(b\)-bounded function \(f\) with \(L\)-Lipschitz gradient, and \(\varepsilon \geq 0\), our algorithm requires poly\((b, L, 1/\varepsilon)\) gradient and function oracle calls to converge to an \((\varepsilon, Q)\)-approximate local equilibrium (Theorem 3.3). The number of oracle calls required by our algorithm is independent of the dimension \(d\). Gradient-based algorithms that converge in (almost) dimension independent number of iterations are required for machine-learning applications where the dimension \(d\), equal to the number of trainable parameters, can be very large.

The equilibrium point our algorithm converges to depends on the choice of proposal distribution \(Q\). In the special case when \(f(x, y)\) is strongly convex in \(x\) and strongly concave in  \(y\), there is a choice of proposal distribution \(Q\)– the (deterministic or stochastic) gradients with mean \(-\nabla_x f\)– for which our \((\varepsilon, Q)\)-equilibrium corresponds to an (approximate) global min-max point with duality gap roughly \(O(\varepsilon)\) (Theorem B.2 in the appendix). Our algorithm can find such a point in time that is polynomial in \(\frac{1}{\varepsilon}\) and independent of dimension (Corollary B.10). This motivates using (stochastic) gradients for proposal distributions in more general settings as well, and, when training GANs we choose the proposal distribution to be the distribution of stochastic gradients.

Empirically, we show that our algorithm converges on test functions (Wang et al., 2019) on which other popular gradient-based min-max optimization algorithms such as GDA and optimistic mirror descent (OMD) (Daskalakis et al., 2018) are known to either diverge or cycle (Figure 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), 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 Supplementary Material.

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–i.e., paths along which \(f(x^*, \cdot)\) is increasing–initialized at the value \(y^*\). Additionally, \(y^*\) is an approximate local maximum of \(f(x^*, \cdot)\). In particular, any point \((x^*, y^*)\) which is a local min-max point is also an equilibrium point for our algorithm (see Section 3).

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., Nesterov & Polyak, 2006), 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, e.g. cross entropy (Goodfellow et al., 2014) 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., Nemirovski, 2004; Lu et al., 2020; Heusel et al., 2017; Mescheder et al., 2017), previous works propose local frameworks where the max-player is able to choose its move after the min-player. These include the local stackelberg equilibrium (Fiez et al., 2020) and the closely related
local min-max point (Jin et al. 2020). In the local min-max, 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 follows a path along which \( f \) is continuously increasing.

Convergence Guarantees. Several works have studied the convergence properties of GDA dynamics (Nagarajan & Kolter 2017; Mescheder et al. 2017; Balduuzzi et al. 2018; Daskalakis & Panageas 2018; Jin et al. 2020; Li et al. 2018), 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 analyze algorithms based on Optimistic Mirror Descent (OMD), Extra-gradient (EG) methods, or similar approaches (Gidel et al. 2019; Daskalakis & Panageas 2018; Liang & Stokes 2019; Daskalakis & Panageas 2019; Mokhtari et al. 2020b). For instance, (Daskalakis et al. 2018) guarantee convergence of OMD to a global min-max point on bilinear losses \( f(x, y) = x^\top A y \), and (Mokhtari et al. 2020a) also show convergence of OMD and EG methods on strongly convex-strongly concave \( f \). However, as observed in (Wang et al. 2019), 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).

Many works make additional assumptions to prove convergence. (Mertikopoulos et al. 2019) show asymptotic convergence of OMD under a “coherence” assumption, and (Balduuzzi et al. 2018) show convergence of a second-order algorithm if \( f \) corresponds to a Hamiltonian game. Some works assume there exists a “variational inequality” solution \((x^*, y^*)\) such that, roughly, the component of the gradient field \((-\nabla_x f, \nabla_y f)\) in the direction away from \((x^*, y^*)\) is very small (Dang & Lan 2015; Liu et al. 2019; Song et al. 2020; Diakonikolas et al. 2021; Liu et al. 2021). Other works (Yang et al. 2020) assume a “PL” condition which says that magnitude of \( \nabla_x f(x, y) \) is at least \( f(x, y) - \min_x f(x, y) \). However, many simple functions, e.g. \( f(x, y) = \sin(x) \sin(y) \), do not satisfy Hamiltonian game, coherence, variational, and PL assumptions.

For this reason, multiple works show convergence to an (approximate) local min-max point. For instance (Heusel et al. 2017) 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 (Mescheder et al. 2017) show convergence if their algorithm is initialized in a small neighborhood of a local min-max point. In addition, many works provide convergence guarantees to a local stackleberg or local minimax point, if their algorithm is provided with a starting point in the region of attraction (Fiez et al. 2019; 2020), or a small enough neighborhood (Wang et al. 2019), of such an equilibrium. And other works (Nemirovski & Yudin 1978; Kinderlehrer & Stampacchia 1982; Nemirovski 2004; Raiche et al. 2021; Lu et al. 2020; Lin et al. 2020; Nouiehed et al. 2019; Thekumparambil et al. 2019; Kong & Monteiro 2021) show convergence 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 \( f \), 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 (Bubeck 2017; Ge et al. 2015; Vishnoi 2021). Similar to our approach, some algorithms make multiple player updates each iteration (e.g., (Nouiehed et al. 2019)). However, these algorithms are not guaranteed to converge from any initial point on any nonconvex-nonconcave smooth bounded \( f \); to overcome this, our algorithm introduces a randomized accept-reject procedure.

Greedy Paths. (Mangoubi & Vishnoi 2021) also consider a framework where the max-player makes updates in a greedy manner. The “greedy paths” considered in their work are defined such that at every point along these paths, \( f \) is nondecreasing, and the first derivative of \( f \) is at least \( \varepsilon \) or the second 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, (Mangoubi & Vishnoi 2021) 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 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 non-convergence behavior such as cycling when training GANs using GDA (Goodfellow et al. 2014). (Daskalakis et al. 2018) show OMD can mitigate cycling when training GANs with Wasserstein loss. In contrast to both GDA and OMD, where at each iteration the min- and max-players 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. (Metz et al. 2017) introduced Unrolled GANs, where the min-player optimizes an “unrolled” loss 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 an (approximate) local minimum, 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

As a first step towards obtaining a computationally tractable variant of the min-max framework, we consider the local min-max point studied in prior work—that is, any point \((x, y)\) such that \(x\) is local minimum of \(f(\cdot, y)\) and \(y\) is a local maximum of \(f(x, \cdot)\). Unfortunately, local min-max points may not exist even on smooth and bounded functions. For instance, the function \(f(x, y) = \sin(x + y)\) has no local min-max points. This is because, while \(f(x, y) = \sin(x + y)\) has local maximum in \(y\) at all points along the collection of lines \(S = \{(x, y) : x + y = \frac{2k\pi}{3}, k \in \mathbb{N}\}\), \(\sin(x + y)\) does not have a local minimum in \(x\) at any of these points. However, the points \(S\) are all global min-max points of \(f(x, y) = \sin(x + y)\), since for every \((x, y) \in S\), \(x\) is a global minimum of \(\max_{y \in \mathbb{R}} f(\cdot, y)\), and \(y\) is a global maximum of \(f(x, \cdot)\). This is true even though \(x\) is neither a global nor a local minimum of \(f(\cdot, y)\).

On the other hand, an (approximate) global min-max point is always guaranteed to exist for smooth and bounded \(f\). This is because, in the global min-max framework, before the min-player considers whether to choose a value \(x\), it is able to “look ahead” and anticipate the response \(\arg \max_{y \in \mathbb{R}^d} f(x, y)\) of the max-player. Thus, for any smooth and bounded function \(f\), one can always find an (approximate) global min-max point by first finding an (approximate) global minimum \(x\) of the function \(\max_{y \in \mathbb{R}^d} f(\cdot, y)\), and then finding a value of \(y\) which maximizes \(f(x, \cdot)\). In order to guarantee existence for our framework, we would therefore ideally like to allow the min-player to anticipate the max-player’s response \(\max_{y \in \mathbb{R}^d} f(\cdot, y)\) to any value of \(x\) proposed by the min-player. Unfortunately, computing the global maximum \(\max_{y \in \mathbb{R}^d} f(\cdot, y)\) is intractable.

**Framework and Equilibrium.** To get around this problem, 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 \(Y\) 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 \(f : X \times Y \rightarrow \mathbb{R}\), where \(X, Y \subseteq \mathbb{R}^d\) are convex, an equilibrium for our framework is a point \((x^*, y^*) \in X \times Y\) such that

\[
\begin{align*}
  x^* &\in \arg \min_{x \in X} \max_{y \in P(x, y^*)} f(x, y), \quad (2) \\
  y^* &\in \arg \max_{y \in P(x^*, y)} f(x^*, y). \quad (3)
\end{align*}
\]

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

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

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

**Approximate local equilibrium.** Similar to previous work on local minimization for smooth nonconvex objectives, we would like to solve (4) and (5) to converge to approximate stationary points\(^\dagger\). Toward this end, we can replace \(P(x, y^*)\) in (4) and (5) with the set \(P_\varepsilon(x, y^*)\) of endpoints of paths starting at \(y^*\) along which \(f(x, \cdot)\) increases at some “rate” \(\varepsilon > 0\).

**Definition 3.1.** For any \(x \in X\), \(y \in Y\), and \(\varepsilon \geq 0\), define \(P_\varepsilon(x, y) \subseteq Y\) to be points \(w \in Y\) 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., \(\|\frac{d}{dt} \gamma(t)\| \leq 1\), s.t. at any point on \(\gamma\), \(\frac{d}{dt} f(x, \gamma(t)) \geq \varepsilon\).

The above definition restricts the max-player to updating \(y\) via any “greedy” algorithm, e.g., gradient ascent. Note that, compared to Definition 3.1, the notion of greedy paths in\(^\dagger\) additionally requires \(\frac{d}{dt} f(x, \gamma(t)) \geq \sqrt{\varepsilon}\) so as to achieve the goal of converging to an approximate second-order local equilibrium. Our goal, on the other hand, is for the max-player’s updates to approximate paths taken by first-order greedy algorithms, hence, the condition on first derivative in Definition 3.1 suffices.

While we would also like to replace the local minimum in\(^\dagger\) with an approximate stationary point, the min-player’s objective, \(L_\varepsilon(x, y) := \max_{z \in P_\varepsilon(x, y)} f(x, z)\), may not be continuous\(^\dagger\) 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 \(L_\varepsilon(\cdot, \cdot)\) has decreased. Formally, given a choice of sampling distribution \(Q_\varepsilon\) (which may depend on \(x\)), and \(\delta, \omega > 0\), \(x^*\) is said to be an approximate local minimum of a (possibly discontinuous) function \(\tilde{g} : X \rightarrow \mathbb{R}\) if \(\Pr_{\tilde{g} \sim Q_\varepsilon} [\tilde{g}(x^* + \Delta) < \tilde{g}(x^*) - \delta] < \omega\).

Thus, replacing the set \(P\) with \(P_\varepsilon\) in Equations (4) and (5), and the “exact” local minimum in Equation (4) with an approximate local minimum, we arrive at our equilibrium definition:

\(^\dagger\)Consider the example \(f(x, y) = \min(x^2 y^2, 1)\). The min-player’s objective for \(\varepsilon > 0\) is \(L_\varepsilon(x, y) = f(x, y)\) if \(2x^2 y < \varepsilon\), and 1 otherwise. Thus \(L_{1/2}\) is discontinuous at \((1/2, 1)\).
We present an algorithm for our framework w.h.p., we will have that with $O$ the (deterministic or stochastic) gradients $-\nabla_x f$, then the $(\varepsilon, \delta, \omega, Q)$-equilibrium corresponds to an “approximate” global min-max point for $f$ with duality gap $O(\varepsilon + \delta)$ (Theorem B.2). This duality gap does not depend on the dimension. This motivates choosing the proposal distribution to be the (stochastic) gradients $-\nabla_x f$ in the more general setting when $f$ is nonconvex-nonconcave. Another motivation for this choice of $Q$ is that adding stochastic gradient noise to steps taken by deep learning algorithms is known empirically to lead to better outcomes than, e.g., standard Gaussian noise (see Zhu et al., 2019). Empirically, this choice of $Q$ leads to GANs that are able to successfully learn the dataset’s distribution (Section 4).

**Algorithm**. We present an algorithm for our framework (Algorithm 1), along with the gradient ascent subroutine it uses to compute max-player updates (Algorithm 2). In Theorem 3.3, we show it efficiently finds an approximate local equilibrium (Definition 3.2).

**Algorithm 1** Our algorithm for min-max optimization

**input**: Stochastic zeroth-order oracle $F$ for bounded loss function $f: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ with $L$-Lipschitz gradient, stochastic gradient oracle $G_y$ with mean $\nabla_y f$, initial point $(x_0, y_0)$

**output**: A distribution $Q_{x,y}$ and an oracle for sampling from this distribution. 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 $r \leq r_{\text{max}}$
3: Sample $\Delta_i$ from the distribution $Q_{x_i, y_i}$
4: Set $X_{i+1} \leftarrow x_i + \Delta_i \{\text{min-player's proposed update}\}$
5: Run Algorithm 2 with inputs $x_i \leftarrow X_{i+1}, y_0 \leftarrow y_i$, and $e' \leftarrow \varepsilon_i \times (1 - 2\eta L)^{-1} \{\text{max-player's update}\}$
6: Set $y_{i+1} \leftarrow y_{\text{stationary}}$ to be the output of Algorithm 2
7: Set $f_{\text{new}} \leftarrow F(X_{i+1}, y_{i+1}) \{\text{Compute new loss}\}$
8: Set $\text{Accept}_i \leftarrow \text{True}$
9: if $f_{\text{new}} > f_{\text{old}}$ - $\frac{\delta}{4}$ then
10: Set $\text{Accept}_i \leftarrow \text{False}$ with probability $\max(0, 1 - e^{-\eta / \tau_1}) \{\text{Decide to accept or reject}\}$
11: end if
12: if $\text{Accept}_i = \text{True}$
13: Set $x_{i+1} \leftarrow X_{i+1}, y_{i+1} \leftarrow y_{i+1} \{\text{accept the proposed x and y updates}\}$
14: Set $f_{\text{old}} \leftarrow f_{\text{new}}, r \leftarrow 0, \varepsilon_{i+1} \leftarrow \varepsilon_i \times (1 - 2\eta L)^{-2}$
15: else
16: Set $x_{i+1} \leftarrow x_i, y_{i+1} \leftarrow y_i, r \leftarrow r + 1, \varepsilon_{i+1} \leftarrow \varepsilon_i \{\text{Reject the proposed updates}\}$
17: end if
18: Set $i \leftarrow i + 1$
19: end while
20: return $(x^*, y^*) \leftarrow (x_i, y_i)$

We consider bounded loss functions $f : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, 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 $F$ where $\mathbb{E}[F] = f$. We call such an oracle a stochastic zeroth-order oracle for $f$. We are also given randomized oracles $G_x, G_y$ for $\nabla_x f, \nabla_y f$, where $\mathbb{E}[G_x] = \nabla_x f$, 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” $B_i, B_x, B_y \subseteq [m]$ (i.i.d., with replacement) and returning $F = \frac{1}{|B|} \sum_{i \in B} f_i, G_x = \frac{1}{|B_x|} \sum_{i \in B_x} \nabla_x f_i$, 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 each \( f_i : b, L > 0 \) such that \( |f_i(x, y)| \leq b \) and \( \|\nabla f_i(x, y) - \nabla f_i(x', y')\| \leq L\|x-x'| + L\|y-y'\| \) for all \( x, y \) and all \( i \). These bounds imply \( f \) is also \( b \)-bounded, and \( L \)-gradient-Lipschitz.

**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 \( \epsilon' \)

**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 \( \|g_{y,j}\| > \epsilon' \) then
5. if \( y_j+1 \leftarrow y_j + \eta g_{y,j} \)
6. Set \( j \leftarrow j + 1 \)
else
7. Set Stop \( \leftarrow \) True
8. end if
9. end while
10. return \( y_{\text{stationary}} \leftarrow y_j \)

**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 gradients \( -G_x \), 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 \( \epsilon \)-stationary point \( y' \), that is, a point where \( \|\nabla_y f(x + \Delta, y')\| \leq \epsilon \). 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 \( \mathcal{L}_c(x, y') \) in order to satisfy (6). Moreover, it must also ensure that \( y' \) satisfies (7).

Towards this, Algorithm 1 does the following:

1) The algorithm re-uses the same point \( y' \) to compute an approximation \( f(x + \Delta, y') \) for \( \mathcal{L}_c(x + \Delta, y) \) in order to have access to the value of the min-player’s objective \( \mathcal{L}_c \) to be able to minimize it.

2) If \( f(x + \Delta, y') \) is less than \( f(x, y) \) the algorithm concludes that \( \mathcal{L}_c(x + \Delta, y') \) has decreased and, consequently, accepts the updates \( x + \Delta \) and \( y' \); otherwise it rejects both updates. We show that after accepting both \( x + \Delta \) and \( y' \), \( \mathcal{L}_c(x + \Delta, y') < \mathcal{L}_c(x, y) \), implying that the algorithm does not cycle.

3) It then starts the next iteration proposing a 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 \( \mathcal{L}_c \) after roughly \( \frac{1}{\omega} \) attempts, it concludes w.h.p. that the current \( x \) is an approximate local minimum for \( \mathcal{L}_c(x, y) \) 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 \( \mathcal{L}_c \). We also show that the current \( y \) is an \( \epsilon \)-stationary point for \( f(x, \cdot) \).

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 (alternately, one may also be able to substitute other optimization algorithms such as Momentum SGD (Polyak 1964), ADAGrad (Duchi et al., 2011), or Adabelief (Zhuang et al., 2020) for gradient updates). 2) Algorithm 1 uses a randomized accept-reject rule (similar to 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 \( e^{-/\tau_i} \) at each iteration \( i \), where \( \tau_i \) 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 1’s implementation can be simplified by taking a small fixed number of max-player updates at each iteration.

**Convergence Guarantee.**

**Theorem 3.3 (Main result).** Algorithm 1 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 \( \epsilon, \delta, \omega > 0 \), and an oracle for sampling from a distribution \( Q_{x,y} \), with probability at least \( 9/10 \) returns \((x^*, y^*) \in \mathbb{R}^d \times \mathbb{R}^d \) such that, for some \( \epsilon^* \in \left[ \frac{1}{2\delta}, \epsilon \right] \), \((x^*, y^*)\) is an \((\epsilon^*, \delta, \omega, Q)\)-approximate local equilibrium. The number of stochastic gradient, function, and sampling oracle calls required by the algorithm is poly(b, L, \( 1/\epsilon, 1/\delta, 1/\omega \)) and does not depend on the dimension \( d \).

Theorem 3.3 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-nonnconcave \( f \). 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. The exact number of stochastic gradient, function, and sampling oracle calls required by the algorithm is \( O(b^3L^3/(\delta^3\omega^3\epsilon^4)) \). We present a proof overview for Theorem 3.3 next, and the full proof in Appendix A.

In the setting where \( f(x, y) \) is \( \alpha \)-strongly convex in \( x \) and \( \alpha \)-strongly concave in \( y \) and has \( L \)-Lipschitz gradients, Algorithm 3.3 with \( x \)-player updates \( Q_{x,y} \) chosen to be the
Algorithm 1 ensures that, with probability at least $1 - \varepsilon$, the decaying acceptance rate (Line 1 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, this fact together with (10) implies that, with high probability, the proposal distribution $Q_{x^*, y^*}$ satisfies
\[
\Pr_{\Delta \sim Q_{x^*, y^*}} \left[ f(x^* + \Delta, y^*) \geq f(x^*, y^*) - \varepsilon \right] \geq 1 - O(r_{\text{max}}^{-1}) = 1 - O(\varepsilon). \tag{11}
\]
To show (9) holds, we need to replace $f$ in the above equation with the min-player’s objective $\mathcal{L}_e$. Towards this end, we first use the fact that $f$ has $O(1)$-Lipschitz gradient, together with (9), to show that, w.h.p., the stochastic gradient ascent steps of Algorithm 2 form an $\varepsilon$-increasing path, starting at $y^*$ with endpoint $y^*$, along which $f$ increases at rate at least $\varepsilon$ (Proposition A.6). Since $L_e$ is the supremum of $f$ at the endpoints of all such $\varepsilon$-increasing paths starting at $y^*$,
\[
f(x^* + \Delta, y^*) \leq f(x^*, y^*) - \varepsilon \tag{12}
\]
Finally, recall (Section 3) that $\|\nabla_y f(x^*, y^*)\| \leq \varepsilon$ implies that $L_e(x^*, y^*) = f(x^*, y^*)$, and hence (7) holds. Plugging this and (12) into (11) implies that
\[
\Pr_{\Delta \sim Q_{x^*, y^*}} \left[ L_e(x^* + \Delta, y^*) \geq L_e(x^*, y^*) - \varepsilon \right] \geq 1 - O(\varepsilon),
\]
and hence that (9) holds.

4. Empirical Results

Performance on Test Functions. We apply our algorithm to three test loss functions previously considered in [Wang et al., 2019] (Figure 2):
\[
F_1(x, y) = -3x^2 - y^2 + 4xy, F_2(x, y) = 3x^2 + y^2 + 4xy,
\]
\[
F_3(x, y) = 4x^2 - (y - 3x + 0.05x^3)^2 - 0.1y^2 + e^{-0.01(x^2 + y^2)}.
\]
We choose these functions because they are known to be challenging for gradient-based algorithms.
Both $F_1$ and $F_3$ have global min-max at $(0, 0)$, yet popular gradient-based algorithms including GDA, OMD, and extra-gradient (EG) algorithm were shown in (Wang et al., 2019) to 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 $F_2$, $\min_{x \in \mathbb{R}} \max_{y \in \mathbb{R}} f(x, y) = +\infty$. 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 diverges to infinity.

When applying our algorithm we use $\eta=0.05$ and $Q_{x,y} \sim N(0,0.25)$. For GDA and OMD we use learning rate 0.05 (see Appendix E.1).

**Performance when Training GANs.** We apply our algorithm to train GANs to learn from both synthetic and real-world datasets. When training on both datasets, we choose the proposal distribution $Q$ in our algorithm to be the (ADAM) stochastic gradients for $-\nabla_x f$. We formulate GAN using our framework with cross-entropy loss,

$$f(x, y) = - (\log(D_y(\zeta)) + \log(1 - D_y(G_x(\zeta)))),$$

where $x, y$ are the parameters of generator $G$ and discriminator $D$ respectively, $\zeta$ is sampled from data, and $\zeta \sim N(0, I_d)$.

To adapt Algorithm 1 to training GANs, we make certain simplifications: 1) we use a fixed temperature $\tau$ at all iterations $i$, making it simpler to choose a good temperature value, rather than a temperature schedule; 2) we replace the randomized acceptance rule with a deterministic rule: If $f_{\text{new}} \leq f_{\text{old}}$ we accept, and if $f_{\text{new}} > f_{\text{old}}$ we only accept if $i$ is a multiple of $e^{1/k}$ (i.e., average acceptance rate of $e^{-1/k}$); 3) 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 F).

**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 $(0, 1), (1, 0), (-1, 0), (0, -1)$. Since modes in this dataset are well-separated, mode collapse can be clearly detected. We report the number of modes learnt by the GAN from each training algorithm across iterations.

**Baselines.** We compare our algorithm’s performance to...
A Convergent and Dimension-Independent Min-Max Optimization Algorithm

GDA, OMD (Daskalakis et al., 2018), and unrolled GAN (Metz et al., 2017). For the networks and hyperparameter details, see Appendix E.3.

Results on Gaussian mixture dataset. 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 (using Adam updates), 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 shows the samples from generators of the different training algorithms at various iterations (see Appendix E.4 for images from all runs). The GAN returned by our algorithm learns all four modes in 70% of the runs, significantly more than the other training algorithms. Thus, for this synthetic dataset, our algorithm is the most effective in avoiding mode collapse and cycling in comparison to baselines.

Results on Real-World Datasets. While we focus on 2-D and Gaussian mixture GAN simulations in this section to illustrate the convergence properties of our algorithm, we also ran our algorithm on two real-world datasets, 01-MNIST and CIFAR-10. For the 01-MNIST dataset, samples generated from GANs trained using GDA and our algorithm are presented in Figure 4. We observed that GANs trained on the 01-MNIST dataset with the gradient descent-ascent algorithm (GDA) exhibit mode collapse in 77% of the trial runs (Figure 18 in Appendix C), while GANs trained with our algorithm do not exhibit mode collapse in any of the training runs (Figure 19 in Appendix C). For the CIFAR-10 dataset, samples generated from GANs trained using our algorithm are presented in Figure 5. On CIFAR-10, our algorithm achieved a mean Inception score of 4.68 after 50k iterations (across 20 repetitions); in comparison, GDA achieved a mean Inception score of 4.51 and OMD achieved a mean Inception score of 1.96 (Table 2). Detailed results and methodologies used for 01-MNIST and CIFAR-10 datasets are presented in Appendix G and I respectively.

Experiments with GANs also demonstrate that our algorithm scales to high-dimensional parameter spaces; the dimension $d$ of the space of trainable parameters used in the GAN experiments was around $3.5 \times 10^4$ for the GANs trained on the Gaussian mixture dataset, $3 \times 10^6$ for 01-MNIST and $2 \times 10^6$ for CIFAR-10.

5. Conclusion and Future Directions

We introduce a new variant of the min-max optimization framework, and provide a gradient-based algorithm with efficient convergence guarantees to an equilibrium for this framework, for nonconvex-nonconcave losses and from any initial point. Empirically, we observe our algorithm converges on many challenging test functions and shows improved stability when training GANs.

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 the run-time is linear in $b, L$. Moreover, while our guarantees hold for any distribution $Q$, it would be interesting to see if a specialized analysis for adaptively preconditioned distributions leads to improved bounds. Our framework can also be extended to general settings like multi-agent minimization problems arising in meta-learning (Finn et al., 2017).

Acknowledgments

This research was supported in part by NSF CCF-1908347, NSF CCF-2112665, and NSF CCF-2104528 grants, and an AWS ML research award.

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Iteration 5000</th>
<th>Iteration 25000</th>
<th>Iteration 50000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>2.71 (0.28)</td>
<td>4.10 (0.35)</td>
<td>4.68 (0.39)</td>
</tr>
<tr>
<td>GDA</td>
<td>2.80 (0.52)</td>
<td>4.28 (0.77)</td>
<td>4.51 (0.86)</td>
</tr>
<tr>
<td>OMD</td>
<td>1.60 (0.18)</td>
<td>1.73 (0.25)</td>
<td>1.96 (0.26)</td>
</tr>
</tbody>
</table>
A Convergent and Dimension-Independent Min-Max Optimization Algorithm

References


A Convergent and Dimension-Independent Min-Max Optimization Algorithm


A Convergent and Dimension-Independent Min-Max Optimization Algorithm


A. Proof of Theorem 3.3

In this section we give the proof of Theorem 3.3.

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. \( \nu = \frac{1}{10L} \left( \frac{32b(L+1)}{\varepsilon} \right) \left( \frac{1}{\tau_1} \log \left( \frac{128}{\omega} \right) + \frac{2048b}{\omega^2} \log^2 \left( \frac{100}{\omega} \left( \frac{1}{\nu} \right) \left( 8 \frac{b}{3} + 1 \right) \right) + 1 \right) ^{-2} \)

2. \( r_{\max} = \frac{128}{\omega} \log^2 \left( \frac{100}{\omega} \left( \frac{1}{\nu} \right) \left( 8 \frac{b}{3} + 1 \right) + \log \left( \frac{1}{\eta} \right) \right) \)

3. Define \( \mathcal{I} := \frac{1}{\nu} \log \left( \frac{r_{\max}}{\nu} \right) + 8r_{\max} - \frac{b}{3} + 1 \)

4. \( \eta = \min \left( \frac{1}{10L}, \frac{1}{8L} \right) \)

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

6. \( \hat{\varepsilon}_1 = \min (\varepsilon \eta L, \frac{2}{5}) \)

7. \( b_0 = \hat{\varepsilon}_1^{-2}300b^2 \log (1/\nu) \)

8. \( b_y = \varepsilon^{-2} \hat{\varepsilon}_1^{-2}300b^2 \log (1/\nu) \)

In particular, we note that \( \nu \leq \frac{1}{20} \left( 2J\mathcal{I} + 2 \times (r_{\max} - \frac{b}{3} + 1) \right) ^{-1} \), and \( r_{\max} \geq \frac{4}{\omega} \log \left( \frac{100J}{\omega} \right) \). At every iteration \( i \leq \mathcal{I} \), where we set \( \varepsilon' = \varepsilon_i. \)

We also have

\[
\varepsilon' \leq \varepsilon_0 \left( \frac{1}{1 - 2\eta L} \right)^{2i} \leq \varepsilon. \tag{13}
\]

To see why (13) holds, note that since we set the hyperparameter \( \eta \) to be \( \eta = \min \left( \frac{1}{10L}, \frac{1}{8L} \right) \), we have \( 1 - 2\eta L \leq 1 - \frac{1}{4L} \). Since we also set \( \varepsilon_0 = \frac{1}{2} \), we therefore have that for all \( i \leq \mathcal{I} \),

\[
\varepsilon_0 (1 - 2\eta L)^{-2i} \leq \frac{\varepsilon}{2} (1 - \frac{1}{4L})^{-2i} \leq \varepsilon,
\]

where the second inequality holds because \((1 - \frac{1}{4L})^{-t} \leq 2\) for all \( t \geq 1 \).

A.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 A.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 A.4); together, these two bounds imply a \( \text{poly}(b, L, 1/\varepsilon, 1/\delta, 1/\omega) \) bound on the number of gradient, function, and sampling oracle evaluations (Lemma A.5).

**Proposition A.1.** For any \( \hat{\varepsilon}_1, \nu > 0 \), if we use batch sizes \( b_y = \varepsilon^{-2} \hat{\varepsilon}_1^{-2}300b^2 \log (1/\nu) \) and \( b_0 = \hat{\varepsilon}_1^{-2}300b^2 \log (1/\nu) \), we have that

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

and

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

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

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

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

Therefore,

\[ \mathbb{P} \left( \left\| \frac{1}{b_y} \sum_{i \in B_y} [\nabla_y f_i(x, y) - \nabla_y f(x, y)] \right\| \geq \frac{\hat{\varepsilon}_1}{10} \right) < \nu \]

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 $|f_i(x, y)|$.

\[ \square \]

**Proposition A.2.** For every $j$, with probability at least $1 - \nu$ we have that either $\|G_y(x, y_j)\| < \varepsilon$, or that

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

and

\[ \|y_{j+1} - y_j\| = \eta \|G_y(x, y_j)\| \leq 2\eta \|\nabla_y f(x, y_j)\| \]

\[ \square \]

**Proof.** By Proposition A.1 we have that, with probability at least $1 - \nu$, and whenever $\|G_y(x, y_j)\| \geq \varepsilon$,

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

where the first inequality holds by Proposition A.1, the second inequality holds since $\hat{\varepsilon}_1 \leq \varepsilon \eta L$, and the third inequality holds since $\|G_y(x, y_j)\| \geq \varepsilon$ and since (again by Proposition A.1) $\|\nabla_y f(x, y_j) - G_y(x, y_j)\| < \frac{\hat{\varepsilon}_1}{10}$. This proves Inequality (16).

Moreover, we have that, whenever $\|G_y(x, y_j)\| \geq \varepsilon$ and in the same probability $1 - \nu$ event where (16) holds,

\[ \|\eta G_y(x, y_j)\| \leq \eta \|\nabla_y f(x, y_j)\| + \frac{\hat{\varepsilon}_1}{10}. \]

Thus,

\[ 2\eta \|\nabla_y f(x, y_j)\| \geq 2\eta \left( \|G_y(x, y_j)\| - \frac{\hat{\varepsilon}_1}{10} \right) \geq \eta \|G_y(x, y_j)\| = \|y_{j+1} - y_j\|, \]

where the first inequality holds by (18), the second inequality holds since $\|G_y(x, y_j)\| \geq \varepsilon \geq \hat{\varepsilon}_1$, and the equality holds by Step 2 of Algorithm 2. This proves Inequality (17).

\[ \square \]

**Proposition A.3.** Algorithm 2 terminates in at most $J := \frac{16Lb}{\eta^2}$ iterations of its “While” loop, with probability at least $1 - \nu \times J$. 

\[ \square \]
A Convergent and Dimension-Independent Min-Max Optimization Algorithm

Proof. Let \( j_{max} \in \mathbb{N} \cup \{\infty\} \) be the number of iterations of the “While” loop in Algorithm 2.

First, we note that the stopping condition for Algorithm 2 implies that
\[
\|G_y(x, y_j)\| \geq \frac{1}{2} \varepsilon
\]  
for all \( j \leq j_{max} - 1 \).

Since \( f \) has \( L \)-Lipschitz gradient, there exists a vector \( u \), with \( \|u\| \leq L\|y_{j+1} - y_j\| \), such that, for all \( j \leq j_{max} - 1 \)
\[
f(y_{j+1}) - f(y_j) = \langle y_{j+1} - y_j, \nabla_y f(x, y_j) + u \rangle
\]
and
\[
f(y_{j+1}) - f(y_j) = \langle y_{j+1} - y_j, \nabla_y f(x, y_j) \rangle + \langle y_{j+1} - y_j, u \rangle
\]
\[
= \langle \eta G_y(x, y_j), \nabla_y f(x, y_j) \rangle - \langle \eta G_y(x, y_j), \nabla_y f(x, y_j) \rangle + \langle \eta G_y(x, y_j), u \rangle
\]
\[
\geq \eta \|G_y(x, y_j)\|^2 - \eta \|G_y(x, y_j)\| \times \|G_y(x, y_j)\| - \eta \|G_y(x, y_j)\| \times \|u\|
\]
\[
\geq \eta \|G_y(x, y_j)\|^2 - \eta \|G_y(x, y_j)\| \times \|G_y(x, y_j)\| + \|G_y(x, y_j)\| \times L \|y_{j+1} - y_j\|
\]
\[
\geq \eta \|G_y(x, y_j)\|^2 - \frac{1}{10} \eta L \|G_y(x, y_j)\|^2 - \eta \|G_y(x, y_j)\| \times L \|y_{j+1} - y_j\|
\]
\[
\geq \frac{1}{8} \eta \|G_y(x, y_j)\|^2
\]
with probability at least \( 1 - \nu \), where the second-to-last inequality holds since \( \eta \leq \frac{1}{10L} \). Existence of the vector \( u \) in Equation (20) is guaranteed by the fundamental theorem of calculus. Namely, by the fundamental theorem of calculus we have
\[
f(y_{j+1}) - f(y_j) = \int_0^1 \langle y_{j+1} - y_j, \nabla_y f(x, y_j + t(y_{j+1} - y_j)) \rangle dt.
\]
Thus (20) holds for \( u = \int_0^1 \nabla_y f(x, y_j + t(y_{j+1} - y_j)) - \nabla_y f(x, y_j) dt \). Note that this choice of \( u \) satisfies \( \|u\| \leq L \|y_{j+1} - y_j\| \), since \( \|\nabla_y f(x, y_j + t(y_{j+1} - y_j)) - \nabla_y f(x, y_j)\| \leq L \|y_{j+1} - y_j\| \) for \( t \in [0, 1] \) because \( f \) has \( L \)-Lipschitz gradient.

Since \( f \) takes values in \([-b, b] \), Inequality (20) implies that Algorithm 2 terminates in at most \( J := \frac{16b^2}{\eta L} \) iterations of its “While” loop, with probability at least \( 1 - \nu \times J \).

\[\square\]

Proposition A.4. Algorithm 2 terminates in at most \( I := r_{max} \log \left( \frac{r_{max}}{\nu} \right) + 8r_{max} \frac{b}{\delta} + 1 \) iterations of its “While” loop, with probability at least \( 1 - 2 \nu \times \left( r_{max} \frac{2b}{\delta^2} + 1 \right) \).

Proof. For any \( i > 0 \), let \( E_i \) be the “bad” event that both \( f(x_{i+1}, y_{i+1}) - f(x_i, y_i) > -\frac{\delta}{4} \) and \( \text{Accept}_i = \text{True} \).

Then by Proposition A.1 since \( \frac{\delta}{4b} \leq \frac{\delta}{8} \), we have that
\[
P(E_i) \leq e^{-\frac{1}{16t}} + \nu.
\]  
(21)

Define \( \hat{I} := r_{max} \log \left( \frac{r_{max}}{\nu} \right) \).

Then for \( i \geq \hat{I} \), from Line 4 of Algorithm 1 we have by Inequality (21) that
\[
P(E_i) \leq 2\nu.
\]

Define \( h := r_{max} \frac{2b}{\delta^2} + 1 \). Then
\[
P \left( \bigcup_{i=\hat{I}}^{\hat{I}+h} E_i \right) \leq 2\nu \times h.
\]
(22)

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 \( \frac{2b}{3^3} \).

Therefore, since \( h = r_{\max} \frac{2b}{3^3} + 1 \), 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 \( i \in [i, i + r_{\max}] \).

Therefore the condition in the While loop (Line 1) of Algorithm 1 implies that Algorithm 1 terminates after at most \( \max \) iterations of its While loop, as long as \( \bigcup_{i=\hat{I}}^{\hat{I}+h} E_i \) does not occur.

Therefore, Inequality (22) implies that, with probability at least \( 1 - 2\nu \times (r_{\max} \frac{2b}{3^3} + 1) \), Algorithm 1 terminates after at most
\[
\hat{I} + h = \tau_1 \log \left( \frac{r_{\max}}{\nu} \right) + 8r_{\max} \frac{b}{3} + 1
\]
iterations of its “While” loop.

**Lemma A.5.** With probability at least \( 1 - 3\nu J I \), Algorithm 1 terminates at most \( (\tau_1 \log (r_{\max} / \nu) + 4r_{\max} \frac{b}{3} + 1) \times (J \times b_y + b_0 + b_z) \) gradient, function, and sampling oracle evaluations.

**Proof.** Each iteration of the While loop in Algorithm 1 computes one batch gradient with batch size \( b_z \), 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 A.4 and A.5.

### A.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 A.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 A.5).

Recall the paths \( \gamma(t) \) from Definition 5.1. 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 A.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 \( \gamma(t) \):
\[
\gamma(t) = y_j + (t - \sum_{k=1}^{j-1} \|v_k\| \frac{v_j}{\|v_j\|}), \quad \forall t \in \left[ \sum_{k=1}^{j-1} \|v_k\|, \sum_{k=1}^{j} \|v_k\| \right], \quad j \in [j_{\max}],
\]
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
\[
\frac{d}{dt} f(x, \gamma(t)) \geq \left( \frac{\eta L}{10} \|G_y(x, y_j)\| \right) - \frac{\eta L}{10} \|G_y(x, y_j)\| \geq (1 - 2\eta L)\|G_y(x, y_j)\|
\]

\[
\geq \left( 1 - 2\eta L \right) \|G_y(x, y_j)\|
\]

\[
\geq (1 - 2\eta L) \|G_y(x, y_j)\|
\]
\[
\geq (1 - 2\eta L)\varepsilon' \quad \forall t \in \left[\sum_{k=1}^{j-1} \|v_k\|, \sum_{k=1}^{j} \|v_k\|\right],
\]
with probability at least \(1 - \nu\) for some unit vectors \(u, w \in \mathbb{R}^d\).

But by Proposition A.3 we have that \(j_{\text{max}} \leq J\) with probability at least \(1 - \nu \times J\). Therefore inequality (23) implies that
\[
\frac{d}{dt} f(x, \gamma(t)) \geq (1 - 2\eta L)\varepsilon' \quad \forall t \in [0, \sum_{k=1}^{j_{\text{max}}} \|v_k\|],
\]
with probability at least \(1 - 2\nu J\).

**Lemma A.7.** Let \(i^* + 1\) be the last iteration \(i\) of the “While” loop in Algorithm 1 for which \(\text{Accept}_i = \text{True}\). Then with probability at least \(1 - 2\nu J I - 2\nu \times (r_{\max} \frac{2b}{\eta L} + 1)\) we have that
\[
\|\nabla_y f(x^*, y^*)\| \leq (1 - \eta L)\varepsilon_{i^*}. \tag{24}
\]
Moreover, with probability at least \(1 - \frac{\omega}{100} - 2\nu \times (r_{\max} \frac{2b}{\eta L} + 1)\) we have that
\[
\mathbb{P}_{\Delta \sim Q_{i^*, y^*}} \left(\mathcal{E}_{i^*} \left( (x^*, \Delta, y^*) \leq \mathcal{L}_{i^*}(x^*, y^*) - \frac{1}{2}\|x^*, y^*\| \right) \right) \leq \frac{1}{2}\omega. \tag{25}
\]

and that
\[
\frac{\varepsilon}{2} \leq \varepsilon_{i^*} \leq \varepsilon.
\]

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

Let \(H_i\) be the “bad” event that, when Algorithm 1 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 A.6 we have that
\[
\mathbb{P}(H_i) \leq 2\nu J. \tag{27}
\]

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 A.1 and A.3 we have that
\[
\mathbb{P}(K_i) \leq 2\nu J. \tag{28}
\]

Whenever \(K_i^c\) occurs we have that
\[
\|\nabla_y f(x_i, y_i)\| \leq \|G_y(x_i, y_i)\| + \|G_y(x_i, y_i) - \nabla_y f(x_i, y_i)\| \tag{29}
\]
\[
\leq (1 - 2\eta L)\varepsilon_i + \|G_y(x_i, y_i) - \nabla_y f(x_i, y_i)\| \\
\leq (1 - 2\eta L)\varepsilon_i + \frac{\varepsilon_i}{10} \\
\leq (1 - \eta L)\varepsilon_i,
\]
where the second Inequality holds by Line 2 of Algorithm 2 and the last inequality holds since \(\frac{\varepsilon_i}{10} \leq \eta L\).

Therefore, Inequalities (28) and (29) together with Proposition A.4 imply that
\[
\|\nabla_y f(x^*, y^*)\| \leq (1 - \eta L)\varepsilon_{i^*},
\]
with probability at least \(1 - 2\nu J I - 2\nu \times (r_{\max} \frac{2b}{\eta L} + 1)\). This proves Inequality (24).

Inequality (29) also implies that, whenever \(K_i^c\) occurs, the set \(P_{\varepsilon_i}(x_i, y_i)\) of endpoints of \(\varepsilon_i\)-increasing paths with initial point \(y_i\) (and \(x\)-value \(x_i\)) consists only of the single point \(y_i\). Therefore, we have that
\[
\mathcal{L}_{\varepsilon_i}(x_i, y_i) = f(x_i, y_i) \tag{30}
\]
whenever \(K_i^c\) occurs.

Moreover, whenever \(H_i^c\) occurs we have that \(Y_{i+1}\) is the endpoint of an \(\varepsilon_i\)-increasing path with starting point \((x_i + \Delta_i, y_i)\). Now, \(\mathcal{L}_{\varepsilon_i}(x_i + \Delta_i, y_i)\) is the supremum of the value of \(f\) at the endpoints of all \(\varepsilon_i\)-increasing paths with starting point \((x_i + \Delta_i, y_i)\). Therefore, we must have that
\[
\mathcal{L}_{\varepsilon_i}(x_i + \Delta_i, y_i) \geq f(x_i + \Delta_i, Y_{i+1}) \tag{31}
\]
whenever \(H_i^c\) occurs.
Therefore,
\[
\mathbb{P}_{\Delta \sim \mathcal{Q}_{x_i, y_i}} \left( \mathcal{L}_{\epsilon_i}(x_i + \Delta, y_i) > \mathcal{L}_{\epsilon_i}(x_i, y_i) - \frac{1}{2} \left| x_i, y_i \right| \right)
\geq \mathbb{P}_{\Delta \sim \mathcal{Q}_{x_i, y_i}} \left( f(x_i + \Delta, y_{i+1}) > f(x_i, y_i) - \frac{1}{2} \left| x_i, y_i \right| - \mathbb{P}(H_i) - \mathbb{P}(K_i) \right)
\geq \mathbb{P} \left( \text{Accept}_i = \text{False} \mid x_i, y_i \right) - 2\nu - \mathbb{P}(H_i) - \mathbb{P}(K_i)
\geq \mathbb{P} \left( \text{Accept}_i = \text{False} \mid x_i, y_i \right) - 2\nu - 2\nu \mathcal{J} - 2\nu \mathcal{F}, \ \forall i \leq I,
\]
where the second inequality holds by Proposition \[A.1\] since \( \frac{\delta_i}{10} \leq \frac{\delta}{8} \).

Define
\[
p_i := \mathbb{P}_{\Delta \sim \mathcal{Q}_{x_i, y_i}} \left( \mathcal{L}_{\epsilon_i}(x_i + \Delta, y_i) > \mathcal{L}_{\epsilon_i}(x_i, y_i) - \frac{1}{2} \left| x_i, y_i \right| \right)
\]
for every \( i \in \mathbb{N} \). Then Inequality \[32\] implies that
\[
\mathbb{P} \left( \text{Accept}_i = \text{False} \mid x_i, y_i \right) \leq p_i + \nu(4\mathcal{J} + 2) \leq p_i + \frac{1}{8} \omega \quad \forall i \leq I,
\]
since \( \nu \leq \frac{\omega}{3\mathcal{J} + 16} \).

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 \[33\] that
\[
\mathbb{P} \left( \bigcap_{s=0}^{r_{\text{max}}} \{ \text{Accept}_{i+s} = \text{False} \} \mid p_i \leq 1 - \frac{1}{2} \omega \right) \leq (1 - \frac{1}{4} \omega)^{r_{\text{max}}} \leq \frac{\omega}{100}\mathcal{I} \quad \forall i \leq I - r_{\text{max}}
\]
since \( r_{\text{max}} \geq \frac{4}{\omega} \log\left( \frac{100\mathcal{I}}{\omega} \right) \).

Therefore, with probability at least \( 1 - \frac{\omega}{100} \times \mathcal{I} = 1 - \frac{\omega}{100} \), we have that the event \( \bigcap_{s=0}^{r_{\text{max}}} \{ \text{Accept}_{i+s} = \text{False} \} \) does not occur for any \( i \leq I - r_{\text{max}} \) for which \( p_i \leq 1 - \frac{1}{2} \omega \).

Recall from Proposition \[A.5\] that Algorithm \[A\] terminates in at most \( \mathcal{I} \) iterations of its “While” loop, with probability at least \( 1 - 2\nu \times (r_{\text{max}} \frac{2b}{4\beta} + 1) \).

Therefore,
\[
\mathbb{P} \left( p_{i^*} > 1 - \frac{1}{2} \omega \right) \geq 1 - \frac{\omega}{100} - 2\nu \times \left( r_{\text{max}} \frac{2b}{4\beta} + 1 \right).
\]
In other words, by the definition of \( p_{i^*} \), Inequality \[34\] implies that with probability at least \( 1 - \frac{\omega}{100} - 2\nu \times (r_{\text{max}} \frac{2b}{4\beta} + 1) \), the point \( (x^*, y^*) \) is such that
\[
\mathbb{P}_{\Delta \sim \mathcal{Q}_{x_i, y_i}} \left( \mathcal{L}_{\epsilon_i}(x^* + \Delta, y^*) \leq \mathcal{L}_{\epsilon_i}(x^*, y^*) - \frac{1}{2} \left| x^*, y^* \right| \right) \leq \frac{1}{2} \omega.
\]
This completes the proof of inequality \[25\].

Finally we note that when Algorithm \[A\] terminates in at most \( \mathcal{I} \) iterations of its “While” loop, we have
\[
\varepsilon_{i^*} = \varepsilon_0 \left( \frac{1}{1 - 2\eta L} \right)^{2i^*} \leq \varepsilon_0 \left( \frac{1}{1 - 2\eta L} \right)^{2\mathcal{I}} \leq \varepsilon,
\]
since \( \eta \leq \frac{1}{8\mathcal{I}} \). This completes the proof of Inequality \[26\].

We can now complete the proof of the main theorem:

**Proof of Theorem \[3.3\]** First, by Lemma \[A.5\] with probability at least \( 1 - 3\nu \mathcal{I} \mathcal{J} \geq \frac{99}{100} \), our algorithm converges to some
point \((x^*, y^*)\) after at most \((\tau_1 \log(\frac{r_{\text{max}}}{\epsilon}) + 4r_{\text{max}}b + 1) \times (J \times b_\nu + b_0 + b_\omega)\) gradient, function, and sampling oracle evaluations, which is polynomial in \(b, L, 1/\epsilon, 1/\delta, 1/\omega,\) and does not depend on the dimension \(d.\)

By Lemma A.7 if we set \(\epsilon^* = \epsilon^1\), we have that Inequalities (7) and (6) hold for parameters \(\epsilon^* \in [\frac{1}{4}\epsilon, \epsilon]\), \(\delta, \omega\) and distribution \(Q\), with probability at least \(1 - 2\nu J I - 2\nu \times (r_{\text{max}} + \frac{b}{\epsilon} + 1)\geq \frac{10}{3\nu^2}, \) since \(\nu \leq \frac{1}{3\nu^2}(2J I + 2 \times (r_{\text{max}} + \frac{b}{\epsilon} + 1))^{-1}.\)

### B. Equilibrium in the Strongly Convex-Strongly Concave Setting

In this section we show that, if \(f\) is strongly-convex strongly-concave with Lipschitz gradient, if we choose \(Q\) to be the distribution of (either deterministic or stochastic) gradients with mean \(-\nabla_x f\), then an \((\epsilon, \delta, \omega, Q)\)-equilibrium corresponds to an “approximate” global min-max point (Theorem B.1). We then show that this fact, together with the proof of our main result, implies that when our algorithm is applied to \(\alpha\)-strongly-convex \(\alpha\)-strongly-concave objective functions \(f\) with \(L\)-Lipschitz gradient, it finds an “approximate” global min-max point with duality gap \(O(\epsilon)\) in a number of gradient evaluations that is polynomial in \(L, \frac{1}{\epsilon}, \frac{1}{\alpha}, L, D\) (Corollary B.10).

For any \(L > 0\), we say that a function \(\psi : \mathbb{R}^d \rightarrow \mathbb{R}\) has \(L\)-Lipschitz gradient (equivalently, “\(L\)-smooth”) if for any \(\theta, \xi \in \mathbb{R}^d,\)

\[
\|\nabla \psi(x) - \nabla \psi(\theta)\| \leq L \times \|x - \theta\|.
\]

And for any \(\alpha > 0\) we say that \(\psi\) is \(\alpha\)-strongly convex if for any \(\theta, \xi \in \mathbb{R}^d,\)

\[
(\nabla \psi(x) - \nabla \psi(\theta))^\top (x - \theta) \geq \alpha \|x - \theta\|^2.
\]

Similarly, we say that \(\psi\) is \(\alpha\)-strongly concave if \(-\psi\) is \(\alpha\)-strongly-convex.

In the following, we assume that the proposal distribution is a stochastic gradient for \(-\nabla_x f\) with some variance \(\sigma^2 \geq 0\); for simplicity we set \(\sigma^2 = 0\) in Corollary B.10, although this is not strictly necessary.

**Assumption B.1.** \((\sigma \geq 0)\) For every \((x, y) \in \mathbb{R}^d \times \mathbb{R}^d\), the distribution \(Q_{x,y}\) satisfies \(\mathbb{E}_{\Delta \sim Q_{x,y}} \|\Delta\| = -\frac{1}{2L} \nabla_x f(x, y)\) and \(\mathbb{E}_{\Delta \sim Q_{x,y}} \|\Delta\|^2 \leq \sigma^2.\)

**Theorem B.2.** Suppose that \(f : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}\) is \(\alpha\)-strongly convex in \(x\) and \(\alpha\)-strongly concave in \(y\), with \(L\)-Lipschitz gradient in both variables for some \(L \geq \alpha\). Then, for any \(\epsilon, \delta, \omega > 0\) with \(\omega \leq \frac{1}{2}\), and any proposal distribution \(Q_{x,y}\) satisfying Assumption B.1 for some \(\sigma \geq 0\), we have that any point \((x^*, y^*)\) which is an \((\epsilon, \delta, \omega, Q)\)-approximate local equilibrium of \(f\) also has duality gap satisfying

\[
\max_{z \in \mathbb{R}^d} f(x^*, y) - \min_{x \in \mathbb{R}^d} f(x, y^*) \leq \frac{L^2}{2\alpha^2} + \frac{L}{\alpha^2} \left(\sqrt{\delta + L \left(\frac{\epsilon}{\alpha} + \frac{1}{\alpha} (L \frac{\epsilon}{\alpha} + 2\sigma)\right) + L \frac{\epsilon^2}{2\alpha^2} + L \frac{\epsilon}{\alpha} + L \frac{\epsilon}{\alpha}}\right)^2.\tag{36}
\]

Before proving Theorem B.2, we first show a number of Lemmas.

**Lemma B.3.** For any \(x, w \in \mathbb{R}^d,\)

\[
\|\operatorname{argmax}_{z \in \mathbb{R}^d} f(x, z) - \operatorname{argmax}_{z \in \mathbb{R}^d} f(\theta, z)\| \leq \frac{L}{\alpha} \|\theta - x\|\tag{37}
\]

**Proof.** Since \(f(x, \cdot)\) is concave, we have that a point \(z\) is a global maximum if and only if \(\nabla_y f(x, z) = 0\). Since \(f(x, \cdot)\) is \(\alpha\)-strongly concave for \(\alpha > 0\), this global maximum point is unique.

Let \(z^*\) be the global maximum of \(f(x, \cdot)\), and let \(\zeta^*\) be the global maximum of \(f(\theta, \cdot)\).

Then, since \(f(x, \cdot)\) is \(\alpha\)-strongly concave, \(\|\nabla_y f(x, z)\| \geq \alpha \|z - z^*\|\) for all \(z \in \mathbb{R}^d\).

Moreover, since \(f\) is \(L\)-smooth, we also have that \(\|\nabla_y f(x, z) - \nabla_y f(\theta, z)\| \leq L \|\theta - x\|\) for every \(x, \theta, z \in \mathbb{R}^d\).

Therefore,

\[
\|\nabla_y f(\theta, z)\| \geq \|\nabla_y f(x, z)\| - \|\nabla_y f(\theta, z) - \nabla_y f(x, z)\| \geq \|\nabla_y f(x, z)\| - L \|\theta - x\| \geq \alpha \|z - z^*\| - L \|\theta - x\|.\tag{38}
\]
Since \( \alpha \| z - z^* \| - L \| \theta - x \| > 0 \) for any \( z \in \mathbb{R}^d \) such that \( \| z - z^* \| > \frac{L}{\alpha} \| \theta - x \| \), \((38)\) implies that
\[
\| \argmax_{z \in \mathbb{R}^d} f(x, z) - \argmax_{z \in \mathbb{R}^d} f(\theta, z) \| \leq \frac{L}{\alpha} \| \theta - x \|
\]  
(39)

\[\square\]

**Lemma B.4.** If \((x, w) \in \mathbb{R}^d \times \mathbb{R}^d\) are such that \(\| \nabla_y f(x, w) \| \leq \varepsilon\) then
\[
\| w - \argmax_{z \in \mathbb{R}^d} f(x, z) \| \leq \frac{\varepsilon}{\alpha}
\]

**Proof.** Let \(z^*\) be the unique global maximum point of \(f(x, \cdot)\).
Since \(f(x, \cdot)\) is \(\alpha\)-strongly concave, we have that
\[
\alpha \| w - z^* \| \leq \| \nabla_y f(x, w) \| \leq \varepsilon
\]  
(40)

Therefore, \((40)\) implies that
\[
\| w - \argmax_{z \in \mathbb{R}^d} f(x, z) \| = \| w - z^* \| \leq \frac{\varepsilon}{\alpha}
\]  
(41)

\[\square\]

**Lemma B.5.** For any \(x, \theta \in \mathbb{R}^d\),
\[
\| \mathcal{L}_\varepsilon(x, y) - \mathcal{L}_\varepsilon(\theta, y) \| \leq L \left( \frac{2 \varepsilon}{\alpha} + \frac{L}{\alpha} \| \theta - x \| \right)
\]

**Proof.** Denote by \(\hat{P}_\varepsilon(x, y) \subseteq P_\varepsilon(x, y)\) the collection of endpoints of \(\varepsilon\)-greedy paths where the endpoint \(z\) of the path satisfies \(\nabla_y f(x, z) = \varepsilon\). Since \(f\) is smooth, we have that \(\sup_{z \in \hat{P}_\varepsilon(x, y)} f(x, z) = \sup_{z \in \hat{P}_\varepsilon(x, y)} f(x, z)\) (this is true since, if the endpoint \(z\) of an \(\varepsilon\)-greedy path does not satisfy \(\| \nabla_y f(x, z) \| = \varepsilon\), then the \(\varepsilon\)-greedy path can be extended to achieve a higher value of \(f\)).

Thus, we have that
\[
\| \mathcal{L}_\varepsilon(x, y) - \mathcal{L}_\varepsilon(\theta, y) \| = \| \sup_{z \in \hat{P}_\varepsilon(x, y)} f(x, z) - \sup_{w \in P_\varepsilon(\theta, y)} f(\theta, w) \|
\]
(42)

\[
= \| \sup_{z \in \hat{P}_\varepsilon(x, y)} f(x, z) - \sup_{w \in P_\varepsilon(\theta, y)} f(\theta, w) \|
\]
\[
\leq \sup_{z \in \hat{P}_\varepsilon(x, y), w \in P_\varepsilon(\theta, y)} | f(x, z) - f(\theta, w) |
\]
\[
\leq \sup_{z \in \hat{P}_\varepsilon(x, y), w \in P_\varepsilon(\theta, y)} \| f(x, z) - f(\theta, w) \|
\]
\[
\leq \sup_{z \in \hat{P}_\varepsilon(x, y), w \in P_\varepsilon(\theta, y)} \| w - z \|
\]
\[
\leq \sup_{z \in \hat{P}_\varepsilon(x, y), w \in P_\varepsilon(\theta, y)} L \times \| w - \argmax_{\zeta \in \mathbb{R}^d} f(\theta, \zeta) \|
\]
\[
+ \| \argmax_{\zeta \in \mathbb{R}^d} f(\theta, \zeta) - \argmax_{\zeta \in \mathbb{R}^d} f(x, \zeta) \| \leq \sup_{z \in \hat{P}_\varepsilon(x, y), w \in P_\varepsilon(\theta, y)} L \times \left( \| w - \argmax_{\zeta \in \mathbb{R}^d} f(\theta, \zeta) \| + \| \argmax_{\zeta \in \mathbb{R}^d} f(x, \zeta) - z \| \right)
\]
\[
\leq L \times \left( \frac{\varepsilon}{\alpha} + \frac{L}{\alpha} \| \theta - x \| + \frac{\varepsilon}{\alpha} \right)
\]
\[
= L \times \left( \frac{2 \varepsilon}{\alpha} + \frac{L}{\alpha} \| \theta - x \| \right)
\]

where the last inequality holds by Lemma B.3 and also by Lemma B.4 because \(\| \nabla_y f(x, z) \| = \varepsilon\) whenever \(z \in \hat{P}_\varepsilon(x, y)\) and \(\| \nabla_y f(\theta, w) \| = \varepsilon\) whenever \(w \in P_\varepsilon(\theta, y)\).

**Lemma B.6.** For any \(x, y \in \mathbb{R}^d\) we have
\[
| \mathcal{L}_\varepsilon(x, y) - \mathcal{L}_0(x, y) | \leq L \frac{\varepsilon^2}{2\alpha^2}
\]
Proof.

\[|\mathcal{L}_\varepsilon(x, y) - \mathcal{L}_0(x, y)| = \left| \sup_{z \in \hat{P}(x, y)} f(x, z) - f(x, \arg\max_{z \in \mathbb{R}^d} f(x, z)) \right| \tag{43}\]
\[\leq \sup_{z \in \hat{P}(x, y)} |f(x, z) - f(x, \arg\max_{z \in \mathbb{R}^d} f(x, z))| \leq \sup_{z \in \hat{P}(x, y)} \frac{L}{2} \|z - \arg\max_{z \in \mathbb{R}^d} f(x, z)\|^2 \]
\[\leq \frac{L}{2} \left( \frac{\varepsilon}{\alpha} \right)^2, \]

where the last inequality holds by Lemma B.4 because \(\|\nabla_y f(x, z)\| = \varepsilon\) whenever \(z \in \hat{P}_\varepsilon(x, y)\).

\[\square\]

**Lemma B.7.** \(\mathcal{L}_0(x^*, y^*)\) is differentiable and

\[\|\nabla_x f(x^*, y^*) - \nabla_x \mathcal{L}_0(x^*, y^*)\| \leq \frac{L \varepsilon}{\alpha}.\]

**Proof.** Since \(f(x, \cdot)\) is concave, we have that \(\mathcal{L}_0(x^*, y^*) = \max_{z \in \mathbb{R}^d} f(x^*, z)\).

Moreover, by strong concavity \(f(x^*, \cdot)\) has a unique maximizer \(\arg\max_{z \in \mathbb{R}^d} f(x^*, z)\).

Thus, since Daskin’s Theorem ([Daskin, 1966]), we have that \(\mathcal{L}_0(x^*, \cdot)\) is differentiable and that \(\nabla_x \mathcal{L}_0(x^*, y^*) = \nabla_x f(x^*, \arg\max_{z \in \mathbb{R}^d} f(x^*, z))\).

Therefore, since \(f\) is \(L\)-smooth,

\[\|\nabla_x f(x^*, y^*) - \nabla_x \mathcal{L}_0(x^*, y^*)\| \leq L \times \|y^* - \arg\max_{z \in \mathbb{R}^d} f(x^*, z)\| \leq L \times \frac{\varepsilon}{\alpha},\]

where the last inequality holds by Lemma B.4 since \(\|\nabla_y f(x^*, y^*)\| \leq \varepsilon\).

\[\square\]

**Proof of Theorem B.2** Since \((x^*, y^*)\) is an \((\varepsilon, \delta, \omega, Q)\)-approximate local equilibrium of \(f\), we have that,

\[\|\nabla_y f(x^*, y^*)\| \leq \varepsilon, \tag{46}\]

and that, with probability at least \(1 - \omega\),

\[\mathcal{L}_\varepsilon(x^* - \Delta, y^*) \geq f(x^*, y^*) - \delta, \tag{47}\]

where \(\Delta \sim Q_{x^*, y^*}\).

Thus, since by Assumption B.1 \(\mathbb{E}_{\Delta \sim Q_{x^*, y^*}} |\Delta| = \mu \nabla_x f(x, y)\) and \(\mathbb{E}_{\Delta \sim Q_{x^*, y}} \|\Delta - \mu \nabla_x\|^2 \leq \sigma^2\), by Chebyshev’s inequality, we have that, with probability at least \(1 - \omega - \frac{1}{4}\),

\[\mathcal{L}_\varepsilon(x^* - \mu \nabla_x f(x^*, y^*) + \nu, y^*) \geq f(x^*, y^*) - \delta, \tag{48}\]

for some \(\nu \in \mathbb{R}^d\) such that \(\|\nu\| \leq 2\sigma\).

Since \(\omega \leq \frac{1}{2}\), we have \(1 - \omega - \frac{1}{2} \geq \frac{1}{4}\). Thus, (48) holds with probability at least \(\frac{1}{2}\). Since (48) holds with probability at least \(\frac{1}{4}\) yet contains no random variables, it must also hold with probability 1.

Therefore, by plugging in Lemma B.6 to the LHS of (48) and applying Lemma B.4 together with the fact that \(f\) is \(L\)-smooth to the RHS of (48), we have that

\[\mathcal{L}_0(x^* - \mu \nabla_x f(x^*, y^*) + \nu, y^*) + \frac{L \varepsilon^2}{2\alpha^2} \geq \mathcal{L}_0(x^*, y^*) - L \frac{\varepsilon}{\alpha} - \delta. \tag{49}\]

Therefore, applying Lemma B.7 to (49) we get that

\[\mathcal{L}_0(x^* - \mu \nabla_x \mathcal{L}_0(x^*, y^*) + \nu, y^*) + \frac{L \varepsilon^2}{2\alpha^2} \geq \mathcal{L}_0(x^*, y^*) - L \frac{\varepsilon}{\alpha} - \delta, \tag{50}\]

for some \(\nu \in \mathbb{R}^d\) such that \(\|\nu\| \leq L \frac{\varepsilon}{\alpha} + 2\sigma\).
We also have that
\[ L_0(x^* - \mu \nabla_x L_0(x^*, y^*)) + L \left( \frac{2\varepsilon}{\alpha} + \mu \frac{L \varepsilon}{\alpha} + 2\sigma \right) + L \frac{\varepsilon^2}{2\alpha^2} \geq L_0(x^*, y^*) - L \frac{\varepsilon}{\alpha} - \delta, \tag{51} \]
since \( \|v\| \leq L \frac{\varepsilon}{\alpha} + 2\sigma. \)

We also have that
\[ L_0(x^* - \mu \nabla_x L_0(x^*, y^*)) = L_0(x^*, y^*) - (\nabla_x L_0(x^*, y^*) + u)^\top \mu \nabla_x L_0(x^*, y^*), \tag{52} \]
for some \( u \in \mathbb{R}^d \) such that \( \|u\| \leq L \|\mu \nabla_x L_0(x^*, y^*)\|, \) since \( f \) is \( L \)-smooth.

Therefore (52) implies that
\[ L_0(x^* - \mu \nabla_x L_0(x^*, y^*)) \leq L_0(x^*, y^*) - (\mu - \mu^2 L) \|\nabla_x L_0(x^*, y^*)\|^2, \tag{53} \]
Plugging (53) into (51), we get that (since \( \mu = \frac{1}{2L} \) implies that \( \mu - \mu^2 L > 0 \),
\[ \|\nabla_x L_0(x^*, y^*)\| \leq \frac{1}{\mu - \mu^2 L} \delta + L \left( \frac{2\varepsilon}{\alpha} + \mu \frac{L \varepsilon}{\alpha} + 2\sigma \right) + L \frac{\varepsilon^2}{2\alpha^2} + L \frac{\varepsilon}{\alpha}, \tag{54} \]
But from (44) we have that
\[ \nabla_x L_0(x^*, y^*) = \nabla_x f(x^*, \text{argmax}_{z \in \mathbb{R}^d} f(x^*, z)). \tag{55} \]

Thus,
\[ \|\nabla_x L_0(x^*, y^*) - \nabla_x f(x^*, y^*)\| \leq L \|y^* - \text{argmax}_{z \in \mathbb{R}^d} f(x^*, z)\| \]
\[ \leq L \frac{\varepsilon}{\alpha}, \tag{56} \]
where the first inequality holds since \( f \) is \( L \)-smooth, and the second inequality holds by Lemma B.4 since \( \|\nabla_y f(x^*, y^*)\| \leq \varepsilon. \)

Plugging in (56) into (54), we get
\[ \|\nabla_x f(x^*, y^*)\| \leq \frac{1}{\mu - \mu^2 L} \sqrt{\delta + L \left( \frac{2\varepsilon}{\alpha} + \mu \frac{L \varepsilon}{\alpha} + 2\sigma \right) + L \frac{\varepsilon^2}{2\alpha^2} + L \frac{\varepsilon}{\alpha}}, \tag{57} \]
Now, since \( f(\cdot, y^*) \) is \( \alpha \)-strongly convex, by Lemma B.4 (applied to \(-f\) instead of \(f\), we have
\[ \|x^* - \text{argmin}_{\theta \in \mathbb{R}^d} f(\theta, y^*)\| \leq \frac{\|\nabla_x f(x^*, y^*)\|}{\alpha}. \tag{58} \]

Since \( f \) is \( L \)-smooth, (58) implies that
\[ f(x^*, y^*) - \min_{\theta \in \mathbb{R}^d} f(\theta, y^*) \leq L \frac{1}{2} \|x^* - \text{argmin}_{\theta \in \mathbb{R}^d} f(\theta, y^*)\|^2 \]
\[ \leq \frac{L}{2} \left( \left( \frac{\|\nabla_x f(x^*, y^*)\|}{\alpha} \right)^2. \tag{59} \]

Moreover, since \( f(x^*, \cdot) \) is \( \alpha \)-strongly concave and \( \|\nabla_y f(x^*, y^*)\| \leq \varepsilon \) we have by Lemma B.4 that
\[ \|y^* - \text{argmax}_{z \in \mathbb{R}^d} f(x^*, z)\| \leq \frac{\varepsilon}{\alpha}. \tag{60} \]

Since \( f \) is \( L \)-smooth, (60) implies that
\[ \max_{z \in \mathbb{R}^d} f(x^*, z) - f(x^*, y^*) \leq L \frac{1}{2} \|y^* - \text{argmax}_{z \in \mathbb{R}^d} f(x^*, z)\|^2 \]
\[ \leq \frac{L}{2} \left( \left( \frac{\varepsilon}{\alpha} \right)^2. \tag{61} \]
Thus, adding (59) to (61), and plugging in (57), we get that

\[
\max_{z \in \mathbb{R}^d} f(x^*, z) - \min_{\theta \in \mathbb{R}^d} f(\theta, y^*) \leq \frac{L}{2} \left( \frac{\|\nabla_x f(x^*, y^*)\|}{\alpha} \right)^2 + \frac{L}{2} \left( \frac{\varepsilon}{\alpha} \right)^2
\]

(62)\]

Since \( \mu = \frac{1}{2L} \), we get that

\[
\max_{z \in \mathbb{R}^d} f(x^*, z) - \min_{\theta \in \mathbb{R}^d} f(\theta, y^*) \leq \frac{L \varepsilon^2}{2\alpha^2} + \frac{L}{2\alpha^2} \left( \frac{1}{\mu - \mu^2} \sqrt{\delta + L \left( \frac{2\varepsilon}{\alpha} + \mu \left( \frac{L \varepsilon}{\alpha} + 2\sigma \right) \right)} + \frac{L \varepsilon^2}{2\alpha^2} + \frac{L}{2\alpha} \right)^2
\]

(63)

\]

B.1. Runtime in Strongly Convex-Strongly Concave Setting

Suppose that \( f(x, y) \) is \( L \)-smooth in \( (x, y) \), \( \alpha \)-strongly convex in \( x \) and \( \alpha \)-strongly concave in \( y \).

In this section we assume that hyper-parameter \( \tau_1 \) in Algorithm 1 is set to \( \tau_1 = \infty \), and that Algorithm 2 takes as input exact gradients for \( \nabla_y f \) (i.e., the “stochastic” gradients have variance set to 0).

**Lemma B.8.** The function \( \max_{x \in \mathbb{R}^d} f(\cdot, z) \) is \( \alpha \)-strongly convex.

**Proof.** Define the “global max” function \( \psi(x) := \max_{z \in \mathbb{R}^d} f(x, z) \) for all \( x \in \mathcal{X} \). We start by showing that the function \( \psi(x) \) is \( \alpha \)-strongly convex. Indeed, for any \( x_1, x_2 \in \mathcal{X} \) and any \( \lambda \in [0, 1] \) we have

\[
\begin{align*}
\lambda \psi(\lambda x_1 + (1 - \lambda)x_2) &= \max_{y \in \mathbb{R}^d} f(\lambda x_1 + (1 - \lambda)x_2, y) \\
\quad &\leq \max_{y \in \mathbb{R}^d} \left[ \lambda f(x_1, y) + (1 - \lambda)f(x_2, y) - \frac{1}{2} \alpha \lambda(1 - \lambda)\|x_1 - x_2\|^2 \right] \\
\quad &\leq \lambda \max_{y \in \mathbb{R}^d} f(x_1, y) + (1 - \lambda)\max_{y \in \mathbb{R}^d} f(x_2, y) - \frac{1}{2} \alpha \lambda(1 - \lambda)\|x_1 - x_2\|^2 \\
\quad &= \lambda \psi(x_1) + (1 - \lambda)\psi(x_2) - \frac{1}{2} \alpha \lambda(1 - \lambda)\|x_1 - x_2\|^2,
\end{align*}
\]

where the first inequality holds by the \( \alpha \)-strong convexity of \( f(\cdot, y) \). Thus \( \psi \) is \( \alpha \)-strongly convex. \( \square \)

**Lemma B.9.** Denote by \( y_{i,j} \) the point \( y_j \) in Algorithm 2 when it is called at the \( i \)'th iteration of Algorithm 1. Let \( (x^i, y^i) \) be the global min-max point of \( f \), and define \( D := \| (x_0, y_0) - (x^i, y^i) \| \) and \( \mathcal{D} := 2\max(D + \frac{L^2}{\alpha^2} + \frac{\varepsilon}{\alpha}, \frac{L \alpha^2}{\alpha \sqrt{\varepsilon}}, \frac{\varepsilon \sqrt{\alpha}}{\alpha \varepsilon}) \). Then, as long as \( \eta \leq \frac{1}{L} \), at every iteration \( i \) of Algorithm 1 and at every iteration \( j \) of its subroutine Algorithm 2 we have that \( \| (x_i, y_i) - (x^i, y^i) \| \leq \mathcal{D} \) and \( \| (x_i, y_{i,j}) - (x^i, y^i) \| \leq \mathcal{D} \).

**Proof.** Bounding the distance \( \| (x_i, y_i) - (x^i, y^i) \| \).

Since at the global min-max point \( (x^i, y^i) \) we have \( \nabla_y f(x^i, y^i) = 0 \), and since \( f \) is \( L \)-Lipschitz, we have that

\[
\| \nabla_y f(x_0, y_0) \| \leq L \| (x_0, y_0) - (x^i, y^i) \| \leq LD.
\]

Thus, since \( f(x_0, \cdot) \) is \( \alpha \)-strongly concave, we have by Lemma B.4 that

\[
\| y_0 - \arg\max_{z \in \mathbb{R}^d} f(x_0, z) \| \leq \frac{LD}{\alpha},
\]

(64)
And since (by definition) \( x_0 = x_1 \), and \( \nabla_y f(x_0, y_1) = \nabla_y f(x_1, y_1) \leq \varepsilon \), we have that (again by Lemma B.4),

\[
\|y_1 - \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)\| \leq \frac{\varepsilon}{\alpha}.
\]  

(65)

Thus, combining (64) and (65), we have that (since \( x_0 = x_1 \)),

\[
\|(x_1, y_1) - (x^\dagger, y^\dagger)\| \leq \|(x_0, y_0) - (x^\dagger, y^\dagger)\| + \|(x_0, y_0) - (x_0, y_1)\|
\leq D + \|y_0 - y_1\|
\leq D + \|y_1 - \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)\| + \|y_0 - \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)\|
\leq D + \frac{LD}{\alpha} + \frac{\varepsilon}{\alpha}
\leq \mathcal{D}.
\]

(66)

Bounding the distance \( \|x_i - x^\dagger\| \).

At each iteration \( i > 1 \) of Algorithm 1 we have that \( \|\nabla_y f(x_i, y_i)\| \leq \varepsilon \). Thus, by Lemma B.4 we have that

\[
\|y_i - \text{argmax}_{z \in \mathbb{R}^d} f(x_i, z)\| \leq \frac{\varepsilon}{\alpha}.
\]

(67)

Thus, since \( f \) is \( L \)-smooth,

\[
\max_{z \in \mathbb{R}^d} f(x_i, z) - f(x_i, y_i) \leq L \times \|y_i - \text{argmax}_{z \in \mathbb{R}^d} f(x_i, z)\|^2
\leq L \frac{\varepsilon^2}{\alpha^2}.
\]

(68)

But \( f(x_{i+1}, y_{i+1}) \leq f(x_i, y_i) \) at each iteration \( i \) (since, if the proposed update to \( x_i \) does not lead to a decrease in the value of \( f \) we have that the proposed update to \( x_i \) would be rejected and \( x_i = x_{i+1} \) and \( y_i = y_{i+1} \)). Therefore,

\[
f(x_i, y_i) \leq f(x_1, y_1) \quad \forall i \geq 1.
\]

(69)

Therefore, (68) and (69) imply that

\[
\max_{z \in \mathbb{R}^d} f(x_i, z) \leq f(x_i, y_i) + L \frac{\varepsilon^2}{\alpha^2}
\leq f(x_1, y_1) + L \frac{\varepsilon^2}{\alpha^2}
\leq \max_{z \in \mathbb{R}^d} f(x_1, z) + L \frac{\varepsilon^2}{\alpha^2}
= f(x_0, \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)) + L \frac{\varepsilon^2}{\alpha^2}
\leq f(x_0, y_0) + L \times \frac{LD}{\alpha} + L \frac{\varepsilon^2}{\alpha^2};
\]

since \( (x_0 = x_1) \), and where the last inequality holds by (64) since \( f \) is \( L \)-smooth.

But since \( \nabla_x f(x^\dagger, y^\dagger) = \nabla_y f(x^\dagger, y^\dagger) = 0 \), and \( f \) is \( L \)-smooth,

\[
f(x^\dagger, y^\dagger) - f(x_0, y_0) \leq L \|(x^\dagger, y^\dagger) - (x_0, y_0)\| \leq LD.
\]

(71)

Thus, plugging in (71) into (70), we get

\[
\psi(x_i) - \min_{\theta \in \mathbb{R}^d} \psi(\theta) = \max_{z \in \mathbb{R}^d} f(x_i, z) - \min_{\theta \in \mathbb{R}^d} \max_{z \in \mathbb{R}^d} f(\theta, z)
= \max_{z \in \mathbb{R}^d} f(x_i, z) - f(x^\dagger, y^\dagger)
\leq LD + \frac{L^2D}{\alpha} + L \frac{\varepsilon^2}{\alpha^2};
\]

(72)
Thus, since \( \nabla \) implies that
\[
\|x_i - x^i\| = \|x_i - \text{argmin}_{\theta \in \mathbb{R}^d} \psi(\theta)\| \leq \sqrt{L D + \frac{L^2 D}{\alpha} + L \frac{z^2}{\alpha^2}} \leq \mathfrak{D}.
\]

**Bounding the distance** \( \|y_{i,j} - y^i\| \).

Now, since \( \eta \leq \frac{1}{L} \), we have that \( f(x_i, y_{i,j}) \) is nondecreasing at each iteration \( j \) of Algorithm \( 2 \)
\[
f(x_i, y_{i,j+1}) \geq f(x_i, y_{i,j}) \quad \forall i \geq 0, j \geq 1.
\]

First we consider the case when \( i = 0 \). Since \( y_{1,0} = y_0 \), by (64) we have that
\[
\|y_{1,0} - \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)\| \leq \frac{L D}{\alpha}.
\]

Thus, \( f(x_0, \cdot) \) is \( L \)-smooth, (75) implies that
\[
\max_{z \in \mathbb{R}^d} f(x_0, z) - f(x_0, y_{1,0}) \leq L \left( \frac{\sqrt{L D}}{\alpha} \right)^2.
\]

Thus, by (74) we have that
\[
\max_{z \in \mathbb{R}^d} f(x_0, z) - f(x_0, y_{0,j}) \leq L \left( \frac{\sqrt{L D}}{\alpha} \right)^2 \quad \forall j \geq 1.
\]

Thus, since \( \nabla_y f(x_0, \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)) = 0 \), and since \( f(x_0, \cdot) \) is \( \alpha \)-strongly concave, we have that
\[
\|y_{i,j} - \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)\| \leq \sqrt{\frac{L}{\alpha} \left( \frac{\sqrt{L D}}{\alpha} \right)^2} = \frac{L \sqrt{D}}{\alpha \sqrt{\alpha}} \leq \mathfrak{D} \quad \forall j \geq 1.
\]

Next, we consider the case when \( i \geq 1 \). At each \( i \geq 1 \), we have that \( \|\nabla_y f(x_i, y_i)\| = \|\nabla_y f(x_i, y_{i,0})\| \leq \varepsilon \). Therefore, since \( f(x_i, \cdot) \) is \( \alpha \)-strongly concave, we have by Lemma \( 3.4 \) that
\[
\|y_{i,0} - \text{argmax}_{z \in \mathbb{R}^d} f(x_i, z)\| \leq \frac{\varepsilon}{\alpha}.
\]

Thus, \( f(x_i, \cdot) \) is \( L \)-smooth, (79) implies that
\[
\max_{z \in \mathbb{R}^d} f(x_i, z) - f(x_i, y_{i,0}) \leq L \left( \frac{\varepsilon}{\alpha} \right)^2.
\]

Thus by (74) we have that
\[
\max_{z \in \mathbb{R}^d} f(x_i, z) - f(x_i, y_{i,j}) \leq L \left( \frac{\varepsilon}{\alpha} \right)^2 \quad \forall j \geq 0.
\]

Thus, since \( \nabla_y f(x_0, \text{argmax}_{z \in \mathbb{R}^d} f(x_0, z)) = 0 \), and since \( f(x_0, \cdot) \) is \( \alpha \)-strongly concave, we have that
\[
\|y_{i,j} - \text{argmax}_{z \in \mathbb{R}^d} f(x_i, z)\| \leq \sqrt{\frac{L}{\alpha} \left( \frac{\varepsilon}{\alpha} \right)^2} = \frac{\varepsilon \sqrt{L}}{\alpha \sqrt{\alpha}} \leq \mathfrak{D} \quad \forall j \geq 1.
\]

Therefore, from (73), (77), and (82), and since \( y_i = y_{i,0} \) for every \( i \), we have that \( \|(x_i, y_i) - (x^i, y^i)\| \leq \mathfrak{D} \) and \( \|(x_i, y_{i,j}) - (x^i, y^i)\| \leq \mathfrak{D} \) for every \( i \geq 0 \) and every \( j \geq 0 \).
**Corollary B.10.** Suppose that $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is such that $f(\cdot, y)$ is $\alpha$-strongly convex for every $y \in \mathbb{R}^d$ and $f(x, \cdot)$ is $\alpha$-strongly concave for every $x \in \mathbb{R}^d$, and that $f$ has $L$-Lipschitz gradients for some $L \geq \alpha > 0$. And suppose that the proposal distribution $Q_{x,y}$ of Algorithm 1, with appropriate parameters, outputs a point $(x^\star, y^\star)$ which is an approximate global min-max point of $f$ with duality gap $\max_{y \in \mathbb{R}^d} f(x^\star, y) - \min_{x \in \mathbb{R}^d} f(x, y^\star) \leq \epsilon'$. Then, given any $\epsilon' > 0$ and any initial point $(x_0, y_0) \in \mathbb{R}^d \times \mathbb{R}^d$, Algorithm 1, with appropriate parameters, outputs a point $(x^\star, y^\star)$ which is an approximate global min-max point of $f$. Without loss of generality we may assume that $\epsilon' \leq L_1/\sqrt{2}$ and $\epsilon \leq \sqrt{\alpha/\epsilon'}$.

**Proof.** Set the parameters $\epsilon = \epsilon' = \frac{1}{10} \min \left( \frac{\alpha}{\sqrt{L}}, \frac{\eta}{2\sqrt{2}} \epsilon' \right)$, $\delta = \frac{1}{10} \frac{\alpha^2}{\sqrt{L}} \epsilon'$, and $\omega = \frac{1}{4}$.

Define $\mathfrak{D} := 2 \max \left( D + \frac{LD}{\alpha} + \frac{\epsilon}{\alpha}, \sqrt{\frac{LD + \epsilon^2}{\alpha} + \frac{L^2 \epsilon^2}{\alpha^2} \sqrt{\frac{L \sqrt{D}}{\alpha \sqrt{\alpha}}} + \frac{\epsilon \sqrt{D}}{\alpha \sqrt{\alpha}}} \right)$.

Define $b := 4L\mathfrak{D}$, and $L_1 := 2L\mathfrak{D}$.

Set hyperparameters $\tau_1 = \infty$ (so that the rejection probability in line 1 of Algorithm 1 is 1).

Set the remaining hyperparameters as in Items 1-8 in Appendix A with the parameter “$L$” in Items 1-8 replaced by $\min(L, \frac{L_1}{\sqrt{2}})$. Since $\eta \leq \frac{1}{10L}$, by Lemma B.9, we have that every step $i$ of Algorithm 1 and every step $j$ of its subroutine Algorithm 2 satisfy

$$\| (x_i, y_i) - (x^\star, y^\star) \| \leq \mathfrak{D} \quad \text{and} \quad \| (x_i, y_{i,j}) - (x^\star, y^\star) \| \leq \mathfrak{D} \quad \text{for all} \quad i \geq 0 \quad \text{and} \quad j \geq 0.$$ 

Thus, Algorithm 1 and its subroutine Algorithm 2 remain inside the ball $B((x^\star, y^\star), \mathfrak{D})$ of radius $\mathfrak{D}$ with center at the global min-max point $(x^\star, y^\star)$ of $f$.

Since $(x^\star, y^\star)$ is the global min-max point of $f$, we have that $\nabla_x f(x^\star, y^\star) = \nabla_y f(x^\star, y^\star) = 0$.

Without loss of generality we may assume that $f(x^\star, y^\star) = 0$ (we can assume this since each step of the algorithm remains the same if we add a constant to $f$).

Thus, since $f$ is $L$-smooth on all of $\mathbb{R}^d \times \mathbb{R}^d$, we have that

$$| f(x, y) | \leq L \times 4\mathfrak{D}^2 \quad \forall (x, y) \in B((x^\star, y^\star), 2\mathfrak{D})$$

and

$$\| (\nabla_x f(x, y), \nabla_y f(x, y)) \| \leq L \times 2\mathfrak{D} \quad \forall (x, y) \in B((x^\star, y^\star), 2\mathfrak{D}).$$

Since $f$ is $b$-bounded with $L_1$-Lipschitz gradient on the ball $B((x^\star, y^\star), 2\mathfrak{D})$, and since every step of the algorithm remains inside the ball $B((x^\star, y^\star), \mathfrak{D}) \subseteq B((x^\star, y^\star), 2\mathfrak{D})$, each step of the proof of Theorem 3.3 holds if we replace the parameter “$L$” in that proof with $\min(L, \frac{L_1}{\sqrt{2}})$ (since the parameter “$L$” in the proof of Theorem 3.3 is required to be $\leq \frac{L_1}{\sqrt{2}}$, and setting that parameter “$L$” to be $\min(L, \frac{L_1}{\sqrt{2}})$ ensures that this assumption holds).

Therefore, the conclusion of Theorem 3.3 must also hold and we have that Algorithm 1 returns a point $(x^\star, y^\star) \in \mathbb{R}^d \times \mathbb{R}^d$ such that, for some $\epsilon^\star \in [\frac{1}{2} \epsilon, \epsilon]$, $(x^\star, y^\star)$ is an $(\epsilon^\star, \delta, \omega, Q)$-equilibrium. The number of gradient and function evaluations required by the algorithm is $\text{poly}(b, \min(L, \frac{L_1}{\sqrt{2}}), \frac{1}{\sqrt{1/4}})$ and does not depend on the dimension $d$.

Note that, since we assume the gradients and proposal distribution are deterministic, each step of the algorithm is also deterministic, and the conclusion must hold with probability 1.

But $\frac{1}{2}, b, \frac{1}{\sqrt{2}} = \text{poly}(L, \frac{1}{\sqrt{2}}, \frac{1}{2}, D)$ and $\text{min}(L, \frac{L_1}{\sqrt{2}}) = \text{poly}(L, \frac{1}{\sqrt{2}}, \frac{1}{2}, D)$. Therefore, the number of gradient and function evaluations is also $\text{poly}(L, \frac{1}{\sqrt{2}}, \frac{1}{2}, D)$.

We have now shown that Algorithm 1 returns a point $(x^\star, y^\star)$ which is an $(\epsilon^\star, \delta, \omega, Q)$-equilibrium for $f$ where $\epsilon^\star \in [\frac{1}{2} \epsilon, \epsilon]$ (and in particular, $\epsilon^\star, \delta = \text{poly}(\epsilon', \alpha, \frac{1}{2})$).

Therefore, by Theorem B.2, we have that since $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is $\alpha$-strongly convex in $x$ and $\alpha$-strongly concave in $y$, with $L$-Lipschitz gradient in both variables, the point $(x^\star, y^\star)$, which is an $(\epsilon^\star, \delta, \omega, Q)$-equilibrium, also satisfies the duality gap.
\[
\max_{y \in \mathbb{R}^d} f(x^*, y) - \min_{x \in \mathbb{R}^d} f(x, y^*) \leq \frac{L(e^*)^2}{2n^2} + \frac{L^3}{\alpha^2} \left( \sqrt{\delta + L \left( \frac{e^*}{\alpha} + \frac{1}{\alpha} (L e^* + 2\sigma) \right)} + L \left( \frac{e^*}{2n^2} + \frac{L e^*}{\alpha} + \frac{L e^*}{\alpha} \right)^2 \right) 
\leq \epsilon'.
\]  
(85)

C. Examples of Functions Where Global Min-Max Satisfies Definition 3.2 but not Other Local Equilibrium Notions

In this section, we expand upon the examples mentioned in Section 3. In particular, we provide example functions for which there exists min-max points that satisfy Definition 3.2 but which do not satisfy other common notions of local equilibrium.

Functions for which global min-max points are not first-order stationary points. \( f(x, y) = \sin(x) \times \sin(y) - \sum_{m,n \in \mathbb{Z}} \operatorname{Bump}(x + m\pi, y + n\pi) \), where \( \operatorname{Bump}(x, y) := e^{-1/(1-100(x^2+y^2))} \) for \( x^2 + y^2 < \frac{1}{100} \) and \( \operatorname{Bump}(x, y) = 0 \) everywhere else. This function has a global min-max point at \( (x, y) = (0, 1) \) and this point also satisfies Definition 3.2 (and \( \epsilon \) also has such a point at all points along the line \( x = 0 \) except for the intervals \( (-\frac{1}{10} + n\pi, \frac{1}{10} + n\pi) \) for integers \( n \), and yet \( \nabla_x f(0, 1) = \cos(0) \times \sin(1) = 0.84 \) meaning that \( (x, y) = (0, 1) \) is not a first-order stationary point for \( x \). In fact, every global min-max point of this function is not a first-order stationary point in \( x \).

Functions for which global min-max points are not second-order equilibrium points. For \( f(x, y) = \sin(x + y) \) and \( f(x, y) = 10^3 \cdot \sum_{k \in \mathbb{Z}} e^{-(x+y+2+9k)^2} + 2e^{-(x+y+2+9k)^2} - e^{-(x+6k)^2} \), there are no \( \epsilon \)-approximate local min/max points for \( \epsilon < \frac{1}{2} \), and yet, an equilibrium point from Definition 3.2 is guaranteed to exist for such functions. Note that these functions are indeed smooth and bounded.

D. Comparison of Local Equilibrium Point and Local Min-Max Point

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

Since \( y^* \) is a local maximum of \( f(x^*, \cdot) \), there is only one \( \epsilon \)-greedy path with initial point \( y^* \), namely, the path \( \{y^*\} \) consisting of the single point \( y^* \) (since \( f \) must increase at rate at least \( \epsilon \) at every point on an \( \epsilon \)-greedy path).

Thus,
\[
P_\epsilon(x^*, y^*) = \{y^*\}
\]  
(86)

Hence, \( 86 \) implies that
\[
y^* \in \arg\max_{y \in P_\epsilon(x^*, y^*)} f(x^*, y)
\]  
(87)

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
\[
f(z, y^*) \geq f(x^*, y^*) \quad \forall z \in B(x^*, \nu)
\]  
(88)

Since \( y^* \in P_\epsilon(x, y^*) \) for all \( x \in \mathcal{X} \), we have that
\[
\max_{y \in P_\epsilon(x, y^*)} f(x, y) \geq f(x, y^*) \quad \forall x \in \mathcal{X},
\]  
(89)

and hence that
\[
\min_{x \in B(x^*, \nu) \cap \mathcal{X}} \max_{y \in P_\epsilon(x, y^*)} f(x, y) \geq \min_{x \in B(x^*, \nu^*)} f(x, y^*) \quad \text{Eq. 88}
\]
\[
\min_{x \in B(x^*, \nu) \cap \mathcal{X}} \max_{y \in P_\epsilon(x, y^*)} f(x, y) \geq \min_{x \in B(x^*, \nu^*)} f(x, y^*) \quad \text{Eq. 88}
\]
\[
\min_{x \in B(x^*, \nu) \cap \mathcal{X}} \max_{y \in P_\epsilon(x, y^*)} f(x, y) \geq \min_{x \in B(x^*, \nu^*)} f(x, y^*)
\]  
(90)
A Convergent and Dimension-Independent Min-Max Optimization Algorithm

Figure 6. Different runs of our algorithm over function $F_1$ for random starting points.

Figure 7. Different runs of our algorithm over function $F_2$ for random starting points.

\[
\max_{y \in P_\varepsilon(x^*, y^*)} f(x^*, y),
\]
which proves Equation (4).

**Corollary D.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 (91), for instance the distribution $Q_{x, y} \sim \mathcal{N}(0, \sigma^2 I_d)$ for $\sigma = O(\nu \log^{-1}(\frac{1}{\omega}))$.

**Proof.** By Inequality (91) in the proof of Lemma D.1, there exists $\nu > 0$ such that

\[
\min_{x \in B(x^*, \nu) \cap \mathcal{X}} \max_{y \in P_\varepsilon(x, y)} f(x, y) \geq \max_{y \in P_\varepsilon(x^*, y^*)} f(x^*, y),
\]
whence, for any proposal distribution $Q$ which satisfies Inequality (91), Inequality (92) implies that, for any $\delta \geq 0$,

\[
\Pr_{\Delta \sim Q_{x^*, y^*}} \left[ \max_{y \in P_\varepsilon(x^* + \Delta, y)} f(x^* + \Delta, y) < \max_{y \in P_\varepsilon(x^*, y^*)} f(x^*, y) - \delta \right]
\leq \Pr_{\Delta \sim Q_{x^*, y^*}} (\|\Delta\| \geq \nu)
\leq \omega.
\]

This proves Inequality (4).

Inequality (7) follows directly from Inequality (87) in the proof of Lemma D.1.
E. Additional Empirical Details and Results for Test Functions and Gaussian Mixture Dataset

E.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.

E.2. Additional Simulation Results for Low-Dimensional Test functions

We also run our algorithm for toy functions $F_1, F_2, F_3$ (defined in Section 4) on random initial points. The results are present in Figures 6, 7, 8 for functions $F_1, F_2, F_3$, respectively. For all starting points, our algorithm converges to global min-max point $(0, 0)$ for functions $F_1, F_3$, and diverges to $\infty$ for function $F_2$.

E.3. 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 [Metz et al., 2017] (github.com/poolio/unrolled_gan), 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 [Metz et al., 2017]: 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 [Metz et al., 2017], 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 (github.com/vsyrgkanis/optimistic_GAN_training/).

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.

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.
A Convergent and Dimension-Independent Min-Max Optimization Algorithm

GDA with 1 discriminator step

Figure 9. The generated points at the 1500'th iteration for all runs of GDA with $k = 1$ discriminator steps.

GDA with 6 discriminator steps

Figure 10. The generated points at the 1500'th iteration for all runs of the GDA algorithm, with $k = 6$ discriminator steps, for the simulation mentioned in Figure 3. At the 1500'th iteration, GDA had learned two modes 65% of the runs, one mode 20% of the runs, and four modes 15% of the runs.

E.4. 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 plot points from the generated distribution at iteration 1,500. Figure 9 gives the results for GDA with $k = 1$ discriminator step. Figure 10 gives the results for GDA with $k = 6$ discriminator steps. Figure 11 gives the results for the Unrolled GANs algorithm. Figure 12 gives the results for the OMD algorithm. Figure 13 gives the results for our algorithm.

F. Empirical Results for CIFAR-10 Dataset

This real-world dataset contains 60K color images from 10 classes. Previous works [Borji 2019, Metz et al., 2017, Srivastava et al. 2017] 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 [Salimans et al., 2018] 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.

Unrolled GANs with 6 unrolling steps

Figure 11. 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.
A Convergent and Dimension-Independent Min-Max Optimization Algorithm

OMD

Figure 12. The generated points at the 1500'th iteration for all runs of OMD algorithm.

Our algorithm

Figure 13. 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 \( \frac{1}{\tau} = \frac{1}{4} \). 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 14. 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.

Table 3. 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>5000</th>
<th>10000</th>
<th>25000</th>
<th>50000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>2.71 (0.28)</td>
<td>3.57 (0.26)</td>
<td>4.10 (0.35)</td>
<td><strong>4.68</strong> (0.39)</td>
</tr>
<tr>
<td>GDA</td>
<td>2.80 (0.52)</td>
<td>3.56 (0.64)</td>
<td>4.28 (0.77)</td>
<td>4.51 (0.86)</td>
</tr>
<tr>
<td>OMD</td>
<td>1.60 (0.18)</td>
<td>1.80 (0.37)</td>
<td>1.73 (0.25)</td>
<td>1.96 (0.26)</td>
</tr>
</tbody>
</table>
Figure 15. 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 16. 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 17. GAN trained using OMD. We repeated this simulation multiple times; here we display images generated from some of the resulting generators for OMD.
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 [Brownlee, 2019], 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 $4096$ parameters, followed by a leaky RELU layer, whose activations are reshaped into $246 \times 4 \times 4$ feature maps. The feature maps are then upsampled 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.

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.

Results for CIFAR-10. 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 3 reports the Inception Scores at iteration 5000, 10000, 25000, and 50000, while Figure 14 provides the complete plot for Inception Score vs. training iterations. Sample images from all three algorithms are also provided in Figures 15, 16, 17.

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 16, 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. 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 17).

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.

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 G.

G. Empirical Results for MNIST Dataset

This dataset consists of 60k images of hand-written digits (LeCun et al., 2010). We use two versions of this dataset: the full dataset and the dataset restricted to 0-1 digits.

3We could not replicate the performance of OMD reported in (Daskalakis et al., 2018), even with the implementation provided here - [https://github.com/vsyrgkanis/optimistic_GAN_training](https://github.com/vsyrgkanis/optimistic_GAN_training).
A Convergent and Dimension-Independent Min-Max Optimization Algorithm

Figure 18. Images generated at the 1000'th iteration of the 13 runs of the GDA simulation on the 01-MNIST dataset. In 77% of the runs the generator seems to be generating only 1’s at the 1000’th iteration.

Figure 19. Images generated at the 1000'th iteration of each of the 22 runs of our algorithm on the 01-MNIST dataset.

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 (Khandelwal, 2019) and Rowel Atienza (Atienza, 2017), 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 (Figures 4, 18, 19). 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 18 for images generated by all the runs of GDA, and Figure 19 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 $k = 1$ 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. 20 for generated images).
Figure 20. 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.

Figure 21. 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 \( e^{-\frac{1}{\tau_i}} = \frac{1}{4 + e^{(i/20000)^2}} \). 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 22 for results from the other four runs).

H. Randomized Acceptance Rule with Decreasing Temperature

In this section we give the simulations mentioned in the paragraph towards the beginning of Section 4, 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 21).

Figure 22. Images generated at the 39,000'th iteration of each of the 5 runs of our algorithm for the simulation mentioned in Figure 21 with a randomized acceptance rule with a temperature schedule of \( e^{-\frac{1}{\tau_i}} = \frac{1}{4 + e^{(i/20000)^2}} \).