TAMING GANS WITH LOOKAHEAD

Tatjana Chavdarova Matteo Pagliardini
Martin Jaggi Francois Fleuret

Idiap-RR-20-2020

SEPTEMBER 2020

Taming GANs with Lookahead

Tatjana Chavdarova[*] Idiap, EPFL

Mattéo Pagliardini[*] EPFL

Martin Jaggi EPFL

François Fleuret Idiap, EPFL

Abstract

Generative Adversarial Networks are notoriously challenging to train. The underlying minimax optimization is highly susceptible to the variance of the stochastic gradient and the rotational component of the associated game vector field. We empirically demonstrate the effectiveness of the Lookahead meta-optimization method for optimizing games, originally proposed for standard minimization. The backtracking step of Lookahead naturally handles the rotational game dynamics, which in turn enables the gradient ascent descent method to converge on challenging toy games often analyzed in the literature. Moreover, it implicitly handles high variance without using large mini-batches, known to be essential for reaching state of the art performance. Experimental results on MNIST, SVHN, and CIFAR-10, demonstrate a clear advantage of combining Lookahead with Adam or extragradient, in terms of performance, memory footprint, and improved stability. Using 30-fold fewer parameters and 16-fold smaller minibatches we outperform the reported performance of the class-dependent BigGAN on CIFAR-10 by obtaining FID of 13.65 *without* using the class labels, bringing state-of-the-art GAN training within reach of common computational resources.

1 Introduction

Gradient-based methods are the workhorse of machine learning. These methods optimize the parameters of a model with respect to a single objective $f : \mathcal{X} \to \mathbb{R}$. However, an increasing interest for multi-objective optimization arises in various domains–such as mathematics, economics, multi-agent reinforcement learning (Omidshafiei et al., 2017)–where several agents aim at optimizing their own cost function $f_i : \mathcal{X}_1 \times \cdots \times \mathcal{X}_N \to \mathbb{R}$ simultaneously. One particularly successful such class of algorithms are the Generative Adversarial Networks (Goodfellow et al., 2014, (GANs)), which consist of two players referred to as a generator and a discriminator. GANs were originally formulated as minimax optimization $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ (Von Neumann and Morgenstern, 1944), where the generator and the discriminator aim at minimizing and maximizing the same value function, see § 2. A natural generalization of gradient descent for minimax problems is the gradient descent ascent (GDA) algorithm, which alternates between a gradient descent step for the min-player and a gradient ascent step for the max-player. In the ideal case, minimax training aims at finding a Nash equilibrium where no player has the incentive of changing its parameters.

Despite the impressive quality of the generated samples that GANs have demonstrated–relative to classical maximum likelihood-based generative models, these algorithms remain *notoriously difficult to train*. In particular, poor performance (sometimes manifesting as "mode collapse"), brittle dependency on hyperparameters, or divergence are often reported. Consequently, obtaining state-of-the-art performance was shown to require large computational resources (Brock et al., 2019), making well-performing models unavailable for common budgets of computational resources.

To train GANs, practitioners originally adopted methods that are known to perform well on standard single-objective minimization. However, an understanding of the fundamental differences in terms

[*]equal contribution, correspondence to {firstname.lastname}@epfl.ch

Preprint. Under review.

of optimization as well as the stationary points of a game is currently missing (Jin et al., 2019). Moreover, it was recently empirically shown that: (i) GANs often converge to a locally stable stationary point that is *not* a differential Nash equilibrium (Berard et al., 2020); (ii) increased batch size improves GAN performances (Brock et al., 2019) in contrast to minimization (Defazio and Bottou, 2018; Shallue et al., 2018). A principal reason behind these differences is attributed to the *rotations* arising due to the adversarial component of the associated vector field of the gradient of the two player's parameters (Mescheder et al., 2018; Balduzzi et al., 2018), which are atypical for minimization. More precisely, the Jacobian of the associated vector field (see definition in § 2) can be decomposed into a symmetric and antisymmetric component (Balduzzi et al., 2018), which behave as a "potential" (Monderer and Shapley, 1996) and a Hamiltonian game. For our purposes, a "potential" game can be seen as standard minimization, where gradient converges in contrast to the Hamiltonian game where GDA exhibits *cyclic* behavior on the level sets of the Hamiltonian scalar function $\mathcal{H}$. While gradient and gradient descent on $\mathcal{H}$ converge on potential and Hamiltonian game, respectively, games are often combination of the two, making this general case hard to solve. In the context of minimization, Zhang et al. (2019) recently proposed the "Lookahead" algorithm, which intuitively uses an update direction by "looking ahead" at the sequence of parameters that change with higher variance due to stochastic gradient estimates–generated by some inner optimizer. Lookahead was shown to improve the stability during training and to reduce the variance of the so called "slow" weights.

Contributions. In this paper we investigate extensions of the Lookahead algorithm to minimax problems, and empirically benchmark their combination with currently used optimization methods for GANs. Our contributions can be summarized as follows:

- We extend the Lookahead algorithm to a meta-optimizer for minimax, called "lookahead–minimax", in a way that takes into account the rotational component of the associated vector field, yielding an algorithm that is straightforward to implement.
- We motivate the use of Lookahead for games by considering the extensively studied toy bilinear example (Goodfellow, 2016) and show that: (i) the use of lookahead allows for convergence of the otherwise diverging GDA on the classical bilinear game in full-batch setting (see § 3.1.1), as well as (ii) it yields good performance on challenging stochastic variants of this game, despite the high variance (see § 3.1.2).
- We empirically investigate the performance of lookahead on GANs on three standard datasets–MNIST, CIFAR-10, and SVHN as well as with standard optimization methods for GANs–GDA and Extragradient (both using Adam, Kingma and Ba, 2015), called LA–AltGAN and LA–ExtraGradient, respectively. We observe consistent performance and stability improvements at a negligible additional cost that does not require additional forward and backward passes, see § 4.
- Finally, we report a new state of the art result on CIFAR-10, while outperforming the class-conditional BigGAN (Brock et al., 2019) using 30 times smaller model and 16 times smaller minibatches on *unconditional* image generation (known to be harder than using the class labels) , by obtaining FID score of 13.65, see Table 2.

2 Background

GANs formulation. Given the data distribution p_d, the generator is a mapping $G : z \mapsto x$, where z is sampled from a known distribution $z \sim p_z$ and ideally $x \sim p_d$. The discriminator $D : x \mapsto D(x) \in [0, 1]$ is a binary classifier whose output represents a conditional probability estimate that an x sampled from a balanced mixture of real data from p_d and G-generated data is actually real.

The optimization of GAN is formulated as a differentiable two-player game where the generator G with parameters $\boldsymbol{\theta}$, and the discriminator D with parameters $\boldsymbol{\varphi}$, aim at minimizing their own cost function $\mathcal{L}^{\boldsymbol{\theta}}$ and $\mathcal{L}^{\boldsymbol{\varphi}}$, respectively, as follows:

$$\boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}\in\Theta} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}^\star) \qquad \text{and} \qquad \boldsymbol{\varphi}^\star \in \arg\min_{\boldsymbol{\varphi}\in\Phi} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}^\star, \boldsymbol{\varphi})\,. \tag{2P-G}$$

When $\mathcal{L}^{\boldsymbol{\varphi}} = -\mathcal{L}^{\boldsymbol{\theta}} =: \mathcal{L}$ this game is called a *zero-sum game* and (2P-G) is a minimax problem:

$$\min_{\boldsymbol{\theta}\in\Theta} \max_{\boldsymbol{\varphi}\in\Phi} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\varphi}) \tag{SP}$$

Minimax optimization methods. As GDA does *not* converge for some simple convex-concave game, Korpelevich (1976) proposed the *extragradient* method, where a "prediction" step is performed to obtain an extrapolated point $(\boldsymbol{\theta}_{t+\frac{1}{2}}, \boldsymbol{\varphi}_{t+\frac{1}{2}})$ using GDA, and the gradients at the *extrapolated* point are then applied to the current iterate $(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$ as follows:

$$\text{Extrapolation:} \begin{cases} \boldsymbol{\theta}_{t+\frac{1}{2}} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t) \\ \boldsymbol{\varphi}_{t+\frac{1}{2}} = \boldsymbol{\varphi}_t - \eta \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t) \end{cases} \text{Update:} \begin{cases} \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}_{t+\frac{1}{2}}, \boldsymbol{\varphi}_{t+\frac{1}{2}}) \\ \boldsymbol{\varphi}_{t+1} = \boldsymbol{\varphi}_t - \eta \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}_{t+\frac{1}{2}}, \boldsymbol{\varphi}_{t+\frac{1}{2}}) \end{cases}, \tag{EG}$$

where η denotes the step size. In the context of zero-sum games, the extragradient method converges for any *convex-concave* function $\mathcal{L}$ and any closed convex sets Θ and Φ, (see Harker and Pang, 1990, Thm. 12.1.11).

The joint vector field. Mescheder et al. (2017) and Balduzzi et al. (2018) argue that the vector field obtained by concatenating the gradients of the two players gives more insights of the dynamics than studying the loss surface. The joint vector field (JVF) and the Jacobian[2] of JVF are defined as:

$$v(\boldsymbol{\theta}, \boldsymbol{\varphi}) = \begin{pmatrix} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) \\ \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) \end{pmatrix}, \text{and} \quad v'(\boldsymbol{\theta}, \boldsymbol{\varphi}) = \begin{pmatrix} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) & \nabla_{\boldsymbol{\varphi}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) \\ \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) & \nabla^2_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}) \end{pmatrix}, \text{ resp.} \tag{JVF}$$

Rotational component of the game vector field. Berard et al. (2020) show empirically that GANs converge to a *locally stable stationary point* (Verhulst, 1990, LSSP) that is not a differential Nash equilibrium–defined as a point where the norm of the Jacobian is zero and where the Hessian of both the players are *definite positive*. LSSP is defined as a point $(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star)$ where:

$$v(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star) = 0, \quad \text{and} \quad \mathcal{R}(\lambda) > 0, \forall \lambda \in Sp(v'(\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star)), \tag{LSSP}$$

where $Sp(\cdot)$ denotes the spectrum of $v'(\cdot)$ and $\mathcal{R}(\cdot)$ the real part. In summary, (i) if all the eigenvalues of $v'(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$ have positive real part the point $(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$ is LSSP, and (ii) if the eigenvalues of $v'(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$ have imaginary part, the dynamics of the game exhibit rotations.

Impact of noise due to the stochastic gradient estimates on games. Chavdarova et al. (2019) point out that relative to minimization, noise impedes more the game optimization, and show that there exists a class of zero-sum games for which the *stochastic* extragradient method *diverges*. Intuitively, bounded noise of the stochastic gradient hurts the convergence as with higher probability the noisy gradient points in a direction that makes the algorithm to diverge from the equilibrium, due to the properties of $v'(\cdot)$ (see Fig.1, Chavdarova et al., 2019).

3 Lookahead for minimax objectives

Lookahead for single objective optimization. In the context of minization, Zhang et al. (2019) recently proposed the "Lookahead" algorithm where at every step t: (i) a copy of the current iterate $\boldsymbol{\omega}_t$ is made: $\boldsymbol{\omega}_t^{\mathcal{P}} \leftarrow \boldsymbol{\omega}_t$ (where $\mathcal{P}$ stands for "prediction"), (ii) $\boldsymbol{\omega}_t^{\mathcal{P}}$ is then updated for k times yielding $\boldsymbol{\omega}_{t+k}^{\mathcal{P}}$, and finally (iii) the actual update $\boldsymbol{\omega}_{t+1}$ is obtained as a *point that lies on a line between the two iterates*: the current $\boldsymbol{\omega}_t$ and the predicted one $\boldsymbol{\omega}_{t+k}^{\mathcal{P}}$:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t + \alpha(\boldsymbol{\omega}_{t+k}^{\mathcal{P}} - \boldsymbol{\omega}_t), \qquad \text{where } \alpha \in [0, 1]. \tag{LA}$$

Note that Lookahead uses two additional hyperparameters: (i) k–the number of steps to do prediction, as well as (ii) α–controls how large step we make towards the predicted iterate $\boldsymbol{\omega}^{\mathcal{P}}$: the larger the closest, and when $\alpha = 1$ (LA) is equivalent to regular optimization (has no impact). Besides the extra hyperparameters, Lookahead was shown to help the used optimizer to be more resilient to the choice of its hyperparameters, as well as to reduce the variance of the gradient estimates (Zhang et al., 2019).

Using lookahead, Zhang et al. (2019) were able to achieve faster convergence across different tasks, with minimal computational overhead. Recently, by viewing Lookahead as a multi-agent optimization with two agents, Wang et al. (2020) proved under certain assumptions that Lookahead converges to a first order stationary point.

[2]Note that in general $v(\cdot)$ is not a gradient vector field and unlike the Hessian, $v'(\cdot)$ is non-symmetric.

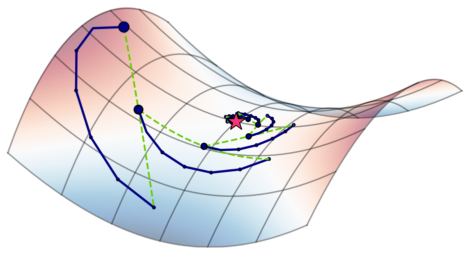


Figure 1: Illustration of Lookahead–minimax (Alg.1) with GDA on: $\min_x \max_y x \cdot y$, with α=0.5. The solution, trajectory $\{x_t, y_t\}_{t=1}^T$, and the lines between $(x^{\mathcal{P}}, y^{\mathcal{P}})$ and (x_t, y_t) are shown with red star, blue line, and dashed green line, resp. The backtracking step of Alg. 1 (lines 10 & 11) allows the otherwise non-converging GDA to converge.

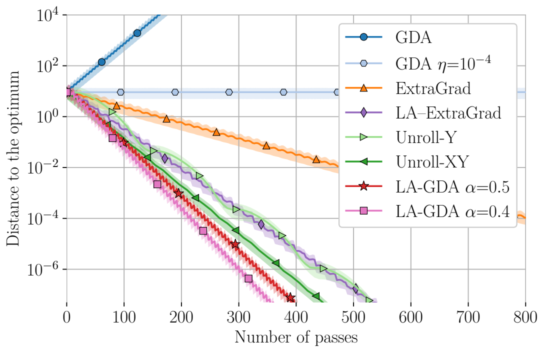


Figure 2: Distance to the optimum of (1) using different *full–batch* optimization methods, averaged over 5 runs. Unless otherwise specified, the learning rate is fixed to $\eta = 0.3$. See § 3.1.1 for details.

Lookahead–minimax. As games in the general case are a combination of "potential" (attraction) and Hamiltonian vector field, it is natural to consider the extension of Lookahead on games, as besides reducing the variance, taking a point on a line between two points on a cyclic trajectory would bring us closer to the solution, as illustrated in Fig. 1. Alg. 1 summarizes the proposed Lookahead–minimax algorithm. For the purpose of fair comparison, as a step we count each update of both the players, while covering the case of using different update ratio r for the two players–lines 4–6. To mitigate oscillations, we extend (LA) in joint parameter space $(\boldsymbol{\theta}, \boldsymbol{\varphi})$. More precisely, every k steps, given a previous stored checkpoint $(\boldsymbol{\theta}, \boldsymbol{\varphi})$ we perform a backtracking step using (LA), see lines 10 & 11.

Algorithm 1 Lookahead–Minimax pseudocode.

```
1:  Input: Stopping time $T$, learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\varphi}}$, initial weights $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, lookahead hyperparameters
    $k$ and $\alpha$, losses $\mathcal{L}^{\boldsymbol{\theta}}$, $\mathcal{L}^{\boldsymbol{\varphi}}$, update ratio $r$, real–data distribution $p_d$, noise–data distribution $p_z$.
2:  $(\boldsymbol{\theta}^{\mathcal{P}}, \boldsymbol{\varphi}^{\mathcal{P}}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\varphi})$                                   (store a copy)
3:  for $t \in 1, \ldots, T$ do
4:      for $i \in 1, \ldots, r$ do
5:          $\boldsymbol{x} \sim p_d$, $\boldsymbol{z} \sim p_z$
6:          $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} - \eta_{\boldsymbol{\varphi}} \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{x}, \boldsymbol{z})$            (update $\boldsymbol{\varphi}$ $r$ times)
7:      $\boldsymbol{z} \sim p_z$
8:      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{z})$                 (update $\boldsymbol{\theta}$ once)
9:      if $t\%k == 0$ then
10:         $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi}^{\mathcal{P}} + \alpha_{\boldsymbol{\varphi}}(\boldsymbol{\varphi} - \boldsymbol{\varphi}^{\mathcal{P}})$     (backtracking on interpolated line $\boldsymbol{\varphi}^{\mathcal{P}}$, $\boldsymbol{\varphi}$)
11:         $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{\mathcal{P}} + \alpha_{\boldsymbol{\theta}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathcal{P}})$     (backtracking on interpolated line $\boldsymbol{\theta}^{\mathcal{P}}$, $\boldsymbol{\theta}$)
12:         $(\boldsymbol{\theta}^{\mathcal{P}}, \boldsymbol{\varphi}^{\mathcal{P}}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\varphi})$                          (update checkpoints)
13: Output: $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$
```

Lookahead Vs. Lookahead–minimax. Note how Alg. 1 differs from applying Lookahead to both the players *separately*. The obvious difference is for the case $r \neq 1$, as the backtracking is done at different number of updates of $\boldsymbol{\varphi}$ and $\boldsymbol{\theta}$. The key difference is in fact that after applying (LA) to one of the players, we do *not* use the resulting interpolated point to update the parameters of the other player–a version we refer to as "Alternating–Lookahead", see § C. Instead, (LA) is applied to both the players *at the same time*, which we found that outperforms the former. Unless otherwise emphasized, we focus on the "joint" version, as described in Alg. 1.

3.1 Motivating example: the bilinear game

We argue that Lookahead-minimax allows for improved stability and performance on minimax problems due to two main reasons: (i) It allows for faster optimization in presence of a Hamiltonian vector field associated with minimax optimization; as well as (ii) it reduces the noise due to making more conservative steps.

In the following, we disentangle the two, and show in § 3.1.1 that Lookahead-minimax converges fast in the full-batch setting, without presence of noise. Moreover, besides that the GDA algorithm is known to diverge on this example, we show that when combined with lookahead it converges.

In § 3.1.2 we consider the challenging problem of (Chavdarova et al., 2019), specifically designed to have high variance of the gradient methods, and we show that besides therein proposed Stochastic Variance Reduced Extragradient (Chavdarova et al., 2019, SVRE), Lookahead-minimax is the only method that converges on this experiment, while considering all methods of (Gidel et al., 2019a, §7.1). More precisely, we consider the following bilinear problem:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d}\max_{\boldsymbol{\varphi}\in\mathbb{R}^d}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\varphi}) = \min_{\boldsymbol{\theta}\in\mathbb{R}^d}\max_{\boldsymbol{\varphi}\in\mathbb{R}^d}\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{\theta}^\top\boldsymbol{b}_i + \boldsymbol{\theta}^\top\boldsymbol{A}_i\boldsymbol{\varphi} + \boldsymbol{c}_i^\top\boldsymbol{\varphi}), \tag{1}$$

with $\boldsymbol{\theta},\boldsymbol{\varphi},\boldsymbol{b},\boldsymbol{c}\in\mathbb{R}^d$ and $\boldsymbol{A}\in\mathbb{R}^{n\times d\times d}$. We set $n=d=100$, and draw $[\boldsymbol{A}_i]_{kl}=\delta_{kli}$ and $[\boldsymbol{b}_i]_k,[\boldsymbol{c}_i]_k\sim\mathcal{N}(0,1/d)$, $1\leq k,l\leq d$, where $\delta_{kli}=1$ if $k=l=i$, and 0 otherwise.

3.1.1 The full-batch setting

In the batch setting each parameter update uses the full dataset. In Fig. 2 we compare: (i) *GDA* with learning rate $\eta=10^{-4}$ and $\eta=0.3$ (in blue), which oscilates arount the optimum with small enough learning rate, and diverges otherwise; (ii) *Unroll-Y* where the max-player is unrolled k steps, before updating the min player, as in (Metz et al., 2017); (iii) *Unroll-XY* where both the players are unrolled k steps with fixed opponent, and the actual updates are done with un unrolled opponent (see § A); (iv) *LA–GDA* with $\alpha=0.5$ and $\alpha=0.4$ (in red and pink, resp.) which combines Alg. 1 with GDA. (v) *ExtraGradient*–Eq. EG; as well as (vi) *LA–ExtraGrad*, which combines Alg. 1 with ExtraGradient. See § A for definition of all the algorithms used, as well as details on the implementation. Note that all algorithms are normalized by the number of passes, where as one pass we count a forward and backward pass. Interestingly, we observe that Lookahead–Minimax allows GDA to converge on this example, and moreover speeds up the convergence of ExtraGradient.

3.1.2 The stochastic setting

In this section, we show that besides SVRE, Lookahead–minimax also converges on (1). In addition, we test all the methods of (Gidel et al., 2019a, §7.1) using minibatches of several sizes $B=1,16,64$, and sampling without replacement. In particular, we tested: (i) the *Adam* method combined with GDA (shown in blue); (ii) *ExtraGradient*–Eq. EG; as well as (iii) *ExtraAdam* proposed by (Gidel et al., 2019a); (iv) our proposed method *LA-GDA* (Alg. 1) combined with GDA; as well as (v) *SVRE* (Chavdarova et al., 2019, Alg.1) for completeness. Fig. 3 depicts our results. See § A for details on the implementation and choice of hyperparameters. We observe that besides the good performance of LA-GDA on games in the batch setting, it also has the property to cope well large variance of the gradient estimates, and it converges *without* using restarting.

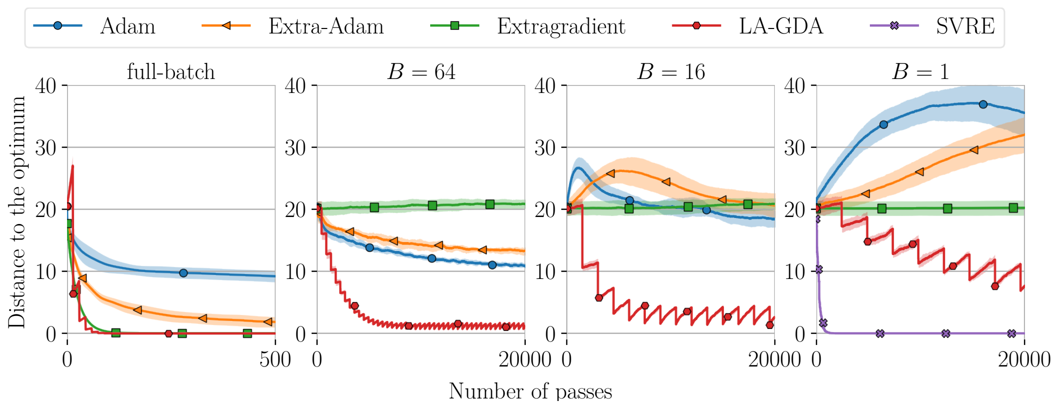


Figure 3: Convergence of *Adam*, *ExtraAdam*, *Extragradient*, *SVRE* and *LA-GDA*, on stochastic (1), for several minibatch sizes B. All methods are normalized by the number of passes (x-axis), where as one pass we count a forward and backward pass. Each experiment is averaged over 5 runs, where we randomly initialize both the initial point of the parameters, as well as the data points ($\boldsymbol{A}$, $\boldsymbol{b}$ and $\boldsymbol{c}$).

4 Experiments

In this section, we empirically benchmark Lookahead–minimax (Alg. 1) for *training GANs*.

4.1 Experimental setup

Datasets. For empirical comparison we used the following image datasets: (i) **MNIST** (Lecun and Cortes, 1998), (ii) **CIFAR-10** (Krizhevsky, 2009, §3), and (iii) **SVHN** (Netzer et al., 2011). using resolution of 28×28 for **MNIST**, and $3 \times 32 \times 32$ for the rest of the datasets.

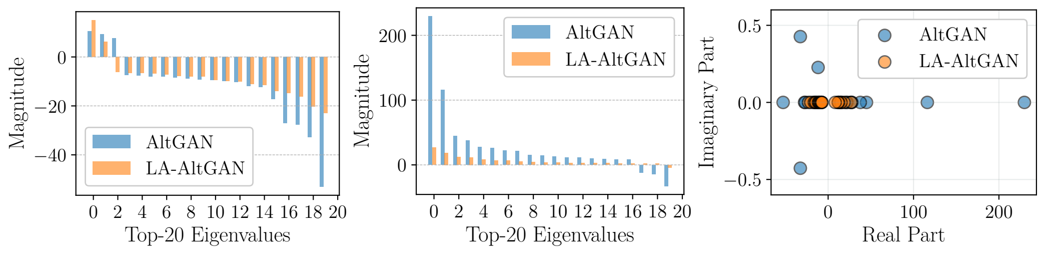
Metrics. We used the **Inception score** (IS, Salimans et al., 2016) and the **Fréchet Inception distance** (FID, Heusel et al., 2017) as most commonly used performance metrics for image synthesis. We used their respective original implementations, and sample size of 50000, see § D for details. We compute FID and IS at every 10000 iterations of each algorithm. See § D.1 for details.

DNN architectures. For experiments on **MNIST**, we used the DCGAN architectures (Radford et al., 2016), described in § D.2.1. For SVHN and CIFAR-10, we used the ResNet architectures, replicating the setup of (Miyato et al., 2018; Chavdarova et al., 2019), described in details in D.2.2.

Optimization methods. We conduct experiments using the following optimization methods for GANs: (i) **AltGan:** the standard alternating GAN, (ii) **ExtraGrad:** the extragradient method, as well as (iii) **UnrolledGAN:** proposed by Metz et al. (2017). We combine Lookahead-minimax with (i) and (ii), and we refer to these as **LA–AltGAN** and **LA–ExtraGrad**, respectively or for both as **LA–GAN** for brevity. *All methods use the Adam optimizer* (Kingma and Ba, 2015). We also compute Exponential Moving Average (EMA) and uniform averaging of the running iterates, see their definitions in § B.

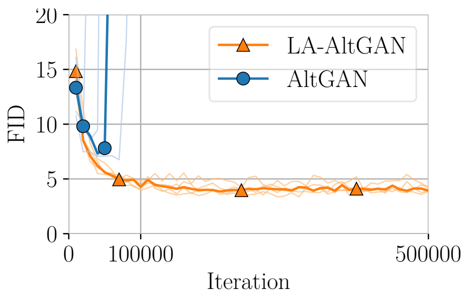
4.2 Results and Discussion

Comparison with baselines. Table 1 summarizes our comparison of combining Alg. 1 with AltGAN and ExtraGrad. On CIFAR-10, we observe that the iterates (column "no avg") of LA–AltGAN and LA–ExtraGrad obtain notably better performances than those of the corresponding baselines, and using EMA on LA–AltGAN and LA–ExtraGrad further improved the FID and IS scores obtained with LA–AltGAN. On SVHN, we observe similar relative comparisons between LA–GAN and baselines, but also–as opposed to CIFAR-10–we see that in some cases uniform averaging reduces performance and that EMA for AltGAN did not provide competitive results, as the iterates diverged relatively early. Although we obtain our best results with EMA, we see that the iterates of LA–GAN without averaging reach state-of-art results, and that the relative improvement of EMA is reduced compared to the baseline. Finally, we report our results on MNIST, where the training of all baselines is stable, to investigate if Lookahead-minimax allows for obtaining better *final* performances. Hence, we run each experiment for 100K iterations, despite that all methods converge earlier. The best FID scores of the iterates (column "no avg") are obtained with *LA–ExtraGrad*, and for EMA using *LA–ExtraGrad* and *Unrolled–GAN*. Note, however, that Unrolled–GAN is computationally much more expensive (in the order of the ratio $4:22$–as we used 20 steps of unrolling what gave best results, see § D). This confirms that in the absence of noise and for stable baselines, LA–GAN yields improvement for games. Fig. 4 depicts that after convergence LA–GAN shows no rotations.

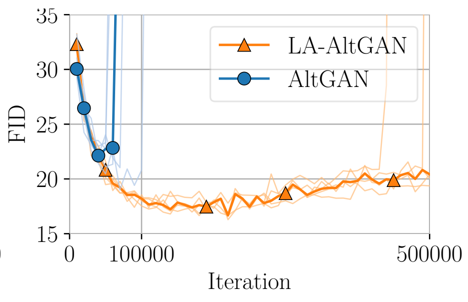


(a) Generator. (b) Discriminator. (c) Eigenvalues of $v'(\boldsymbol{\theta}, \boldsymbol{\varphi})$.

Figure 4: Analysis on MNIST at 100K iterations. Fig. 4a & 4b: Largest 20 eigenvalues of the Hessian of the generator and the discriminator. Fig. 4c: Eigenvalues of the Jacobian of JVF, indicating no rotations at the point of convergence of LA–AltGAN (see § 2).



(a) SVHN dataset.



(b) CIFAR-10 dataset.

Figure 5: Improved stability of LA–AltGAN relative to its AltGAN baseline on SVHN and CIFAR-10, over 5 runs. The median and the individual runs are illustrated with ticker solid lines and with transparent lines, respectively. See § 4.2 for discussion.

CIFAR-10	Fréchet Inception distance			Inception score		
Method	no avg	uniform avg	EMA	no avg	uniform avg	EMA
AltGAN	21.37 ± 1.60	19.25 ± 1.72	16.92 ± 1.16	$7.41 \pm .16$	$8.23 \pm .17$	$8.03 \pm .13$
LA–AltGAN	$16.74 \pm .46$	$15.02 \pm .81$	$\mathbf{14.40 \pm .48}$	$8.05 \pm .43$	$8.45 \pm .32$	$7.97 \pm .20$
ExtraGrad	$18.49 \pm .99$	16.22 ± 1.59	15.47 ± 1.82	$7.61 \pm .07$	$\mathbf{8.46 \pm .08}$	$\mathbf{8.05 \pm .09}$
LA–ExtraGrad	$\mathbf{15.25 \pm .30}$	$\mathbf{14.95 \pm .44}$	$14.68 \pm .30$	$\mathbf{8.42 \pm .26}$	$8.13 \pm .18$	$7.99 \pm .04$
Unrolled–GAN	21.04 ± 1.08	18.25 ± 1.60	17.51 ± 1.08	$7.43 \pm .07$	$8.26 \pm .15$	$7.88 \pm .12$
SVHN						
AltGAN	7.84 ± 1.21	10.83 ± 3.20	6.83 ± 2.88	$3.10 \pm .09$	$3.12 \pm .14$	$3.19 \pm .09$
LA–AltGAN	$3.87 \pm .09$	10.84 ± 1.04	$3.28 \pm .09$	$3.16 \pm .02$	$\mathbf{3.38 \pm .09}$	$\mathbf{3.22 \pm .08}$
ExtraGrad	$4.08 \pm .11$	8.89 ± 1.07	$3.22 \pm .09$	$\mathbf{3.21 \pm .02}$	$3.21 \pm .04$	$3.16 \pm .02$
LA–ExtraGrad	$\mathbf{3.20 \pm .09}$	$\mathbf{7.66 \pm 1.54}$	$\mathbf{3.16 \pm .14}$	$3.20 \pm .02$	$3.32 \pm .13$	$3.19 \pm .03$
MNIST						
AltGAN	$.094 \pm .006$	$\mathbf{.167 \pm .033}$	$.031 \pm .002$	$8.92 \pm .01$	$8.88 \pm .02$	$\mathbf{8.99 \pm .01}$
LA–AltGAN	$.058 \pm .003$	$.176 \pm .024$	$.031 \pm .002$	$\mathbf{8.93 \pm .01}$	$\mathbf{8.92 \pm .01}$	$8.96 \pm .02$
ExtraGrad	$.094 \pm .013$	$.182 \pm .024$	$.032 \pm .003$	$8.90 \pm .01$	$8.88 \pm .03$	$8.98 \pm .01$
LA–ExtraGrad	$\mathbf{.055 \pm .009}$	$.180 \pm .024$	$\mathbf{.030 \pm .002}$	$8.91 \pm .01$	$\mathbf{8.92 \pm .02}$	$8.95 \pm .01$
Unrolled–GAN	$.077 \pm .006$	$.224 \pm .016$	$\mathbf{.030 \pm .002}$	$8.91 \pm .02$	$8.91 \pm .02$	$\mathbf{8.99 \pm .01}$

Table 1: Comparison of the LA-GAN optimizer with its respective baselines AltGAN and ExtraGrad (see § 4.1 for naming), using FID (lower is better) and IS (higher is better). EMA denotes *exponential moving average* (with fixed $\beta = 0.9999$, see § B). We use the *Adam* (Kingma and Ba, 2015) optimizer for all the methods. Results are averaged over 5 runs. We run each experiment on MNIST for 100K iterations, and for 500K iterations for the rest of the datasets. See § D and § 4.2 for details on architectures and hyperparameters and for discussion on the results, resp. Our overall best obtained FID scores are 13.649 on CIFAR-10 and 2.823 on SVHN, see § E for samples of these generators.

Benchmark on CIFAR-10 using reported results. Table 2 summarizes the recently reported *best* obtained FID and IS scores on CIFAR-10. Although using the class labels–Conditional GAN is known to improve GAN performances (Radford et al., 2016), we outperform BigGAN (Brock et al., 2019) on CIFAR-10. Notably, our model and BigGAN have 5.1M and 158.3M parameters in total, respectively, and we use minibatch size of 128, whereas BigGAN uses 2048 samples.

Additional memory & computational cost. The additional memory cost of Lookahead-minimax is negligible as it only requires storing one copy per model ($\boldsymbol{\theta}^{\mathcal{P}}$ and $\boldsymbol{\varphi}^{\mathcal{P}}$ in Alg. 1). Note that EMA and uniform averaging have the same extra memory requirement–both of which are updated each step whereas LA–GAN is updated *once every* k iterations.

On the choice of α and k. In all our experiments we fixed $\alpha = 0.5$, and we tested with few values of k, while keeping k fixed throughout the training. We observe that all values of k improve upon the baseline, both in terms of stability and performance. We also observed that having smaller value of k makes AltGAN more stable, as if k is large, the algorithm quickly diverges as it becomes similar to AltGAN. On the other hand, when combining Lookahead-minimax with ExtraGradient we could use larger k as ExtraGradient is more stable, and usually diverges later then AltGAN.

	Unconditional GANs						Conditional GANs	
	SNGAN Miyato et al.	Prog.GAN Karras et al.	NCSN Song and Ermon	WS-SVRE Chavdarova et al.	ExtraAdam Gidel et al.	LA-AltGAN (ours)	SNGAN Miyato et al.	BigGAN Brock et al.
FID	21.7	–	25.32	16.77	$16.78 \pm .21$	**13.65**	25.5	14.73
IS	8.22	$8.80 \pm .05$	8.87	–	$8.47 \pm .10$	8.78	8.60	**9.22**

Table 2: Summary of the most recent competitive state-of-art reported best scores on CIFAR-10 and benchmark with LA–GAN, *using published results*. Note that the architectures used are *not* identical for all the methods, see § 4.2.

Stability of convergence. We observe that LA–GAN consistently improved the stability of its respective baseline, see Fig. 5. Although we used update ratio of $1:5$ for $G:D$–known to improve stability, our baselines diverged in all the experiments, whereas only few LA–GAN experiments diverged (later in the training relative to the baseline), see additional results in § E.

5 Related work

Parameter averaging. In the context of convex single-objective optimization, taking an arithmetic average of the parameters as by Polyak and Juditsky (1992); Ruppert (1988) is well-known to yield faster convergence for convex functions and allowing the use of larger constant step-sizes in the case of stochastic optimization (Dieuleveut et al., 2017). Parameter averaging has recently gained more interest in deep learning in general (Garipov et al., 2018), in natural language processing (Merity et al., 2018), and particularly in GANs (Yazıcı et al., 2019) where researchers often report the performance of a running uniform or exponential moving average of the iterates. Such averaging as a post-processing after training is fundamentally different from immediately applying averages during training. Lookahead (Zhang et al., 2019) as of our interest here in spirit is closer to extrapolation methods (Korpelevich, 1976) which rely on gradients taken not at the current iterate but at an extrapolated point for the current trajectory. For highly complex optimization landscapes such as in deep learning, the effect of using gradients at perturbations of the current iterate has a desirable smoothing effect which is known to help training speed and stability in the case of non-convex single-objective optimization (Wen et al., 2018; Haruki et al., 2019)

GANs. Several proposed methods for GANs are motivated by the "recurrent dynamics". For example, (i) Gidel et al. (2019a) and Yadav et al. (2018) use prediction steps to stabilize GANs, (ii) Metz et al. (2017) update the generator using "unrolled" version of the discriminator, (iii) Daskalakis et al. (2018) propose *Optimistic Mirror Decent* (OMD) for training Wasserstein GANs (Arjovsky et al., 2017), (iv) Balduzzi et al. (2018) propose the *Symplectic Gradient Adjustment* (SGA), and (v) Chavdarova et al. (2019) propose the *SVRE* method, which combines the extragradient method with stochastic variance reduced gradient (SVRG, Johnson and Zhang, 2013). Besides its simplicity, the key benefit of LA–GAN is that it handles well *both* the rotations of the vector field as well as noise from stochasticity, thus performing well on real–world applications.

6 Conclusion

Motivated by the adversarial component of games and the negative impact of noise on games, we proposed and extension of the Lookahead algorithm to games, called "Lookahead–minimax". On the bilinear toy example we observe that combining Lookahead–minimax with standard gradient methods converges, and that Lookahead–minimax handles well high variance of the gradient estimates.

Exponential moving averaging of the iterates is known to help obtain improved performances for GANs, yet it does not impact the actual iterates, hence does not stop the algorithm from (early) divergence. Lookahead–minimax goes beyond such averaging, requires less computation than running averages, and it is *straightforward to implement*. It can be applied to any optimization method, and in practice it *consistently* improves the stability of its respective baseline. Performance-wise, using Lookahead–minimax we obtained new state–of–art result on CIFAR–10 of 13.65 FID, outperforming BigGAN which uses the annotated classes, and requires 30–times larger models.

As Lookahead–minimax uses two additional hyperparameters, future directions include developing adaptive schemes of obtaining these coefficients throughout training, which could speed up further the convergence of Lookahead–minimax.

Broader Impact

By simplifying the training process and lowering its computational requirements, we hope this work will stimulate the research in new fields of applications for GANs and minimax problems in general. Furthermore, we hope it will bridge the gap between obtaining well-performing GANs using large computational resources and common setups of limited computation.

By improving GANs performances, inevitably we are also increasing the potential for misuses of this technology, those we already know today such as fake impersonation through video and audio generation, and those still to come that we need to prevent. As a mitigation strategy, we believe the scientific community should promote better education in the matter, more accountable information streams, and more adequate laws. We are hopeful that new arising challenges will create new research areas (e.g. ASVspoof challenge) helping to restore balance.

Acknowledgments and Disclosure of Funding

TC was funded in part by the grant 200021-169112 from the Swiss National Science Foundation, and MP was funded by the grant 40516.1 from the Swiss Innovation Agency. TC would like to thank Hugo Berard for insightful discussions on the optimization landscape of GANs and sharing their code, as well as David Balduzzi for insightful discussions on n-player differentiable games.

References

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel. The mechanics of n-player differentiable games. In *ICML*, 2018.

H. Berard, G. Gidel, A. Almahairi, P. Vincent, and S. Lacoste-Julien. A closer look at the optimization landscapes of generative adversarial networks. In *ICLR*, 2020.

A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

R. E. Bruck. On the weak convergence of an ergodic iteration for the solution of variational inequalities for monotone operators in hilbert space. *Journal of Mathematical Analysis and Applications*, 1977.

T. Chavdarova, G. Gidel, F. Fleuret, and S. Lacoste-Julien. Reducing noise in gan training with variance reduced extragradient. In *NeurIPS*, 2019.

C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng. Training GANs with optimism. In *ICLR*, 2018.

A. Defazio and L. Bottou. On the ineffectiveness of variance reduced optimization for deep learning. *arXiv:1812.04529*, 2018.

A. Dieuleveut, A. Durmus, and F. Bach. Bridging the gap between constant step size stochastic gradient descent and markov chains. *arXiv:1707.06386*, 2017.

T. Garipov, P. Izmailov, D. Podoprikhin, D. Vetrov, and A. G. Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *arXiv:1802.10026*, 2018.

G. Gidel, H. Berard, P. Vincent, and S. Lacoste-Julien. A variational inequality perspective on generative adversarial nets. In *ICLR*, 2019a.

G. Gidel, R. A. Hemmat, M. Pezeshki, R. L. Priol, G. Huang, S. Lacoste-Julien, and I. Mitliagkas. Negative momentum for improved game dynamics. In *AISTATS*, 2019b.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv:1701.00160*, 2016.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

P. T. Harker and J.-S. Pang. Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications. *Mathematical programming*, 1990.

K. Haruki, T. Suzuki, Y. Hamakawa, T. Toda, R. Sakai, M. Ozawa, and M. Kimura. Gradient Noise Convolution (GNC): Smoothing Loss Function for Distributed Large-Batch SGD. *arXiv:1906.10822*, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

C. Jin, P. Netrapalli, and M. I. Jordan. Minmax optimization: Stable limit points of gradient descent ascent are locally optimal. *arXiv:1902.00618*, 2019.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, 2013.

T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*. OpenReview.net, 2018.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

G. Korpelevich. The extragradient method for finding saddle points and other problems. *Matecon*, 1976.

A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, 2009.

Y. Lecun and C. Cortes. The MNIST database of handwritten digits. 1998. URL `http://yann.lecun.com/exdb/mnist/`.

S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing LSTM language models. In *ICLR*, 2018.

L. Mescheder, S. Nowozin, and A. Geiger. The numerics of GANs. In *NIPS*, 2017.

L. Mescheder, A. Geiger, and S. Nowozin. Which Training Methods for GANs do actually Converge? In *ICML*, 2018.

L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In *ICLR*, 2017.

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

D. Monderer and L. S. Shapley. Potential games. *Games and economic behavior*, 1996.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011. URL `http://ufldl.stanford.edu/housenumbers/`.

S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *ICML*, 2017.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 1992. doi: 10.1137/0330046.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

D. Ruppert. Efficient estimations from a slowly convergent robbins-monro process. *Technical report, Cornell University Operations Research and Industrial Engineering*, 1988.

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In *NIPS*, 2016.

C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training. *arXiv:1811.03600*, 2018.

Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, pages 11895–11907, 2019.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *arXiv:1512.00567*, 2015.

F. Verhulst. *Nonlinear Differential Equations and Dynamical Systems*. 1990. ISBN 0387506284.

J. Von Neumann and O. Morgenstern. *Theory of games and economic behavior.* Princeton University Press, 1944.

J. Wang, V. Tantia, N. Ballas, and M. Rabbat. Lookahead converges to stationary points of smooth non-convex functions. In *ICASSP*, 2020.

W. Wen, Y. Wang, F. Yan, C. Xu, C. Wu, Y. Chen, and H. Li. Smoothout: Smoothing out sharp minima to improve generalization in deep learning. *arXiv:1805.07898*, 2018.

A. Yadav, S. Shah, Z. Xu, D. Jacobs, and T. Goldstein. Stabilizing adversarial nets with prediction methods. 2018.

Y. Yazıcı, C.-S. Foo, S. Winkler, K.-H. Yap, G. Piliouras, and V. Chandrasekhar. The unusual effectiveness of averaging in gan training. In *ICLR*, 2019.

M. Zhang, J. Lucas, J. Ba, and G. E. Hinton. Lookahead optimizer: k steps forward, 1 step back. In *NeurIPS*. 2019.

A Experiments on the bilinear example

In this section we list the details regarding our implementation of the experiments on the bilinear example of (1) that were presented in § 3.1. In particular: (i) in § A.1 we list the implementational details of the benchmarked algorithms, (ii) in § A.2 and A.3 we list the hyperparameters used in § 3.1.1 and § 3.1.2, respectively and finally (iii) in § A.4 we present visualizations in aim to improve the reader's intuition on how Lookahead-minimax works on games.

A.1 Implementation details

Gradient Descent Ascent (GDA). We use an alternating implementation of GDA where the players are updated *sequentially*, as follows:

$$\varphi_{t+1} = \varphi_t - \eta \nabla_{\varphi} \mathcal{L}(\boldsymbol{\theta}_t, \varphi_t), \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t, \varphi_{t+1}) \tag{GDA}$$

ExtraGrad. Our implementation of extragradient follows (EG), with $\mathcal{L}^{\boldsymbol{\theta}}(\cdot) = -\mathcal{L}^{\varphi}(\cdot)$, thus:

$$\text{Extrapolation:} \begin{cases} \boldsymbol{\theta}_{t+\frac{1}{2}} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t, \varphi_t) \\ \varphi_{t+\frac{1}{2}} = \varphi_t + \eta \nabla_{\varphi} \mathcal{L}(\boldsymbol{\theta}_t, \varphi_t) \end{cases} \text{Update:} \begin{cases} \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_{t+\frac{1}{2}}, \varphi_{t+\frac{1}{2}}) \\ \varphi_{t+1} = \varphi_t + \eta \nabla_{\varphi} \mathcal{L}(\boldsymbol{\theta}_{t+\frac{1}{2}}, \varphi_{t+\frac{1}{2}}) \end{cases}. \tag{EG–ZS}$$

Unroll-Y. Unrolling was introduced by Metz et al. (2017) as a way to mitigate mode collapse of GANs. It consists of finding an optimal max–player $\varphi^\star$ for a fixed min–player $\boldsymbol{\theta}$, *i.e.* $\varphi^\star(\boldsymbol{\theta}) = \arg\max_{\varphi} \mathcal{L}^{\varphi}(\boldsymbol{\theta}, \varphi)$ through "unrolling" as follows:

$$\varphi_t^0 = \varphi_t, \qquad \varphi_t^{m+1}(\boldsymbol{\theta}) = \varphi_t^m - \eta \nabla_{\varphi} \mathcal{L}^{\varphi}(\boldsymbol{\theta}_t, \varphi_t^m), \qquad \varphi_t^\star(\boldsymbol{\theta}_t) = \lim_{m \to \infty} \varphi_t^m(\boldsymbol{\theta}) .$$

In practice m is a finite number of unrolling steps, yielding φ_t^m. The min–player $\boldsymbol{\theta}_t$, *e.g.* the generator, can be updated using the unrolled φ_t^m, while the update of φ_t is unchanged:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}_t, \varphi_t^m), \qquad \varphi_{t+1} = \varphi_t - \eta \nabla_{\varphi} \mathcal{L}^{\varphi}(\boldsymbol{\theta}_t, \varphi_t) \tag{UR–X}$$

Unroll-XY. While Metz et al. (2017) only unroll one player (the discriminator in their GAN setup), we extended the concept of unrolling to games and for completeness also considered unrolling *both* players. For the bilinear experiment we also have that $\mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}_t, \varphi_t) = -\mathcal{L}^{\varphi}(\boldsymbol{\theta}_t, \varphi_t)$.

Adam. Adam (Kingma and Ba, 2015) computes an exponentially decaying average of both past gradients m_t and squared gradients v_t, for each parameter of the model as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 , \tag{3}$$

where the hyperparameters $\beta_1, \beta_2 \in [0, 1]$, $m_0 = 0$, $v_0 = 0$, and t denotes the iteration $t = 1, \ldots T$. m_t and v_t are respectively the estimates of the first and the second moments of the stochastic gradient. To compensate the bias toward 0 due to their initialization to $m_0 = 0$, $v_0 = 0$, Kingma and Ba (2015) propose to use bias-corrected estimates of these first two moments:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} . \tag{5}$$

Finally, the Adam update rule for all parameters at t-th iteration $\boldsymbol{\omega}_t$ can be described as:

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \eta \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} . \tag{Adam}$$

Algorithm 2 Pseudocode for Restarted SVRE.

```
1:  Input: Stopping time $T$, learning rates $\eta_{\boldsymbol\theta}, \eta_{\boldsymbol\varphi}$, losses $\mathcal{L}^{\boldsymbol\theta}$ and $\mathcal{L}^{\boldsymbol\varphi}$, probability of restart $p$, dataset $\mathcal{D}$,
    noise dataset $\mathcal{Z}$, with $|\mathcal{D}| = |\mathcal{Z}| = n$.
2:  Initialize: $\boldsymbol\varphi$, $\boldsymbol\theta$, $t = 0$                          ▷ $t$ is for the online average computation.
3:  for $e = 0$ to $T-1$ do
4:      Draw restart $\sim \mathrm{B}(p)$.                                  ▷ Check if we restart the algorithm.
5:      if restart and $e > 0$ then
6:          $\boldsymbol\varphi \leftarrow \bar{\boldsymbol\varphi}$,  $\boldsymbol\theta \leftarrow \bar{\boldsymbol\theta}$ and $t = 1$
7:      $\boldsymbol\varphi^{\mathcal{S}} \leftarrow \boldsymbol\varphi$ and $\boldsymbol\mu_{\boldsymbol\varphi}^{\mathcal{S}} \leftarrow \frac{1}{|\mathcal{D}|}\sum_{i=1}^{n} \nabla_{\boldsymbol\varphi}\mathcal{L}_i^{\boldsymbol\varphi}(\boldsymbol\theta, \boldsymbol\varphi^{\mathcal{S}})$
8:      $\boldsymbol\theta^{\mathcal{S}} \leftarrow \boldsymbol\theta$ and $\boldsymbol\mu_{\boldsymbol\theta}^{\mathcal{S}} \leftarrow \frac{1}{|\mathcal{Z}|}\sum_{i=1}^{n} \nabla_{\boldsymbol\theta}\mathcal{L}_i^{\boldsymbol\theta}(\boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}})$
9:      $N \sim \mathrm{Geom}\big(1/n\big)$                                            ▷ Length of the epoch.
10:     for $i = 0$ to $N-1$ do
11:         Sample $i_{\boldsymbol\theta} \sim \pi_{\boldsymbol\theta}$, $i_{\boldsymbol\varphi} \sim \pi_{\boldsymbol\varphi}$, do extrapolation:
12:         $\tilde{\boldsymbol\varphi} \leftarrow \boldsymbol\varphi - \eta_{\boldsymbol\theta} \boldsymbol d_{\boldsymbol\varphi}(\boldsymbol\theta, \boldsymbol\varphi, \boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}})$ , $\tilde{\boldsymbol\theta} \leftarrow \boldsymbol\theta - \eta_{\boldsymbol\varphi} \boldsymbol d_{\boldsymbol\theta}(\boldsymbol\theta, \boldsymbol\varphi, \boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}})$      ▷ (6) and (7)
13:         Sample $i_{\boldsymbol\theta} \sim \pi_{\boldsymbol\theta}$, $i_{\boldsymbol\varphi} \sim \pi_{\boldsymbol\varphi}$, do update:
14:         $\boldsymbol\varphi \leftarrow \boldsymbol\varphi - \eta_{\boldsymbol\theta} \boldsymbol d_{\boldsymbol\varphi}(\tilde{\boldsymbol\theta}, \tilde{\boldsymbol\varphi}, \boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}})$ , $\boldsymbol\theta \leftarrow \boldsymbol\theta - \eta_{\boldsymbol\varphi} \boldsymbol d_{\boldsymbol\theta}(\tilde{\boldsymbol\theta}, \tilde{\boldsymbol\varphi}, \boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}})$      ▷ (6) and (7)
15:         $\bar{\boldsymbol\theta} \leftarrow \frac{t}{t+1}\bar{\boldsymbol\theta} + \frac{1}{t+1}\boldsymbol\theta$ and $\bar{\boldsymbol\varphi} \leftarrow \frac{t}{t+1}\bar{\boldsymbol\varphi} + \frac{1}{t+1}\boldsymbol\varphi$      ▷ Online computation of the average.
16:         $t \leftarrow t + 1$                          ▷ Increment $t$ for the online average computation.
17: Output: $\boldsymbol\theta, \boldsymbol\varphi$
```

Extra-Adam. Gidel et al. (2019a) adjust Adam for extragradient (EG) and obtain the empirically motivated *ExtraAdam* which re-uses the same running averages of (Adam) when computing the extrapolated point $\boldsymbol\omega_{t+\frac{1}{2}}$ as well as when computing the new iterate $\boldsymbol\omega_{t+1}$ (see Alg.4, Gidel et al., 2019a). We used the provided implementation by the authors.

SVRE. Chavdarova et al. (2019) propose SVRE as a way to cope with variance in games that may cause divergence otherwise. We used the restarted version of SVRE as used for the problem of (1) described in (Alg3, Chavdarova et al., 2019), which we describe in Alg. 2 for completeness–where $d_{\boldsymbol\theta}$ and $d_{\boldsymbol\varphi}$ denote "variance corrected" gradient:

$$\boldsymbol d_{\boldsymbol\varphi}(\boldsymbol\theta, \boldsymbol\varphi, \boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}}) := \boldsymbol\mu_{\boldsymbol\varphi} + \nabla_{\boldsymbol\varphi}\mathcal{L}^{\boldsymbol\varphi}(\boldsymbol\theta, \boldsymbol\varphi, \mathcal{D}[n_d], \mathcal{Z}[n_z]) - \nabla_{\boldsymbol\varphi}\mathcal{L}^{\boldsymbol\varphi}(\boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}}, \mathcal{D}[n_d], \mathcal{Z}[n_z]) \tag{6}$$

$$\boldsymbol d_{\boldsymbol\theta}(\boldsymbol\theta, \boldsymbol\varphi, \boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}}) := \boldsymbol\mu_{\boldsymbol\theta} + \nabla_{\boldsymbol\theta}\mathcal{L}^{\boldsymbol\theta}(\boldsymbol\theta, \boldsymbol\varphi, \mathcal{Z}[n_z]) - \nabla_{\boldsymbol\theta}\mathcal{L}^{\boldsymbol\theta}(\boldsymbol\theta^{\mathcal{S}}, \boldsymbol\varphi^{\mathcal{S}}, \mathcal{Z}[n_z])\,, \tag{7}$$

where $\boldsymbol\theta^{\mathcal{S}}$ and $\boldsymbol\varphi^{\mathcal{S}}$ are the snapshots and $\boldsymbol\mu_{\boldsymbol\theta}$ and $\boldsymbol\mu_{\boldsymbol\varphi}$ their respective gradients. $\mathcal{D}$ and $\mathcal{Z}$ denote the finite data and noise datasets. With a probability p (fixed) before the computation of $\boldsymbol\mu_{\boldsymbol\varphi}^{\mathcal{S}}$ and $\boldsymbol\mu_{\boldsymbol\theta}^{\mathcal{S}}$, we decide whether to restart SVRE (by using the averaged iterate as the new starting point–Alg. 2, Line 6–$\bar{\boldsymbol\omega}_t$) or computing the batch snapshot at a point $\boldsymbol\omega_t$. For consistency, we used the provided implementation by the authors.

A.2 Hyperparameters used for the full-batch setting

Optimal α. In the full-batch bilinear problem, it is possible to derive the optimal α parameter for a small enough η. Given the optimum $\boldsymbol\omega^\star$, the current iterate $\boldsymbol\omega$, and the "previous" iterate $\boldsymbol\omega^{\mathcal{P}}$ before k steps, let $\boldsymbol x = \boldsymbol\omega^{\mathcal{P}} + \alpha(\boldsymbol\omega - \boldsymbol\omega^{\mathcal{P}})$ be the next iterate selected to be on the interpolated line between $\boldsymbol\omega^{\mathcal{P}}$ and $\boldsymbol\omega$. We aim at finding $\boldsymbol x$ (or in effect α) that is closest to $\boldsymbol\omega^\star$. For an infinitesimally small learning rate, a GDA iterate would revolve around $\boldsymbol\omega^\star$, hence $\|\boldsymbol\omega - \boldsymbol\omega^\star\| = \|\boldsymbol\omega^{\mathcal{P}} - \boldsymbol\omega^\star\| = r$. The shortest distance between $\boldsymbol x$ and $\boldsymbol\omega^\star$ would be according to:

$$r^2 = \|\boldsymbol{\omega}^{\mathcal{P}} - \boldsymbol{x}\|^2 + \|\boldsymbol{x} - \boldsymbol{\omega}^\star\|^2 = \|\boldsymbol{\omega} - \boldsymbol{x}\|^2 + \|\boldsymbol{x} - \boldsymbol{\omega}^\star\|^2$$

Hence the optimal $\boldsymbol{x}$, for any k, would be obtained for $\|\boldsymbol{\omega} - \boldsymbol{x}\| = \|\boldsymbol{\omega}^{\mathcal{P}} - \boldsymbol{x}\|$, which is given for $\alpha = 0.5$.

In the case of larger learning rate, for which the GDA iterates diverge, we would have $\|\boldsymbol{\omega}^{\mathcal{P}} - \boldsymbol{\omega}^\star\| = r_1 < \|\boldsymbol{\omega} - \boldsymbol{\omega}^\star\| = r_2$ as we are diverging. Hence the optimal $\boldsymbol{x}$ would follow $\|\boldsymbol{\omega}^{\mathcal{P}} - \boldsymbol{x}\| < \|\boldsymbol{\omega} - \boldsymbol{x}\|$, which is given for $\alpha < 0.5$. In Fig. 2 we indeed observe LA-GDA with $\alpha = 0.4$ converging faster than with $\alpha = 0.5$.

Hyperparameters. Unless otherwise specified the learning rate used is fixed to $\eta = 0.3$. For both Unroll-Y and Unroll-XY, we use 6 unrolling steps. When combining Lookahead-minimax with GDA or Extragradient, we use a k of 6 and and α of 0.5 unless otherwise emphasized.

A.3 Hyperparameters used for the stochastic setting

The hyperparameters used in the stochastic bilinear experiment of (1) are listed in Table 3. We tuned the hyperparameters of each method independently, for each batch-size. We tried η ranging from 0.005 to 1. When for all values of η the method diverges, we set $\eta = 0.005$ in Fig. 3. To tune the first moment estimate of Adam β_1, we consider values ranging from -1 to 1, as Gidel et al. reported that negative β_1 can help in practice. We used $\alpha \in \{0.3, 0.5\}$ and $k \in [5, 3000]$.

Batch-size	Parameter	Adam	Extra-Adam	Extragradient	LA-GDA	SVRE
full-batch	η	0.005	0.02	0.8	0.2	-
	Adam β_1	-0.9	-0.6	-	-	-
	Lookahead k	-	-	-	15	-
	Lookahead α	-	-	-	0.3	-
64	η	0.005	0.01	0.005	0.005	-
	Adam β_1	-0.6	-0.2	-	-	-
	Lookahead k	-	-	-	450	-
	Lookahead α	-	-	-	0.3	-
16	η	0.005	0.005	0.005	0.01	-
	Adam β_1	-0.3	0.0	-	-	-
	Lookahead k	-	-	-	1500	-
	Lookahead α	-	-	-	0.3	-
1	η	0.005	0.005	0.005	0.05	0.1
	Adam β_1	0.0	0.0	-	-	-
	Lookahead k	-	-	-	2450	-
	Lookahead α	-	-	-	0.3	-
	restart probability p	-	-	-	-	0.1

Table 3: List of hyperparameters used in Figure 3. η denotes the learning rate, β_1 is defined in (2), and α and k in Alg. 1.

Fig. 6 depicts the final performance of Lookahead–Minimax, using different values of k. Note that, the choice of plotting the distance to the optimum at a particular final iteration is causing the frequent oscillations of the depicted performances, since the iterate gets closer to the optimum only after the "backtracking" step. Besides the misleading oscillations, one can notice the trend of how the choice of k affects the final distance to the optimum. Interestingly, the case of $B = 16$ in Fig. 6 captures the periodicity of the rotating vector field, what sheds light on future directions in finding methods with adaptive k.

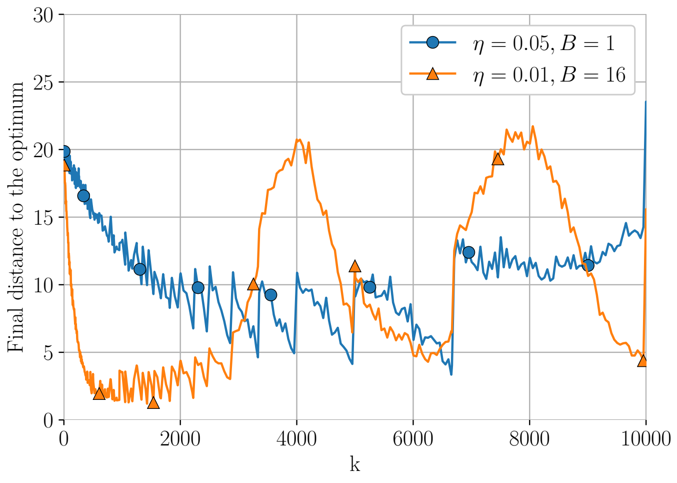


Figure 6: Sensitivity of LA-GDA to the value of the hyperparameter k in Alg. 1 for two combinations of batch sizes and η. The y-axis is the distance to the optimum at 20000 passes. The jolting of the curves is due to the final value being affected by how close it is to the last (LA) step, *i.e.* lines 10 and 11 of Alg. 1.

A.4 Illustrations of GAN optimization with lookahead

In Fig. 7 we consider a 2D bilinear game $\min_x \max_y x \cdot y$, and we illustrate the convergence of Lookahead–Minimax. Interestingly, Lookahead makes use of the rotations of the game vector field caused by the adversarial component of the game. Although standard-GDA diverges with all three shown learning rates, Lookahead–Minimax converges. Moreover, we see Lookahead–Minimax with larger learning rate of $\eta = 0.4$ (and fixed k and α) in fact converges faster then the case $\eta = 0.1$, what indicates that Lookahead–Minimax is also sensitive to the value of η, besides that it introduces additional hyperparameters k and α

B Parameter averaging

Polyak parameter averaging was shown to give fastest convergence rates among all stochastic gradient algorithms for convex functions, by minimizing the asymptotic variance induced by the algorithm (Polyak and Juditsky, 1992). This, so called *Ruppet–Polyak* averaging, is computed as the arithmetic average of the parameters:

$$\tilde{\boldsymbol{\theta}}_{RP} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{\theta}^{(t)} \,, \quad T \geq 1 \,. \tag{RP–Avg}$$

In the context of games, weighted averaging was proposed by Bruck (1977) as follows:

$$\tilde{\boldsymbol{\theta}}_{\text{WA}}^{(T)} = \frac{\sum_{t=1}^{T} \rho^{(t)} \boldsymbol{\theta}^{(t)}}{\sum_{t=1}^{T} \rho^{(t)}} \,. \tag{W–Avg}$$

Eq. (W–Avg) can be computed efficiently *online* as: $\boldsymbol{\theta}_{\text{WA}}^{(t)} = (1-\gamma^{(t)})\boldsymbol{\theta}_{\text{WA}}^{(t-1)} + \gamma^{(t)}\boldsymbol{\theta}^{(t)}$ with $\gamma \in [0,1]$. With $\gamma = \frac{1}{t}$ we obtain the Uniform Moving Averages (UMA) whose performance is reported in our experiments in

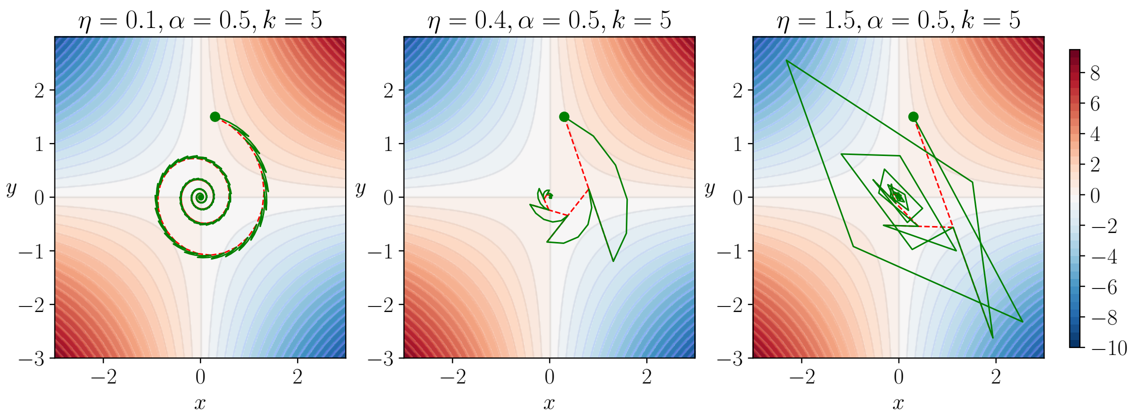


Figure 7: Illustration of Lookahead-minimax on the bilinear game $\min_x \max_y x \cdot y$, for different values of the learning rate $\eta \in \{0.1, 0.4, 1.5\}$, with fixed $k = 5$ and $\alpha = 0.5$. The trajectory of the iterates is depicted with green line, whereas the the interpolated line between $(\boldsymbol{\omega}^{\mathcal{P}}, \boldsymbol{\omega})$ with $\boldsymbol{\omega} = (\boldsymbol{\theta}, \boldsymbol{\varphi})$ is shown with dashed red line. The transparent lines depict the level curves of the loss function, and $\boldsymbol{\omega}^\star = (0.0)$. See § A.4 for discussion.

§ 4 and is computed as follows:

$$\boldsymbol{\theta}^t_{\text{UMA}} = (1 - \frac{1}{t})\boldsymbol{\theta}^{(t-1)}_{\text{UMA}} + \frac{1}{t}\boldsymbol{\theta}^{(t)}\,, \quad t = 1, \dots, T\,. \tag{UMA}$$

Analogously, we compute the Exponential Moving Averages (EMA) in an online fashion using $\gamma = 1 - \beta < 1$, as follows:

$$\boldsymbol{\theta}^t_{\text{EMA}} = \beta\boldsymbol{\theta}^{(t-1)}_{\text{EMA}} + (1 - \beta)\boldsymbol{\theta}^{(t)}\,, \quad t = 1, \dots, T\,. \tag{EMA}$$

In *all* our experiments, following related works (Yazıcı et al., 2019; Gidel et al., 2019a; Chavdarova et al., 2019), we fix $\beta = 0.9999$.

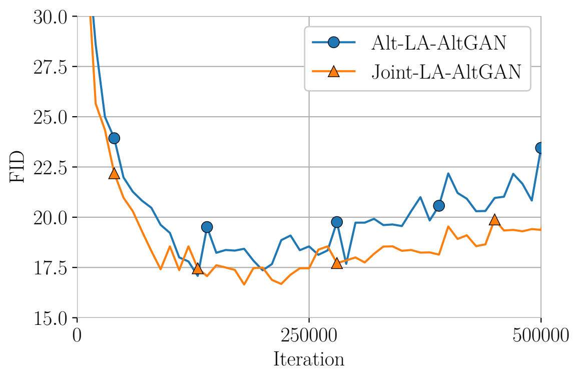
C Details on the Lookahead–minimax algorithm and alternatives

For completeness, in this section we consider an alternative implementation of Lookahead-minimax, which naively applies (LA) on each player separately, which we refer to as "alternating–lookahead". This in turn uses a "backtracked" iterate to update the opponent, rather than performing the "backtracking" step at the same time for both the players. In other words, the fact that line 9 of Alg. 3 is executed before updating $\boldsymbol{\theta}$ in line 12, and vice versa, does not allow for Lookahead to help deal with the rotations typical for games.

Algorithm 3 Alternating Lookahead-minimax pseudocode.

```
1: Input: Stopping time $T$, learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\varphi}}$, initial weights $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, $k_{\boldsymbol{\theta}}$, $k_{\boldsymbol{\varphi}}$, $\alpha_{\boldsymbol{\theta}}$, $\alpha_{\boldsymbol{\varphi}}$, losses $\mathcal{L}^{\boldsymbol{\theta}}$, $\mathcal{L}^{\boldsymbol{\varphi}}$,
   update ratio $r$, real–data distribution $p_d$, noise–data distribution $p_z$.
2: $\boldsymbol{\theta}^{\mathcal{P}} \leftarrow \boldsymbol{\theta}$                                   (store copy)
3: $\boldsymbol{\varphi}^{\mathcal{P}} \leftarrow \boldsymbol{\varphi}$
4: for $t \in 0, \ldots, T-1$ do
5:    for $i \in 1, \ldots, r$ do
6:       $\boldsymbol{x} \sim p_d$, $\boldsymbol{z} \sim p_z$
7:       $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} - \eta_{\boldsymbol{\varphi}} \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{x}, \boldsymbol{z})$                (update $\boldsymbol{\varphi}$ $k$ times)
8:       if $(t * r + i) \% k_{\boldsymbol{\varphi}} == 0$ then
9:          $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi}^{\mathcal{P}} + \alpha_{\boldsymbol{\varphi}}(\boldsymbol{\varphi} - \boldsymbol{\varphi}^{\mathcal{P}})$          (backtracking on line $\boldsymbol{\varphi}^{\mathcal{P}}$, $\boldsymbol{\varphi}$)
10:         $\boldsymbol{\varphi}^{\mathcal{P}} \leftarrow \boldsymbol{\varphi}$
11:   $\boldsymbol{z} \sim p_z$
12:   $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{z})$                   (update $\boldsymbol{\theta}$ once)
13:   if $t \% k_{\boldsymbol{\theta}} == 0$ then
14:      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{\mathcal{P}} + \alpha_{\boldsymbol{\theta}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathcal{P}})$             (backtracking on line $\boldsymbol{\theta}^{\mathcal{P}}$, $\boldsymbol{\theta}$)
15:      $\boldsymbol{\theta}^{\mathcal{P}} \leftarrow \boldsymbol{\theta}$
16: Output: $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$
```

On SVHN and CIFAR-10, the joint Lookahead-minimax consistently gave us the best results, as can be seen in Figure 8 and 9. On MNIST, the alternating and joint implementations worked equally well, see Figure 9.



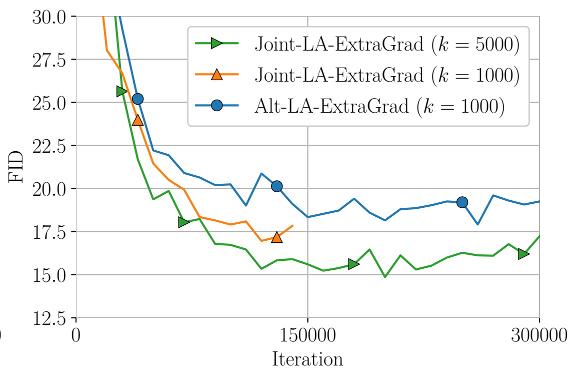
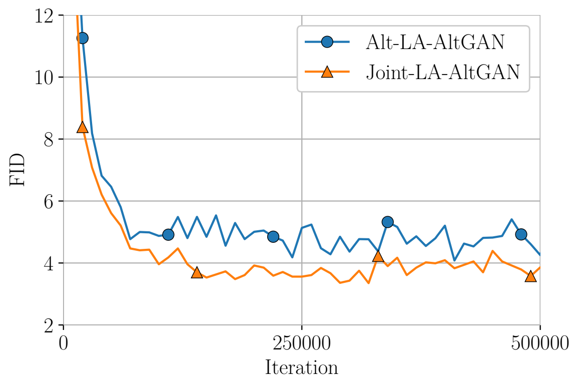
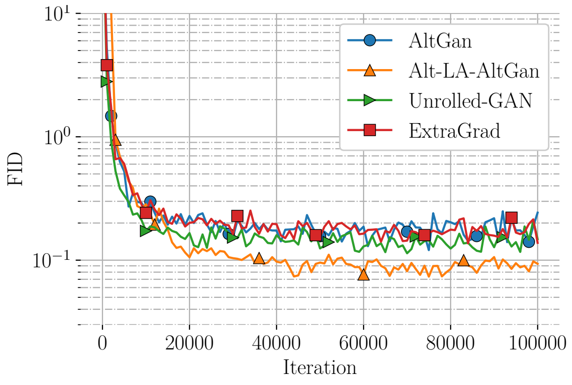


Figure 8: On the CIFAR-10 dataset, comparison of the joint Lookahead-minimax implementation (*Joint* prefix, see Algorithm 1) and the alternating Lookahead-minimax implementation (*Alt* prefix, see Algorithm 3). We can see some significant improvements in FID when using the joint implementation, for both LA-AltGAN and LA-ExtraGrad.



(a) SVHN dataset.



(b) MNIST dataset.

Figure 9: (a): Comparison of the joint Lookahead-minimax implementation (*Joint* prefix, see Algorithm 1) and the alternating Lookahead-minimax implementation (*Alt* prefix, see Algorithm 3) on the SVHN dataset. (b): Results obtained with the different methods introduced in §4 as well as an alternating implementation of Lookahead-minimax, on the MNIST dataset. Each curve is obtained averaged over 5 runs. The results of the alternating implementation differ very little from the joint implementation, the curve for Alt-LA-AltGAN matches results in Table 1.

D Details on the implementation

For our experiments, we used the PyTorch[3] deep learning framework, whereas for computing the FID and IS metrics, we used the provided implementations in Tensorflow[4] for consistency with related works.

D.1 Metrics

We provide more details about the metrics enumerated in § 4. Both FID and IS use: (i) the *Inception v3 network* (Szegedy et al., 2015) that has been trained on the ImageNet dataset consisting of ∼1 million RGB images of 1000 classes, $C = 1000$. (ii) a sample of m generated images $x \sim p_g$, where usually $m = 50000$.

D.1.1 Inception Score

Given an image x, IS uses the softmax output of the Inception network $p(y|x)$ which represents the probability that x is of class $c_i, i \in 1 \dots C$, i.e., $p(y|x) \in [0,1]^C$. It then computes the marginal class distribution $p(y) = \int_x p(y|x)p_g(x)$. IS measures the Kullback–Leibler divergence $\mathbb{D}_{KL}$ between the predicted conditional label distribution $p(y|x)$ and the marginal class distribution $p(y)$. More precisely, it is computed as follows:

$$IS(G) = \exp\big(\mathbb{E}_{x\sim p_g}[\mathbb{D}_{KL}(p(y|x)||p(y))]\big) = \exp\big(\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C} p(y_c|x_i)\log\frac{p(y_c|x_i)}{p(y_c)}\big). \tag{8}$$

It aims at estimating (i) if the samples look realistic i.e., $p(y|x)$ should have low entropy, and (ii) if the samples are diverse (from different ImageNet classes) i.e., $p(y)$ should have high entropy. As these are combined using the Kullback–Leibler divergence, the higher the score is, the better the performance. Note that the range of IS scores at convergence varies across datasets, as the Inception network is pretrained on the

[3] https://pytorch.org/
[4] https://www.tensorflow.org/

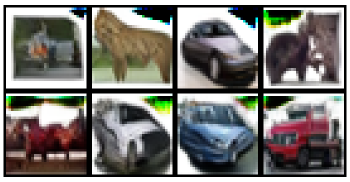


Figure 10: Samples from our generator model with the highest IS score. We can clearly see some unrealistic artefacts. We observed that the IS metric does not penalize these artefacts, whereas FID does penalize them.

ImageNet classes. For example, we obtain low IS values on the SVHN dataset as a large fraction of classes are numbers, which typically do not appear in the ImageNet dataset. Since **MNIST** has greyscale images, we used a classifier trained on this dataset and used $m = 5000$. For the rest of the datasets, we used the original implementation[5] of IS in TensorFlow, and $m = 50000$.

As the Inception Score considers the classes as predicted by the Inception network, it can be prone not to penalize visual artefacts as long as those do not alter the predicted class distribution. In Fig. 10 we show some images generated by our best model according to IS. Those images exhibit some visible unrealistic artifacts, while enough of the image is left for us to recognise a potential image label. For this reason we consider that the Fréchet Inception Distance is a more reliable estimator of image quality. However, we reported IS for completeness.

D.1.2 Fréchet Inception Distance

Contrary to IS, FID aims at comparing the synthetic samples $x \sim p_g$ with those of the training dataset $x \sim p_d$ in a feature space. The samples are embedded using the first several layers of the Inception network. Assuming p_g and p_d are multivariate normal distributions, it then estimates the means $\boldsymbol{m}_g$ and $\boldsymbol{m}_d$ and covariances C_g and C_d, respectively for p_g and p_d in that feature space. Finally, FID is computed as:

$$\mathbb{D}_{\text{FID}}(p_d, p_g) \approx d^2((\boldsymbol{m}_d, C_d), (\boldsymbol{m}_g, C_g)) = \|\boldsymbol{m}_d - \boldsymbol{m}_g\|_2^2 + Tr(C_d + C_g - 2(C_d C_g)^{\frac{1}{2}}), \tag{9}$$

where d^2 denotes the Fréchet Distance. Note that as this metric is a distance, the lower it is, the better the performance. We used the original implementation of FID[6] in Tensorflow, along with the provided statistics of the datasets.

D.2 Architectures & Hyperparameters

Description of the architectures. We describe the models we used in the empirical evaluation of Lookahead-minimax by listing the layers they consist of, as adopted in GAN works, e.g. (Miyato et al., 2018). With "conv." we denote a convolutional layer and "transposed conv" a transposed convolution layer (Radford et al., 2016). The models use Batch Normalization (Ioffe and Szegedy, 2015) and Spectral Normalization layers (Miyato et al., 2018).

D.2.1 Architectures for experiments on MNIST

For experiments on the **MNIST** dataset, we used the DCGAN architectures (Radford et al., 2016), listed in Table 4, and the parameters of the models are initialized using PyTorch default initialization. For experiments

[5]`https://github.com/openai/improved-gan/`
[6]`https://github.com/bioinf-jku/TTUR`

Generator
Input: $z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
transposed conv. (ker: 3×3, $128 \rightarrow 512$; stride: 1)
Batch Normalization
ReLU
transposed conv. (ker: 4×4, $512 \rightarrow 256$, stride: 2)
Batch Normalization
ReLU
transposed conv. (ker: 4×4, $256 \rightarrow 128$, stride: 2)
Batch Normalization
ReLU
transposed conv. (ker: 4×4, $128 \rightarrow 1$, stride: 2, pad: 1)
$Tanh(\cdot)$

Discriminator
Input: $x \in \mathbb{R}^{1 \times 28 \times 28}$
conv. (ker: 4×4, $1 \rightarrow 64$; stride: 2; pad:1)
LeakyReLU (negative slope: 0.2)
conv. (ker: 4×4, $64 \rightarrow 128$; stride: 2; pad:1)
Batch Normalization
LeakyReLU (negative slope: 0.2)
conv. (ker: 4×4, $128 \rightarrow 256$; stride: 2; pad:1)
Batch Normalization
LeakyReLU (negative slope: 0.2)
conv. (ker: 3×3, $256 \rightarrow 1$; stride: 1)
$Sigmoid(\cdot)$

Table 4: DCGAN architectures (Radford et al., 2016) used for experiments on **MNIST**. We use *ker* and *pad* to denote *kernel* and *padding* for the (transposed) convolution layers, respectively. With $h \times w$ we denote the kernel size. With $c_{in} \rightarrow y_{out}$ we denote the number of channels of the input and output, for (transposed) convolution layers.

on this dataset, we used the *non saturating* GAN loss as proposed (Goodfellow et al., 2014):

$$\mathcal{L}_D = \mathbb{E}_{x \sim p_d} \log(D(x)) + \mathbb{E}_{z \sim p_z} \log(D(G(z))) \tag{10}$$

$$\mathcal{L}_G = \mathbb{E}_{z \sim p_z} \log(D(G(z))), \tag{11}$$

where p_d and p_z denote the data and the latent distributions (the latter to be predefined).

D.2.2 ResNet architectures for Cifar-10 and SVHN

We replicate the experimental setup described for **CIFAR-10** and **SVHN** in (Miyato et al., 2018; Chavdarova et al., 2019), as listed in Table 6. This setup uses the hinge version of the adversarial non-saturating loss, see (Miyato et al., 2018). As a reference, our ResNet architectures for **CIFAR-10** have approximately 85 layers–in total for G and D, including the non linearity and the normalization layers.

D.2.3 Unrolling implementation

In Section A.1 we explained how we implemented unrolling for our full-batch bilinear experiments. Here we describe our implementation for our MNIST and CIFAR-10 experiments.

Unrolling is computationally intensive, which can become a problem for large architectures. The computation of $\nabla_{\varphi} \mathcal{L}^{\varphi}(\boldsymbol{\theta}_t^m, \boldsymbol{\varphi}_t)$, with m unrolling steps, requires the computation of higher order derivatives which comes with a $\times m$ memory footprint and a significant slowdown. Due to limited memory, one can only backpropagate through the last unrolled step, bypassing the computation of higher order derivatives. We empirically see the gradient is small for those derivatives. In this approximate version, unrolling can be seen as of the same family as extragradient, computing its extrapolated points using more than a single step. We tested both true and approximate unrolling on MNIST, with a number of unrolling steps ranging from 5 to 20. The full unrolling that performs the backpropagation on the unrolled discriminator was implemented using the Higher[7] library. On CIFAR-10 we only experimented with approximate unrolling over 5 to 10 steps due to the large memory footprint of the ResNet architectures used for the generator and discriminator, making the other approach infeasible given our resources.

[7] https://github.com/facebookresearch/higher

G–ResBlock
Bypass:
Upsample($\times$2)
Feedforward:
Batch Normalization
ReLU
Upsample($\times$2)
conv. (ker: 3×3, 256 $\rightarrow$ 256; stride: 1; pad: 1)
Batch Normalization
ReLU
conv. (ker: 3×3, 256 $\rightarrow$ 256; stride: 1; pad: 1)

D–ResBlock (ℓ–th block)
Bypass:
[AvgPool (ker:2$\times$2)], if $\ell = 1$
conv. (ker: 1×1, $3_{\ell=1}/128_{\ell\neq1}$ $\rightarrow$ 128; stride: 1)
Spectral Normalization
[AvgPool (ker:2$\times$2, stride:2)], if $\ell \neq 1$
Feedforward:
[ReLU], if $\ell \neq 1$
conv. (ker: 3×3, $3_{\ell=1}/128_{\ell\neq1}$ $\rightarrow$ 128; stride: 1; pad: 1)
Spectral Normalization
ReLU
conv. (ker: 3×3, 128 $\rightarrow$ 128; stride: 1; pad: 1)
Spectral Normalization
AvgPool (ker:2$\times$2)

Table 5: ResNet blocks used for the ResNet architectures (see Table 6), for the Generator (left) and the Discriminator (right). Each ResNet block contains skip connection (bypass), and a sequence of convolutional layers, normalization, and the ReLU non–linearity. The skip connection of the ResNet blocks for the Generator (left) upsamples the input using a factor of 2 (we use the default PyTorch upsampling algorithm–nearest neighbor), whose output is then added to the one obtained from the ResNet block listed above. For clarity we list the layers sequentially, however, note that the bypass layers operate in parallel with the layers denoted as "feedforward" (He et al., 2015). The ResNet block for the Discriminator (right) differs if it is the first block in the network (following the input to the Discriminator), $\ell = 1$, or a subsequent one, $\ell > 1$, so as to avoid performing the ReLU non–linearity immediate on the input.

Generator	**Discriminator**
Input: $z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$	*Input:* $x \in \mathbb{R}^{3\times32\times32}$
Linear(128 $\rightarrow$ 4096)	D–ResBlock
G–ResBlock	D–ResBlock
G–ResBlock	D–ResBlock
G–ResBlock	D–ResBlock
Batch Normalization	ReLU
ReLU	AvgPool (ker:8$\times$8)
conv. (ker: 3×3, 256 $\rightarrow$ 3; stride: 1; pad:1)	Linear(128 $\rightarrow$ 1)
$Tanh(\cdot)$	Spectral Normalization

Table 6: *Deep* ResNet architectures used for experiments on **SVHN** and **CIFAR-10**, where G–ResBlock and D–ResBlock for the Generator (left) and the Discriminator (right), respectively, are described in Table 5. The models' parameters are initialized using the Xavier initialization (Glorot and Bengio, 2010).

D.2.4 Hyperparameters used on MNIST

Table 7 lists the hyperparameters that we used for our experiments on the MNIST dataset.

Parameter	AltGAN	LA-AltGAN	ExtraGrad	LA-ExtraGrad	Unrolled-GAN
η_G	0.001	0.001	0.001	0.001	0.001
η_D	0.001	0.001	0.001	0.001	0.001
Adam β_1	0.05	0.05	0.05	0.05	0.05
Batch-size	50	50	50	50	50
Update ratio r	1	1	1	1	1
Lookahead k	-	1000	-	1000	-
Lookahead α	-	0.5	-	0.5	-
Unrolling steps	-	-	-	-	20

Table 7: Hyperparameters used on MNIST.

D.2.5 Hyperparameters used on SVHN

Parameter	AltGAN	LA-AltGAN	ExtraGrad	LA-ExtraGrad
η_G	0.0002	0.0002	0.0002	0.0002
η_D	0.0002	0.0002	0.0002	0.0002
Adam β_1	0.0	0.0	0.0	0.0
Batch-size	128	128	128	128
Update ratio r	5	5	5	5
Lookahead k	-	5	-	5000
Lookahead α	-	0.5	-	0.5

Table 8: Hyperparameters used on SVHN.

Table 8 lists the hyperparameters used for experiments on SVHN. These values were selected for each algorithm *independently* after tuning the hyperparameters for the baseline.

D.2.6 Hyperparameters used on CIFAR-10

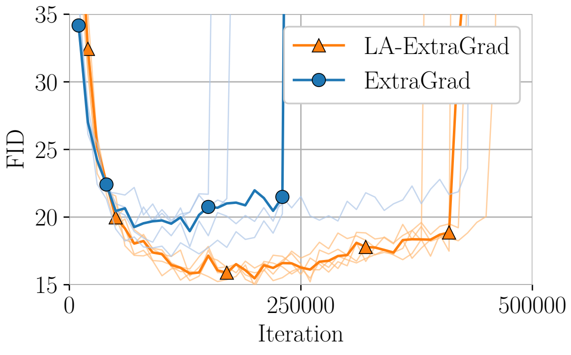
Parameter	AltGAN	LA-AltGAN	ExtraGrad	LA-ExtraGrad	Unrolled-GAN
η_G	0.0002	0.0002	0.0002	0.0002	0.0002
η_D	0.0002	0.0002	0.0002	0.0002	0.0002
Adam β_1	0.0	0.0	0.0	0.0	0.0
Batch-size	128	128	128	128	128
Update ratio r	5	5	5	5	5
Lookahead k	-	5	-	5000	-
Lookahead α	-	0.5	-	0.5	-
Unrolling steps	-	-	-	-	5

Table 9: Hyperparameters that we used for our experiments on CIFAR-10.

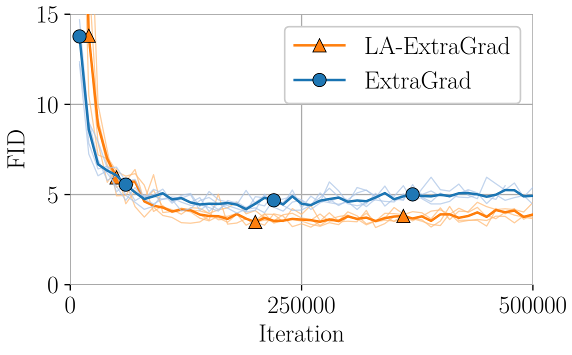
The reported results on CIFAR-10 were obtained using the hyperparameters listed in Table 9. These values were selected for each algorithm *independently* after tuning the hyperparameters. For the baseline methods we selected the hyperparameters giving the best performances. Consistent with the results reported by related works, we also observed that using larger ratio of updates of the discriminator and the generator improves the stability of the baseline, and we used $r = 5$. We observed that using learning rate decay delays the divergence, but does not improve the best FID scores, hence we did not use it in our reported models.

E Additional experimental results

In Fig. 5 we compared the stability of LA–AltGAN methods against their AltGAN baselines on both the CIFAR-10 and SVHN datasets. Analogously, in Fig. 11 we report the comparison between LA–ExtraGrad and ExtraGrad over the iterations. We observe that the experiments on SVHN with ExtraGrad are more stable than those of CIFAR-10. Interestingly, we observe that: (i) LA–ExtraGradient improves both the stability and the performance of the baseline on CIFAR-10, see Fig. 11a, and (ii) when the stability of the baseline is relatively good as on the SVHN dataset, LA–Extragradient still improves its performances, see Fig. 11b.



(a) CIFAR-10 dataset.



(b) SVHN dataset.

Figure 11: Improved stability of LA–ExtraGrad relative to its ExtraGrad baseline on SVHN and CIFAR-10, over 5 runs. The median and the individual runs are illustrated with ticker solid lines and with transparent lines, respectively. See § E and D for discussion and details on the implementation, resp.

E.1 Samples of LA–GAN Generators

In this section we present random samples of the generators of our LAGAN experiments trained on CIFAR-10 and SVHN. Figures 12, **??** and 13 are generated by some of our LA-AltGAN models trained on CIFAR-10 with and without EMA. Similarly, Figures 14 and 15 are generated by some of our LA-ExtraGrad models trained on CIFAR-10 with and without EMA. Finally, we show samples of our LA-ExtraGrad models trained on the SVHN dataset with and without EMA, see Figures 16 and 17 respectively.

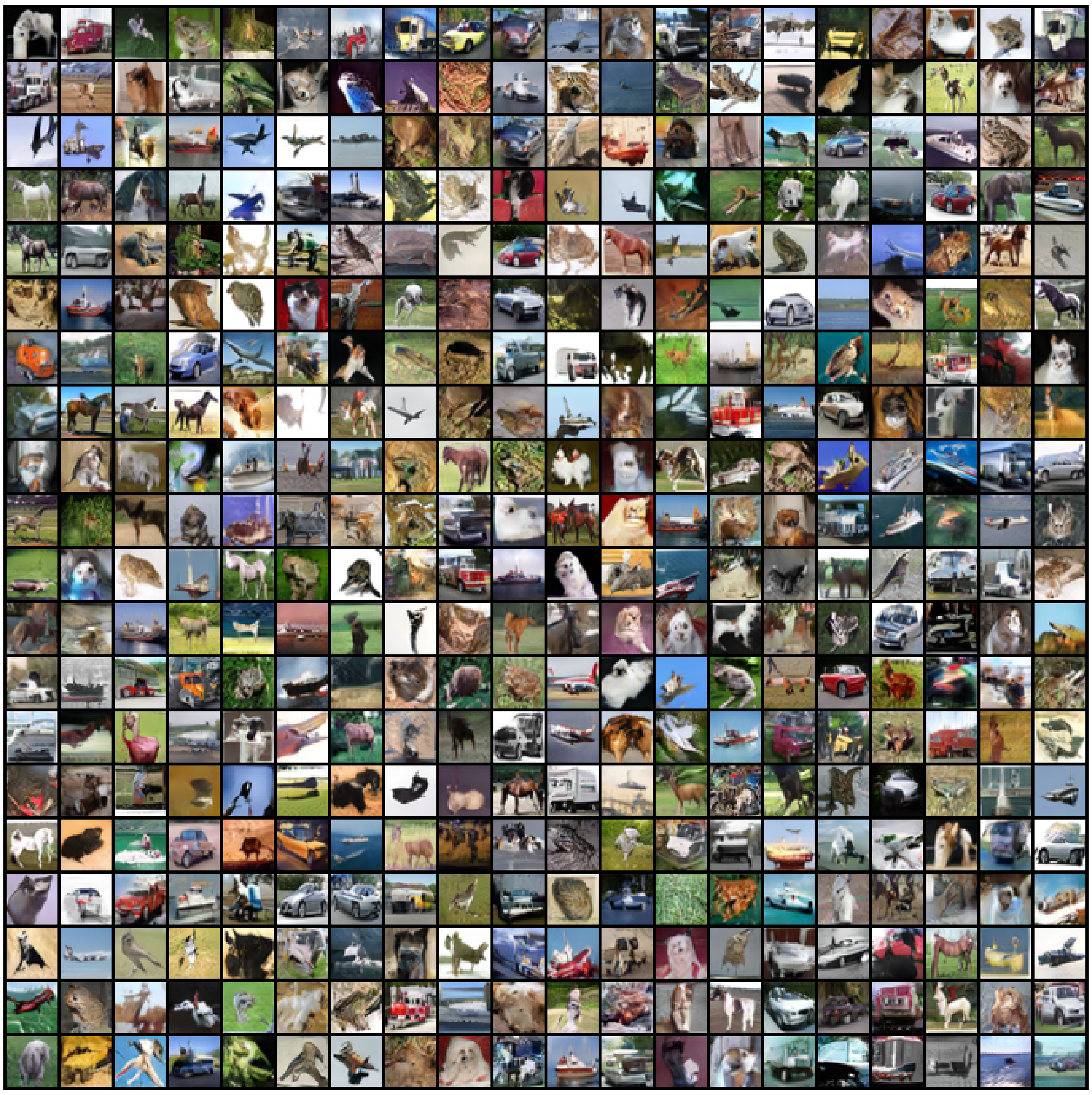


Figure 12: Images generated by our best LA-AltGAN + EMA model (FID of 13.65) trained on CIFAR-10.

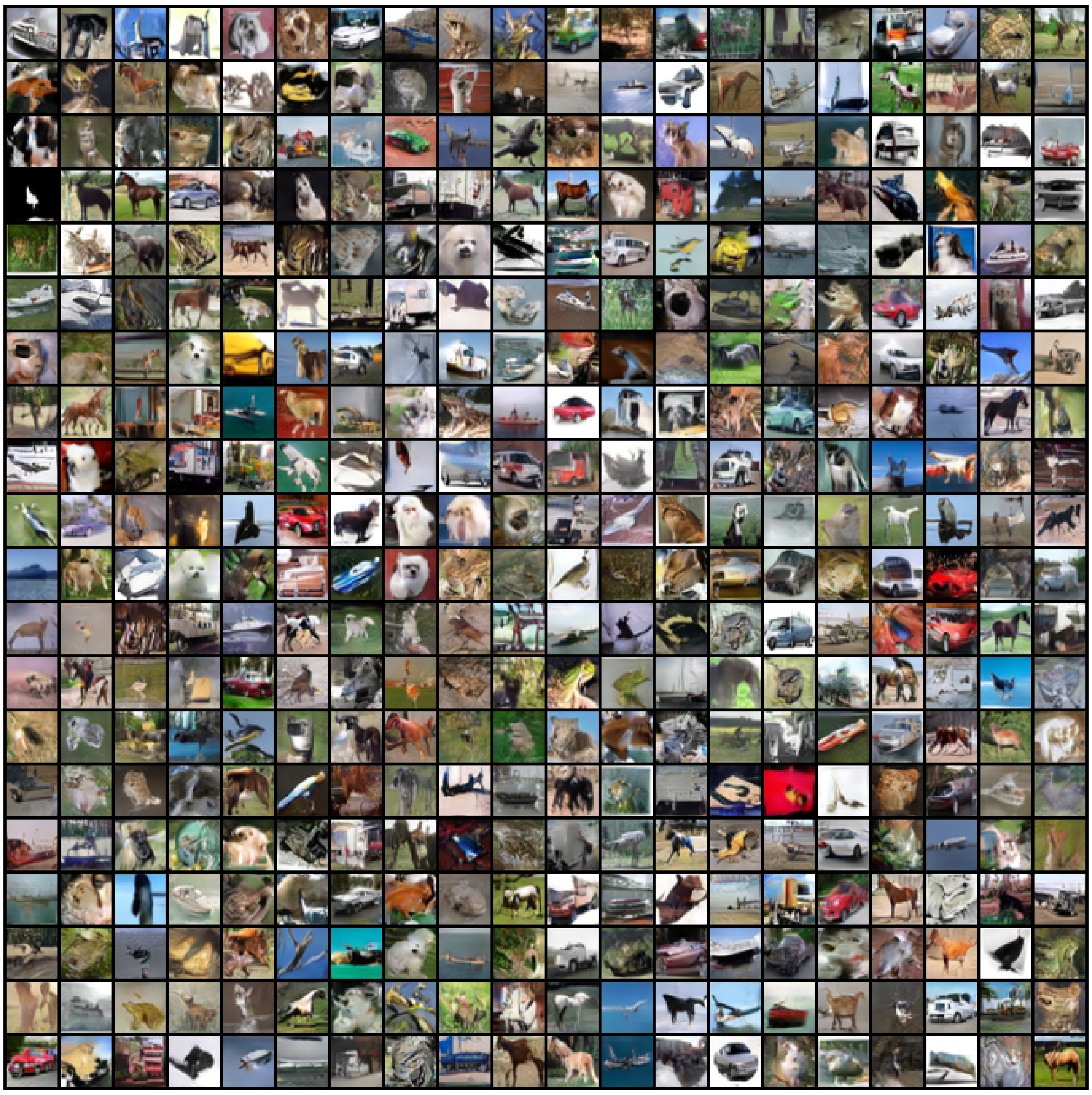


Figure 13: Images generated by the best LA-AltGAN generator we obtained according to FID (trained on CIFAR-10), for which we obtain FID of 16.277.

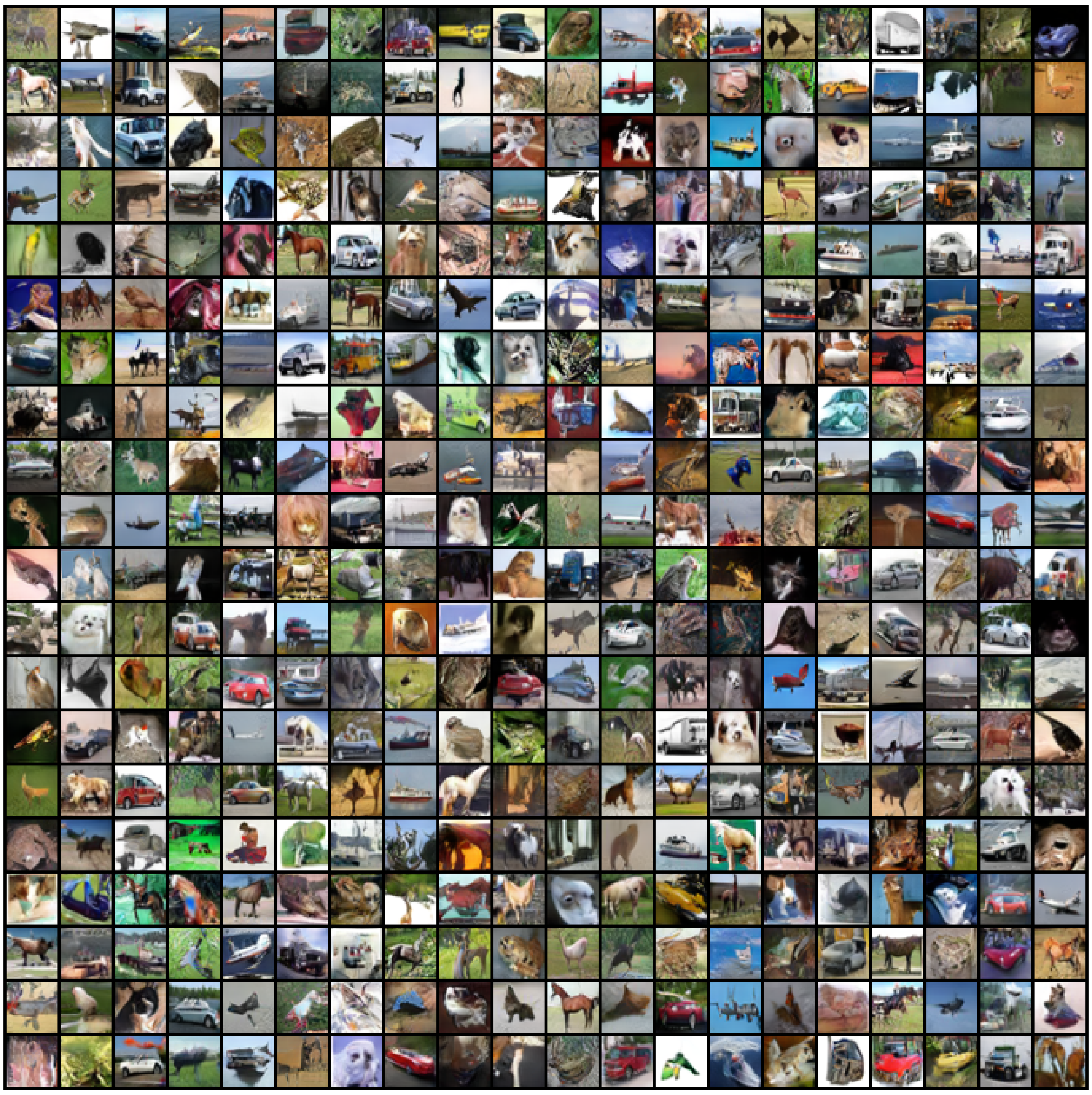


Figure 14: Images generated by our best LA-ExtraGrad + EMA model (FID of 14.25) trained on CIFAR-10.

Figure 15: Images generated by the LA-ExtraGrad generator without averaging from the same experiment of Fig. 14 (trained on CIFAR-10), for which we obtain FID of 14.86.

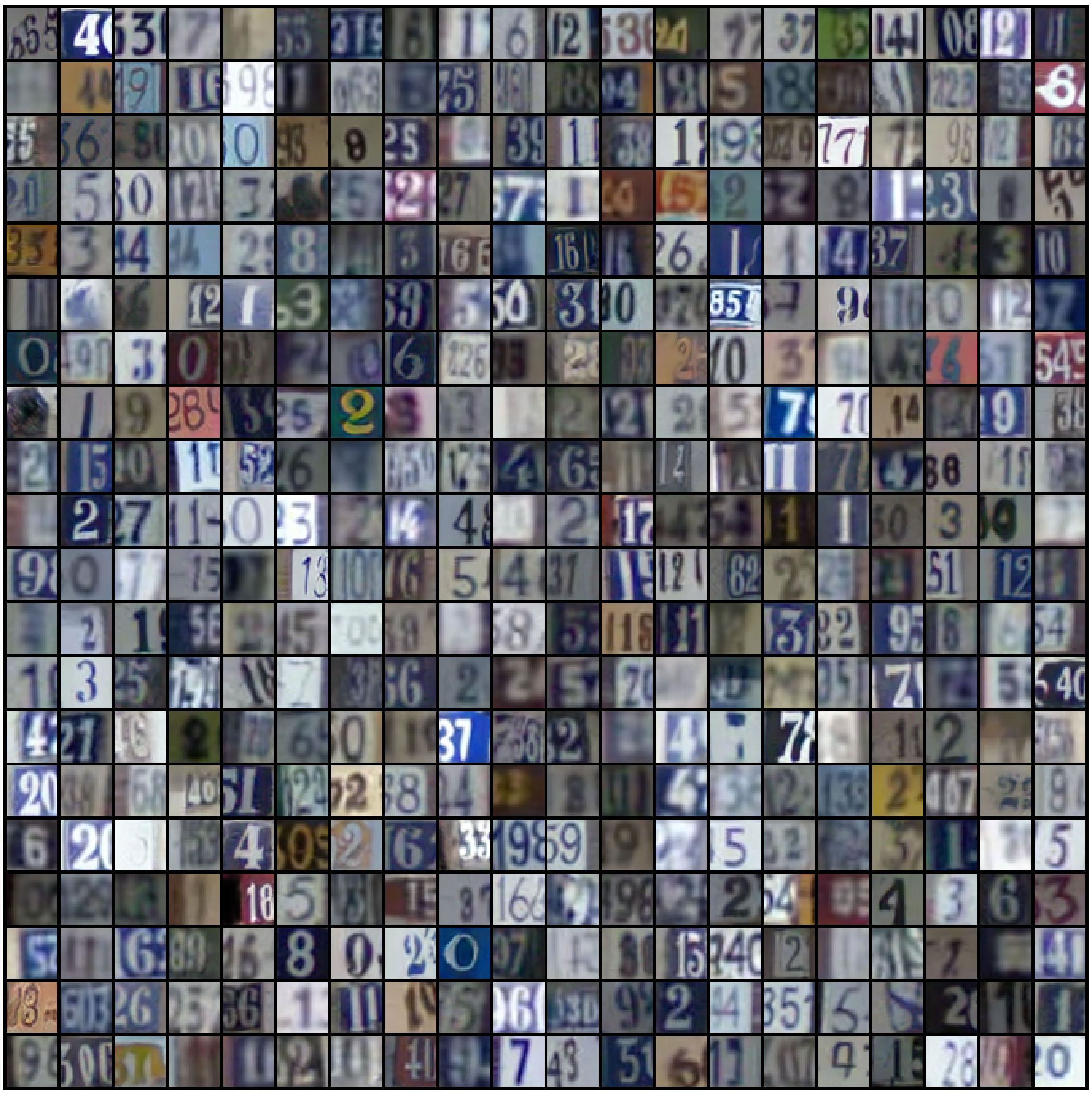


Figure 16: Images generated by one of our best LA-ExtraGrad + EMA model (FID of 2.94) trained on SVHN.

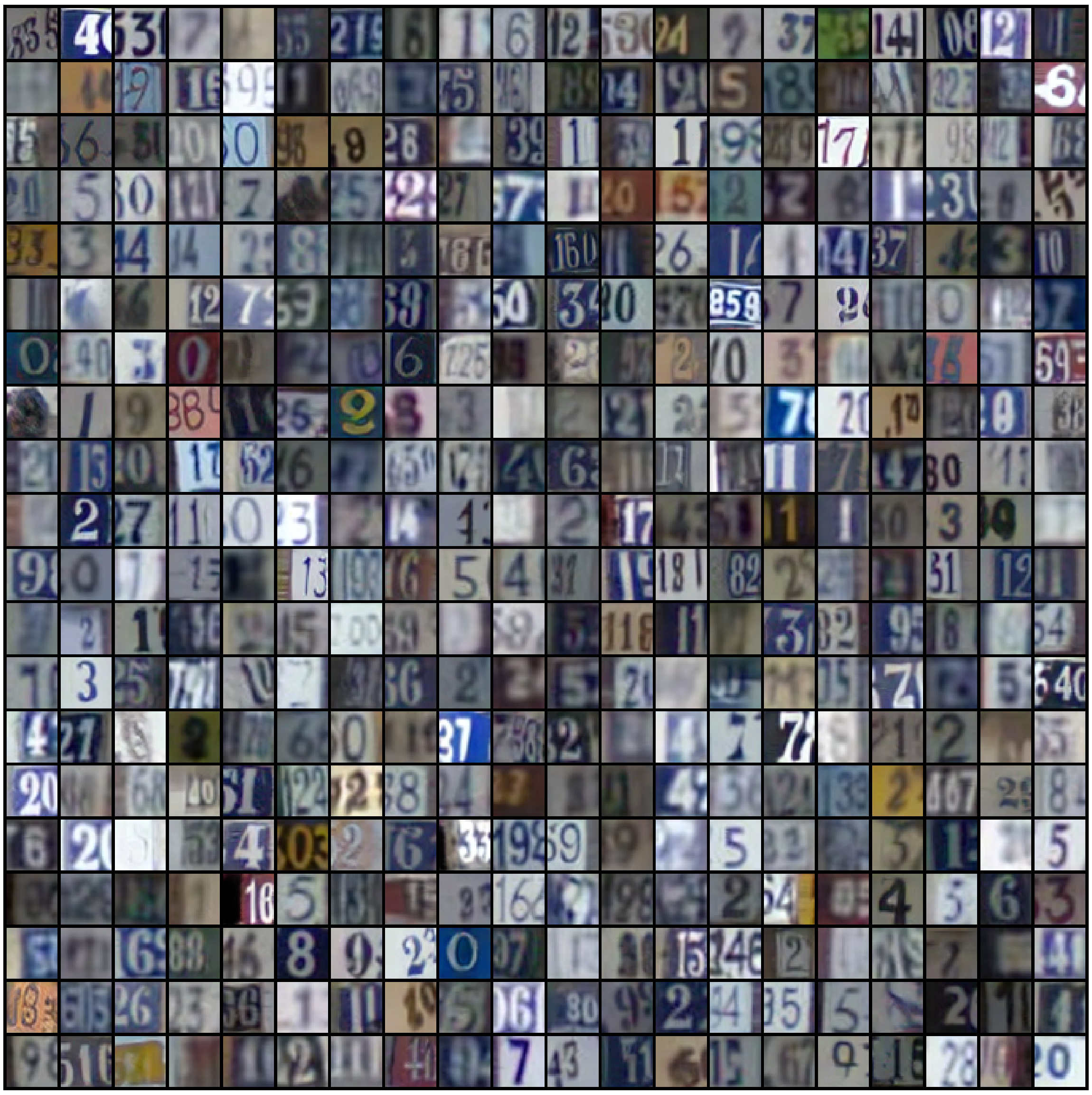


Figure 17: Images generated by the iterate generator (without any averaging) of the same LA-ExtraGrad experiment as in Fig. 16 (trained on SVHN), for which we obtain FID of 3.17.