TAMING GANS WITH LOOKAHEAD–MINMAX

Tatjana Chavdarova* EPFL
Mattéo Pagliardini* EPFL
Sebastian U. Stich EPFL
François Fleuret University of Geneva
Martin Jaggi EPFL

ABSTRACT

Generative Adversarial Networks are notoriously challenging to train. The underlying minmax optimization is highly susceptible to the variance of the stochastic gradient and the rotational component of the associated game vector field. To tackle these challenges, we propose the Lookahead algorithm for minmax optimization, originally developed for single objective minimization only. The backtracking step of our Lookahead–minmax naturally handles the rotational game dynamics, a property which was identified to be key for enabling gradient ascent descent methods to converge on challenging examples 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, CIFAR-10, and ImageNet demonstrate a clear advantage of combining Lookahead–minmax with Adam or extragradient, in terms of performance and improved stability, for negligible memory and computational cost. 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 12.19 *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 $f\colon \mathcal{X} \to \mathbb{R}$. 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 $f_i\colon \mathcal{X}_1 \times \cdots \times \mathcal{X}_N \to \mathbb{R}$ simultaneously.

A particularly successful class of algorithms of this kind 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 minmax optimization $f\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ (Von Neumann & 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 minmax problems is the gradient descent ascent algorithm (GDA), which alternates between a gradient descent step for the min-player and a gradient ascent step for the max-player. This minmax training aims at finding a Nash equilibrium where no player has the incentive of changing its parameters.

Despite the impressive quality of the samples generated by the GANs—relative to classical maximum likelihood-based generative models—these models 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 computational budgets.

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

*Equal contributions. Correspondence to `firstname.lastname@epfl.ch`.

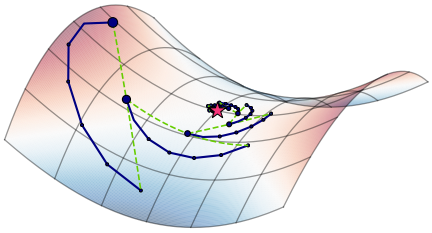


Figure 1: Illustration of Lookahead–minmax (Alg.1) with GDA on: $\min_x \max_y x \cdot y$, with α=0.5. The solution, trajectory $\{x_t, y_t\}_{t=1}^{T}$, and the lines between $(x^{\mathcal{P}}, y^{\mathcal{P}})$ and (x_t, y_t) are shown with red star, blue line, and dashed green line, resp. The backtracking step of Alg. 1 (lines 10 & 11) allows the otherwise non-converging GDA to converge, see § 4.1.1.

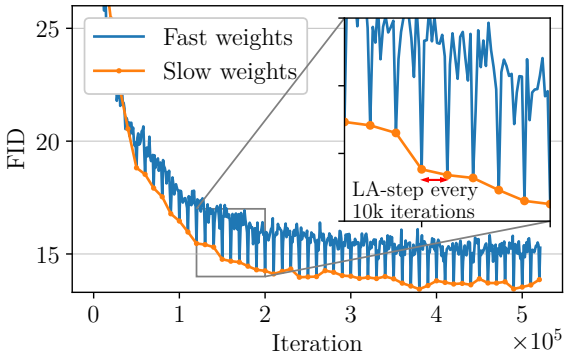


Figure 2: FID (↓) of Lookahead–minmax on 32×32 ImageNet (§ 5), see Fig. 9 for IS. We observe *significant* improvements after each LA-step, what empirically confirms the existence of rotations and thus the intuition of the behaviour of LA-minmax on games, as well as the relevance of the bilinear game for real world applications–Fig.1.

GAN performances (Brock et al., 2019) in contrast to minimization (Defazio & Bottou, 2019; 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 & Shapley, 1996) and a *Hamiltonian* game, resp. Games are often combination of the two, making this general case hard to solve.

In the context of single objective 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 the "stochasticity" of the gradient estimates–called *fast* weights–generated by an inner optimizer[1]. Lookahead was shown to improve the stability during training and to reduce the variance of the so called *slow* weights.

Contributions. Our contributions can be summarized as follows:

- Building on insights of Wang et al. (2020), who argue that Lookahead can be interpreted as an instance of local SGD, we derive improved convergence guarantees for the Lookahead algorithm in the context of single objective minimization.
- We propose Lookahead–minmax for optimizing minmax problems, that applies extrapolation in the *joint* parameter space (see Alg 1), so as to account for the rotational component of the associated game vector field (defined in § 2).
- We motivate the use of Lookahead–minmax 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 § 4.1.1), as well as (ii) it yields good performance on challenging stochastic variants of this game, despite the high variance (see § 4.1.2).
- We empirically benchmark Lookahead–minmax on GANs on four standard datasets—MNIST, CIFAR-10, SVHN and ImageNet—*on two different models* (DCGAN & ResNet), with standard optimization methods for GANs, GDA and Extragradient, called LA–AltGAN and LA–ExtraGradient, resp. We *consistently* observe both stability and performance improvements at a negligible additional cost that does not require additional forward and backward passes, see § 5.

[1] As Lookahead uses iterates generated by inner optimizer we also refer it is a *meta-optimizer* or *wrapper*.

2 BACKGROUND

GAN formulation. Given the data distribution p_d, the generator is a mapping $G\colon z \mapsto x$, where z is sampled from a known distribution $z \sim p_z$ and ideally $x \sim p_d$. The discriminator $D\colon x \mapsto D(x) \in [0,1]$ 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 a 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 $\mathcal{L}^{\boldsymbol{\varphi}} = -\mathcal{L}^{\boldsymbol{\theta}}$ the game is called a *zero-sum* and equation 2P-G is a minmax problem.

Minmax 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 & Pang, 1990).

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 of JVF are defined as:

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

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 SINGLE OBJECTIVE

In the context of single objective minimization, Zhang et al. (2019) recently proposed the Lookahead algorithm where at every step t: (i) a copy of the current iterate $\tilde{\boldsymbol{\omega}}_t$ is made: $\tilde{\boldsymbol{\omega}}_t \leftarrow \boldsymbol{\omega}_t$, (ii) $\tilde{\boldsymbol{\omega}}_t$ is then updated for $k \geq 1$ times yielding $\tilde{\boldsymbol{\omega}}_{t+k}$, 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 $\tilde{\boldsymbol{\omega}}_{t+k}$:

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

Algorithm 1 Lookahead–Minmax pseudocode.

Input: Stopping time T, learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\varphi}}$, initial weights $\boldsymbol{\theta}_0$, $\boldsymbol{\varphi}_0$, lookahead k and α, losses $\mathcal{L}^{\boldsymbol{\theta}}$, $\mathcal{L}^{\boldsymbol{\varphi}}$, update ratio R, p_d and p_z real and noise–data distribution, resp.
$\tilde{\boldsymbol{\theta}}_{0,0}, \tilde{\boldsymbol{\varphi}}_{0,0} \leftarrow \boldsymbol{\theta}_0, \boldsymbol{\varphi}_0$
for $t \in 0, \ldots, T-1$ **do**
  for $i \in 0, \ldots, k-1$ **do**
    Sample $\boldsymbol{x}, \boldsymbol{z} \sim p_d, p_z$
    $\tilde{\boldsymbol{\varphi}}_{t,i+1} = \tilde{\boldsymbol{\varphi}}_{t,i} - \eta_{\boldsymbol{\varphi}} \nabla_{\tilde{\boldsymbol{\varphi}}} \mathcal{L}^{\boldsymbol{\varphi}}(\tilde{\boldsymbol{\theta}}_{t,i}, \tilde{\boldsymbol{\varphi}}_{t,i}, \boldsymbol{x}, \boldsymbol{z})$
    Sample $\boldsymbol{z} \sim p_z$
    $\tilde{\boldsymbol{\theta}}_{t,i+1} = \tilde{\boldsymbol{\theta}}_{t,i+1} - \eta_{\boldsymbol{\theta}} \nabla_{\tilde{\boldsymbol{\theta}}} \mathcal{L}^{\boldsymbol{\theta}}(\tilde{\boldsymbol{\theta}}_{t,i}, \tilde{\boldsymbol{\varphi}}_{t,i}, \boldsymbol{z})$
  end for
  $\boldsymbol{\varphi}_{t+1} = \boldsymbol{\varphi}_t + \alpha_{\boldsymbol{\varphi}}(\tilde{\boldsymbol{\varphi}}_{t,k} - \boldsymbol{\varphi}_t)$
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_{\boldsymbol{\theta}}(\tilde{\boldsymbol{\theta}}_{t,k} - \boldsymbol{\theta}_t)$
  $\tilde{\boldsymbol{\theta}}_{t+1,0}, \tilde{\boldsymbol{\varphi}}_{t+1,0} \leftarrow \boldsymbol{\theta}_{t+1}, \boldsymbol{\varphi}_{t+1}$
end for
Output: $\boldsymbol{\theta}_T, \boldsymbol{\varphi}_T$

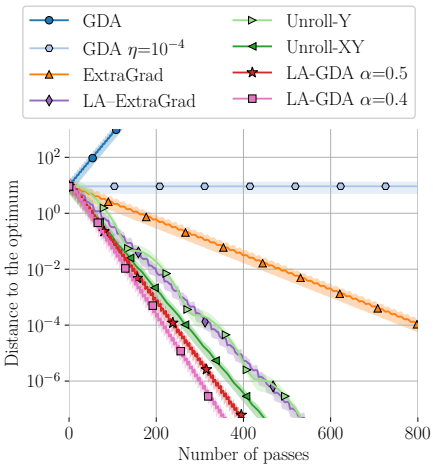


Figure 3: Distance to the optimum of equation SB–G using different *full–batch* methods, averaged over 5 runs. Unless otherwise specified, the learning rate is $\eta = 0.3$. See § 4.1.1.

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

Theoretical Analysis. The theoretical understanding of the LA optimizer (in the variant where k SGD steps are used for prediction) is still under development: Zhang et al. (2019) study the performance of LA on quadratic functions and Wang et al. (2020) recently provided an analysis for general smooth non-convex functions. One of their main observations is that LA can be viewed as an instance of local SGD (or parallel SGD, Stich, 2019; Koloskova et al., 2020; Woodworth et al., 2020b). Relying on the insights developed in these works, we can further improve upon the results by Wang et al. (2020), and we show in Appendix A that the number of gradient updates for converging to a stationary point $\mathbb{E}\|\nabla f(\boldsymbol{\omega})\|^2 \leq \varepsilon$, assuming stochastic gradient variance bounded by σ^2, can be bounded as:

$$\text{LA on smooth } f\text{: } \mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{1}{\varepsilon} + \frac{1-\alpha}{\alpha}\left(\frac{\sigma\sqrt{k-1}}{\varepsilon^{3/2}} + \frac{k}{\varepsilon}\right)\right), \quad \begin{array}{l}\text{LA on quadratic } f\text{:} \\ \text{SGD on smooth } f\text{:}\end{array} \mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2} + \frac{1}{\varepsilon}\right).$$

The asymptotically most significant term, $\mathcal{O}\left(\frac{\sigma^2}{\varepsilon^2}\right)$, of matches with the corresponding term in the SGD convergence rate for all choices of $\alpha \in (0, 1]$, and when $\alpha \to 1$, the same convergence guarantees as for SGD can be attained. When $\sigma^2 = 0$ the rate improves to $\mathcal{O}\left(\frac{1}{\epsilon}\right)$, in contrast to $\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ in (Wang et al., 2020).[2] For small values of α, the worst-case complexity estimates of LA can in general be k times worse than for SGD (except for quadratic functions, where the rates match). Deriving tighter analyses for LA that corroborate the observed practical advantages is still an open problem.

4 Lookahead for Minmax Objectives

Although the simplistic bilinear game does *not* capture all the challenges of minmax optimization, it is instructive to consider it in order to understand the behavior of the optimizers around stationary points. In this bilinear game, taking a point on a line between two points on a cyclic trajectory would reduce the distance to the optimum—as illustrated in Fig. 1—what motivates the extension of Lookahead on games. Thus we use equation LA in joint parameter space $(\boldsymbol{\theta}, \boldsymbol{\varphi})$. Alg. 1 summarizes the proposed

[2]Our results rely on optimally tuned stepsize γ for every choice of α. Wang et al. (2020) state a slightly different observation but keep the stepize γ fixed for different choices of α.

Lookahead–minmax algorithm which for clarity does *not* cover *different update ratios* for the two players–see Alg. 3 for this case. At every step $t = 1, \dots, T$, we first compute k-step "predicted" iterates $(\tilde{\boldsymbol{\theta}}_{t,k}, \tilde{\boldsymbol{\varphi}}_{t,k})$—also called *fast* weights—using an inner optimizer (stochastic GDA in Alg. 1). The so called *slow* weights $\boldsymbol{\theta}_{t+1}, \boldsymbol{\varphi}_{t+1}$ are then obtained as a point on a line between $(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$ and $(\tilde{\boldsymbol{\theta}}_{t,k}, \tilde{\boldsymbol{\varphi}}_{t,k})$—*i.e.* equation LA, see lines 10 & 11 (herein called *backtracking* or LA–step). Alg. 1 can we written *equivalently* as Alg. 3, and it differs from using it separately per player, see § D.2. Moreover, it can be applied several times using *nested* LA–steps with (different) k, see Alg. 4.

4.1 Motivating example: the bilinear game

We argue that Lookahead-minmax allows for improved stability and performance on minmax problems due to two main reasons: (i) it handles well rotations, as well as (ii) it reduces the noise due to making more conservative steps. In the following, we disentangle the two, and show in § 4.1.1 that Lookahead-minmax converges fast in the full-batch setting, without presence of noise as each parameter update uses the full dataset. Moreover, besides that the GDA algorithm is known to diverge on this example, we show that when combined with lookahead it does converge.

In § 4.1.2 we consider the challenging problem of Chavdarova et al. (2019), designed to have high variance. We show that besides therein proposed *Stochastic Variance Reduced Extragradient* (SVRE), Lookahead-minmax 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:

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

with $\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^d$ and $\boldsymbol{A} \in \mathbb{R}^{n\times d\times d}$, $n{=}d{=}100$, and we 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. As one pass we count a forward and backward pass, and all results are normalized. See § B.1 for hyperparameter choice.

4.1.1 The full-batch setting

In Fig. 3 we compare: (i) *GDA* with learning rate $\eta = 10^{-4}$ and $\eta = 0.3$ (in blue), which oscilates around the optimum with small enough step size, 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 § B); (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 § B for description of the algorithms and their implementation. Interestingly, we observe that Lookahead–Minmax allows GDA to converge on this example, and moreover speeds up the convergence of ExtraGradient.

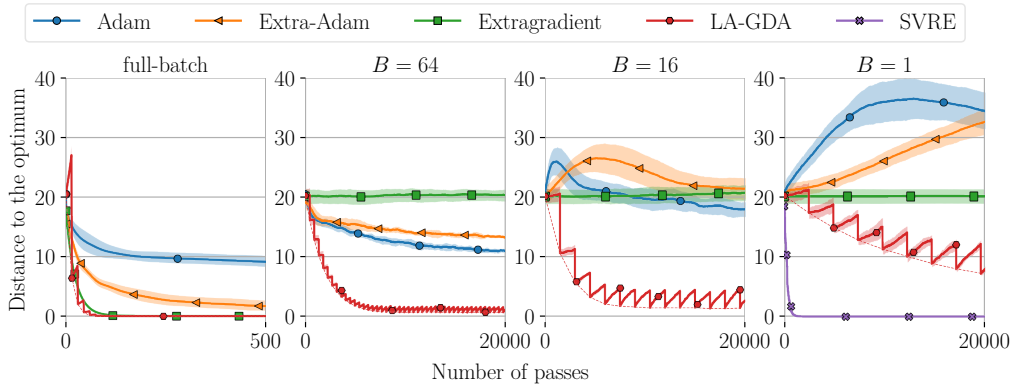


Figure 4: Convergence of *Adam*, *ExtraAdam*, *Extragradient*, *SVRE* and *LA-GDA*, on equation SB–G, for several minibatch sizes B, averaged over 5 runs–with random initialization of both the parameters and the data points ($\boldsymbol{A}$, $\boldsymbol{b}$ and $\boldsymbol{c}$). Fast $(\tilde{\boldsymbol{\varphi}}, \tilde{\boldsymbol{\theta}})$ and slow $(\boldsymbol{\varphi}, \boldsymbol{\theta})$ weights of LA–GDA are shown with solid and dashed lines, resp.

	Fréchet Inception distance ↓			Inception score ↑		
(32×32) **ImageNet**	no avg	EMA	EMA-slow	no avg	EMA	EMA-slow
AltGAN	$15.63 \pm .46$	$14.16 \pm .27$	–	$7.23 \pm .13$	$7.81 \pm .07$	–
LA–AltGAN	$14.37 \pm .20$	$\mathbf{13.06 \pm .20}$	$\mathbf{12.53 \pm .06}$	$7.58 \pm .07$	$\mathbf{7.97 \pm .11}$	$\mathbf{8.42 \pm .11}$
NLA–AltGAN	$\mathbf{13.14 \pm .25}$	$\mathbf{13.07 \pm .25}$	$12.71 \pm .13$	$\mathbf{7.85 \pm .05}$	$\mathbf{7.87 \pm .01}$	$8.10 \pm .05$
ExtraGrad	$15.48 \pm .44$	$14.15 \pm .63$	–	$7.31 \pm .06$	$7.85 \pm .10$	–
LA–ExtraGrad	$14.53 \pm .27$	$14.13 \pm .23$	$14.09 \pm .28$	$7.62 \pm .06$	$7.70 \pm .07$	$7.89 \pm .04$
NLA–ExtraGrad	$15.05 \pm .96$	$14.79 \pm .93$	$13.88 \pm .45$	$7.39 \pm .12$	$7.48 \pm .12$	$7.76 \pm .15$
CIFAR-10						
AltGAN	21.37 ± 1.60	16.92 ± 1.16	–	$7.41 \pm .16$	$8.03 \pm .13$	–
LA–AltGAN	$16.74 \pm .46$	$\mathbf{13.98 \pm .47}$	$\mathbf{12.67 \pm .57}$	$\mathbf{8.05 \pm .43}$	$\mathbf{8.19 \pm .05}$	$\mathbf{8.55 \pm .04}$
ExtraGrad	$18.49 \pm .99$	15.47 ± 1.82	–	$7.61 \pm .07$	$8.05 \pm .09$	–
LA–ExtraGrad	$\mathbf{15.25 \pm .30}$	$14.68 \pm .30$	$13.39 \pm .23$	$\mathbf{7.99 \pm .03}$	$8.04 \pm .04$	$8.40 \pm .05$
Unrolled–GAN	21.04 ± 1.08	17.51 ± 1.08	–	$7.43 \pm .07$	$7.88 \pm .12$	–
SVHN						
AltGAN	7.84 ± 1.21	6.83 ± 2.88	–	$3.10 \pm .09$	$\mathbf{3.19 \pm .09}$	–
LA–AltGAN	$3.87 \pm .09$	$\mathbf{3.28 \pm .09}$	$\mathbf{3.21 \pm .19}$	$3.16 \pm .02$	$\mathbf{3.22 \pm .08}$	$\mathbf{3.30 \pm .07}$
ExtraGrad	$4.08 \pm .11$	$\mathbf{3.22 \pm .09}$	–	$\mathbf{3.21 \pm .02}$	$\mathbf{3.16 \pm .02}$	–
LA–ExtraGrad	$\mathbf{3.20 \pm .09}$	$\mathbf{3.16 \pm .14}$	$\mathbf{3.15 \pm .31}$	$\mathbf{3.20 \pm .02}$	$\mathbf{3.19 \pm .03}$	$3.20 \pm .04$

Table 1: Comparison of LA–GAN with its respective baselines AltGAN and ExtraGrad (see § 5.1 for naming), using FID (lower is better) and IS (higher is better). EMA denotes *exponential moving average*, see § C. We use the *Adam* (Kingma & Ba, 2015) optimizer for all the methods. Results are averaged over 5 runs. We run each experiment for 500K iterations. See § E and § 5.2 for details on architectures and hyperparameters and for discussion on the results, resp. Our overall best obtained FID scores are 12.19 on CIFAR-10 and 2.823 on SVHN, see § F for samples of these generators. Best scores obtained for each metric and dataset are highlighted in yellow. For each column the best score is in bold along with any score within its standard deviation reach.

	Unconditional GANs						Conditional GANs	
	SNGAN Miyato et al.	Prog.GAN Karras et al.	NCSN Song & 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	$16.78 \pm .21$	**12.19**	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 recently reported best scores on CIFAR-10 and benchmark with LA–GAN, *using published results*. Note that the architectures are *not* identical for all the methods–see § 5.2.

4.1.2 The stochastic setting

In this section, we show that besides SVRE, Lookahead–minmax also converges on equation SB–G. 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. 4 depicts our results. See § B 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.

5 Experiments

In this section, we empirically benchmark Lookahead–minmax–Alg. 1 for *training GANs*. For the purpose of fair comparison, as an *iteration* we count each update of *both* the players, see Alg. 3.

5.1 Experimental setup

Datasets. We used the following image datasets: (i) **MNIST** (Lecun & Cortes, 1998), (ii) **CIFAR-10** (Krizhevsky, 2009, §3), (iii) **SVHN** (Netzer et al., 2011), and (iv) **ImageNet** ILSVRC 2012 (Russakovsky et al., 2015), using resolution of 28×28 for **MNIST**, and $3 \times 32 \times 32$ for the rest. **Metrics.** We use **Inception score** (IS, Salimans et al., 2016) and **Fréchet Inception distance**

(FID, Heusel et al., 2017) as these are most commonly used metrics for image synthesis, see § E.1 for details. On datasets other than ImageNet, IS is less consistent with the sample quality (see E.1.1).

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

Optimization methods. We use: (i) **AltGan:** the standard alternating GAN, (ii) **ExtraGrad:** the extragradient method, as well as (iii) **UnrolledGAN:** (Metz et al., 2017). We combine Lookahead-minmax with (i) and (ii), and we refer to these as **LA–AltGAN** and **LA–ExtraGrad**, respectively or for both as **LA–GAN** for brevity. We denote *nested* LA–GAN with prefix **NLA**, see § D.1. All methods in this section use *Adam* (Kingma & Ba, 2015). We compute Exponential Moving Average (EMA, see def. in § C) of both the fast and slow weights–called **EMA** and **EMA–slow**, resp. See Appendix for results of uniform iterate averaging and *RAdam* (Liu et al., 2020).

5.2 Results and Discussion

Comparison with baselines. Table 1 summarizes our comparison of combining Alg. 1 with AltGAN and ExtraGrad. On all datasets, we observe that the iterates (column "no avg") of LA–AltGAN and LA–ExtraGrad perform notably better than the corresponding baselines, and using EMA on LA–AltGAN and LA–ExtraGrad further improves the FID and IS scores obtained with LA–AltGAN. As the performances improve after each LA–step, see Fig. 2 computing EMA solely on the slow weights further improves the scores. It is interesting to note that on most of the datasets, the scores of the iterates of LA–GAN (column *no-avg*) outperform the EMA scores of the respective baselines. In some cases, EMA for AltGAN does not provide improvements, as the iterates diverge relatively early. In our baseline experiments, ExtraGrad outperforms AtlGAN–while requiring twice the computational time of the latter per iteration. The addition of Lookahead–minimax stabilizes AtlGAN making it competitive to LA–ExtraGrad while using *half* of the computational time.

In Table 3 we report our results on MNIST, where–different from the rest of the datasets–the training of the baselines is stable, to gain insights if LA–GAN still yields improved performances. The best FID scores of the iterates (column "no avg") are obtained with LA–GAN. Interestingly, although we obtain that the last iterates are not LSSP (which could be due to stochasticity), from Fig. 5–which depicts the eigenvalues of JVF, we observe that after convergence LA–GAN shows no rotations.

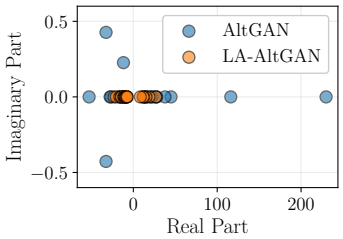
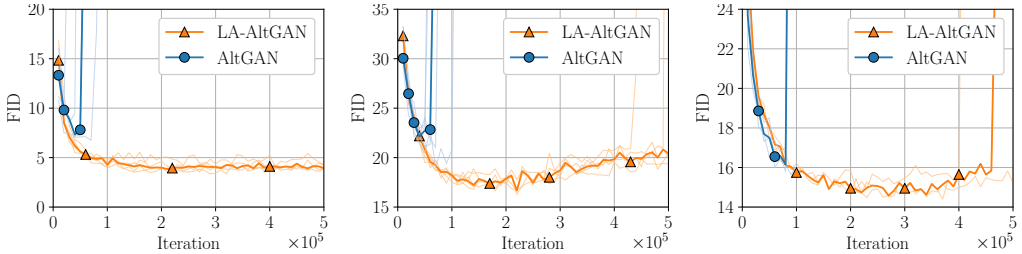


Figure 5: Eigenvalues of $v'(\boldsymbol{\theta}, \boldsymbol{\varphi})$ at 100K iterations on MNIST, see § 5.2.

	no avg	EMA	EMA-slow
AltGAN	$.094 \pm .006$	$\mathbf{.031 \pm .002}$	–
LA–AltGAN	$\mathbf{.053 \pm .004}$	$\mathbf{.029 \pm .002}$	$.032 \pm .002$
ExtraGrad	$.094 \pm .013$	$.032 \pm .003$	–
LA–ExtraGrad	$\mathbf{.053 \pm .005}$	$.032 \pm .002$	$.034 \pm .001$
Unrolled–GAN	$.077 \pm .006$	$\mathbf{.030 \pm .002}$	–

Table 3: FID (lower is better) results on MNIST, averaged over 5 runs. Each experiment is trained for 100K iterations. Note that Unrolled–GAN is computationally more expensive: in the order of the ratio 4 : 22–as we used 20 steps of unrolling what gave best results. See 5.2 & E for implementation and discussion, resp.



(a) SVHN dataset. (b) CIFAR-10 dataset. (c) ImageNet (32 × 32) dataset.

Figure 6: Improved stability of LA–AltGAN relative to its baselines on SVHN, CIFAR-10 and ImageNet, over 5 runs. The median and the individual runs are illustrated with ticker solid lines and with transparent lines, respectively. See § 5.2 for discussion.

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. The competitive result of (Yazıcı et al., 2019) of average 12.56 FID uses ~ 3.5 times larger model then ours and is omitted from Tab. 2 as it does not use the standard metrics pipeline.

Additional memory & computational cost. Lookahead-minmax requires the same extra memory footprint as EMA and uniform averaging (one additional set of parameters per player)—both of which are updated each step whereas LA—GAN is updated *once every* k iterations.

On the choice of α and k. We fixed $\alpha = 0.5$, and we experimented with several values of k–while once selected, keeping k fixed throughout the training. We observe that all values of k improve the baseline. Using larger k in large part depends on the stability of the base-optimizer: if it quickly diverges, smaller k is necessary to stabilize the training. Thus we used $k = 5$ and $k = 5000$ for LA–AltGAN and LA–ExtraGrad, resp. When using larger values of k we noticed that the obtained scores would periodically drastically improve every k iterations–after an LA step, see Fig.2. Interestingly, complementary to the results of (Berard et al., 2020)–who locally analyze the vector field, Fig.2 confirms the presence of rotation *while taking into account the used optimizer*, and empirically validates the geometric justification of Lookahead–minmax illustrated in Fig.1. The necessity of small k for unstable baselines motivates *nested* LA–Minmax using two different values of k, of which one is relatively small (the inner LA–step) and the other is larger (the outer LA–step). Intuitively, the inner and outer LA–steps in such nested LA–GAN tackle the variance and the rotations, resp.

Stability of convergence. Fig. 6 shows that LA–GAN *consistently improved the stability of its respective baseline*. Despite using $1 : 5$ update ratio for $G : D$–known to improve stability, our baselines always diverge (also reported by Chavdarova et al., 2019; Brock et al., 2019). On the other hand, LA–GAN diverged only few times and notably later in the training relative to the same baseline, see additional results in § F. The stability further improves using nested LA–steps as in Fig.2.

6 Related work

Parameter averaging. In the context of convex single-objective optimization, taking an arithmetic average of the parameters as by Polyak & 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). It 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 report the performance of a 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 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". Apart from the already introduced works, (i) Yadav et al. (2018) use prediction steps, (ii) Daskalakis et al. (2018) propose *Optimistic Mirror Decent* (OMD) for training Wasserstein GANs (Arjovsky et al., 2017), (iii) Balduzzi et al. (2018) propose the *Symplectic Gradient Adjustment* (SGA), among others. 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.

7 Conclusion

Motivated by the adversarial component of games and the negative impact of noise on games, we proposed an extension of the Lookahead algorithm to games, called "Lookahead–minmax". On the bilinear toy example we observe that combining Lookahead–minmax with standard gradient

methods converges, and that Lookahead–minmax copes very well with the 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–minmax 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. While we do *not* aim at obtaining a new state-of-the-art result, it is remarkable that Lookahead–minmax obtains competitive result on CIFAR–10 of 12.19 FID–outperforming the 30–times larger BigGAN.

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

ACKNOWLEDGMENTS

This work was in part done while TC and FF were affiliated with Idiap research institute. 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

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *ICML*, 2018.

Hugo Berard, Gauthier Gidel, Amjad Almahairi, Pascal Vincent, and Simon Lacoste-Julien. A closer look at the optimization landscapes of generative adversarial networks. In *ICLR*, 2020.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

Ronald 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.

Tatjana Chavdarova, Gauthier Gidel, François Fleuret, and Simon Lacoste-Julien. Reducing noise in GAN training with variance reduced extragradient. In *NeurIPS*, 2019.

Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training GANs with optimism. In *ICLR*, 2018.

Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. In *NeurIPS*, 2019.

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

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In *NeurIPS*, 2018.

Gauthier Gidel, Hugo Berard, Pascal Vincent, and Simon Lacoste-Julien. A variational inequality perspective on generative adversarial nets. In *ICLR*, 2019a.

Gauthier Gidel, Reyhane Askari Hemmat, Mohammad Pezeshki, Rémi Le Priol, Gabriel Huang, Simon Lacoste-Julien, and Ioannis Mitliagkas. Negative momentum for improved game dynamics. In *AISTATS*, 2019b.

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

Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

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

Kosuke Haruki, Taiji Suzuki, Yohei Hamakawa, Takeshi Toda, Ryuji Sakai, Masahiro Ozawa, and Mitsuhiro Kimura. Gradient Noise Convolution (GNC): Smoothing Loss Function for Distributed Large-Batch SGD. *arXiv:1906.10822*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

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

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018.

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

Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian U. Stich. A unified theory of decentralized SGD with changing topology and local updates. *ICML*, 2020.

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

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

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

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *ICLR*, 2020.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *ICLR*, 2018.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of GANs. In *NIPS*, 2017.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which Training Methods for GANs do actually Converge? In *ICML*, 2018.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In *ICLR*, 2017.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

Dov Monderer and Lloyd S Shapley. Potential games. *Games and economic behavior*, 1996.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011. URL `http://ufldl.stanford.edu/housenumbers/`.

Shayegan Omidshafiei, Jason Pazis, Chris Amato, Jonathan P. How, and John 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.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

David Ruppert. Efficient estimations from a slowly convergent Robbins-Monro process. *Technical report, Cornell University Operations Research and Industrial Engineering*, 1988.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016.

Christopher J Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv:1811.03600*, 2018.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, pp. 11895–11907, 2019.

Sebastian U. Stich. Local SGD converges fast and communicates little. *ICLR*, 2019.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv:1512.00567*, 2015.

Ferdinand Verhulst. *Nonlinear Differential Equations and Dynamical Systems*. 1990.

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.

Wei Wen, Yandan Wang, Feng Yan, Cong Xu, Chunpeng Wu, Yiran Chen, and Hai Li. Smoothout: Smoothing out sharp minima to improve generalization in deep learning. *arXiv:1805.07898*, 2018.

Blake Woodworth, Kumar Kshitij Patel, and Nathan Srebro. Minibatch vs local SGD for heterogeneous distributed learning. *arXiv*, 2020a.

Blake Woodworth, Kumar Kshitij Patel, Sebastian U. Stich, Zhen Dai, Brian Bullins, H. Brendan McMahan, Ohad Shamir, and Nathan Srebro. Is local SGD better than minibatch SGD? *ICML*, 2020b.

Abhay Yadav, Sohil Shah, Zheng Xu, David Jacobs, and Tom Goldstein. Stabilizing adversarial nets with prediction methods. In *ICLR*, 2018.

Yasin Yazıcı, Chuan-Sheng Foo, Stefan Winkler, Kim-Hui Yap, Georgios Piliouras, and Vijay Chandrasekhar. The unusual effectiveness of averaging in GAN training. In *ICLR*, 2019.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *NeurIPS*, 2019.

A THEORETICAL ANALYSIS OF LOOKAHEAD FOR SINGLE OBJECTIVE MINIMIZATION

In this section we give the theoretical justification for the convergence results claimed in Section 3. We consider the LA optimizer, with k SGD prediction steps, that is,

$$\tilde{\boldsymbol{\omega}}_{t+k} = \tilde{\boldsymbol{\omega}}_t - \gamma \sum_{i=1}^{k} g(\tilde{\boldsymbol{\omega}}_{t+i-1})$$

where $g\colon \mathcal{X} \to \mathbb{R}$ is a stochastic gradient estimator, $\mathbb{E}g(\boldsymbol{\omega}) = \nabla f(\boldsymbol{\omega})$, with bounded variance $\mathbb{E}\|g(\boldsymbol{\omega})\|^2 \leq \sigma^2$. Further, we assume that f is L-smooth for a constant $L \geq 0$ (but not necessarily convex). We count the total number of gradient evaluations K. This matches the setting considered in (Wang et al., 2020).

Wang et al. (2020, Theorem 2) show that if the stepsize γ satisfies

$$\alpha\gamma L + (1-\alpha)^2\gamma^2 L^2 k(k-1) \leq 1\,, \tag{1}$$

then the expected squared gradient norm of the LA iterates after T updates of the slow sequence, i.e. $K = kT$ gradient evaluations, can be bounded as

$$\mathcal{O}\left(\frac{F_0}{\alpha\gamma K} + \alpha\gamma L\sigma^2 + (1-\alpha)^2\gamma^2 L^2\sigma^2(k-1)\right) \tag{2}$$

where $F_0 := f(\omega_0) - f_{\inf}$ denotes an upper bound on the on the optimality gap. Now, departing from Wang et al. (2020), we can directly derive the claimed bound in the main text by choosing γ such as to minimze equation 2, while respecting the constraints given in equation 1 (two necessary constraints are e.g. $\gamma^2 \leq \frac{1}{(1-\alpha)^2L^2k(k-1)}$ and $\gamma \leq \frac{1}{\alpha L}$). For this minimization with respect to γ, see for instance (Koloskova et al., 2020, Lemma 15), where we plug in $r_0 = \frac{F_0}{\alpha}$, $b = \alpha L\sigma^2$, $e = (1-\alpha)^2L^2\sigma^2(k-1)$ and $d = \min\{\frac{1}{\alpha L}, \frac{1}{(1-\alpha)Lk}\}$.

The improved result for the quadratic case follows from the observations in Woodworth et al. (2020b) who analyze local SGD on quadratic functions (they show that linear update algorithms (such as local SGD on quadratics) benefit from variance reduction, allowing to recover the same convergence guarantees as single-threaded SGD. These observations also hold for non-convex quadratic functions.

We point out, that the the choice of the stepsize $\gamma = \frac{1}{\sqrt{kT}}$ proposed in Theorem 2 in (Wang et al., 2020) does not necessarily satisfy their constraint given in equation 1 for small values of T. Whilst their conclusions remain valid when k is a constant (or in the limit, $T \to \infty$), the current analyses (including our slightly improved bound) do not show that LA can in general match the performance upper bounds of SGD when k is large. Whilst casting LA as an instance of local SGD was useful to derive the first general convergence guarantees, we believe—in regard to recently derived lower bounds (Woodworth et al., 2020a) that show limitations on convex functions—that this approach is limited and more carefully designed analyses are required to derive tighter results for LA in future works.

B EXPERIMENTS ON THE BILINEAR EXAMPLE

In this section we list the details regarding our implementation of the experiments on the bilinear example of equation SB–G that were presented in § 4.1. In particular: (i) in § B.1 we list the implementational details of the benchmarked algorithms, (ii) in § B.2 and B.3 we list the hyperparameters used in § 4.1.1 and § 4.1.2, respectively and finally (iii) in § B.4 we present visualizations in aim to improve the reader's intuition on how Lookahead-minmax works on games.

B.1 IMPLEMENTATION DETAILS

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

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

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

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

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 $\boldsymbol{\varphi}^\star$ for a fixed min–player $\boldsymbol{\theta}$, *i.e.* $\boldsymbol{\varphi}^\star(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi})$ through "unrolling" as follows:

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

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

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

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 $\mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t) = -\mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$.

Adam. Adam (Kingma & 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:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \,, \tag{4}$$

where the hyperparameters $\beta_1, \beta_2 \in [0, 1]$, $m_0 = 0$, $v_0 = 0$, and t denotes the iteration $t = 1, \dots T$. 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 & Ba (2015) propose to use bias-corrected estimates of these first two moments:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{5}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{6}$$

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

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

Extra-Adam. Gidel et al. (2019a) adjust Adam for extragradient equation EG and obtain the empirically motivated *ExtraAdam* which re-uses the same running averages of equation 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 equation SB–G 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:

$$\boldsymbol{d}_{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}}) := \boldsymbol{\mu}_{\boldsymbol{\varphi}} + \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \mathcal{D}[n_d], \mathcal{Z}[n_z]) - \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}}, \mathcal{D}[n_d], \mathcal{Z}[n_z]) \tag{7}$$

$$\boldsymbol{d}_{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}}) := \boldsymbol{\mu}_{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \mathcal{Z}[n_z]) - \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}}, \mathcal{Z}[n_z]) \,, \tag{8}$$

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.

Algorithm 2 Pseudocode for Restarted SVRE.

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 $|\mathcal{D}| = |\mathcal{Z}| = n$.
Initialize: $\boldsymbol{\varphi}$, $\boldsymbol{\theta}$, $t = 0$ ▷ t is for the online average computation.
for $e = 0$ **to** $T-1$ **do**
  Draw $\texttt{restart} \sim \mathrm{B}(p)$. ▷ Check if we restart the algorithm.
  if $\texttt{restart}$ **and** $e > 0$ **then**
    $\boldsymbol{\varphi} \leftarrow \bar{\boldsymbol{\varphi}}$, $\boldsymbol{\theta} \leftarrow \bar{\boldsymbol{\theta}}$ and $t = 1$
  end if
  $\boldsymbol{\varphi}^{\mathcal{S}} \leftarrow \boldsymbol{\varphi}$ and $\boldsymbol{\mu}_{\boldsymbol{\varphi}}^{\mathcal{S}} \leftarrow \frac{1}{|\mathcal{D}|}\sum_{i=1}^{n} \nabla_{\boldsymbol{\varphi}} \mathcal{L}_i^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}^{\mathcal{S}})$
  $\boldsymbol{\theta}^{\mathcal{S}} \leftarrow \boldsymbol{\theta}$ and $\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\mathcal{S}} \leftarrow \frac{1}{|\mathcal{Z}|}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i^{\boldsymbol{\theta}}(\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}})$
  $N \sim \mathrm{Geom}\left(1/n\right)$ ▷ Length of the epoch.
  for $i = 0$ **to** $N-1$ **do**
    Sample $i_{\boldsymbol{\theta}} \sim \pi_{\boldsymbol{\theta}}$, $i_{\boldsymbol{\varphi}} \sim \pi_{\boldsymbol{\varphi}}$, do **extrapolation:**
    $\tilde{\boldsymbol{\varphi}} \leftarrow \boldsymbol{\varphi} - \eta_{\boldsymbol{\theta}} \boldsymbol{d}_{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}})$, $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} - \eta_{\boldsymbol{\varphi}} \boldsymbol{d}_{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}})$ ▷ equation 7 and equation 8
    Sample $i_{\boldsymbol{\theta}} \sim \pi_{\boldsymbol{\theta}}$, $i_{\boldsymbol{\varphi}} \sim \pi_{\boldsymbol{\varphi}}$, do **update:**
    $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} - \eta_{\boldsymbol{\theta}} \boldsymbol{d}_{\boldsymbol{\varphi}}(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\varphi}}, \boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}})$, $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\boldsymbol{\varphi}} \boldsymbol{d}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\varphi}}, \boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}})$ ▷ equation 7 and equation 8
    $\bar{\boldsymbol{\theta}} \leftarrow \frac{t}{t+1}\bar{\boldsymbol{\theta}} + \frac{1}{t+1}\boldsymbol{\theta}$ and $\bar{\boldsymbol{\varphi}} \leftarrow \frac{t}{t+1}\bar{\boldsymbol{\varphi}} + \frac{1}{t+1}\boldsymbol{\varphi}$ ▷ Online computation of the average.
    $t \leftarrow t + 1$ ▷ Increment t for the online average computation.
  end for
end for
Output: $\boldsymbol{\theta}, \boldsymbol{\varphi}$

B.2 Hyperparameters used for the full-batch setting

We now consider minmax optimization, departing from the theoretical setting in Appendix A.

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. 3 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-minmax with GDA or Extragradient, we use a k of 6 and and α of 0.5 unless otherwise emphasized.

B.3 Hyperparameters used for the stochastic setting

The hyperparameters used in the stochastic bilinear experiment of equation SB–G are listed in Table 4. 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. 4. To tune

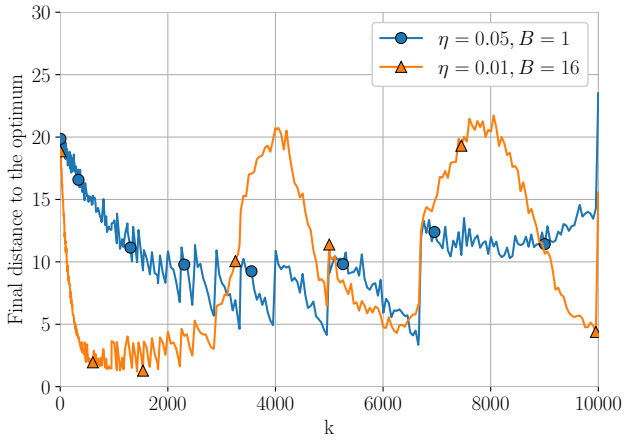


Figure 7: Sensitivity of LA-GDA to the value of the hyperparameter k in Alg. 1 for two combinations of batch sizes and η. The y-axis is the distance to the optimum at 20000 passes. The jolting of the curves is due to the final value being affected by how close it is to the last equation LA step, *i.e.* lines 10 and 11 of Alg. 1.

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 4: List of hyperparameters used in Figure 4. η denotes the learning rate, β_1 is defined in equation 3, and α and k in Alg. 1.

Fig. 7 depicts the final performance of Lookahead–minmax, 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. 7 captures the periodicity of the rotating vector field, what sheds light on future directions in finding methods with adaptive k.

B.4 Illustrations of GAN optimization with Lookahead

In Fig. 8 we consider a 2D bilinear game $\min_x \max_y x \cdot y$, and we illustrate the convergence of Lookahead–Minmax. 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–minmax converges. Moreover, we see Lookahead–minmax 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–minmax is also sensitive to the value of η, besides that it introduces additional hyperparameters k and α

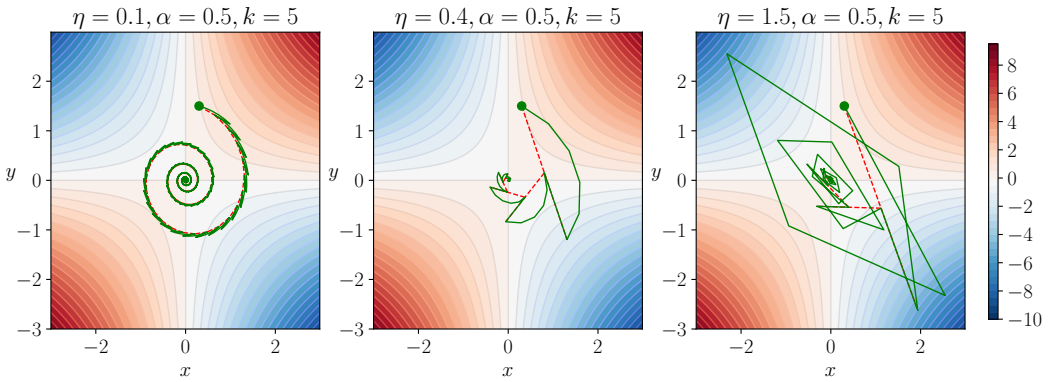


Figure 8: Illustration of Lookahead-minmax on the bilinear game $\min_x \max_y x \cdot y$, for different values of the learning rate $\eta \in \{0.1, 0.4, 1.5\}$, with fixed $k = 5$ and $\alpha = 0.5$. The trajectory of the iterates is depicted with green line, whereas the the interpolated line between $(\boldsymbol{\omega}_t, \tilde{\boldsymbol{\omega}}_{t,k})$, $t = 1, \ldots, T$, $k \in \mathbb{R}$ with $\boldsymbol{\omega}_t = (\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$ is shown with dashed red line. The transparent lines depict the level curves of the loss function, and $\boldsymbol{\omega}^\star = (0.0)$. See § B.4 for discussion.

C 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 & 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. equation 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 § 5 and is computed as follows:

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

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

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

In our experiments, following related works (Yazıcı et al., 2019; Gidel et al., 2019a; Chavdarova et al., 2019), we fix $\beta = 0.9999$ for computing equation EMA *on all the fast weights* and when computing it on the slow weights in the experiments with small k (e.g. when $k = 5$). The remaining case of computing equation EMA *on the slow weights* of the experiments that use large k produce only *few* slow weight iterates, thus using such large $\beta = 0.9999$ results in largely weighting the parameters at initialization. We used fixed $\beta = 0.8$ for those cases.

D DETAILS ON THE LOOKAHEAD–MINMAX ALGORITHM AND ALTERNATIVES

Alg. 3 reformulates Alg. 1 by replacing the inner fast-weight loop by a modulo operation of k–for the purpose of clearly demonstrating to the reader the negligible modification of the source code of the base-optimizer. Note that instead of the notation of fast weights $\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\varphi}}$ we use snapshot networks $\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}}$ (past iterates) to denote the slow weights, and the parameters $\boldsymbol{\theta}, \boldsymbol{\varphi}$ for the fast weights. Alg. 3 also covers the possibility of using different update ratio $r \in \mathbb{Z}$ for the two players. For the purpose of fair comparison, all our empirical LA–GAN results use the convention of Alg. 3, *i.e.* as one step we count one update of both players (rather than one update of the slow weights–as it is t in Alg. 1).

Algorithm 3 Alternative formulation of Lookahead–Minmax (**equivalent** to Alg. 1)

Input: Stopping time T, learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\varphi}}$, initial weights $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, lookahead hyperparameters k and α, losses $\mathcal{L}^{\boldsymbol{\theta}}$, $\mathcal{L}^{\boldsymbol{\varphi}}$, update ratio r, real–data distribution p_d, noise–data distribution p_z.
$\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}} \leftarrow \boldsymbol{\theta}, \boldsymbol{\varphi}$ *(store snapshots)*
for $t \in 1, \dots, T$ **do**
  for $i \in 1, \dots, r$ **do**
    $\boldsymbol{x} \sim p_d, \boldsymbol{z} \sim p_z$
    $\boldsymbol{\varphi} = \boldsymbol{\varphi} - \eta_{\boldsymbol{\varphi}} \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{x}, \boldsymbol{z})$ *(update $\boldsymbol{\varphi}$ r times)*
  end for
  $\boldsymbol{z} \sim p_z$
  $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{z})$ *(update $\boldsymbol{\theta}$ once)*
  if $t\%k == 0$ **then**
    $\boldsymbol{\varphi} = \boldsymbol{\varphi}^{\mathcal{S}} + \alpha_{\boldsymbol{\varphi}}(\boldsymbol{\varphi} - \boldsymbol{\varphi}^{\mathcal{S}})$ *(backtracking on interpolated line $\boldsymbol{\varphi}^{\mathcal{S}}$, $\boldsymbol{\varphi}$)*
    $\boldsymbol{\theta} = \boldsymbol{\theta}^{\mathcal{S}} + \alpha_{\boldsymbol{\theta}}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathcal{S}})$ *(backtracking on interpolated line $\boldsymbol{\theta}^{\mathcal{S}}$, $\boldsymbol{\theta}$)*
    $\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}} \leftarrow \boldsymbol{\theta}, \boldsymbol{\varphi}$ *(update snapshots)*
  end if
end for
Output: $\boldsymbol{\theta}^{\mathcal{S}}, \boldsymbol{\varphi}^{\mathcal{S}}$

Unroll-GAN Vs. Lookahead–Minmax (LA–MM). LAGAN differs from UnrolledGAN (and approximated UnrolledGAN which does not backprop through the unrolled steps) as it: (i) *does not fix one player* to update the other player k times. (ii) at each step t, it does *not* take gradient at future iterate $t + k$ to apply this gradient at the current iterate t (as extragradient does too). (iii) contrary to UnrolledGAN, LA–MM uses point on a *line between two iterates*, closeness to which is controlled with parameter α, hence deals with rotations in a different way. Note that LA–MM can be applied to UnrolledGAN, however the heavy computational cost associated to UnrolledGAN prevented us from running this experiment (see note in Tab. 3 for computational comparison).

D.1 NESTED LOOKAHEAD–MINMAX

In our experiments we explored the effect of different values of k. On one hand, we observed that less stable baselines such as Alt-GAN tend to perform better using small k (*e.g.* $k = 5$) when combined with Lookahead–Minmax, as it seems to be the necessary to prevent early divergence (what in turn achieves better iterate and EMA performances). On the other hand, we observed that stable baselines combined with Lookahead–Minmax with large k (*e.g.* $k = 10000$) tend to take better advantage of the rotational behavior of games, as indicated by the notable improvement after each LA-step, see Fig. 2 and Fig. 9.

This motivates combination of both a small *"slow"* k_s and a large *"super slow"* k_{ss}, as shown in Alg. 4. In this so called "nested" Lookahead–minimax version–denoted with *NLA* prefix, we store two copies of the each player, one corresponding to the traditional slow weights updated every k_s, and another for the so called *"super slow"* weights updated every k_{ss}. When computing EMA on the slow weights for NLA–GAN methods we use the super-slow weights as they correspond to the best performing iterates. However, better results could be obtain by computing EMA on the slow weights as EMA performs best with larger β parameter and averaging over many iterates (whereas we obtain relatively small number of super–slow weights). We empirically find Nested Lookahead–minimax to be more stable than its non-nested counterpart, see Fig.10.

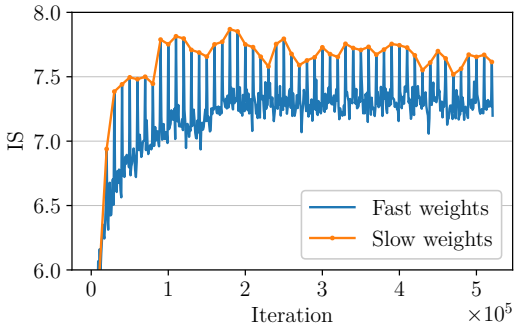


Figure 9: IS (higher is better) of LA–GAN on ImageNet with relatively large $k = 10000$. The backtracking step is significantly improving the model's performance every 10000 iterations. This shows how a large k can take advantage of the rotating gradient vector field.

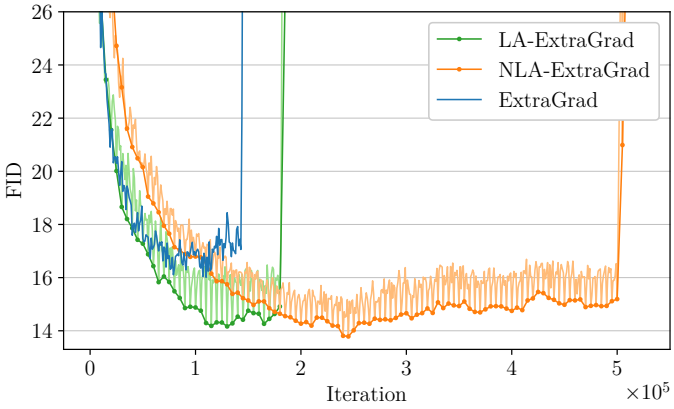


Figure 10: LA-ExtraGrad, NLA-ExtraGrad and Extragrad models trained on ImageNet. For LA-Extragrad ($k = 5000$), the lighter and darker colors represent the fast and slow weights respectively. For NLA-ExtraGrad ($k_s = 5, k_{ss} = 5000$), the lighter and darker colors represent the fast and super-slow weights respectively. In our experiments on ImageNet, while LA-ExtraGrad is more stable than ExtraGrad, it still has a tendency to diverge early. Using Alg. 4 we notice a clear improvement in stability.

D.2 Alternating–Lookahead–Minmax

Lookahead Vs. Lookahead–minmax. 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 equation 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 § D. Instead, equation 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.

For completeness, in this section we consider an alternative implementation of Lookahead-minmax, which naively applies equation 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. 5 is executed before updating $\boldsymbol{\theta}$ in line 14, and vice versa, does not allow for Lookahead to help deal with the rotations typical for games.

Algorithm 4 Pseudocode of Nested Lookahead–Minmax.

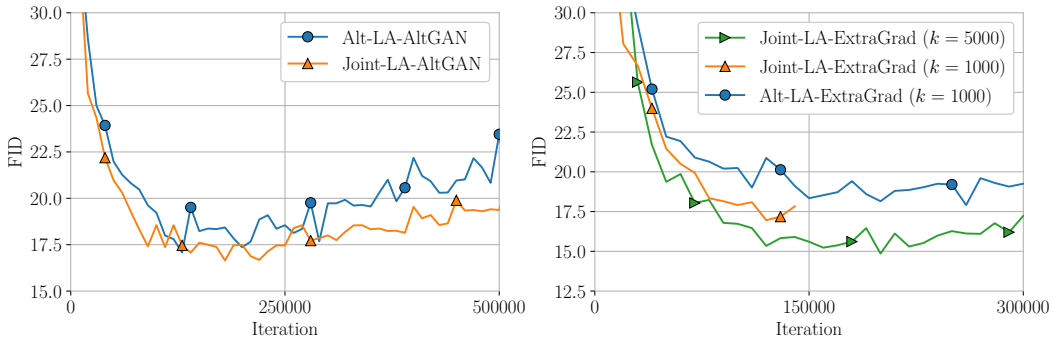
Input: Stopping time T, learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\varphi}}$, initial weights $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, lookahead hyperparameters k_s, k_{ss} and α, losses $\mathcal{L}^{\boldsymbol{\theta}}$, $\mathcal{L}^{\boldsymbol{\varphi}}$, update ratio r, real–data distribution p_d, noise–data distribution p_z.

$(\boldsymbol{\theta}_s, \boldsymbol{\theta}_{ss}, \boldsymbol{\varphi}_s, \boldsymbol{\varphi}_{ss}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{\varphi})$ *(store copies for slow and super-slow)*
for $t \in 1, \ldots, T$ **do**
  for $i \in 1, \ldots, r$ **do**
    $\boldsymbol{x} \sim p_d$, $\boldsymbol{z} \sim p_z$
    $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} - \eta_{\boldsymbol{\varphi}} \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{x}, \boldsymbol{z})$ *(update $\boldsymbol{\varphi}$ r times)*
  end for
  $\boldsymbol{z} \sim p_z$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{z})$ *(update $\boldsymbol{\theta}$ once)*
  if $t \% k_s == 0$ **then**
    $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi}_s + \alpha_{\boldsymbol{\varphi}}(\boldsymbol{\varphi} - \boldsymbol{\varphi}_s)$ *(backtracking on interpolated line $\boldsymbol{\varphi}_s$, $\boldsymbol{\varphi}$)*
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_s + \alpha_{\boldsymbol{\theta}}(\boldsymbol{\theta} - \boldsymbol{\theta}_s)$ *(backtracking on interpolated line $\boldsymbol{\theta}_s$, $\boldsymbol{\theta}$)*
    $(\boldsymbol{\theta}_s, \boldsymbol{\varphi}_s) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\varphi})$ *(update slow checkpoints)*
  end if
  if $t \% k_{ss} == 0$ **then**
    $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi}_{ss} + \alpha_{\boldsymbol{\varphi}}(\boldsymbol{\varphi} - \boldsymbol{\varphi}_{ss})$ *(backtracking on interpolated line $\boldsymbol{\varphi}_{ss}$, $\boldsymbol{\varphi}$)*
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{ss} + \alpha_{\boldsymbol{\theta}}(\boldsymbol{\theta} - \boldsymbol{\theta}_{ss})$ *(backtracking on interpolated line $\boldsymbol{\theta}_{ss}$, $\boldsymbol{\theta}$)*
    $(\boldsymbol{\theta}_{ss}, \boldsymbol{\varphi}_{ss}) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\varphi})$ *(update super-slow checkpoints)*
    $(\boldsymbol{\theta}_s, \boldsymbol{\varphi}_s) \leftarrow (\boldsymbol{\theta}, \boldsymbol{\varphi})$ *(update slow checkpoints)*
  end if
end for
Output: $\boldsymbol{\theta}_{ss}$, $\boldsymbol{\varphi}_{ss}$

Algorithm 5 Alternating Lookahead-minmax pseudocode.

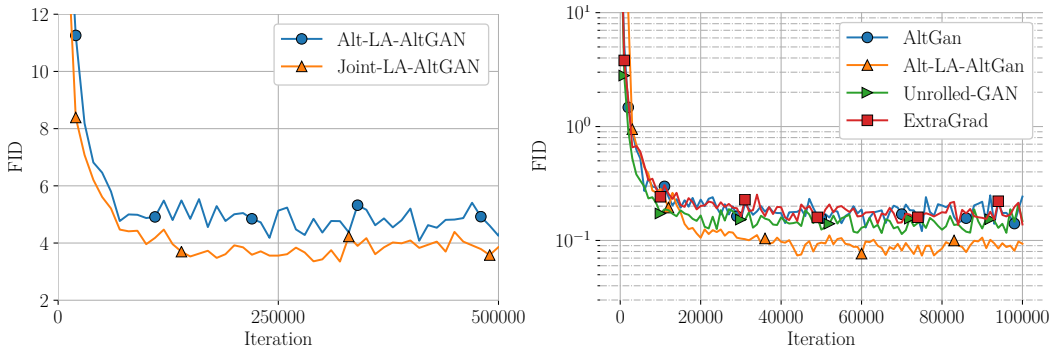
Input: Stopping time T, learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\varphi}}$, initial weights $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, $k_{\boldsymbol{\theta}}$, $k_{\boldsymbol{\varphi}}$, $\alpha_{\boldsymbol{\theta}}$, $\alpha_{\boldsymbol{\varphi}}$, losses $\mathcal{L}^{\boldsymbol{\theta}}$, $\mathcal{L}^{\boldsymbol{\varphi}}$, update ratio r, real–data distribution p_d, noise–data distribution p_z.
$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$ *(store copy)*
$\tilde{\boldsymbol{\varphi}} \leftarrow \boldsymbol{\varphi}$
for $t \in 0, \ldots, T-1$ **do**
  for $i \in 1, \ldots, r$ **do**
    $\boldsymbol{x} \sim p_d$, $\boldsymbol{z} \sim p_z$
    $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} - \eta_{\boldsymbol{\varphi}} \nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{x}, \boldsymbol{z})$ *(update $\boldsymbol{\varphi}$ k times)*
    if $(t * r + i) \% k_{\boldsymbol{\varphi}} == 0$ **then**
      $\boldsymbol{\varphi} \leftarrow \tilde{\boldsymbol{\varphi}} + \alpha_{\boldsymbol{\varphi}}(\boldsymbol{\varphi} - \tilde{\boldsymbol{\varphi}})$ *(backtracking on line $\tilde{\boldsymbol{\varphi}}$, $\boldsymbol{\varphi}$)*
      $\tilde{\boldsymbol{\varphi}} \leftarrow \boldsymbol{\varphi}$
    end if
  end for
  $\boldsymbol{z} \sim p_z$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{z})$ *(update $\boldsymbol{\theta}$ once)*
  if $t \% k_{\boldsymbol{\theta}} == 0$ **then**
    $\boldsymbol{\theta} \leftarrow \tilde{\boldsymbol{\theta}} + \alpha_{\boldsymbol{\theta}}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})$ *(backtracking on line $\tilde{\boldsymbol{\theta}}$, $\boldsymbol{\theta}$)*
    $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$
  end if
end for
Output: $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$

On SVHN and CIFAR-10, the joint Lookahead-minmax consistently gave us the best results, as can be seen in Figure 11 and 12. On MNIST, the alternating and joint implementations worked equally well, see Figure 12.



(a) LA–AltGAN. (b) LA–ExtraGrad.

Figure 11: Comparison between different extensions of Lookahead to games on CIFAR-10. We use prefix *joint* and *alt* to denote Alg. 1 and Alg. 5, respectively of which the former is the one presented in the main paper. We can see some significant improvements in FID when using the joint implementation, for both LA-AltGAN (left) and LA-ExtraGrad (right).



(a) SVHN dataset. (b) MNIST dataset.

Figure 12: (a): Comparison of the joint Lookahead-minmax implementation (*Joint* prefix, see Algorithm 1) and the alternating Lookahead-minmax implementation (*Alt* prefix, see Algorithm 5) on the SVHN dataset. (b): Results obtained with the different methods introduced in §5 as well as an alternating implementation of Lookahead-minmax, 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.

Figure 13: 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.

E Details on the implementation

For our experiments, we used the PyTorch[3] deep learning framework. For experiments on CIFAR-10 and SVHN, we compute the FID and IS metrics using the provided implementations in Tensorflow[4] for consistency with related works. For experiments on ImageNet we use a faster-running PyTorch implementation[5] of FID and IS by (Brock et al., 2019) which allows for more frequent evaluation.

E.1 Metrics

We provide more details about the metrics enumerated in § 5. Both FID and IS use: (i) the *Inception v3 network* (Szegedy et al., 2015) that has been trained on the ImageNet dataset consisting of $\sim$1 million RGB images of 1000 classes, $C = 1000$. (ii) a sample of m generated images $x \sim p_g$, where usually $m = 50000$.

E.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:

$$IS(G) = \exp\left(\mathbb{E}_{x\sim p_g}[\mathbb{D}_{KL}(p(y|x)||p(y))]\right) = \exp\left(\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C} p(y_c|x_i)\log\frac{p(y_c|x_i)}{p(y_c)}\right). \quad (9)$$

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 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 CIFAR-10 and SVHN, we used the original implementation[6] 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. 13 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.

[3] https://pytorch.org/

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

[5] https://github.com/ajbrock/BigGAN-PyTorch

[6] https://github.com/openai/improved-gan/

Generator	Discriminator
Input: $z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$	*Input:* $x \in \mathbb{R}^{1\times28\times28}$
transposed conv. (ker: 3×3, $128 \rightarrow 512$; stride: 1)	conv. (ker: 4×4, $1 \rightarrow 64$; stride: 2; pad:1)
Batch Normalization	LeakyReLU (negative slope: 0.2)
ReLU	conv. (ker: 4×4, $64 \rightarrow 128$; stride: 2; pad:1)
transposed conv. (ker: 4×4, $512 \rightarrow 256$, stride: 2)	Batch Normalization
Batch Normalization	LeakyReLU (negative slope: 0.2)
ReLU	conv. (ker: 4×4, $128 \rightarrow 256$; stride: 2; pad:1)
transposed conv. (ker: 4×4, $256 \rightarrow 128$, stride: 2)	Batch Normalization
Batch Normalization	LeakyReLU (negative slope: 0.2)
ReLU	conv. (ker: 3×3, $256 \rightarrow 1$; stride: 1)
transposed conv. (ker: 4×4, $128 \rightarrow 1$, stride: 2, pad: 1)	$Sigmoid(\cdot)$
$Tanh(\cdot)$	

Table 5: 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.

E.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:

$$\mathbb{D}_{\text{FID}}(p_d, p_g) \approx d^2((\boldsymbol{m}_d, C_d), (\boldsymbol{m}_g, C_g)) = \|\boldsymbol{m}_d - \boldsymbol{m}_g\|_2^2 + Tr(C_d + C_g - 2(C_d C_g)^{\frac{1}{2}}), \quad (10)$$

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[7] in Tensorflow, along with the provided statistics of the datasets.

E.2 ARCHITECTURES & HYPERPARAMETERS

Description of the architectures. We describe the models we used in the empirical evaluation of Lookahead-minmax 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 & Szegedy, 2015) and Spectral Normalization layers (Miyato et al., 2018).

E.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 5, and the parameters of the models are initialized using PyTorch default initialization. For experiments on this dataset, we used the *non saturating* GAN loss as proposed (Goodfellow et al., 2014):

$$\mathcal{L}_D = \mathbb{E}_{x\sim p_d} \log(D(x)) + \mathbb{E}_{z\sim p_z} \log(D(G(z))) \quad (11)$$

$$\mathcal{L}_G = \mathbb{E}_{z\sim p_z} \log(D(G(z))), \quad (12)$$

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

E.2.2 RESNET ARCHITECTURES FOR IMAGENET, 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 7. 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.

[7] https://github.com/bioinf-jku/TTUR

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:2×2)], if $\ell = 1$
conv. (ker: 1×1, $3_{\ell=1}/128_{\ell\neq1} \rightarrow 128$; stride: 1)
Spectral Normalization
[AvgPool (ker:2×2, 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:2×2)

Table 6: ResNet blocks used for the ResNet architectures (see Table 7), 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., 2016). 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:8×8)
conv. (ker: 3×3, $256 \rightarrow 3$; stride: 1; pad:1)	Linear($128 \rightarrow 1$)
$Tanh(\cdot)$	Spectral Normalization

Table 7: *Deep* ResNet architectures used for experiments on **ImageNet**, **SVHN** and **CIFAR-10**, where G–ResBlock and D–ResBlock for the Generator (left) and the Discriminator (right), respectively, are described in Table 6. The models' parameters are initialized using the Xavier initialization (Glorot & Bengio, 2010). For ImageNet experiments, the generator's input is of dimension 512 instead of 128.

E.2.3 UNROLLING IMPLEMENTATION

In Section B.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 $\nabla_{\boldsymbol{\varphi}} \mathcal{L}^{\boldsymbol{\varphi}}(\boldsymbol{\theta}_t^m, \boldsymbol{\varphi}_t)$, 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[8] 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.

E.2.4 HYPERPARAMETERS USED ON MNIST

Table 8 lists the hyperparameters that we used for our experiments on the MNIST dataset.

Table 8: Hyperparameters used on MNIST.

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

E.2.5 HYPERPARAMETERS USED ON SVHN

Table 9: 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 9 lists the hyperparameters used for experiments on SVHN. These values were selected for each algorithm *independently* after tuning the hyperparameters for the baseline.

[8] `https://github.com/facebookresearch/higher`

E.2.6 HYPERPARAMETERS USED ON CIFAR-10

Table 10: Hyperparameters that we used for our experiments 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

The reported results on CIFAR-10 were obtained using the hyperparameters listed in Table 10. 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.2.7 HYPERPARAMETERS USED ON IMAGENEET

Table 11: Hyperparameters used for our AltGAN experiments on ImageNet.

Parameter	AltGAN	LA-AltGAN	DLA-AltGAN
η_G	0.0002	0.0002	0.0002
η_D	0.0002	0.0002	0.0002
Adam β_1	0.0	0.0	0.0
Batch-size	256	256	256
Update ratio r	5	5	5
Lookahead k	-	5	5
Lookahead k'	-	-	10000
Lookahead α	-	0.5	0.5
Unrolling steps	-	-	-

Table 12: Hyperparameters used for our ExtraGrad experiments on ImageNet.

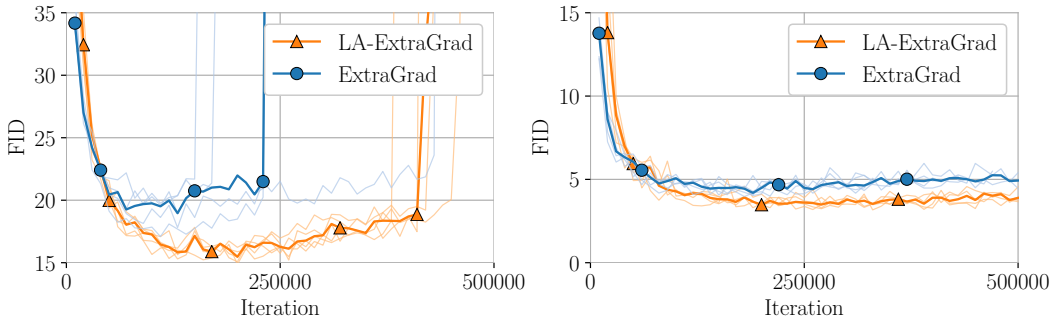
Parameter	ExtraGrad	LA-ExtraGrad	DLA-ExtraGrad
η_G	0.0002	0.0002	0.0002
η_D	0.0002	0.0002	0.0002
Adam β_1	0.0	0.0	0.0
Batch-size	256	256	256
Update ratio r	5	5	5
Lookahead k	-	5	5
Lookahead k'	-	-	5000
Lookahead α	-	0.5	0.5
Unrolling steps	-	-	-

The reported results on ImageNet were obtained using the hyperparameters listed in Tables 11 and 12.

F ADDITIONAL EXPERIMENTAL RESULTS

In Fig. 6 we compared the stability of LA–AltGAN methods against their AltGAN baselines on both the CIFAR-10 and SVHN datasets. Analogously, in Fig. 14 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. 14a, and (ii) when the stability of the baseline is relatively good as on the SVHN dataset, LA–Extragradient still improves its performances, see Fig. 14b.

For completeness to Fig. 5 in Fig 15 we report the respective eigenvalue analyses on MNIST, that are summarized in the main paper.

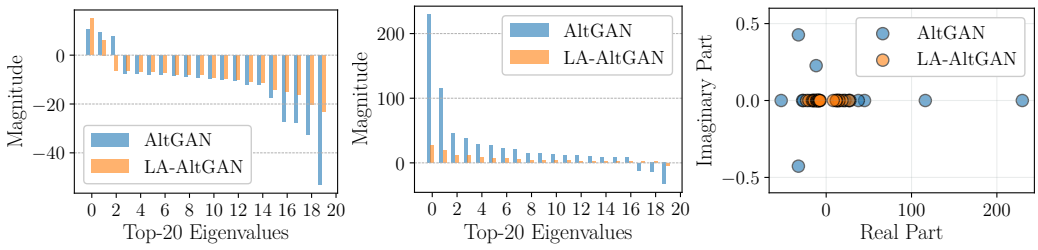


(a) CIFAR-10 dataset. (b) SVHN dataset.

Figure 14: 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 § F and E for discussion and details on the implementation, resp.

F.1 COMPLETE BASELINE COMPARISON

In § 5.2 we omitted uniform averaging of the iterates for clarity of the presented results–selected as it down-performs the exponential moving average in our experiments. In this section, for completeness we report the uniform averaging results. Table 13 lists these results, including experiments using the RAdam (Kingma & Ba, 2015) optimizer instead of Adam.



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

Figure 15: Analysis on MNIST at 100K iterations. Fig. 15a & 15b: Largest 20 eigenvalues of the Hessian of the generator and the discriminator. Fig. 15c: Eigenvalues of the Jacobian of JVF, indicating no rotations at the point of convergence of LA–AltGAN (see § 2).

CIFAR-10	Fréchet Inception distance			Inception score		
Method	no avg	uniform avg	EMA	no avg	uniform avg	EMA
AltGAN–R	23.27 ± 1.65	19.81 ± 2.58	17.82 ± 1.31	$7.32. \pm .30$	$8.14 \pm .20$	$7.99 \pm .13$
LA–AltGAN–R	$17.31 \pm .58$	16.68 ± 1.45	16.68 ± 1.45	$7.81 \pm .09$	$8.61 \pm .08$	$8.13 \pm .07$
ExtraGrad–R	19.15 ± 1.13	$16.17 \pm .63$	$15.40 \pm .94$	$7.59 \pm .13$	$8.55 \pm .05$	$8.05 \pm .15$
LA–ExtraGrad–R	$15.38 \pm .76$	$14.75 \pm .61$	$14.99 \pm .66$	$7.93 \pm .05$	$8.51 \pm .08$	$8.01 \pm .09$
AltGAN–A	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–A	$16.74 \pm .46$	$15.02 \pm .81$	$13.98 \pm .47$	$8.05 \pm .43$	$8.45 \pm .32$	$8.19 \pm .05$
ExtraGrad–A	$18.49 \pm .99$	16.22 ± 1.59	15.47 ± 1.82	$7.61 \pm .07$	$8.46 \pm .08$	$8.05 \pm .09$
LA–ExtraGrad–A	$15.25 \pm .30$	$14.95 \pm .44$	$14.68 \pm .30$	$7.99 \pm .03$	$8.13 \pm .18$	$8.04 \pm .04$
Unrolled–GAN–A	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–A	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–A	$3.87 \pm .09$	10.84 ± 1.04	$3.28 \pm .09$	$3.16 \pm .02$	$3.38 \pm .09$	$3.22 \pm .08$
ExtraGrad–A	$4.08 \pm .11$	8.89 ± 1.07	$3.22 \pm .09$	$3.21 \pm .02$	$3.21 \pm .04$	$3.16 \pm .02$
LA–ExtraGrad–A	$3.20 \pm .09$	7.66 ± 1.54	$3.16 \pm .14$	$3.20 \pm .02$	$3.32 \pm .13$	$3.19 \pm .03$
MNIST						
AltGAN–A	$.094 \pm .006$	$.167 \pm .033$	$.031 \pm .002$	$8.92 \pm .01$	$8.88 \pm .02$	$8.99 \pm .01$
LA–AltGAN–A	$.053 \pm .004$	$.176 \pm .024$	$.029 \pm .002$	$8.93 \pm .01$	$8.92 \pm .01$	$8.96 \pm .02$
ExtraGrad–A	$.094 \pm .013$	$.182 \pm .024$	$.032 \pm .003$	$8.90 \pm .01$	$8.88 \pm .03$	$8.98 \pm .01$
LA–ExtraGrad–A	$.053 \pm .005$	$.180 \pm .024$	$.032 \pm .002$	$8.91 \pm .01$	$8.92 \pm .02$	$8.95 \pm .01$
Unrolled–GAN–A	$.077 \pm .006$	$.224 \pm .016$	$.030 \pm .002$	$8.91 \pm .02$	$8.91 \pm .02$	$8.99 \pm .01$

Table 13: Comparison of the LA-GAN optimizer with its respective baselines AltGAN and ExtraGrad (see § 5.1 for naming), using FID (lower is better) and IS (higher is better). EMA denotes *exponential moving average* (with fixed $\beta = 0.9999$, see § C). With suffix –R and –A we denote that we use *RAdam* (Liu et al., 2020) and *Adam* (Kingma & Ba, 2015) optimizer, respectively. 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 § E and § 5.2 for details on architectures and hyperparameters and for discussion on the results, resp.

F.2 Samples of LA–GAN Generators

In this section we show random samples of the generators of our LAGAN experiments trained on ImageNet, CIFAR-10 and SVHN.

The samples in Fig. 16 are generated by our best performing LA-AltGAN models trained on CIFAR-10. Similarly, Fig. 17 & 18 depict such samples of generators trained on ImageNet and SVHN, respectively.

Figure 16: Samples generated by our best performing trained generator on **CIFAR-10**, using **LA-AltGAN** and exponential moving average (EMA) on the *slow* weights. The obtained FID score is 12.193.

Figure 17: Images generated by our best model trained on **32×32 ImageNet**, obtained with **LA-AltGAN** and EMA of the *slow weights*, yielding FID of 12.44.

Figure 18: Images generated by one of our best **LA-ExtraGrad** & EMA model (FID of 2.94) trained on **SVHN**.