Complex Momentum for Optimization in Games

Jonathan Lorraine[1,2] **David Acuna**[1,3] **Paul Vicol**[1,2] **David Duvenaud**[1,2]
University of Toronto[1] Vector Institute[2] NVIDIA[3]

Abstract

We generalize gradient descent with momentum for optimization in differentiable games to have complex-valued momentum. We give theoretical motivation for our method by proving convergence on bilinear zero-sum games for simultaneous and alternating updates. Our method gives real-valued parameter updates, making it a drop-in replacement for standard optimizers. We empirically demonstrate that complex-valued momentum can improve convergence in realistic adversarial games—like generative adversarial networks—by showing we find better solutions with an almost identical computational cost. We also show a practical complex-valued Adam variant, which we use to train BigGAN to improve inception scores on CIFAR-10.

1 Introduction

Gradient-based optimization has been critical for the success of machine learning, updating a single set of parameters to minimize a single loss. A growing number of applications require learning in games, which generalize single-objective optimization. Common examples are GANs (Goodfellow et al., 2014), actor-critic models (Pfau and Vinyals, 2016), curriculum learning (Baker et al., 2019; Balduzzi et al., 2019; Sukhbaatar et al., 2018), hyperparameter optimization (Lorraine and Duvenaud, 2018; Lorraine et al., 2020; MacKay et al., 2019; Raghu et al., 2020), adversarial examples (Bose et al., 2020; Yuan et al., 2019), learning models (Rajeswaran et al., 2020; Abachi et al., 2020; Nikishin et al., 2021), domain adversarial adaptation (Acuna et al., 2021), neural architecture search (Grathwohl et al., 2018; Adam and Lorraine, 2019), and meta-learning (Ren et al., 2018, 2020).

Games consist of multiple players, each with parameters and objectives. We often want solutions where no player gains from changing their strategy unilaterally, e.g., Nash equilibria (Morgenstern and Von Neumann, 1953) or Stackelberg equilibria (Von Stackelberg, 2010). Classical gradient-based learning often fails to find these equilibria due to rotational dynamics (Berard et al., 2019). Numerous saddle point finding algorithms for zero-sum games have been proposed (Arrow et al., 1958; Freund and Schapire, 1999).

Gidel et al. (2019) generalize GD with momentum to games, showing we can use a negative momentum to converge if the eigenvalues (EVals) of the Jacobian of the gradient vector field have a large imaginary part. We use terminology in Gidel et al. (2019) and say *(purely) cooperative or adversarial games* for games with (purely) real or imaginary EVals. Setups like GANs are not purely adversarial, but rather have both *purely cooperative and adversarial eigenspaces* (ESpaces) – i.e., ESpaces with purely real or imaginary EVals. More generally, we simple say an ESpace is cooperative when the real part of the EVal is larger than the imaginary part and adversarial otherwise. In cooperative ESpaces classical optimization methods perform best, while in adversarial ESpaces methods customized for games work best – see Fig. 4.

We want a method that converges with simultaneous and alternating updates in purely adversarial games – a setup where existing momentum methods fail. Also, we want a method that robustly converges with different mixtures of adversarial and cooperative ESpaces – see Fig. 5 – because finding all ESpaces depends on an eigendecomposition which can be intractable. To solve this we unify and generalize existing momentum methods (Lucas et al., 2018; Gidel et al., 2019) to recurrently linked momentum – a setup with multiple recurrently linked momentum buffers with potentially negative coefficients – see Fig. 7c.

Selecting two of these recurrently linked buffers with appropriate momentum coefficients can be interpreted as the real and imaginary parts of a single *complex buffer* and *complex momentum coefficient* – see App. Fig. 7d. This setup (a) allows us to converge in adversarial games with simultaneous updates, (b) only introduces one new optimizer parameter – the phase or

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

Actual JAX implementation: changes in green

```
mass = .8 + .3j
def momentum(step_size, mass):
 ...
 def update(i, g, state):
  x, velocity = state
  velocity = mass * velocity + g
  x=x-jnp.real(step_size(i)*velocity)
  return x, velocity
 ...
```

Figure 1: How to modify JAX's SGD with momentum here to use complex momentum. The only changes are in green. `jnp.real` gets the real part of `step_size` times the momentum buffer (called `velocity` here). We use a complex `mass` for our method in this case $\beta = |\beta| \exp(i \arg(\beta)) = 0.9 \exp(i\pi/8) \approx .8 + .3i$.

arg of our momentum, (c) allows us to gain intuitions via complex analysis, (d) is trivial to implement in libraries supporting complex arithmetic, and (e) robustly converges for different ESpace mixtures.

Intuitively, our complex buffer stores historical gradient information, oscillating between adding or subtracting at a frequency dictated by the momentum coefficient. Classical momentum only adds gradients, and negative momentum changes between adding or subtracting each iteration, while we oscillate at an arbitrary (fixed) frequency – see Fig. 2a. This reduces rotational dynamics during training by canceling out opposing updates.

Our contributions include:

- Providing generalizations and variants of classical (Polyak, 1964), negative (Gidel et al., 2019), and aggregated (Lucas et al., 2018) momentum for learning in differentiable games.
- Showing our method converges on adversarial games, including bilinear zero-sum games and Dirac-GAN, with simultaneous and alternating updates.
- Illustrating a robustness during optimization, converging faster and over a larger range of mixtures of cooperative and adversarial games than existing first-order methods.
- Giving a practical extension of our method to a complex-valued Adam (Kingma and Ba, 2014) variant, which we use to train a BigGAN (Brock et al., 2018) on CIFAR-10, improving their inception scores.

2 Background

Appendix Table 2 summarizes our notation. Consider the optimization problem:

$$\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{1}$$

We can find local minima of loss $\mathcal{L}$ using (stochastic) gradient descent with step size α. We denote the loss gradient at parameters $\boldsymbol{\theta}^j$ by $\boldsymbol{g}^j := \boldsymbol{g}(\boldsymbol{\theta}^j) := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}^j}$.

$$\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^j - \alpha \boldsymbol{g}^j \tag{SGD}$$

Momentum can generalize SGD. For example, Polyak's Heavy Ball (Polyak, 1964):

$$\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^j - \alpha \boldsymbol{g}^j + \beta(\boldsymbol{\theta}^j - \boldsymbol{\theta}^{j-1}) \tag{2}$$

Which can be equivalently written with momentum buffer $\boldsymbol{\mu}^j = {}^{(\boldsymbol{\theta}^j - \boldsymbol{\theta}^{j-1})}/_{\alpha}$.

$$\boldsymbol{\mu}^{j+1} = \beta \boldsymbol{\mu}^j - \boldsymbol{g}^j, \qquad \boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^j + \alpha \boldsymbol{\mu}^{j+1} \tag{SGDm}$$

We can also generalize SGDm to aggregated momentum (Lucas et al., 2018), shown in App. Alg. 3.

2.1 Game Formulations

Another class of problems is learning in *games*, which includes problems like generative adversarial networks (GANs). We focus on 2-player games —with players denoted by A and B—where each player minimizes their loss $\mathcal{L}_A, \mathcal{L}_B$ with their parameters $\boldsymbol{\theta}_A, \boldsymbol{\theta}_B$. If $\mathcal{L}_A$ and $\mathcal{L}_B$ are differentiable in $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$ we say the game is differentiable. In deep learning, losses are non-convex with many parameters, so we focus on finding local solutions. If $\boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A)$ denotes player B's best-response function, then solutions can be defined as:

$$\begin{aligned} \boldsymbol{\theta}_A^* &:= \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A)), \\ \boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A) &:= \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B) \end{aligned} \tag{3}$$

We may be able to approximately find $\boldsymbol{\theta}_A^*$ efficiently if we can do SGD on:

$$\mathcal{L}_A^*(\boldsymbol{\theta}_A) := \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A)) \tag{4}$$

Unfortunately, SGD would require computing ${}^{d\mathcal{L}_A^*}/_{d\boldsymbol{\theta}_A}$, which often requires ${}^{d\boldsymbol{\theta}_B^*}/_{d\boldsymbol{\theta}_A}$, but $\boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A)$ and its Jacobian are typically intractable. A common optimization algorithm to analyze for finding solutions is simultaneous SGD (SimSGD) – sometimes called gradient descent ascent for zero-sum games – where $\boldsymbol{g}_A^j := \boldsymbol{g}_A(\boldsymbol{\theta}_A^j, \boldsymbol{\theta}_B^j)$ and $\boldsymbol{g}_B^j := \boldsymbol{g}_B(\boldsymbol{\theta}_A^j, \boldsymbol{\theta}_B^j)$ are estimators for $\nabla_{\boldsymbol{\theta}_A} \mathcal{L}_A|_{\boldsymbol{\theta}_A^j, \boldsymbol{\theta}_B^j}$ and $\nabla_{\boldsymbol{\theta}_B} \mathcal{L}_B|_{\boldsymbol{\theta}_A^j, \boldsymbol{\theta}_B^j}$:

$$\boldsymbol{\theta}_A^{j+1} = \boldsymbol{\theta}_A^j - \alpha \boldsymbol{g}_A^j, \quad \boldsymbol{\theta}_B^{j+1} = \boldsymbol{\theta}_B^j - \alpha \boldsymbol{g}_B^j \tag{SimSGD}$$

We simplify notation with the concatenated or joint-parameters $\boldsymbol{\omega} := [\boldsymbol{\theta}_A, \boldsymbol{\theta}_B] \in \mathbb{R}^d$ and the joint-gradient vector field $\hat{\boldsymbol{g}} : \mathbb{R}^d \to \mathbb{R}^d$, which at the j^{th} iteration is the joint-gradient denoted:

$$\hat{\boldsymbol{g}}^j := \hat{\boldsymbol{g}}(\boldsymbol{\omega}^j) := [\boldsymbol{g}_A(\boldsymbol{\omega}^j), \boldsymbol{g}_B(\boldsymbol{\omega}^j)] = [\boldsymbol{g}_A^j, \boldsymbol{g}_B^j] \tag{5}$$

We extend to n-player games by treating $\boldsymbol{\omega}$ and $\hat{\boldsymbol{g}}$ as concatenations of the players' parameters and loss gradients, allowing for a concise expression of the SimSGD update with momentum (SimSGDm):

$$\boldsymbol{\mu}^{j+1} = \beta\boldsymbol{\mu}^{j} - \hat{\boldsymbol{g}}^{j}, \boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^{j} + \alpha\boldsymbol{\mu}^{j+1} \quad \text{(SimSGDm)}$$

Gidel et al. (2019) show classical momentum choices of $\beta \in [0, 1)$ do not improve convergence rate over SimSGD in some games, while negative momentum helps if the Jacobian of the joint-gradient vector field $\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}}$ has complex EVals. Thus, for purely adversarial games with imaginary EVals, any non-negative momentum and step size will not converge. For cooperative games – i.e., minimization – $\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}}$ has real EVals because it is a Hessian of a loss, so classical momentum works well.

2.2 Limitations of Existing Methods

Higher-order: Methods using higher-order gradients are often harder to parallelize across GPUs (Osawa et al., 2019), get attracted to bad saddle points (Mescheder et al., 2017), require estimators for inverse Hessians (Schäfer and Anandkumar, 2019; Wang et al., 2019), are complicated to implement, have numerous optimizer parameters, and can be more expensive in iteration and memory cost (Hemmat et al., 2020; Wang et al., 2019; Schäfer and Anandkumar, 2019; Schäfer et al., 2020; Czarnecki et al., 2020; Zhang et al., 2020a). Instead, we focus on first-order methods.

First-order: Some first-order methods such as extra-gradient (Korpelevich, 1976) require a second, costly, gradient evaluation per step. Similarly, methods alternating player updates must wait until after the first player's gradient is used to evaluate the second player's gradient. But, many deep learning setups can parallelize computation of both players' gradients, making alternating updates effectively cost another gradient evaluation. We want are interested in convergence speed in number of gradient queries, so we naturally looked at methods which updates with the effective cost of one gradient evaluation – see App. Fig. 9. Also, simultaneous updates are a standard choice in some settings (Acuna et al., 2021).

Robust convergence: We want our method to converge in purely adversarial game's with simultaneous updates – a setup where existing momentum methods fail (Gidel et al., 2019). Furthermore, computing a games eigendecomposition is often infeasibly expensive, so we want methods that robustly converge over different mixtures of adversarial and cooperative ESpaces – see Fig. 5. We are interested in ESpace mixtures that are relevant during GAN training – see Fig. 6 and App. Fig. 10.

2.3 Coming up with our Method

Combining existing methods: Given the preceding limitations, we would like a robust first-order method using a single, simultaneous gradient evaluation. We looked at combining aggregated (Lucas et al., 2018) with negative (Gidel et al., 2019) momentum by allowing negative coefficients, because these methods are first-order and use a single gradient evaluation – see Fig. 7b. Also, aggregated momentum provides robustness during optimization by converging quickly on problems with wide range of conditioning, while negative momentum works in adversarial setups. We hoped to combine their benefits, gaining robustness to different mixtures of adversarial and cooperative ESpaces. However, with this setup we could not find solutions that converge with simultaneous updates in purely adversarial games.

Generalize to allow solutions: We generalized the setup to allow recurrent connections between momentum buffers, with potentially negative coefficients – see Fig. 7c and App. Alg. 4. There are optimizer parameters so this converges with simultaneous updates in purely adversarial games, while being first-order with a single gradient evaluation – see Corollary 1. However, in general, this setup could introduce many optimizer parameters, have unintuitive behavior, and not be amenable to analysis. So, we choose a special case of this method to help solve these problems.

A simple solution: With two momentum buffers and correctly chosen recurrent weights, we can conveniently interpret our buffers as the real and imaginary part of one complex buffer – see App. Fig. 7d. This method is (a) capable of converging in purely adversarial games with simultaneous updates – Corollary 1, (b) only introduces one new optimizer parameter – the phase of the momentum coefficient, (c) is tractable to analyze and have intuitions for with Euler's formula – ex., Eq. (8), (d) is trivial to implement in libraries supporting complex arithmetic – see Fig. 1, and (e) can be robust to games with different mixtures of cooperative and adversarial ESpaces – see Figs. 4 and 5.

3 Complex Momentum

We describe our proposed method, where the momentum coefficient $\beta \in \mathbb{C}$, step size $\alpha \in \mathbb{R}$, momentum buffer $\boldsymbol{\mu} \in \mathbb{C}^d$, and player parameters $\boldsymbol{\omega} \in \mathbb{R}^d$. The simultaneous (or Jacobi) update is:

$$\boldsymbol{\mu}^{j+1} = \beta\boldsymbol{\mu}^{j} - \hat{\boldsymbol{g}}^{j}, \quad \boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^{j} + \Re(\alpha\boldsymbol{\mu}^{j+1}) \quad \text{(SimCM)}$$

There are many ways to get a real-valued update from $\boldsymbol{\mu} \in \mathbb{C}$, but we only consider updates equivalent to classical momentum when $\beta \in \mathbb{R}$. It is possible for the update to use the imaginary part of the momentum

Algorithm 1 SimCM Momentum

$\beta, \alpha \in \mathbb{C}, \boldsymbol{\mu}^0 \in \mathbb{C}^d, \boldsymbol{\omega}^0 \in \mathbb{R}^d$
for $j = 1 \dots N$ **do**
  $\boldsymbol{\mu}^{j+1} = \beta \boldsymbol{\mu}^j - \hat{\boldsymbol{g}}^j$
  $\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j + \Re(\alpha \boldsymbol{\mu}^{j+1})$
return $\boldsymbol{\omega}^N$

buffer, which also works and could yield better solutions. However, we only use the real component of the momentum – $\Re(\boldsymbol{\mu})$ – because this is the simplest setup and sufficient to work.

We show the SimCM update in Alg. 1 and visualize it in App. Fig. 7d. We also show the alternating (or Gauss-Seidel) update, which is common for GAN training:

$$\boldsymbol{\mu}_A^{j+1} = \beta \boldsymbol{\mu}_A^j - \hat{\boldsymbol{g}}_A(\boldsymbol{\omega}^j), \boldsymbol{\theta}_A^{j+1} = \boldsymbol{\theta}_A^j + \Re(\alpha \boldsymbol{\mu}_A^{j+1}) \quad \text{(AltCM)}$$
$$\boldsymbol{\mu}_B^{j+1} = \beta \boldsymbol{\mu}_B^j - \hat{\boldsymbol{g}}_B(\boldsymbol{\theta}_A^{j+1}, \boldsymbol{\theta}_B^j), \boldsymbol{\theta}_B^{j+1} = \boldsymbol{\theta}_B^j + \Re(\alpha \boldsymbol{\mu}_B^{j+1})$$

Generalizing negative momentum: Consider the negative momentum from Gidel et al. (2019): $\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j - \alpha \hat{\boldsymbol{g}}^j + \beta(\boldsymbol{\omega}^j - \boldsymbol{\omega}^{j-1})$. Expanding (SimCM) with $\boldsymbol{\mu}^j = {}^{(\boldsymbol{\omega}^j - \boldsymbol{\omega}^{j-1})}/_{\alpha}$ for real momentum shows the negative momentum method of Gidel et al. (2019) is a special case of our method:

$$\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j + \Re(\alpha({}^{\beta(\boldsymbol{\omega}^j - \boldsymbol{\omega}^{j-1})}/_{\alpha} - \hat{\boldsymbol{g}}^j)) \quad (6)$$
$$= \boldsymbol{\omega}^j - \alpha \hat{\boldsymbol{g}}^j + \beta(\boldsymbol{\omega}^j - \boldsymbol{\omega}^{j-1}) \quad (7)$$

3.1 Dynamics of Complex Momentum

For simplicity, we assume NumPy-style (Harris et al., 2020a) component-wise broadcasting for operations like taking the real-part $\Re(\boldsymbol{z})$ of vector $\boldsymbol{z} = [z_1, \dots, z_n] \in \mathbb{C}^n$, with proofs in the Appendix. Expanding the buffer updates with the polar components of β gives intuition for complex momentum:

$$\boldsymbol{\mu}^{j+1} = \beta \boldsymbol{\mu}^j - \hat{\boldsymbol{g}}^j \iff \boldsymbol{\mu}^{j+1} = \beta(\beta(\cdot\,\cdot) - \hat{\boldsymbol{g}}^{j-1}) - \hat{\boldsymbol{g}}^j$$
$$\iff \boldsymbol{\mu}^{j+1} = -\sum_{k=0}^{k=j} \beta^k \hat{\boldsymbol{g}}^{j-k} \iff$$
$$\Re(\boldsymbol{\mu}^{j+1}) = -\sum_{k=0}^{k=j} |\beta|^k \cos(k \arg(\beta)) \hat{\boldsymbol{g}}^{j-k},$$
$$\Im(\boldsymbol{\mu}^{j+1}) = -\sum_{k=0}^{k=j} |\beta|^k \sin(k \arg(\beta)) \hat{\boldsymbol{g}}^{j-k} \quad (8)$$

The final line is simply by Euler's formula (27). (8) shows how β controls the momentum buffer $\boldsymbol{\mu}$ by having $|\beta|$ dictate prior gradient decay rates, while $\arg(\beta)$ controls oscillation frequency between adding and subtracting prior gradients, which we visualize in Fig. 2a.

Expanding the parameter updates with the Cartesian components of α and β is key for Theorem 1, which characterizes the convergence rate:

$$\boldsymbol{\mu}^{j+1} = \beta \boldsymbol{\mu}^j - \hat{\boldsymbol{g}}^j \iff$$
$$\Re(\boldsymbol{\mu}^{j+1}) = \Re(\beta)\Re(\boldsymbol{\mu}^j) - \Im(\beta)\Im(\boldsymbol{\mu}^j) - \Re(\hat{\boldsymbol{g}}^j), \quad (9)$$
$$\Im(\boldsymbol{\mu}^{j+1}) = \Im(\beta)\Re(\boldsymbol{\mu}^j) + \Re(\beta)\Im(\boldsymbol{\mu}^j)$$

$$\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j + \Re(\alpha \boldsymbol{\mu}^{j+1}) \iff \quad (10)$$
$$\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j - \alpha \hat{\boldsymbol{g}}^j + \Re(\alpha\beta)\Re(\boldsymbol{\mu}^j) - \Im(\alpha\beta)\Im(\boldsymbol{\mu}^j) \quad (11)$$

So, we can write the next iterate with a fixed-point operator:

$$[\Re(\boldsymbol{\mu}^{j+1}), \Im(\boldsymbol{\mu}^{j+1}), \boldsymbol{\omega}^{j+1}] = \boldsymbol{F}_{\alpha,\beta}([\Re(\boldsymbol{\mu}^j), \Im(\boldsymbol{\mu}^j), \boldsymbol{\omega}^j]) \quad (12)$$

(9) and (10) allow us to write the Jacobian of $\boldsymbol{F}_{\alpha,\beta}$ which can be used to bound convergence rates near fixed points, which we name the Jacobian of the augmented dynamics of buffer $\boldsymbol{\mu}$ and joint-parameters $\boldsymbol{\omega}$ and denote with:

$$\boldsymbol{R} := \nabla_{[\boldsymbol{\mu},\boldsymbol{\omega}]} \boldsymbol{F}_{\alpha,\beta} = \begin{bmatrix} \Re(\beta)\boldsymbol{I} & -\Im(\beta)\boldsymbol{I} & -\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}} \\ \Im(\beta)\boldsymbol{I} & \Re(\beta)\boldsymbol{I} & 0 \\ \Re(\alpha\beta)\boldsymbol{I} & -\Im(\alpha\beta)\boldsymbol{I} & \boldsymbol{I} - \alpha\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}} \end{bmatrix} \quad (13)$$

So, for quadratic losses our parameters evolve via:

$$[\Re(\boldsymbol{\mu}^{j+1}), \Im(\boldsymbol{\mu}^{j+1}), \boldsymbol{\omega}^{j+1}]^\top = \boldsymbol{R}\,[\Re(\boldsymbol{\mu}^j), \Im(\boldsymbol{\mu}^j), \boldsymbol{\omega}^j]^\top \quad (14)$$

We can bound convergence rates near fixed points by using the spectrum of $\boldsymbol{R}$ with Theorem 1.

Theorem 1 (Consequence of Prop. 4.4.1 (Bertsekas, 2008))**.** *Convergence rate of complex momentum: If the spectral radius $\rho(\nabla \boldsymbol{F}_{\alpha,\beta}(\boldsymbol{\mu}^*, \boldsymbol{\omega}^*)) < 1$, then, for $[\boldsymbol{\mu}, \boldsymbol{\omega}]$ in a neighborhood of $[\boldsymbol{\mu}^*, \boldsymbol{\omega}^*]$, the distance of $[\boldsymbol{\mu}^j, \boldsymbol{\omega}^j]$ to the stationary point $[\boldsymbol{\mu}^*, \boldsymbol{\omega}^*]$ converges at a linear rate $\mathcal{O}((\rho(\boldsymbol{R}) + \epsilon)^j), \forall \epsilon > 0$.*

Linear convergence means $\lim_{j\to\infty} {}^{\|\boldsymbol{\omega}^{j+1} - \boldsymbol{\omega}^*\|}/_{\|\boldsymbol{\omega}^j - \boldsymbol{\omega}^*\|} \in (0,1)$ here, where $\boldsymbol{\omega}^*$ is a fixed point. We should select optimization parameters α, β so that the augmented dynamics spectral radius $\mathrm{Sp}(\boldsymbol{R}(\alpha, \beta)) < 1$—with the dependence on α and β now explicit. We may want to express $\mathrm{Sp}(\boldsymbol{R}(\alpha, \beta))$ in terms of the spectrum $\mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$, as in Theorem 3 in Gidel et al. (2019):

$$\boldsymbol{f}(\mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}}), \alpha, \beta) = \mathrm{Sp}(\boldsymbol{R}(\alpha, \beta)) \quad (15)$$

We provide a Mathematica command in App. A.2 for a cubic polynomial p characterizing $\boldsymbol{f}$ with coefficients that are functions of α, β & $\lambda \in \mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$, whose roots

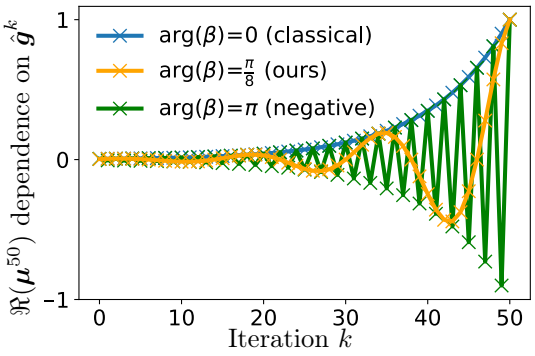


Figure 2(a): We show the real part of our momentum buffer – which dictates the parameter update – at the 50^{th} iteration $\Re(\boldsymbol{\mu}^{50})$ dependence on past gradients $\hat{\boldsymbol{g}}^k$ for $k=1\dots50$. The momentum magnitude is fixed to $|\beta|=0.9$ as in Fig. 3. Euler's formula is used in (8) for finding dependence or coefficient of $\hat{\boldsymbol{g}}^k$ via $\Re(\boldsymbol{\mu}^{50}) = -\sum_{k=0}^{k=50} |\beta|^k \cos(k \arg(\beta))\hat{\boldsymbol{g}}^{j-k}$. Complex momentum allows smooth changes in the buffers dependence on past gradients.

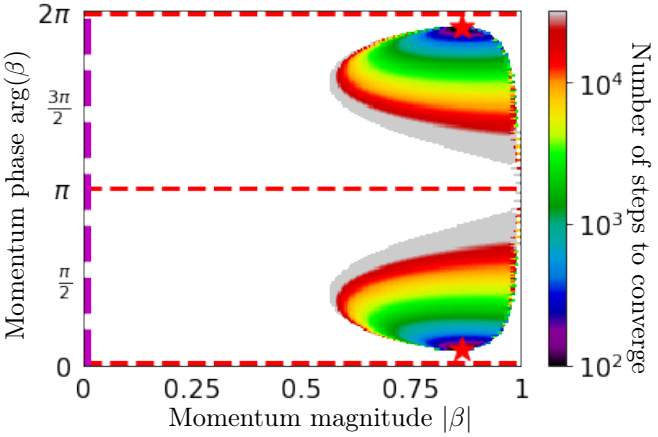


Figure 2(b): How many steps simultaneous complex momentum on a Dirac-GAN takes for a set solution distance. We fix step size $\alpha=0.1$ as in Fig. 3, while varying the phase and magnitude of our momentum $\beta=|\beta|\exp(i\arg(\beta))$. There is a red star at the optima, dashed red lines at real β, and a magenta line for simultaneous gradient descent. There are no real-valued β that converge for this – or any – α with simultaneous updates (Gidel et al., 2019). App. Fig. 9 compares this with alternating updates (AltCM).

are EVals of $\boldsymbol{R}$, which we use in subsequent results. O'donoghue and Candes (2015) and Lucas et al. (2018) mention that we often do not know the condition number, EVals – or the mixture of cooperative and adversarial ESpaces – of a set of functions that we are optimizing, so we try to design algorithms which work over a large range. Sharing this motivation, we consider convergence behavior on games ranging from purely adversarial to cooperative.

In Sec. 4.2 at every non-real β we could select α and $|\beta|$ so Alg. 1 converges. We define *almost-positive* to mean $\arg(\beta)=\epsilon$ for small ϵ, and show there are almost-positive β which converge.

Corollary 1 (Convergence of Complex Momentum)**.** *There exist $\alpha\in\mathbb{R}, \beta\in\mathbb{C}$ so Alg. 1 converges for bilinear zero-sum games. More-so, for small ϵ (we show for $\epsilon=\frac{\pi}{16}$), if $\arg(\beta)=\epsilon$ (i.e., almost-positive) or $\arg(\beta)=\pi-\epsilon$ (i.e., almost-negative), then we can select $\alpha, |\beta|$ to converge.*

Why show this? Our result complements Gidel et al. (2019) who show that for all real α, β Alg. 1 *does not* converge. We include the proof for bilinear zero-sum games, but the result generalizes to some games that are purely adversarial near fixed points, like Dirac GANs (Mescheder et al., 2017). The result's second part shows evidence there is a sense in which the only β that do not converge are real (with simultaneous updates on purely adversarial games). It also suggests a form of robustness, because almost-positive β can approach acceleration in cooperative ESpaces, while converging in adversarial ESpaces, so almost-positive β may be useful when our games have an uncertain or variable mixture of real and imaginary EVals like GANs. Sections 4.2, 4.3, and 4.4 investigate this further.

3.2 What about Acceleration?

With classical momentum, finding the step size α and momentum β to optimize the convergence rate is tractable if $0<l\leqslant L$ and $\text{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})\in[l,L]^d$ (Goh, 2017) – i.e., we have an l-strongly convex and L-Lipschitz loss. The conditioning $\kappa = {}^{L}/_{l}$ can characterize the problem difficulty. Gradient descent with an appropriate α can achieve a convergence rate of $\frac{\kappa-1}{\kappa+1}$, but using momentum with appropriate (α^*,β^*) can achieve an *accelerated* rate of $\rho^*=\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$. However, there is no consensus for constraining $\text{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$ in games for tractable and useful results. Candidate constraints include monotonic vector fields generalizing notions of convexity, or vector fields with bounded EVal norms capturing a kind of sensitivity (Azizian et al., 2020a). Fig. 6 shows $\text{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$ for a GAN, motivating varying α and β for each player as done in Sec. 4.4.

3.3 Implementing Complex Momentum

Complex momentum is trivial to implement with libraries supporting complex arithmetic like JAX (Bradbury et al., 2018) or Pytorch (Paszke et al., 2017). Given an SGD implementation, we often only need to change a few lines of code – see Fig. 1. Also, (9) and (10) can be easily used to implement Alg. 1 in a library without complex arithmetic. More sophisticated optimizers like Adam can trivially support complex optimizer parameters with real-valued updates, which we explore in Sec. 4.4.

3.4 Scope and Limitations

For some games, we need higher than first-order information to converge – ex., pure-response games (Lorraine et al., 2020) – because the first-order information for a player is identically zero. So, momentum methods

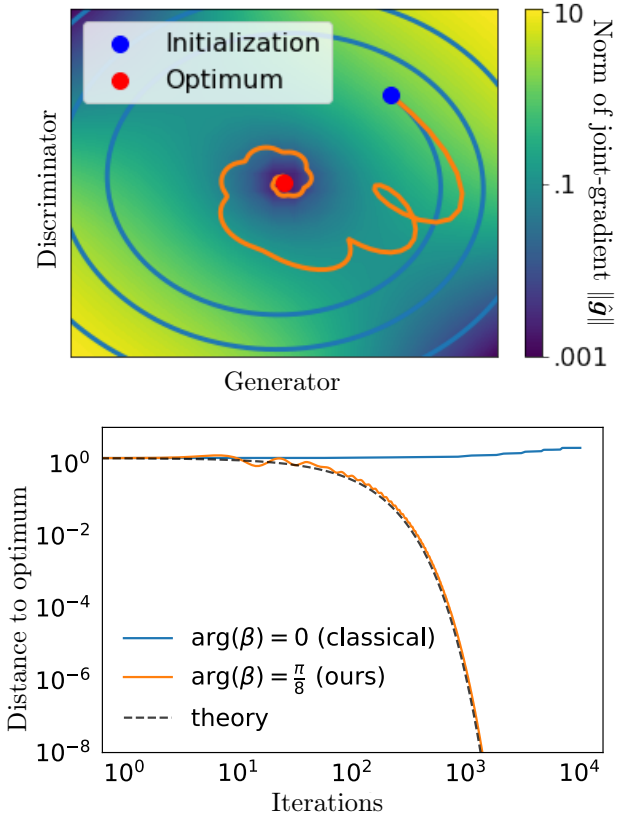


Figure 3: Complex momentum helps correct rotational dynamics when training a Dirac-GAN (Mescheder et al., 2018). *Top:* Parameter trajectories with step size $\alpha = .1$ and momentum $\beta = .9\exp(i\pi/8)$. We include the classical, real and positive momentum which diverges for any α. *Bottom:* The distance to optimum, which has a convergence rate matching our prediction with Thm. 1 and (15).

only using first-order info will not converge in general. However, we can combine methods with second-order information and momentum algorithms (Lorraine et al., 2020; Raghu et al., 2020). Complex momentum's computational cost is almost identical to classical and negative momentum, except we now have a buffer with twice as many real parameters. We require one more optimization hyperparameter than classical momentum, which we provide an initial guess for in Sec. 4.5.

4 Experiments

We investigate complex momentum's performance in training GANs and games with different mixtures of cooperative and adversarial ESpaces, showing improvements over standard baselines. Code for experiments will be available on publication, with reproducibility details in Appendix C.

Overview: We start with a purely adversarial Dirac-GAN and zero-sum games, which have known solutions $\boldsymbol{\omega}^* = (\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B^*)$ and spectrums $\mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$, so we can assess convergence rates. Next, we evaluate GANs generating 2D distributions, because they are simple enough to train with a plain, alternating SGD. Finally, we look at scaling to larger-scale GANs on images which have brittle optimization, and require optimizers like Adam. Complex momentum provides benefits in each setup.

We only compare to first-order optimization methods, despite there being various second-order methods due limitations discussed in Sec. 2.2.

4.1 Opt. in Purely Adversarial Games

Here, we consider the optimizing the Dirac-GAN objective, which is surprisingly hard and where many classical optimization methods fail, because $\mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$ is imaginary near solutions:

$$\min_x \max_y -\log(1+\exp(-xy)) - \log(2) \tag{16}$$

Fig. 3 empirically verifies convergence rates given by Theorem 1 with (15), by showing the optimization trajectories with simultaneous updates.

Fig. 2b shows how the components of the momentum β affect convergence rates with simultaneous updates and a fixed step size. The best β was almost-positive (i.e., $\arg(\beta) = \epsilon$ for small ϵ). We repeat this experiment with alternating updates in App. Fig. 9, which are standard in GAN training. There, almost-positive momentum is best (but negative momentum also converges), and the benefit of alternating updates depends on if we can parallelize player gradient evaluations.

4.2 Adversarialnesses Effect on Convergence

We compare optimization with first-order methods for purely adversarial, cooperative, and mixed games. We use the following game, allowing us to easily interpolate between these regimes, where if $\boldsymbol{\gamma} = \boldsymbol{I}$ it is purely adversarial, while if the $\boldsymbol{\gamma} = \boldsymbol{0}$ it is purely cooperative:

$$\begin{aligned}&\min_{\boldsymbol{x}} \max_{\boldsymbol{y}} \boldsymbol{x}^\top(\boldsymbol{\gamma}\boldsymbol{A})\boldsymbol{y} + \\ &\boldsymbol{x}^\top((\boldsymbol{I}-\boldsymbol{\gamma})\boldsymbol{B}_1)\boldsymbol{x} - \boldsymbol{y}^\top((\boldsymbol{I}-\boldsymbol{\gamma})\boldsymbol{B}_2)\boldsymbol{y}\end{aligned} \tag{17}$$

App. Fig. 8 explores $\mathrm{Sp}(\boldsymbol{R})$ in purely adversarial games for a range of α, β, generalizing Fig. 4 in Gidel et al. (2019). At every non-real β—i.e., $\arg(\beta) \neq \pi$ or 0—we could select $\alpha, |\beta|$ that converge.

Fig. 4 compares first-order algorithms as we interpolate from the purely cooperative games (i.e., minimization) to mixtures of purely adversarial and cooperative ESpaces, because this setup range can occur during GAN training – see Fig. 6. Our baselines are simultaneous SGD (or gradient descent-ascent (GDA)), extragradient (EG) (Korpelevich, 1976), optimistic gradient (OG) (Chiang et al., 2012; Rakhlin and Sridharan, 2013; Daskalakis et al., 2018), and momentum variants.

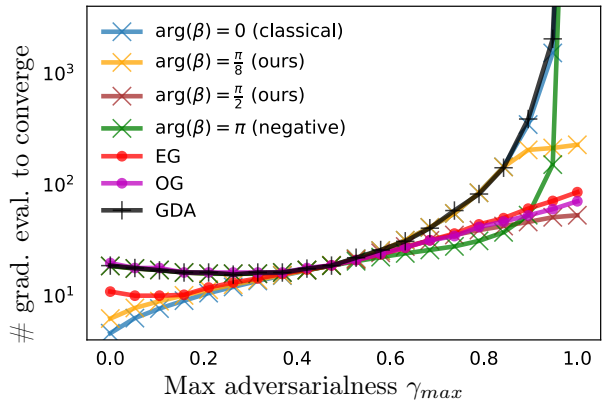


Figure 4: We compare first-order methods convergence rates on the game in (17), with $\boldsymbol{A} = \boldsymbol{B}_1 = \boldsymbol{B}_2$ diagonal and entries linearly spaced in $[1/4, 4]$. We interpolate from purely cooperative to a mixture of purely cooperative and adversarial ESpaces in $\mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$ by making $\boldsymbol{\gamma}$ diagonal with $\gamma_j \sim U[0, \gamma_{max}]$, inducing j^{th} EVal pair to have $\arg(\lambda_j) \approx \pm\gamma_j \frac{\pi}{2}$. So, γ_{max} controls the largest possible EVal arg or *max adversarialness*. Every method generalizes gradient descent-ascent (GDA) by adding an optimizer parameter, tuned via grid search. Positive momentum and negative momentum do not converge if there are purely adversarial ESpaces (i.e., $\gamma_{max} = 1$). Almost-positive momentum $\arg(\beta) = \epsilon > 0$ like $\pi/8$ allows us to approach the acceleration of positive momentum if sufficiently cooperative (i.e., $\gamma_{max} < 0.5$), while still converging if there are purely adversarial ESpaces (i.e., $\gamma_{max} = 1$). Tuning $\arg(\beta)$ with complex momentum performs competitively with extragradient (EG), optimistic gradient (OG) for any adversarialness – ex., $\arg(\beta) = \pi/2$ does well if there are purely adversarial ESpaces (i.e., $\gamma_{max} = 1$).

We additionally *tuned the extrapolation parameters for EG and OG separately* – a non-standard modification so EG and OG are competitive with momentum in cooperative ESpaces; see App. Sec. C.3. We show how many gradient evaluations for a set solution distance, and EG costs two evaluations per update. We optimize convergence rates for each game and method by grid search, as is common for optimization parameters in deep learning.

Takeaway: In the regime where all ESpaces are cooperative – i.e., $\gamma_{max} < .5$ or $\max_{\lambda \in \mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})} |\Im(\lambda)|/|\Re(\lambda)| < 1$ – the best method is classical, positive momentum, otherwise we benefit from a method for learning in games. If we have purely adversarial ESpaces – i.e.,$\gamma_{max} = 1$ – then GDA, positive and negative momentum fail to converge, while EG, OG, and complex momentum can converge. Choosing any non-real momentum β allows robust convergence for every ESpace mixture. More so, almost-positive momentum β allows us to approach acceleration when cooperative, while still converging if there are purely adversarial ESpaces.

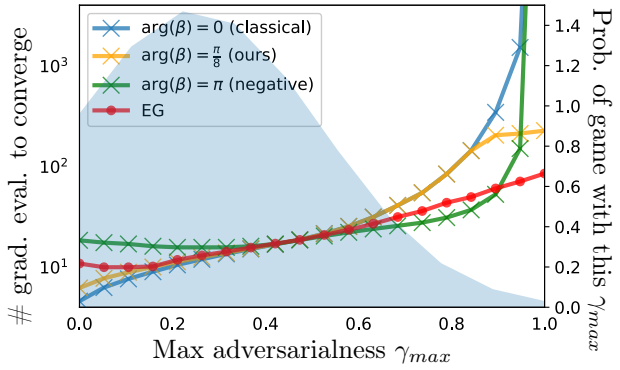


Figure 5: We show a setup where almost-positive, complex momentum with $\arg(\beta) = \pi/8$ minimizes risk over gradient evaluations, by robustly converging for the distribution of games shown in light blue. We compare with methods from Fig. 4. Notably, we believe this distribution of games may be a good proxy for what occurs during GAN training. The distribution was calculated during training the GAN from Fig. 6, by approximating the distribution of γ_{max} over the last 1000 training steps with a truncated Gaussian using the empirical mean and variance. We approximated γ_{max} by looking at the subset of ESpaces the parameters lie in and filtering any ESpaces with negative real part, or where the parameters component in that ESpace was sufficiently smaller than 10^{-5} for simple visualization.

In games like GANs, our eigendecomposition is infeasible to compute and changes during training – see App. Fig. 10 – so we want an optimizer that converges robustly for any potential γ_{max}. A natural goal is to minimize the risk over the games we could observe of the number of gradient evaluations to converge. Fig. 5 displays some methods from Fig. 4 superimposed with a distribution of potential of games. **Takeaway:** The best method is complex momentum, because the bulk of the game distribution is in cooperative ESpaces (i.e., $\gamma_{max} < .5$), but could have purely adversarial ESpaces (i.e., $\gamma_{max} = 1$). Also, we believe the displayed game distribution could be a reasonable proxy for what is encountered in GAN training.

4.3 Training GANs on 2D Distributions

Here, we investigate improving GAN training using alternating gradient descent updates with complex momentum. We look at alternating updates, because they are standard in GAN training (Goodfellow et al., 2014; Brock et al., 2018; Wu et al., 2019). We focus on comparisons to positive and negative momentum, which were the strongest baselines. Note that EG and OG do not have obvious generalizations to alternating updates. We train to generate a 2D mixture of Gaussians, because more complicated distribution require more complicated optimizers than SGD. Fig. 1 shows all changes necessary to use the JAX momentum optimizer for our updates, with full details in App. C.4.

We evaluate the log-likelihood of GAN samples under the mixture as an imperfect proxy for matching.

App. Fig. 11 shows heatmaps for tuning $\arg(\beta)$ and $|\beta|$ with select step sizes. **Takeaway:** The best momentum was found at the almost-positive $\beta \approx 0.7\exp(i\pi/8)$ with step size $\alpha = 0.03$, and for each α we tested a broad range of non-real β outperformed any real β. This suggests we may be able to often improve GAN training with alternating updates and complex momentum.

4.4 Training BigGAN with a Complex Adam

Here, we investigate improving larger-scale GAN training with complex momentum. However, larger-scale GANs train with more complicated optimizers than gradient descent – like Adam (Kingma and Ba, 2014) – have notoriously brittle optimization. We looked at training BigGAN (Brock et al., 2018) on CIFAR-10 (Krizhevsky, 2009), but were unable to succeed with optimizers other than (Brock et al., 2018)-supplied setups, due to brittle optimization. So, we attempted to change the procedure minimally by taking (Brock et al., 2018)-supplied code here which was trained with Adam, and making only the β_1 parameter – analogous to momentum – complex. The modified complex Adam is shown in Alg. 2, where the momentum bias correction is removed to better match our theory. It is an open question on how to best carry over the design of Adam (or other optimizers) to the complex setting. Training each BigGAN took 10 hours on an NVIDIA T4 GPU, so Fig. 13a and Table 1 took about 1000 and 600 GPU hours respectively.

Fig. 13a shows a grid search over $\arg(\beta_1)$ and $|\beta_1|$ for a BigGAN trained with Alg. 2. We only changed β_1 for the discriminator's optimizer. **Takeaway:** The best momentum was at the almost-positive $\beta_1 \approx 0.8\exp(i\pi/8)$, whose samples are in App. Fig. 12b.

We tested the best momentum value over 10 seeds against the author-provided baseline in App. Fig. 13b, with the results summarized in Table 1. **Takeaway:** Our method improves the mean IS with a t-test significance of $p = 0.071$, which shows the desired phenomena – i.e., complex momentum improving training – with a reasonable significance. Also, complex momentum improves the best IS found with 9.25(+.15 over author code, +.03 author reported). Brock et al. (2018) reported a single inception score (IS) on CIFAR-10 of 9.22, but the best we could reproduce over the seeds with the provided PyTorch code and settings was 9.10.

We trained a real momentum $|\beta_1| = 0.8$ to see if the improvement was solely from tuning the momentum magnitude. This occasionally failed to train and decreased the best IS over re-runs, showing we benefit from a non-zero $\arg(\beta_1)$.

4.5 A Practical Initial Guess for $\arg(\beta)$

Here, we propose a practical initial guess for our new hyperparameter $\arg(\beta)$. Corollary 1 shows we can use almost-real momentum coefficients (i.e., $\arg(\beta)$ is close to 0). Fig. 4 shows almost-positive β approach acceleration in cooperative ESpaces, while converging in all ESpaces. Fig. 6 shows GANs can have both cooperative and adversarial ESpaces. Fig. 5 shows a distribution of games – from GAN training – where almost-positive β robustly converges, and minimizes the risk of gradient evaluations. Figures 11 and 13a do a grid search over $\arg(\beta)$ for GANs, finding that almost-positive $\arg(\beta) \approx \pi/8$ works in both cases. Also, by minimally changing $\arg(\beta)$ from 0 to a small ϵ, we can minimally change other hyperparameters in our model, which is useful to adapt existing, brittle setups like in GANs. Based on this, we propose an initial guess of $\arg(\beta) = \epsilon$ for a small $\epsilon > 0$, where $\epsilon = \pi/8$ worked in our GAN experiments.

5 Related Work

Accelerated first-order methods: A broad body of work exists using momentum-type methods (Polyak, 1964; Nesterov, 1983, 2013; Maddison et al., 2018), with a recent focus on deep learning (Sutskever et al., 2013; Zhang and Mitliagkas, 2017; Choi et al., 2019; Zhang et al., 2019; Chen et al., 2020). But, these focus on momentum for minimization as opposed to in games.

Learning in games: Various works approximate response-gradients - some by differentiating through optimization (Foerster et al., 2018; Mescheder et al., 2017; Maclaurin et al., 2015) - or leverage game eigenstructure during optimization (Letcher et al., 2019; Nagarajan et al., 2020; Omidshafiei et al., 2020; Czarnecki et al., 2020; Gidel et al., 2020; Perolat et al., 2020).

First-order methods in games: Zhang et al. (2021, 2020b); Ibrahim et al. (2020); Bailey et al. (2020); Jin et al. (2020); Azizian et al. (2020a); Nouiehed et al. (2019); Zhang et al. (2020c) characterize convergence with first-order methods. Gidel et al. (2019) is the closest work to ours, showing a negative momentum can help in some games. Zhang and Wang (2020) note the suboptimality of negative momentum in a class of games. Azizian et al. (2020b); Domingo-Enrich et al. (2020) investigate acceleration in some games.

Bilinear zero-sum games: Zhang and Yu (2019) study the convergence of gradient methods in bilinear zero-sum games. Their analysis extends (Gidel et al., 2019), showing that we can achieve faster convergence by having separate step sizes and momentum for each player or tuning the extragradient step size. Loizou et al. (2020) provide convergence guarantees for games satisfying a *sufficiently bilinear* condition.

Algorithm 2 Complex Adam variant without momentum bias-correction

$\beta_1 \in \mathbb{C}, \beta_2 \in [0,1)$
$\alpha \in \mathbb{R}^+, \epsilon \in \mathbb{R}^+$
for $j = 1 \dots N$ **do**
  $\boldsymbol{\mu}^{j+1} = \beta_1 \boldsymbol{\mu}^j - \boldsymbol{g}^j$
  $\boldsymbol{v}^{j+1} = \beta_2 \boldsymbol{v}^j + (1-\beta_2)(\boldsymbol{g}^j)^2$
  $\hat{\boldsymbol{v}}^{j+1} = \frac{\boldsymbol{v}^{j+1}}{1-(\beta_2)^j}$
  $\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j + \alpha \frac{\Re(\boldsymbol{\mu}^j)}{\sqrt{\hat{\boldsymbol{v}}^{j+1}} + \epsilon}$
return $\boldsymbol{\omega}^N$

CIFAR-10 BigGAN IS for 10 seeds

Discriminator β_1	Max	Mean	Median	Std.
0, BigGAN default	9.10	8.93	8.89	0.076
$.8\exp(i\pi/8)$, ours	9.25(+.15)	8.97(+.04)	8.97(.08)	0.079(.003)
.8	9.05(−.05)	7.19(−1.7)	8.82(.07)	2.753(2.67)

Table 1: We display the inception scores (IS) found over 10 runs for training BigGAN on CIFAR-10 with various optimizer settings. We use a complex Adam variant outlined in Alg. 2, where we only tuned β_1 for the discriminator. The best parameters found in App. Fig. 13a were $\beta_1 = 0.8\exp(i\pi/8)$, which improved the max IS in our runs, as well as the final mean, and median IS of the BigGAN authors baseline, which was the SoTA optimizer in this setting to best of our knowledge. We tested $\beta_1 = 0.8$ to see if the gain was solely from tuning $|\beta_1|$, which occasionally failed and decreased the IS.

Learning in GANs: Various works make GAN training easier with methods leveraging the game structure (Liu et al., 2020; Peng et al., 2020; Albuquerque et al., 2019; Wu et al., 2019; Hsieh et al., 2019). Metz et al. (2016) approximate the discriminator's response function by differentiating through optimization. Mescheder et al. (2017) find solutions by minimizing the norm of the players' updates. Both of these methods and various others (Qin et al., 2020; Schäfer et al., 2019; Jolicoeur-Martineau and Mitliagkas, 2019) require higher-order information. Daskalakis et al. (2018); Gidel et al. (2018); Chavdarova et al. (2019) look at first-order methods. Mescheder et al. (2018) explore problems for GAN training convergence and Berard et al. (2019) show that GANs have significant rotations in learning.

6 Conclusion

In this paper we provided a generalization of existing momentum methods for learning in differentiable games by allowing a complex-valued momentum with real-valued updates. Our method robustly converges in games with a different range of mixtures of cooperative and adversarial ESpaces than existing methods. We also presented a practical generalization of our method to the Adam optimizer, which we used to improve BigGAN training. More generally, we highlight and lay groundwork for investigating optimizers which work well with various mixtures of cooperative and competitive dynamics in games.

Societal Impact

Our main contribution in this work is methodological – specifically, a scalable algorithm for optimizing in games. Since our focus is on improving optimization methods, we do not expect there to be direct negative societal impacts from this contribution.

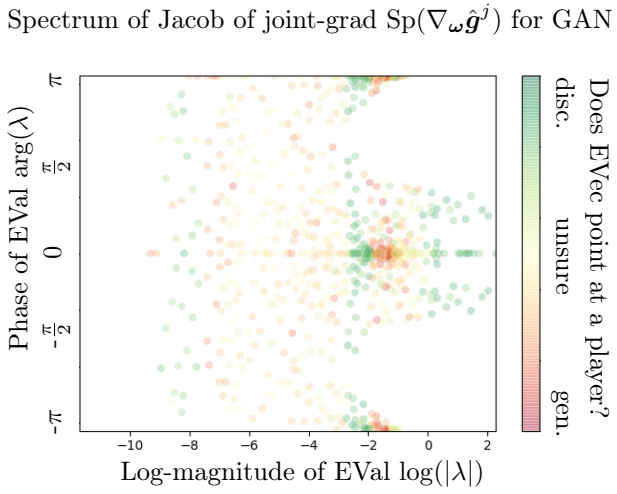


Figure 6: A log-polar visualization reveals structure in the spectrum for a GAN at the end of the training on a 2D mixture of Gaussians with a 1-layer (disc)riminator and (gen)erator, so the joint-parameters $\boldsymbol{\omega} \in \mathbb{R}^{723}$. App. Fig. 10 shows the spectrum through training. There is a mixture of many cooperative (i.e., real or $\arg(\lambda) \approx 0, \pm\pi$) and some adversarial (i.e., imaginary or $\arg(\lambda) \approx \pm\frac{\pi}{2}$) EVals, so – contrary to what the name may suggest – generative adversarial networks are not purely adversarial. We may benefit from optimizers leveraging this structure like complex momentum. EVals are colored if the associated EVec is mostly in one player's part of the joint-parameter space – see App. Fig. 10 for details on this. Many EVecs lie mostly in the the space of (or point at) a one player. The structure of the set of EVals for the disc. (green) is different than the gen. (red), but further investigation of this is an open problem. Notably, this may motivate separate optimizer choices for each player as in Sec. 4.4.

Acknowledgements

Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute. Paul Vicol was supported by an NSERC PGS-D Scholarship. We thank Guodong Zhang, Guojun Zhang, James Lucas, Romina Abachi, Jonah Phillion, Will Grathwohl, Jakob Foerster, Murat Erdogdu, Ken Jackson, and Ioannis Mitliagkis, and Barbara Norton for feedback and helpful discussion. We would also like to thank C. Daniel Freeman, Hérve Jégou, Noam Brown, and David Acuna for feedback on this work and acknowledge the Python community (Van Rossum and Drake Jr, 1995; Oliphant, 2007) for developing the tools that enabled this work, including NumPy (Oliphant, 2006; Van Der Walt et al., 2011; Harris et al., 2020b), and Matplotlib (Hunter, 2007).

References

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014. [Cited on pages 1, 7, and 24]

David Pfau and Oriol Vinyals. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*, 2016. [Cited on page 1]

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*, 2019. [Cited on page 1]

David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*, pages 434–443. PMLR, 2019. [Cited on page 1]

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018. [Cited on page 1]

Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018. [Cited on page 1]

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020. [Cited on pages 1, 5, and 6]

Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-Tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations (ICLR)*, 2019. [Cited on page 1]

Aniruddh Raghu, Maithra Raghu, Simon Kornblith, David Duvenaud, and Geoffrey Hinton. Teaching with commentaries. *arXiv preprint arXiv:2011.03037*, 2020. [Cited on pages 1 and 6]

Avishek Joey Bose, Gauthier Gidel, Hugo Berrard, Andre Cianflone, Pascal Vincent, Simon Lacoste-Julien, and William L Hamilton. Adversarial example games. *arXiv preprint arXiv:2007.00720*, 2020. [Cited on page 1]

Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, 2019. [Cited on page 1]

Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning. *arXiv preprint arXiv:2004.07804*, 2020. [Cited on page 1]

Romina Abachi, Mohammad Ghavamzadeh, and Amirmassoud Farahmand. Policy-aware model learning for policy gradient methods. *arXiv preprint arXiv:2003.00030*, 2020. [Cited on page 1]

Evgenii Nikishin, Romina Abachi, Rishabh Agarwal, and Pierre-Luc Bacon. Control-oriented model-based reinforcement learning with implicit differentiation. *arXiv preprint arXiv:2106.03273*, 2021. [Cited on page 1]

David Acuna, Guojun Zhang, Marc T Law, and Sanja Fidler. f-domain-adversarial learning: Theory and algorithms for unsupervised domain adaptation with neural networks, 2021. URL https://openreview.net/forum?id=WqXAKcwfZtI. [Cited on pages 1 and 3]

Will Grathwohl, Elliot Creager, Seyed Kamyar Seyed Ghasemipour, and Richard Zemel. Gradient-based optimization of neural network architecture. 2018. [Cited on page 1]

George Adam and Jonathan Lorraine. Understanding neural architecture search techniques. *arXiv preprint arXiv:1904.00438*, 2019. [Cited on page 1]

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018. [Cited on page 1]

Mengye Ren, Eleni Triantafillou, Kuan-Chieh Wang, James Lucas, Jake Snell, Xaq Pitkow, Andreas S Tolias, and Richard Zemel. Flexible few-shot learning with contextual similarity. *arXiv preprint arXiv:2012.05895*, 2020. [Cited on page 1]

Oskar Morgenstern and John Von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1953. [Cited on page 1]

Heinrich Von Stackelberg. *Market Structure and Equilibrium*. Springer Science & Business Media, 2010. [Cited on page 1]

Hugo Berard, Gauthier Gidel, Amjad Almahairi, Pascal Vincent, and Simon Lacoste-Julien. A closer look at the optimization landscapes of generative adversarial networks. In *International Conference on Learning Representations*, 2019. [Cited on pages 1 and 9]

Kenneth Joseph Arrow, Hirofumi Azawa, Leonid Hurwicz, and Hirofumi Uzawa. *Studies in Linear and Non-Linear Programming*, volume 2. Stanford University Press, 1958. [Cited on page 1]

Yoav Freund and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999. [Cited on page 1]

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 *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1802–1811. PMLR, 2019. [Cited on pages 1, 2, 3, 4, 5, 6, 8, 16, 18, 20, 21, and 23]

James Lucas, Shengyang Sun, Richard Zemel, and Roger Grosse. Aggregated momentum: Stability through passive damping. In *International Conference on Learning Representations*, 2018. [Cited on pages 1, 2, 3, 5, and 18]

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. [Cited on pages 2, 8, 15, and 18]

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. [Cited on pages 2 and 8]

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018. [Cited on pages 2, 7, and 8]

Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Large-scale distributed second-order optimization using Kronecker-factored approximate curvature for deep convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12359–12367, 2019. [Cited on page 3]

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of GANs. In *Advances in Neural Information Processing Systems*, pages 1825–1835, 2017. [Cited on pages 3, 5, 8, and 9]

Florian Schäfer and Anima Anandkumar. Competitive gradient descent. In *Advances in Neural Information Processing Systems*, pages 7623–7633, 2019. [Cited on page 3]

Yuanhao Wang, Guodong Zhang, and Jimmy Ba. On solving minimax optimization locally: A follow-the-ridge approach. In *International Conference on Learning Representations*, 2019. [Cited on page 3]

Reyhane Askari Hemmat, Amartya Mitra, Guillaume Lajoie, and Ioannis Mitliagkas. LEAD: Least-action dynamics for min-max optimization. *arXiv preprint arXiv:2010.13846*, 2020. [Cited on page 3]

Florian Schäfer, Anima Anandkumar, and Houman Owhadi. Competitive mirror descent. *arXiv preprint arXiv:2006.10179*, 2020. [Cited on page 3]

Wojciech Marian Czarnecki, Gauthier Gidel, Brendan Tracey, Karl Tuyls, Shayegan Omidshafiei, David Balduzzi, and Max Jaderberg. Real world games look like spinning tops. *arXiv preprint arXiv:2004.09468*, 2020. [Cited on pages 3 and 8]

Guojun Zhang, Kaiwen Wu, Pascal Poupart, and Yaoliang Yu. Newton-type methods for minimax optimization. *arXiv preprint arXiv:2006.14592*, 2020a. [Cited on page 3]

GM Korpelevich. The extragradient method for finding saddle points and other problems. *Matecon*, 12:747–756, 1976. [Cited on pages 3, 6, and 24]

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020a. [Cited on page 4]

D Bertsekas. *Nonlinear Programming*. Athena Scientific, 2008. [Cited on pages 4 and 15]

Brendan O'donoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015. [Cited on page 5]

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine learning (ICML)*, pages 3481–3490. PMLR, 2018. [Cited on pages 6 and 9]

Gabriel Goh. Why momentum really works. *Distill*, 2(4): e6, 2017. [Cited on page 5]

Waïss Azizian, Ioannis Mitliagkas, Simon Lacoste-Julien, and Gauthier Gidel. A tight and unified analysis of gradient-based methods for a whole spectrum of differentiable games. In *International Conference on Artificial Intelligence and Statistics*, pages 2863–2873. PMLR, 2020a. [Cited on pages 5 and 8]

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax. [Cited on page 5]

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *Openreview*, 2017. [Cited on page 5]

Chao-Kai Chiang, Tianbao Yang, Chia-Jung Lee, Mehrdad Mahdavi, Chi-Jen Lu, Rong Jin, and Shenghuo Zhu. Online optimization with gradual variations. In *Conference on Learning Theory*, pages 6–1. JMLR Workshop and Conference Proceedings, 2012. [Cited on page 6]

Alexander Rakhlin and Karthik Sridharan. Optimization, learning, and games with predictable sequences. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, pages 3066–3074, 2013. [Cited on page 6]

Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training GANs with Optimism. In *International Conference on Learning Representations (ICLR 2018)*, 2018. [Cited on pages 6, 9, and 24]

Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy Lillicrap. Logan: Latent optimisation for generative adversarial networks. *arXiv preprint arXiv:1912.00953*, 2019. [Cited on pages 7 and 9]

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. [Cited on page 8]

Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983. [Cited on page 8]

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013. [Cited on page 8]

Chris J Maddison, Daniel Paulin, Yee Whye Teh, Brendan O'Donoghue, and Arnaud Doucet. Hamiltonian descent methods. *arXiv preprint arXiv:1809.05042*, 2018. [Cited on page 8]

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013. [Cited on pages 8 and 18]

Jian Zhang and Ioannis Mitliagkas. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017. [Cited on page 8]

Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019. [Cited on page 8]

Michael R Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. *arXiv preprint arXiv:1907.08610*, 2019. [Cited on page 8]

Ricky TQ Chen, Dami Choi, Lukas Balles, David Duvenaud, and Philipp Hennig. Self-tuning stochastic optimization with curvature-aware gradient filtering. *arXiv preprint arXiv:2011.04803*, 2020. [Cited on page 8]

Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130, 2018. [Cited on page 8]

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015. [Cited on page 8]

Alistair Letcher, David Balduzzi, Sébastien Racaniere, James Martens, Jakob N Foerster, Karl Tuyls, and Thore Graepel. Differentiable game mechanics. *Journal of Machine Learning Research*, 20(84):1–40, 2019. [Cited on page 8]

Sai Ganesh Nagarajan, David Balduzzi, and Georgios Piliouras. From chaos to order: Symmetry and conservation laws in game dynamics. In *International Conference on Machine Learning*, pages 7186–7196. PMLR, 2020. [Cited on page 8]

Shayegan Omidshafiei, Karl Tuyls, Wojciech M Czarnecki, Francisco C Santos, Mark Rowland, Jerome Connor, Daniel Hennes, Paul Muller, Julien Pérolat, Bart De Vylder, et al. Navigating the landscape of multiplayer games. *Nature Communications*, 11(1):1–17, 2020. [Cited on page 8]

Gauthier Gidel, David Balduzzi, Wojciech Marian Czarnecki, Marta Garnelo, and Yoram Bachrach. Minimax theorem for latent games or: How I learned to stop worrying about mixed-Nash and love neural nets. *arXiv preprint arXiv:2002.05820*, 2020. [Cited on page 8]

Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, et al. From poincar\'e recurrence to convergence in imperfect information games: Finding equilibrium via regularization. *arXiv preprint arXiv:2002.08456*, 2020. [Cited on page 8]

Guodong Zhang, Yuanhao Wang, Laurent Lessard, and Roger Grosse. Don't fix what ain't broke: Near-optimal local convergence of alternating gradient descent-ascent for minimax optimization. *arXiv preprint arXiv:2102.09468*, 2021. [Cited on page 8]

Guodong Zhang, Xuchao Bao, Laurent Lessard, and Roger Grosse. A unified analysis of first-order methods for smooth games via integral quadratic constraints. *arXiv preprint arXiv:2009.11359*, 2020b. [Cited on page 8]

Adam Ibrahim, Waıss Azizian, Gauthier Gidel, and Ioannis Mitliagkas. Linear lower bounds and conditioning of differentiable games. In *International Conference on Machine Learning*, pages 4583–4593. PMLR, 2020. [Cited on page 8]

James P Bailey, Gauthier Gidel, and Georgios Piliouras. Finite regret and cycles with fixed step-size via alternating gradient descent-ascent. In *Conference on Learning Theory*, pages 391–407. PMLR, 2020. [Cited on page 8]

Chi Jin, Praneeth Netrapalli, and Michael Jordan. What is local optimality in nonconvex-nonconcave minimax optimization? In *International Conference on Machine Learning*, pages 4880–4889. PMLR, 2020. [Cited on page 8]

Maher Nouiehed, Maziar Sanjabi, Tianjian Huang, Jason D Lee, and Meisam Razaviyayn. Solving a class of non-convex min-max games using iterative first order methods. *Advances in Neural Information Processing Systems*, 32: 14934–14942, 2019. [Cited on page 8]

Guojun Zhang, Pascal Poupart, and Yaoliang Yu. Optimality and stability in non-convex smooth games. *arXiv e-prints*, pages arXiv–2002, 2020c. [Cited on page 8]

Guodong Zhang and Yuanhao Wang. On the suboptimality of negative momentum for minimax optimization. *arXiv preprint arXiv:2008.07459*, 2020. [Cited on page 8]

Waïss Azizian, Damien Scieur, Ioannis Mitliagkas, Simon Lacoste-Julien, and Gauthier Gidel. Accelerating smooth games by manipulating spectral shapes. *arXiv preprint arXiv:2001.00602*, 2020b. [Cited on page 8]

Carles Domingo-Enrich, Fabian Pedregosa, and Damien Scieur. Average-case acceleration for bilinear games and normal matrices. *arXiv preprint arXiv:2010.02076*, 2020. [Cited on page 8]

Guojun Zhang and Yaoliang Yu. Convergence of gradient methods on bilinear zero-sum games. In *International Conference on Learning Representations*, 2019. [Cited on page 8]

Nicolas Loizou, Hugo Berard, Alexia Jolicoeur-Martineau, Pascal Vincent, Simon Lacoste-Julien, and Ioannis Mitliagkas. Stochastic Hamiltonian gradient methods for smooth games. In *International Conference on Machine Learning*, pages 6370–6381. PMLR, 2020. [Cited on page 8]

Mingrui Liu, Youssef Mroueh, Jerret Ross, Wei Zhang, Xiaodong Cui, Payel Das, and Tianbao Yang. Towards better understanding of adaptive gradient algorithms in generative adversarial nets. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJxIm0VtwH. [Cited on page 9]

Wei Peng, Yu-Hong Dai, Hui Zhang, and Lizhi Cheng. Training GANs with centripetal acceleration. *Optimization Methods and Software*, 35(5):955–973, 2020. [Cited on page 9]

Isabela Albuquerque, João Monteiro, Thang Doan, Breandan Considine, Tiago Falk, and Ioannis Mitliagkas. Multi-objective training of generative adversarial networks with multiple discriminators. In *International Conference on Machine Learning*, pages 202–211. PMLR, 2019. [Cited on page 9]

Ya-Ping Hsieh, Chen Liu, and Volkan Cevher. Finding mixed Nash equilibria of generative adversarial networks. In *International Conference on Machine Learning*, pages 2810–2819. PMLR, 2019. [Cited on page 9]

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016. [Cited on page 9]

Chongli Qin, Yan Wu, Jost Tobias Springenberg, Andrew Brock, Jeff Donahue, Timothy P Lillicrap, and Pushmeet Kohli. Training generative adversarial networks by solving ordinary differential equations. *arXiv preprint arXiv:2010.15040*, 2020. [Cited on page 9]

Florian Schäfer, Hongkai Zheng, and Anima Anandkumar. Implicit competitive regularization in GANs. *arXiv preprint arXiv:1910.05852*, 2019. [Cited on page 9]

Alexia Jolicoeur-Martineau and Ioannis Mitliagkas. Connections between support vector machines, Wasserstein distance and gradient-penalty GANs. *arXiv preprint arXiv:1910.06922*, 2019. [Cited on page 9]

Gauthier Gidel, Hugo Berard, Gaëtan Vignoud, Pascal Vincent, and Simon Lacoste-Julien. A variational inequality perspective on generative adversarial networks. In *International Conference on Learning Representations*, 2018. [Cited on page 9]

Tatjana Chavdarova, Gauthier Gidel, Francois Fleuret, and Simon Lacoste-Julien. Reducing noise in GAN training with variance reduced extragradient. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2019. [Cited on page 9]

Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995. [Cited on page 10]

Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007. [Cited on page 10]

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006. [Cited on page 10]

Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13 (2):22–30, 2011. [Cited on page 10]

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with NumPy. *Nature*, 585 (7825):357–362, 2020b. [Cited on page 10]

John D Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. [Cited on page 10]

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016. [Cited on page 24]

Simon Foucart. Matrix norm and spectral radius. https://www.math.drexel.edu/~foucart/TeachingFiles/F12/M504Lect6.pdf, 2012. Accessed: 2020-05-21. [Cited on page 15]

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex Optimization*. Cambridge university press, 2004. [Cited on page 18]

Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000. [Cited on page 20]

A Supporting Results

First, some basic results about complex numbers that are used:

$$z = \Re(z) + i\Im(z) = |z| \exp(i \arg(z)) \tag{18}$$

$$\overline{z} = \Re(z) - i\Im(z) = |z| \exp(-i \arg(z)) \tag{19}$$

$$\exp(iz) + \exp(-iz) = 2\cos(z) \tag{20}$$

$$\overline{z_1 z_2} = \overline{z}_1 \overline{z}_2 \tag{21}$$

$$1/2(z + \overline{z}) = \Re(z) \tag{22}$$

$$\Re(z_1 z_2) = \Re(z_1)\Re(z_2) - \Im(z_1)\Im(z_2) \tag{23}$$

$$z_1 + z_2 = (\Re(z_1) + \Re(z_2)) + i(\Im(z_1) + \Im(z_2)) \tag{24}$$

$$z_1 z_2 = (\Re(z_1)\Re(z_2) - \Im(z_1)\Im(z_2)) + i(\Im(z_1)\Re(z_2) + \Re(z_1)\Im(z_2)) \tag{25}$$

$$z_1 z_2 = |z_1||z_2| \exp(i(\arg(z_1) + \arg(z_2))) \tag{26}$$

$$z^k = |z|^k \exp(i \arg(z) k) = |z|^k (\cos(k \arg(z)) + i \sin(k \arg(z)) \tag{27}$$

This Lemma shows how we expand the complex-valued momentum buffer $\boldsymbol{\mu}$ into its Cartesian components as in (9).

Lemma 1.

$$\begin{aligned} &\boldsymbol{\mu}^{j+1} = \beta \boldsymbol{\mu}^j - \hat{\boldsymbol{g}}^j \iff \\ &\Re(\boldsymbol{\mu}^{j+1}) = \Re(\beta)\Re(\boldsymbol{\mu}^j) - \Im(\beta)\Im(\boldsymbol{\mu}^j) - \Re(\hat{\boldsymbol{g}}^j), \Im(\boldsymbol{\mu}^{j+1}) = \Im(\beta)\Re(\boldsymbol{\mu}^j) + \Re(\beta)\Im(\boldsymbol{\mu}^j) - \Im(\hat{\boldsymbol{g}}^j) \end{aligned}$$

Proof.

$$\begin{aligned} & \boldsymbol{\mu}^{j+1} = \beta \boldsymbol{\mu}^j - \hat{\boldsymbol{g}}^j \\ \iff & \boldsymbol{\mu}^{j+1} = (\Re(\beta) + i\Im(\beta)) \left(\Re(\boldsymbol{\mu}^j) + i\Im(\boldsymbol{\mu}^j)\right) - \left(\Re(\hat{\boldsymbol{g}}^j) + i\Im(\hat{\boldsymbol{g}}^j)\right) \\ \iff & \boldsymbol{\mu}^{j+1} = \left(\Re(\beta)\Re(\boldsymbol{\mu}^j) - \Im(\beta)\Im(\boldsymbol{\mu}^j)\right) + \\ & \qquad\qquad i\left(\Im(\beta)\Re(\boldsymbol{\mu}^j) + \Re(\beta)\Im(\boldsymbol{\mu}^j)\right) - \left(\Re(\hat{\boldsymbol{g}}^j) + i\Im(\hat{\boldsymbol{g}}^j)\right) \\ \iff & \boldsymbol{\mu}^{j+1} = \left(\Re(\beta)\Re(\boldsymbol{\mu}^j) - \Im(\beta)\Im(\boldsymbol{\mu}^j) - \Re(\hat{\boldsymbol{g}}^j)\right) + \\ & \qquad\qquad i\left(\Im(\beta)\Re(\boldsymbol{\mu}^j) + \Re(\beta)\Im(\boldsymbol{\mu}^j) - \Im(\hat{\boldsymbol{g}}^j)\right) \\ \iff & \Re(\boldsymbol{\mu}^{j+1}) = \Re(\beta)\Re(\boldsymbol{\mu}^j) - \Im(\beta)\Im(\boldsymbol{\mu}^j) - \Re(\hat{\boldsymbol{g}}^j), \Im(\boldsymbol{\mu}^{j+1}) = \Im(\beta)\Re(\boldsymbol{\mu}^j) + \Re(\beta)\Im(\boldsymbol{\mu}^j) - \Im(\hat{\boldsymbol{g}}^j) \end{aligned}$$

□

We further assume $\Im(\hat{\boldsymbol{g}}^j)$ is 0 - i.e., our gradients are real-valued. This Lemma shows how we can decompose the joint-parameters $\boldsymbol{\omega}$ at the next iterate as a linear combination of the joint-parameters, joint-gradient, and Cartesian components of the momentum-buffer at the current iterate as in (10).

Lemma 2.

$$\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j + \Re(\alpha \boldsymbol{\mu}^{j+1}) \iff \boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j - \Re(\alpha)\hat{\boldsymbol{g}}^j + \Re(\alpha\beta)\Re(\boldsymbol{\mu}^j) - \Im(\alpha\beta)\Im(\boldsymbol{\mu}^j)$$

Proof.

$$
\begin{aligned}
&\Re(\alpha\boldsymbol{\mu}^{j+1}) \\
&= \left(\Re(\alpha)\Re(\boldsymbol{\mu}^{j+1}) - \Im(\alpha)\Im(\boldsymbol{\mu}^{j+1})\right) \\
&= \left(\Re(\alpha)\left(\Re(\beta)\Re(\boldsymbol{\mu}^{j}) - \Im(\beta)\Im(\boldsymbol{\mu}^{j}) - \hat{\boldsymbol{g}}^{j}\right) - \Im(\alpha)\left(\Im(\beta)\Re(\boldsymbol{\mu}^{j}) + \Re(\beta)\Im(\boldsymbol{\mu}^{j})\right)\right) \\
&= -\Re(\alpha)\hat{\boldsymbol{g}}^{j} + \left(\Re(\alpha)\left(\Re(\beta)\Re(\boldsymbol{\mu}^{j}) - \Im(\beta)\Im(\boldsymbol{\mu}^{j})\right) - \Im(\alpha)\left(\Im(\beta)\Re(\boldsymbol{\mu}^{j}) + \Re(\beta)\Im(\boldsymbol{\mu}^{j})\right)\right) \\
&= -\Re(\alpha)\hat{\boldsymbol{g}}^{j} + \left(\Re(\alpha)\Re(\beta) - \Im(\alpha)\Im(\beta)\right)\Re(\boldsymbol{\mu}^{j}) - \left(\Re(\alpha)\Im(\beta) + \Im(\alpha)\Re(\beta)\right)\Im(\boldsymbol{\mu}^{j}) \\
&= -\Re(\alpha)\hat{\boldsymbol{g}}^{j} + \Re(\alpha\beta)\Re(\boldsymbol{\mu}^{j}) - \Im(\alpha\beta)\Im(\boldsymbol{\mu}^{j})
\end{aligned}
$$

Thus,

$$
\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^{j} + \Re(\alpha\boldsymbol{\mu}^{j+1}) \iff \boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^{j} - \Re(\alpha)\hat{\boldsymbol{g}}^{j} + \Re(\alpha\beta)\Re(\boldsymbol{\mu}^{j}) - \Im(\alpha\beta)\Im(\boldsymbol{\mu}^{j})
$$

□

A.1 Theorem 1 Proof Sketch

Theorem 1 (Consequence of Prop. 4.4.1 (Bertsekas, 2008))**.** *Convergence rate of complex momentum: If the spectral radius* $\rho(\nabla \boldsymbol{F}_{\alpha,\beta}(\boldsymbol{\mu}^*, \boldsymbol{\omega}^*)) < 1$*, then, for* $[\boldsymbol{\mu}, \boldsymbol{\omega}]$ *in a neighborhood of* $[\boldsymbol{\mu}^*, \boldsymbol{\omega}^*]$*, the distance of* $[\boldsymbol{\mu}^j, \boldsymbol{\omega}^j]$ *to the stationary point* $[\boldsymbol{\mu}^*, \boldsymbol{\omega}^*]$ *converges at a linear rate* $\mathcal{O}((\rho(\boldsymbol{R}) + \epsilon)^j), \forall \epsilon > 0$*.*

Proof. We reproduce the proof for a simpler case of quadratic games, which is simple case of the well-known method from Polyak (1964) for analyzing the convergence of iterative methods. Bertsekas (2008) generalizes this result from quadratic games to when we are sufficiently close to any stationary point.

For quadratic games, we have that $\hat{\boldsymbol{g}}^j = (\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})^\top \boldsymbol{\omega}^j$. Well, by Lemma 1 and Lemma 2 we have:

$$
\begin{pmatrix} \Re(\boldsymbol{\mu}^{j+1}) \\ \Im(\boldsymbol{\mu}^{j+1}) \\ \boldsymbol{\omega}^{j+1} \end{pmatrix} = \boldsymbol{R} \begin{pmatrix} \Re(\boldsymbol{\mu}^{j}) \\ \Im(\boldsymbol{\mu}^{j}) \\ \boldsymbol{\omega}^{j} \end{pmatrix} \tag{28}
$$

By telescoping the recurrence for the j^{th} augmented parameters:

$$
\begin{pmatrix} \Re(\boldsymbol{\mu}^{j}) \\ \Im(\boldsymbol{\mu}^{j}) \\ \boldsymbol{\omega}^{j} \end{pmatrix} = \boldsymbol{R}^j \begin{pmatrix} \Re(\boldsymbol{\mu}^{0}) \\ \Im(\boldsymbol{\mu}^{0}) \\ \boldsymbol{\omega}^{0} \end{pmatrix} \tag{29}
$$

We can compare $\boldsymbol{\mu}^j$ with the value it converges to $\boldsymbol{\mu}^*$ which exists if $\boldsymbol{R}$ is contractive. We do the same with $\boldsymbol{\omega}$. Because $\boldsymbol{\mu}^* = \boldsymbol{R}\boldsymbol{\mu}^* = \boldsymbol{R}^j\boldsymbol{\mu}^*$:

$$
\begin{pmatrix} \Re(\boldsymbol{\mu}^{j}) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^{j}) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^{j} - \boldsymbol{\omega}^* \end{pmatrix} = \boldsymbol{R}^j \begin{pmatrix} \Re(\boldsymbol{\mu}^{0}) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^{0}) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^{0} - \boldsymbol{\omega}^* \end{pmatrix} \tag{30}
$$

By taking norms:

$$
\left\| \begin{pmatrix} \Re(\boldsymbol{\mu}^{j}) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^{j}) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^{j} - \boldsymbol{\omega}^* \end{pmatrix} \right\|_2 = \left\| \boldsymbol{R}^j \begin{pmatrix} \Re(\boldsymbol{\mu}^{0}) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^{0}) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^{0} - \boldsymbol{\omega}^* \end{pmatrix} \right\|_2 \tag{31}
$$

$$
\implies \left\| \begin{pmatrix} \Re(\boldsymbol{\mu}^{j}) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^{j}) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^{j} - \boldsymbol{\omega}^* \end{pmatrix} \right\|_2 \leqslant \left\| \boldsymbol{R}^j \right\|_2 \left\| \begin{pmatrix} \Re(\boldsymbol{\mu}^{0}) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^{0}) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^{0} - \boldsymbol{\omega}^* \end{pmatrix} \right\|_2 \tag{32}
$$

With Lemma 11 from (Foucart, 2012), we have there exists a matrix norm $\forall \epsilon > 0$ such that:

$$
\left\| \boldsymbol{R}^j \right\| \leqslant \left(\rho\left(\boldsymbol{R}\right) + \epsilon\right)^j \tag{33}
$$

We also have an equivalence of norms in finite-dimensional spaces. So for all norms $\|\cdot\|$, $\exists C \geqslant B > 0$ such that:

$$B\|\boldsymbol{R}^j\| \leqslant \|\boldsymbol{R}^j\|_2 \leqslant C\|\boldsymbol{R}^j\| \tag{34}$$

Combining (33) and (34) we have:

$$\left\| \begin{pmatrix} \Re(\boldsymbol{\mu}^j) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^j) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^j - \boldsymbol{\omega}^* \end{pmatrix} \right\|_2 \leqslant C\left(\rho\left(\boldsymbol{R}\right) + \epsilon\right)^j \left\| \begin{pmatrix} \Re(\boldsymbol{\mu}^0) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^0) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^0 - \boldsymbol{\omega}^* \end{pmatrix} \right\|_2 \tag{35}$$

So, we have:

$$\left\| \begin{pmatrix} \Re(\boldsymbol{\mu}^j) - \Re(\boldsymbol{\mu}^*) \\ \Im(\boldsymbol{\mu}^j) - \Im(\boldsymbol{\mu}^*) \\ \boldsymbol{\omega}^j - \boldsymbol{\omega}^* \end{pmatrix} \right\|_2 = \mathcal{O}((\rho(\boldsymbol{R}) + \epsilon)^j) \tag{36}$$

Thus, we converge linearly with a rate of $\mathcal{O}(\rho(\boldsymbol{R}) + \epsilon)$. □

A.2 Characterizing the Augmented Dynamics Eigenvalues

Here, we present polynomials whose roots are the eigenvalues of our the Jacobian of our augmented dynamics $\mathrm{Sp}(\boldsymbol{R})$, given the eigenvalues of the Jacobian of the joint-gradient vector field $\mathrm{Sp}(\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}})$. We use a similar decomposition as (Gidel et al., 2019).

We can expand $\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}} = PTP^{-1}$ where T is an upper-triangular matrix and λ_i is an eigenvalue of $\nabla_{\boldsymbol{\omega}}\hat{\boldsymbol{g}}$.

$$T = \begin{bmatrix} \lambda_1 & * & \dots & * \\ 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & * \\ 0 & \dots & 0 & \lambda_d \end{bmatrix} \tag{37}$$

We then break up into components for each eigenvalue, giving us submatrices $\mathbf{R}_k \in \mathbb{C}^{3\times 3}$:

$$\boldsymbol{R}_k := \begin{bmatrix} \Re(\beta) & -\Im(\beta) & -\lambda_k \\ \Im(\beta) & \Re(\beta) & 0 \\ \Re(\alpha\beta) & -\Im(\alpha\beta) & 1 - \Re(\alpha)\lambda_k \end{bmatrix} \tag{38}$$

We can get the characteristic polynomial of $\boldsymbol{R}_k$ with the following Mathematica command, where we use substitute the symbols $r + iu = \lambda_k$, $a = \Re(\beta)$, $b = \Im(\beta)$, $c = \Re(\alpha)$, and $d = \Im(\alpha)$.

```
CharacteristicPolynomial[{{a, -b, -(r + u I)}, {b, a, 0}, {a c - b d, -(b c + a d), 1 - c (r + u I)}}, x]
```

The command gives us the polynomial associated with eigenvalue $\lambda_k = r + iu$:

$$p_k(x) = -a^2x + a^2 + acrx + iacux + 2ax^2 - 2ax - b^2x + b^2 + bdrx + ibdux - crx^2 - icux^2 - x^3 + x^2 \tag{39}$$

Consider the case where λ_k is imaginary – i.e, $r = 0$ – which is true in all purely adversarial and bilinear zero-sum games. Then (39) simplifies to:

$$p_k(x) = -a^2x + a^2 + iacux + 2ax^2 - 2ax - b^2x + b^2 + ibdux - icux^2 - x^3 + x^2 \tag{40}$$

Our complex λ_k come in conjugate pairs where $\lambda_k = u_k i$ and $\bar{\lambda}_k = -u_k i$. (40) has the same roots for λ_k and $\bar{\lambda}_k$, which can be verified by writing the roots with the cubic formula. This corresponds to spiraling around the solution in either a clockwise or counterclockwise direction. Thus, we restrict to analyzing λ_k where u_k is positive without loss of generality.

If we make the step size α real – i.e., $d = 0$ – then (40) simplifies to:

$$p_k(x) = x(-a^2 + iacu - 2a - b^2) + a^2 + x^2(2a - icu + 1) + b^2 - x^3 \tag{41}$$

Using a heuristic from single-objective optimization, we look at making step size proportional to the inverse of the magnitude of eigenvalue k – i.e., $\alpha_k = \frac{\alpha'}{|\lambda_k|} = \frac{\alpha'}{u_k}$. With this, (41) simplifies to:

$$p_k(x) = x(-a^2 + ia\alpha' - 2a - b^2) + a^2 + x^2(2a - i\alpha' + 1) + b^2 - x^3 \tag{42}$$

Notably, in (42) there is no dependence on the components of imaginary eigenvalue $\lambda_k = r + iu = 0 + iu$, by selecting a α that is proportional to the eigenvalues inverse magnitude. We can simplify further with $a^2 + b^2 = |\beta|^2$:

$$p_k(x) = x(\Re(\beta)(i\alpha' - 2) - |\beta|^2) + x^2(2\Re(\beta) - i\alpha' + 1) + |\beta|^2 - x^3 \tag{43}$$

We could expand this in polar form for β by noting $\Re(\beta) = |\beta| \cos(\arg(\beta))$:

$$p_k(x) = x(|\beta| \cos(\arg(\beta))(i\alpha' - 2) - |\beta|^2) + x^2(2|\beta| \cos(\arg(\beta)) - i\alpha' + 1) + |\beta|^2 - x^3 \tag{44}$$

We can simplify further by considering an imaginary β – i.e., $\Re(\beta) = 0$ or $\cos(\arg(\beta)) = 0$:

$$p_k(x) = |\beta|^2 - x|\beta|^2 - x^2(i\alpha' - 1) - x^3 \tag{45}$$

The roots of these polynomials can be trivially evaluated numerically or symbolically with the by plugging in β, α, and λ_k then using the cubic formula. This section can be easily modified for the eigenvalues of the augmented dynamics for variants of complex momentum by defining the appropriate $\boldsymbol{R}$ and modifying the Mathematica command to get the characteristic polynomial for each component, which can be evaluated if it is a sufficiently low degree using known formulas.

A.3 Convergence Bounds

Corollary 1 (Convergence of Complex Momentum)**.** *There exist $\alpha \in \mathbb{R}, \beta \in \mathbb{C}$ so Alg. 1 converges for bilinear zero-sum games. More-so, for small ϵ (we show for $\epsilon = \frac{\pi}{16}$), if $\arg(\beta) = \epsilon$ (i.e., almost-positive) or $\arg(\beta) = \pi - \epsilon$ (i.e., almost-negative), then we can select $\alpha, |\beta|$ to converge.*

Proof. Note that Theorem 1 bounds the convergence rate of Alg. 1 by $\text{Sp}(\boldsymbol{R})$. Also, (41) gives a formula for 3 eigenvalues in $\text{Sp}(\boldsymbol{R})$ given α, β, and an eigenvalue $\lambda \in \text{Sp}(\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})$. The formula works by giving outputting a cubic polynomial whose roots are eigenvalues of $\text{Sp}(\boldsymbol{R})$, which can be trivially evaluated with the cubic formula.

We denote the k^{th} eigenspace of $\text{Sp}(\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})$ with eigenvalue $\lambda_k = ic_k$ and $|c_1| \leqslant \cdots \leqslant |c_n|$, because bilinear zero-sum games have purely imaginary eigenvalues due to $\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}}$ being antisymmetric. Eigenvalues come in a conjugate pairs, where $\bar{\lambda_k} = i(-c_k)$

If we select momentum coefficient $\beta = |\beta| \exp(i \arg(\beta))$ and step size $\alpha_k = \frac{\alpha'_k}{|c_k|}$, and use that $\lambda \in \text{Sp}(\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})$ are imaginary, then – as shown in Appendix Section A.2 – (41) simplifies to:

$$p_k(x) = x(|\beta| \cos(\arg(\beta))(i\alpha'_k - 2) - |\beta|^2) + x^2(2|\beta| \cos(\arg(\beta)) - i\alpha'_k + 1) + |\beta|^2 - x^3 \tag{46}$$

So, with these parameter selections, the convergence rate of Alg. 1 in the k^{th} eigenspace is bounded by the largest root of (46).

Note that if we find a separate α'_k that converges in each eigenspace k, then selected the smallest α'_k converges in every eigenspace, because the convergence rate in eigenspace k is a convex function of α that equals 1 when $\alpha = 0$ and is minimized when $\alpha > 0$.

First, consider $\arg(\beta) = \pi - \epsilon$, where $\epsilon = \frac{\pi}{16}$. We select $\alpha'_k = 0.75$ (equivalently, $\alpha_k = \frac{0.75}{|c_k|}$) and $|\beta| = 0.986$ via grid search. Using the cubic formula on the associated $p(x)$ from (46) the maximum magnitude root has size $\approx 0.9998 < 1$, so this selection converges in the k^{th} eigenspace. So, selecting:

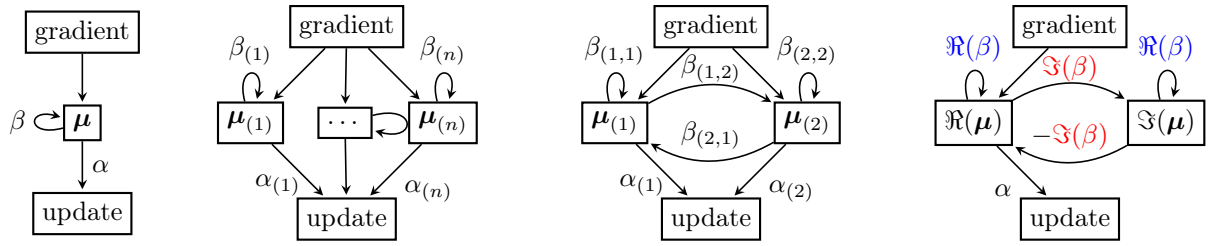
$$\hat{\alpha} \leqslant \min_k \alpha_k \tag{47}$$

$$= \min_k \frac{0.75}{c_k} \tag{48}$$

$$= \frac{0.75}{\max_k c_k} \tag{49}$$

$$= \frac{0.75}{\|\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}}\|_2} \tag{50}$$

with $\beta = 0.986 \exp(i(\pi - \epsilon))$ will converge in each eigenspace.



(a) Classical (Polyak, 1964) (b) Aggregated (Lucas et al., 2018) (c) Recurrently linked (new) (d) Complex (ours)

Figure 7: We show computational diagrams for momentum variants simultaneously updating all players parameters, which update the momentum buffers $\boldsymbol{\mu}$ at iteration $j+1$ with coefficient β via $\boldsymbol{\mu}^{j+1} = (\beta\boldsymbol{\mu}^{j} - \text{gradient})$. Our parameter update is a linear combination of the momentum buffers weighted by step sizes α. *(a)* Classical momentum (Polyak, 1964; Sutskever et al., 2013), with a single buffer and coefficient $\beta \in [0, 1)$. *(b)* Aggregated momentum (Lucas et al., 2018) which adds multiple buffers with different coefficients. *(c)* Recurrently linked momentum, which adds cross-buffer coefficients and updates the buffers with $\boldsymbol{\mu}_{(k)}^{j+1} = (\sum_l \beta_{(l,k)}\boldsymbol{\mu}_{(l)}^{j} - \text{gradient})$. We allow $\beta_{(l,k)}$ to be negative like negative momentum (Gidel et al., 2019) for solutions with simultaneous updates in adversarial games. *(d)* Complex momentum is a special case of recurrently linked momentum with two buffers and $\beta_{(1,1)} = \beta_{(2,2)} = \Re(\beta)$, $\beta_{(1,2)} = -\beta_{(2,1)} = \Im(\beta)$. Analyzing other recurrently linked momentum setups is an open problem.

Now, consider $\arg(\beta) = \epsilon = \frac{\pi}{16}$ with $\alpha_k' = 0.025$ and $|\beta| = 0.9$. Using the cubic formula on the associated $p(x)$ from (46) the maximum magnitude root has size $\approx 0.973 < 1$, so this selection converges in the k^{th} eigenspace. So, selecting:

$$\hat{\alpha} \leqslant \min_k \alpha_k \tag{51}$$

$$= \min_k \frac{0.025}{c_k} \tag{52}$$

$$= \frac{0.025}{\max_k c_k} \tag{53}$$

$$= \frac{0.025}{\|\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}}\|_2} \tag{54}$$

with $\beta = 0.9\exp(i\epsilon)$ will converge in each eigenspace.

Thus, for any of the choices of $\arg(\beta)$ we can select $\hat{\alpha}, |\beta|$ that converges in every eigenspace, and thus converges.

□

In the preceding proof, our prescribed selection of $\hat{\alpha}$ depends on knowing the largest norm eigenvalue of $\text{Sp}(\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})$, because our selections of $\hat{\alpha} \propto \frac{1}{\|\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}}\|_2}$. We may not have access to largest norm eigenvalue of $\text{Sp}(\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})$ in-practice. Nonetheless, this shows that a parameter selection exists to converge, even if it may be difficult to find. Often, in convex optimization we describe choices of α, β in terms of the largest and smallest norm eigenvalues of $\text{Sp}(\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})$ (i.e. the Hessian of the loss) (Boyd et al., 2004).

B Algorithms

Here, we include additional algorithms, which may be of use to some readers. Algorithm 3 show aggregated momentum (Lucas et al., 2018). Algorithm 4 shows the recurrently linked momentum that generalizes and unifies aggregated momentum with negative momentum (Gidel et al., 2019). Algorithm 5 shows our algorithm with alternating updates, which we use for training GANs. Algorithm 6 shows our method with all real-valued objects, if one wants to implement complex momentum in a library that does not support complex arithmetic.

Algorithm 3 Aggregated Momentum

Select number of buffers $K \in \mathbb{N}$
Select $\beta_{(k)} \in [0, 1)$ for $k = 1 \dots K$
Select $\alpha_{(k)} \in \mathbb{R}^+$ for $k = 1 \dots K$
Initialize $\boldsymbol{\mu}_{(k)}^0$ for $k = 1 \dots K$
for $j = 1 \dots N$ **do**
  for $k = 1 \dots K$ **do**
    $\boldsymbol{\mu}_{(k)}^{j+1} = \beta_{(k)} \boldsymbol{\mu}_{(k)}^j - \hat{\boldsymbol{g}}^j$
  $\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j + \sum_{k=1}^K \alpha_{(k)} \boldsymbol{\mu}_{(k)}^{j+1}$
return $\boldsymbol{\omega}_N$

Algorithm 4 Recurrently Linked Momentum

Select number of buffers $K \in \mathbb{N}$
Select $\beta_{(l,k)} \in \mathbb{R}$ for $l = 1 \dots K$ and $k = 1 \dots K$
Select $\alpha_{(k)} \in \mathbb{R}^+$ for $k = 1 \dots K$
Initialize $\boldsymbol{\mu}_{(k)}^0$ for $k = 1 \dots K$
for $j = 1 \dots N$ **do**
  for $k = 1 \dots K$ **do**
    $\boldsymbol{\mu}_{(k)}^{j+1} = \sum_l \beta_{(l,k)} \boldsymbol{\mu}_{(l)}^j - \hat{\boldsymbol{g}}^j$
  $\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j + \sum_{k=1}^K \alpha_{(k)} \boldsymbol{\mu}_{(k)}^{j+1}$
return $\boldsymbol{\omega}_N$

Algorithm 5 (AltCM) Momentum

Select $\beta \in \mathbb{C}, \alpha \in \mathbb{R}^+$
Initialize $\boldsymbol{\mu}_A^0, \boldsymbol{\mu}_B^0$
for $j = 1 \dots N$ **do**
  $\boldsymbol{\mu}_A^{j+1} = \beta \boldsymbol{\mu}_A^j - \boldsymbol{g}_A^j$
  $\boldsymbol{\theta}_A^{j+1} = \boldsymbol{\theta}_A^j + \Re(\alpha \boldsymbol{\mu}_A^{j+1})$
  $\boldsymbol{\mu}_B^{j+1} = \beta \boldsymbol{\mu}_B^j - \boldsymbol{g}_B(\boldsymbol{\theta}_A^{j+1}, \boldsymbol{\theta}_B^j)$
  $\boldsymbol{\theta}_B^{j+1} = \boldsymbol{\theta}_B^j + \Re(\alpha \boldsymbol{\mu}_B^{j+1})$
return $\boldsymbol{\omega}_N$

Algorithm 6 (SimCM) Complex Momentum - $\mathbb{R}$ valued

Select $\Re(\beta), \Im(\beta), \Re(\alpha), \Im(\alpha) \in \mathbb{R}$
Select $\Re(\beta), \Im(\beta), \Re(\alpha), \Im(\alpha) \in \mathbb{R}$
Initialize $\Re(\boldsymbol{\mu})^0, \Im(\boldsymbol{\mu})^0$
for $j = 1 \dots N$ **do**
  $\Re(\boldsymbol{\mu}^{j+1}) = \Re(\beta)\Re(\boldsymbol{\mu}^j) - \Im(\beta)\Im(\boldsymbol{\mu}^j) - \hat{\boldsymbol{g}}^j$
  $\Im(\boldsymbol{\mu}^{j+1}) = \Re(\beta)\Im(\boldsymbol{\mu}^j) + \Im(\beta)\Re(\boldsymbol{\mu}^j)$
  $\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j - \Re(\alpha)\hat{\boldsymbol{g}}^j + \Re(\alpha\beta)\Re(\boldsymbol{\mu}^j) - \Im(\alpha\beta)\Im(\boldsymbol{\mu}^j)$
return $\boldsymbol{\omega}_N$

B.1 Complex Momentum in PyTorch

Our method can be easily implemented in PyTorch 1.6+ by using complex tensors. The only necessary change to the SGD with momentum optimizer is extracting the real-component from momentum buffer as with JAX – see here.

In older versions of Pytorch, we can use a tensor to represent the momentum buffer $\boldsymbol{\mu}$, step size α, and momentum coefficient β. Specifically, we represent the real and imaginary components of the complex number independently. Then, we redefine the operations `__add__` and `__mult__` to satisfy the rules of complex arithmetic – i.e., equations (24) and (25).

C Experiments

C.1 Computing Infrastructure and Runtime

For the purely adversarial experiments in Sections 4.1 and 4.2, we do our computing in CPU. Training each 2D GAN in Section 4.3 takes 2 hours and we can train 10 simultaneously on an NVIDIA T4 GPU. Training each CIFAR GAN in Section 4.4 takes 10 hours and we can only train 1 model per NVIDIA T4 GPU.

C.2 Optimization in Purely Adversarial Games

We include the alternating update version of Fig. 2b in App. Fig. 9, which allows us to contrast simultaneous and alternating updates. With alternating updates on a Dirac-GAN for $\alpha = 0.1$ the best value for the momentum coefficient β was complex, but we could converge with real, negative momentum. Simultaneous updates may be a competitive choice with alternating updates, only if alternating updates cost two gradient evaluations per step, which is common in deep learning setups.

C.3 Adversarialnesses Effect on Convergence

We include the extragradient (EG) update with extrapolation parameter α' and step size α:

$$\begin{aligned} \boldsymbol{\omega}^{j+\frac{1}{2}} &= \boldsymbol{\omega}^j - \alpha' \hat{\boldsymbol{g}}^j \\ \boldsymbol{\omega}^{j+1} &= \boldsymbol{\omega}^j - \alpha \hat{\boldsymbol{g}}^{j+\frac{1}{2}} \end{aligned} \tag{EG}$$

and the optimistic gradient (OG) update with extrapolation parameter α' and step size α:

$$\boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j - 2\alpha \hat{\boldsymbol{g}}^j + \alpha' \hat{\boldsymbol{g}}^{j-1} \tag{OG}$$

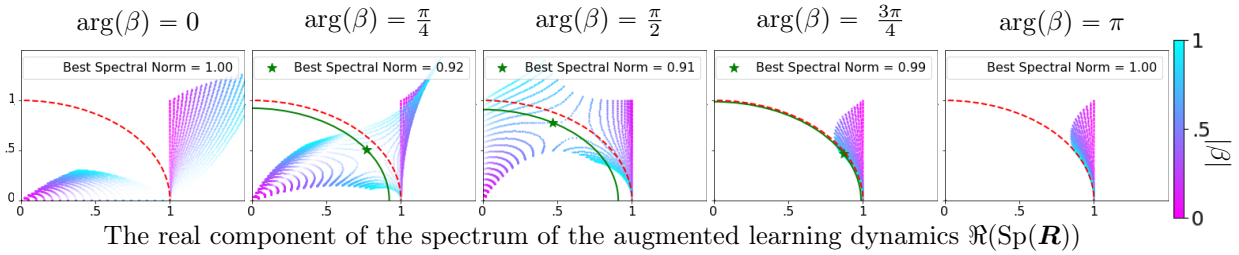


Figure 8: The spectrum of the augmented learning dynamics $\boldsymbol{R}$ is shown, whose spectral norm is the convergence rate in Theorem 1. Each image is a different momentum phase $\arg(\beta)$ for a range of $\alpha, |\beta| \in [0,1]$. The opacity of an EVal is the step size α and the color corresponds to momentum magnitude $|\beta|$. A red unit circle shows where all EVals must lie to converge for a fixed α, β. If the max EVal norm < 1, we draw a green circle whose radius is our convergence rate and a green star at the associated EVal. Notably, at every non-real β we can select $\alpha, |\beta|$ for convergence. The EVals are symmetric over the x-axis, and EVals near $\Re(\lambda) = 1$ dictate convergence rate. EVals near the center are due to state augmentation, have small magnitudes, and do not impact convergence rate. Simultaneous gradient descent corresponds to the magenta values where $|\beta| = 0$.

Often, EG and OG are used with $\alpha = \alpha'$, however we found that this constraint crippled these methods in cooperative games (i.e., minimization). As such, we tuned the extrapolation parameter α' separately from the step size α, so EG and OG were competitive baselines.

We include Fig. 10 which investigates a GANs spectrum throughout training, and elaborates on the information that is shown in Fig. 6. This shows that there are many real and imaginary eigenvalues, so GAN training is neither purely cooperative or purely adversarial. Also, the structure of the set of eigenvalues for the discriminator is different than the generator, which may motivate separate optimizer choices. The structure between the players persists through training, but the eigenvalues grow in magnitude and spread out their phases. This indicates how adversarial the game is can change during training.

C.4 Training GANs on 2D Distributions

For 2D distributions, the data is generated by sampling from a mixture of 8 Gaussian distributions, which are distributed uniformly around the unit circle.

For the GAN, we use a fully-connected network with 4 hidden ReLU (Hahnloser et al., 2000) layers with 256 hidden units. We chose this architecture to be the same as Gidel et al. (2019). Our noise source for the generator is a 4D Gaussian. We trained the models for 100 000 iterations. The performance of the optimizer settings is evaluated by computing the negative log-likelihood of a batch of 100 000 generated 2D samples.

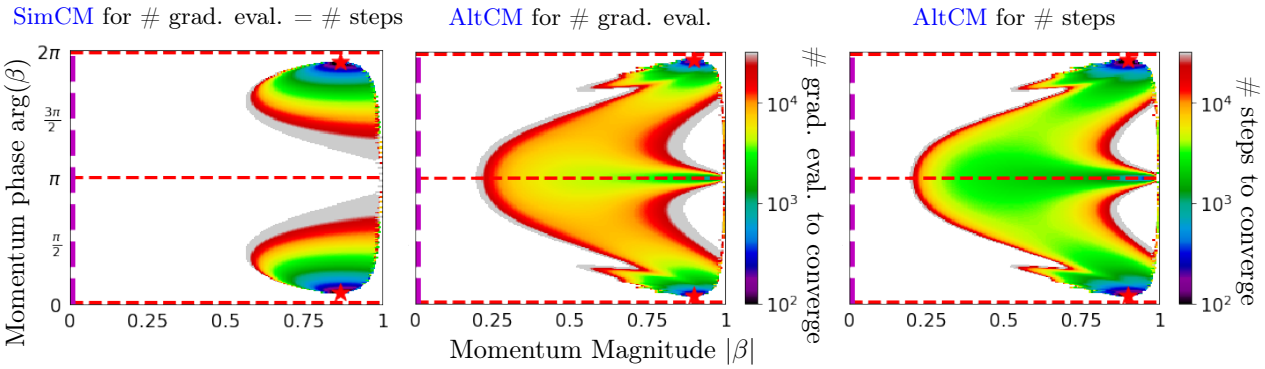


Figure 9: We show many steps and gradient evaluations, both simultaneous and alternating complex momentum on a Dirac-GAN take for a set solution distance. We fix step size $\alpha = 0.1$ as in Fig. 3, while varying the phase and magnitude of our momentum $\beta = |\beta| \exp(i \arg(\beta))$. There is a red star at the optima, dashed red lines at real β, and a dashed magenta line for simultaneous or alternating gradient descent. We only display color for convergent setups. *Left:* Simultaneous complex momentum (SimCM). This is the same as Fig. 2b, which we repeat to contrast with alternating updates. There are no real-valued β that converge for this – or any – α with simultaneous updates (Gidel et al., 2019). Simultaneous updates can parallelize gradient computation for all players at each step, thus costing only one gradient evaluation per step for many deep learning setups. The best rate of convergence per step and gradient evaluation is ≈ 0.955. *Middle:* Alternating complex momentum (AltCM), where we show how many gradient evaluations – as opposed to steps – to reach a set solution distance. Alternating updates are bottlenecked by waiting for first player's update to compute the second players update, effectively costing two gradient evaluations per step for many deep learning setups. Negative momentum can converge here, as shown by Gidel et al. (2019), but the best momentum is still complex. Also, Alternating updates can make the momentum phase $\arg(\beta)$ choice less sensitive to our convergence. The best rate of convergence per gradient evaluation is ≈ 0.965. *Right:* AltCM, where we show how many steps to reach a set solution distance. The best rate of convergence per step is ≈ 0.931. **Takeaway:** If we can parallelize computation of both players gradients we can benefit from SimCM, however if we can not then AltCM can converge more quickly and for a broader set of optimizer parameters. In any case, the best solution uses a complex momentum β for this α.

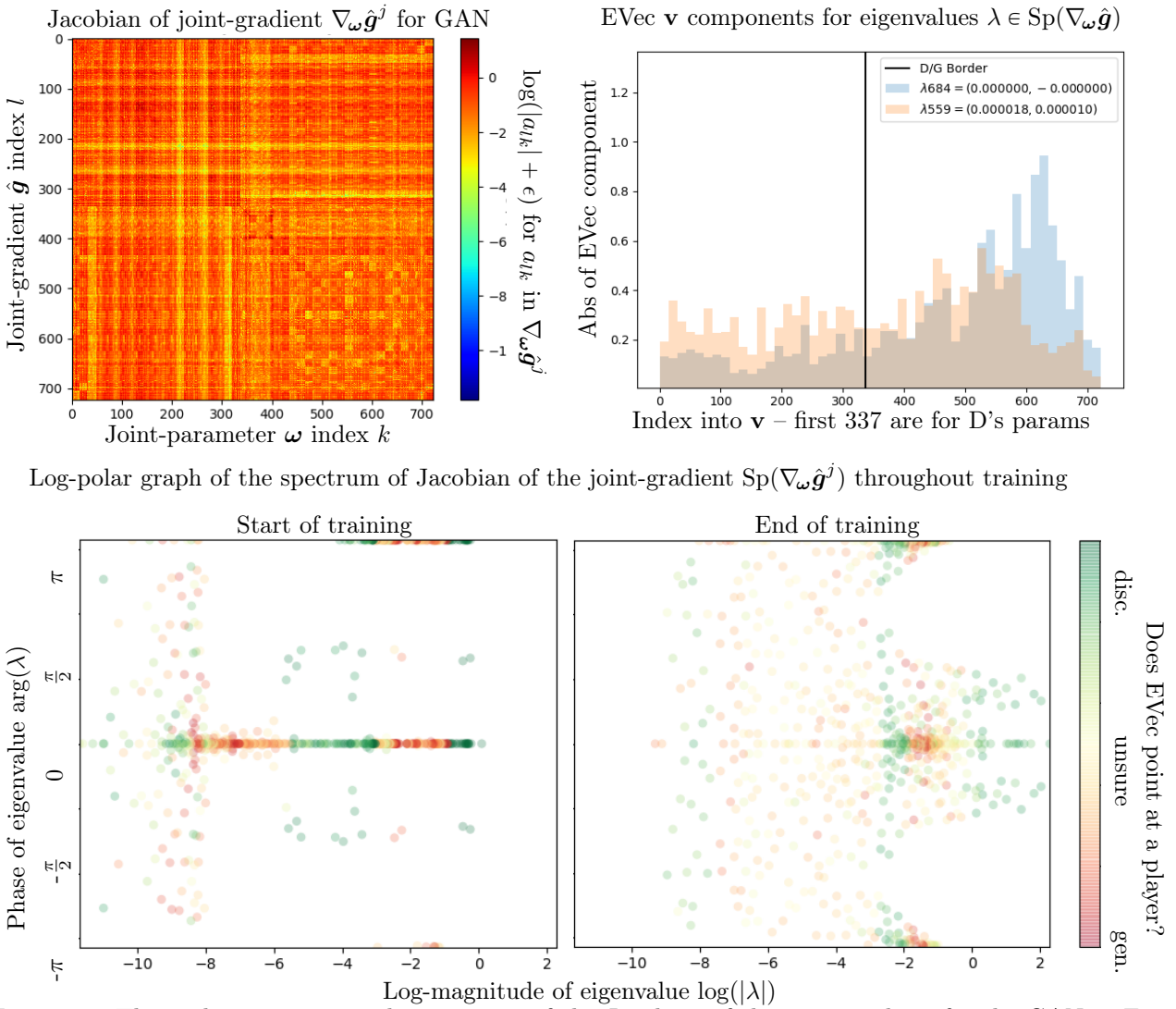


Figure 10: These plots investigate the spectrum of the Jacobian of the joint-gradient for the GAN in Fig. 6 through training. The spectrum is key for bounding convergence rates in learning algorithms.

Top left: The Jacobian $\nabla_{\omega}\hat{g}$ for a GAN on a 2D mixture of Gaussians with a two-layer, fully-connected 16 hidden unit discriminator (D) and generator (G) at the end of training. In the concatenated parameters $\omega \in \mathbb{R}^{723}$, the first 337 are for D, while the last 386 are for G. We display the log of the absolute value of each component plus $\epsilon = 10^{-10}$. The upper left and lower right quadrants are the Hessian of D and G's losses respectively.

Top Right: We visualize two randomly sampled EVecs from $\nabla_{\omega}\hat{g}$. The first part of the parameters is for the discriminator, while the second part is for the generator. Given an eigenvalue with EVec $\mathbf{v}$, we roughly approximate attributing EVecs to players by calculating how much of it lies in D's parameter space with $\frac{\|\mathbf{v}_{1:|\mathrm{D}|}\|_1}{\|\mathbf{v}\|_1} = \frac{\|\mathbf{v}_{1:337}\|_1}{\|\mathbf{v}\|_1}$. If this ratio is near 1 (or 0) and say *the EVec mostly points at D (or G)*. The blue EVec mostly points at G, while the orange EVec is unclear. Finding useful ways to attribute eigenvalues to players is an open problem.

Bottom: The spectrum of the Jacobian of the joint-gradient $\mathrm{Sp}(\nabla_{\omega}\hat{g}^j)$ is shown in log-polar coordinates, because it is difficult to see structure when graphing in Cartesian (i.e., $\Re$ and $\Im$) coordinates, due to eigenvalues spanning orders of magnitude, while being positive and negative. The end of training is when we stop making progress on the log-likelihood. We have imaginary eigenvalues at $\arg(\lambda) = \pm\pi/2$, positive eigenvalues at $\arg(\lambda) = 0$, and negative eigenvalues at $\arg(\lambda) = \pm\pi$.

Takeaway: There is a banded structure for the coloring of the eigenvalues that persists through training. We may want different optimizer parameters for the discriminator and generator, due to asymmetry in their associated eigenvalues. Also, the magnitude of the eigenvalues grows during training, and the args spread out indicating the game can change eigenstructure near solutions.

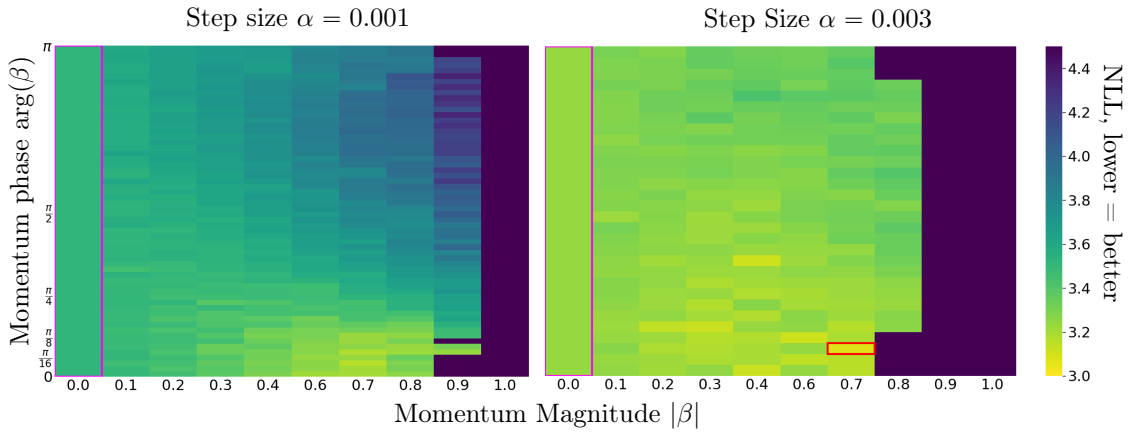


Figure 11: Heatmaps of the negative log-likelihood (NLL) for tuning $\arg(\beta), |\beta|$ with various fixed α on a 2D mixture of Gaussians GAN. We highlight the best performing cell in red, which had $\arg(\beta) \approx \pi/8$. Runs equivalent to alternating SGD are shown in a magenta box. We compare to negative momentum with alternating updates as in Gidel et al. (2019) in the top row with $\arg(\beta) = \pi$. *Left*: Tuning the momentum with $\alpha = 0.001$. *Right*: Tuning the momentum with $\alpha = 0.003$.

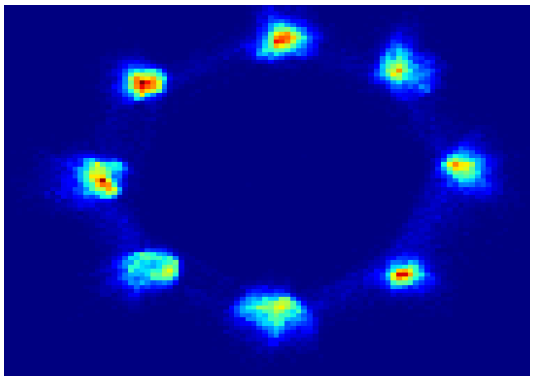


Figure 12(a): Mixture of Gaussian samples from GAN with the best hyperparameters from Fig. 11

Figure 12(b): Class-conditional CIFAR-10 samples from GAN with the best hyperparameters from Fig. 13a

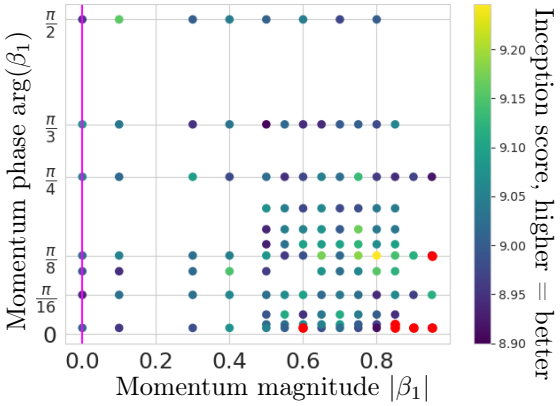


Figure 13(a): The inception score (IS) for a grid search on $\arg(\beta_1)$ and $|\beta_1|$ for training BigGAN on CIFAR-10 with the Adam variant in Algorithm 2. The β_1 is complex for the discriminator, while the generator's optimizer is fixed to author-supplied defaults. Red points are runs that failed to train to the minimum IS in the color bar. The vertical magenta line denotes runs equivalent to alternating SGD. Negative momentum failed to train for any momentum magnitude $|\beta_1| > .5$, so we do not display it for more resolution near values of interest.

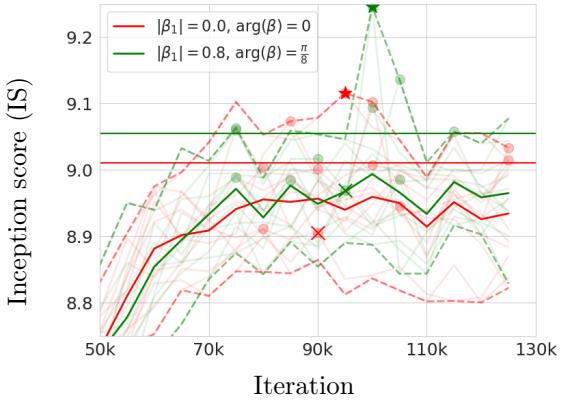


Figure 13(b): We compare the best optimization parameters from grid search Fig. 13a for our complex Adam variant (i.e., Algorithm 2) shown in green, with the author provided values shown in red for the CIFAR-10 BigGAN over 10 seeds. A star is displayed at the best IS over all runs, a cross is displayed at the worst IS over all runs, while a circle is shown at the best IS for each run. Dashed lines are shown at the max/min IS over all runs at each iteration, low-alpha lines are shown for each runs IS, while solid lines are shown for the average IS over all seeds at each iteration. The results are summarized in Table 1.

Table 2: Notation

SGD	Stochastic Gradient Descent
CM	Complex Momentum
SGDm, SimSGDm, ...	... with momentum
SimSGD, SimCM	Simultaneous ...
AltSGD, AltCM	Alternating ...
GAN	Generative Adversarial Network (Goodfellow et al., 2014)
EG	Extragradient (Korpelevich, 1976)
OG	Optimistic Gradient (Daskalakis et al., 2018)
IS	Inception Score (Salimans et al., 2016)
$:=$	Defined to be equal to
$x, y, z, \cdots \in \mathbb{C}$	Scalars
$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \cdots \in \mathbb{C}^n$	Vectors
$\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}, \cdots \in \mathbb{C}^{n \times n}$	Matrices
$\boldsymbol{X}^\top$	The transpose of matrix $\boldsymbol{X}$
$\boldsymbol{I}$	The identity matrix
$\Re(z), \Im(z)$	The real or imaginary component of $z \in \mathbb{C}$
i	The imaginary unit. $z \in \mathbb{C} \implies z = \Re(z) + i\Im(z)$
$\bar{z}$	The complex conjugate of $z \in \mathbb{C}$
$\lvert z \rvert := \sqrt{z\bar{z}}$	The magnitude or modulus of $z \in \mathbb{C}$
$\arg(z)$	The argument or phase of $z \in \mathbb{C} \implies z = \lvert z \rvert \exp(i \arg(z))$
$z \in \mathbb{C}$ is *almost-positive*	$\arg(z) = \epsilon$ for small ϵ respectively
A, B	A symbol for the outer/inner players
$d_A, d_B \in \mathbb{N}$	The number of weights for the outer/inner players
$\boldsymbol{\theta}$	A symbol for the parameters or weights of a player
$\boldsymbol{\theta}_A \in \mathbb{R}^{d_A}, \boldsymbol{\theta}_B \in \mathbb{R}^{d_B}$	The outer/inner parameters or weights
$\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$	A symbol for a loss
$\mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B), \mathcal{L}_B(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)$	The outer/inner losses – $\mathbb{R}^{d_A+d_B} \mapsto \mathbb{R}$
$\boldsymbol{g}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B), \boldsymbol{g}_B(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)$	Gradient of outer/inner losses w.r.t. their weights in $\mathbb{R}^{d_A/d_B}$
$\boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A) := \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)$	The best-response of the inner player to the outer player
$\mathcal{L}_A^*(\boldsymbol{\theta}_A) := \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A))$	The outer loss with a best-responding inner player
$\boldsymbol{\theta}_A^* := \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A^*(\boldsymbol{\theta}_A)$	Outer optimal weights with a best-responding inner player
$d := d_A + d_B$	The combined number of weights for both players
$\boldsymbol{\omega} := [\boldsymbol{\theta}_A, \boldsymbol{\theta}_B] \in \mathbb{R}^d$	A concatenation of the outer/inner weights
$\hat{\boldsymbol{g}}(\boldsymbol{\omega}) := [\boldsymbol{g}_A(\boldsymbol{\omega}), \boldsymbol{g}_B(\boldsymbol{\omega})] \in \mathbb{R}^d$	A concatenation of the outer/inner gradients
$\boldsymbol{\omega}^0 = [\boldsymbol{\theta}_A^0, \boldsymbol{\theta}_B^0] \in \mathbb{R}^d$	The initial parameter values
j	An iteration number
$\hat{\boldsymbol{g}}^j := \hat{\boldsymbol{g}}(\boldsymbol{\omega}^j) \in \mathbb{R}^d$	The joint-gradient vector field at weights $\boldsymbol{\omega}^j$
$\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}}^j := \nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}}\lvert_{\boldsymbol{\omega}^j} \in \mathbb{R}^{d \times d}$	The Jacobian of the joint-gradient $\hat{\boldsymbol{g}}$ at weights $\boldsymbol{\omega}^j$
$\alpha \in \mathbb{C}$	The step size or learning rate
$\beta \in \mathbb{C}$	The momentum coefficient
$\beta_1 \in \mathbb{C}$	The first momentum parameter for Adam
$\boldsymbol{\mu} \in \mathbb{C}^d$	The momentum buffer
$\lambda \in \mathbb{C}$	Notation for an arbitrary EVal
$\mathrm{Sp}(\boldsymbol{M}) \in \mathbb{C}^n$	The spectrum – or set of EVals – of $\boldsymbol{M} \in \mathbb{R}^{n \times n}$
Purely adversarial/cooperative game	$\mathrm{Sp}(\nabla_{\boldsymbol{\omega}} \hat{\boldsymbol{g}})$ is purely real/imaginary
$\rho(\boldsymbol{M}) := \max_{z \in \mathrm{Sp}(\boldsymbol{M})} \lvert z \rvert$	The spectral radius in $\mathbb{R}^+$ of $\boldsymbol{M} \in \mathbb{R}^{n \times n}$
$\boldsymbol{F}_{\alpha,\beta}([\boldsymbol{\mu}, \boldsymbol{\omega}])$	Fixed point op. for CM, or augmented learning dynamics
$\boldsymbol{R} := \nabla_{[\boldsymbol{\mu},\boldsymbol{\omega}]} \boldsymbol{F}_{\alpha,\beta} \in \mathbb{R}^{3d \times 3d}$	Jacobian of the augmented learning dynamics in Corollary 1
$\alpha^*, \beta^* := \arg\min_{\alpha,\beta} \rho(\boldsymbol{R}(\alpha, \beta))$	The optimal step size and momentum coefficient
$\rho^* := \rho(R(\alpha^*, \beta^*))$	The optimal spectral radius or convergence rate
$\kappa := \frac{\max \mathrm{Sp}(\nabla_{\boldsymbol{\omega}} \boldsymbol{g})}{\min \mathrm{Sp}(\nabla_{\boldsymbol{\omega}} \boldsymbol{g})}$	Condition number, for convex single-objective optimization
$\sigma_{min}^2(\boldsymbol{M}) := \max \mathrm{Sp}(\boldsymbol{M}^\top \boldsymbol{M})$	The minimum singular value of a matrix $\boldsymbol{M}$