Complex Momentum for Optimization in Games

Artificial Intelligence and Statistics (AISTATS) 2022

Jonathan Lorraine[1,2], David Acuna[1,2,3], Paul Vicol[1,2], David Duvenaud[1,2]

March 14^{th}, 2022

University of Toronto[1], Vector Institute[2], NVIDIA[3]

Motivation

- *Games* generalize single objective opt. to have multiple objectives, each with their own parameters.

Motivation

- *Games* generalize single objective opt. to have multiple objectives, each with their own parameters.
- But, single-objective opt. methods often don't work for games.

Motivation

- *Games* generalize single objective opt. to have multiple objectives, each with their own parameters.
- But, single-objective opt. methods often don't work for games.
- Gidel et al. [1] show a negative momentum coefficient can help opt. in *adversarial games*.

Motivation

- *Games* generalize single objective opt. to have multiple objectives, each with their own parameters.
- But, single-objective opt. methods often don't work for games.
- Gidel et al. [1] show a negative momentum coefficient can help opt. in *adversarial games*.
- But, can harm performance if the game is *non-adversarial*, and requires alternating updates costing 2 grad. evaluations per iteration.

Benefits of our method

We propose Complex Momentum which:

Benefits of our method

We propose Complex Momentum which:

1. Converges with 1 grad. eval. per iteration in adversarial games.

Benefits of our method

We propose Complex Momentum which:

1. Converges with 1 grad. eval. per iteration in adversarial games.
2. Is tractable to analyze with Euler's formula.

Benefits of our method

We propose Complex Momentum which:

1. Converges with 1 grad. eval. per iteration in adversarial games.
2. Is tractable to analyze with Euler's formula.
3. Is trivial to implement.

Benefits of our method

We propose Complex Momentum which:

1. Converges with 1 grad. eval. per iteration in adversarial games.
2. Is tractable to analyze with Euler's formula.
3. Is trivial to implement.
4. Only introduces a single optimization hyperparameter.

Benefits of our method

We propose Complex Momentum which:

1. Converges with 1 grad. eval. per iteration in adversarial games.
2. Is tractable to analyze with Euler's formula.
3. Is trivial to implement.
4. Only introduces a single optimization hyperparameter.
5. Robustly converges in non-adversarial games.

Actual JAX [2] Complex Momentum 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
 ...
```

Background: Minimization with momentum

- The standard minimization problem for loss $\mathcal{L}$ with parameters $\boldsymbol{\theta}$:

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

Background: Minimization with momentum

- The standard minimization problem for loss $\mathcal{L}$ with parameters $\boldsymbol{\theta}$:

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

- The loss gradient at parameters $\boldsymbol{\theta}^j$ denoted by:

$$\boldsymbol{g}^j := \boldsymbol{g}(\boldsymbol{\theta}^j) := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}^j} \tag{2}$$

Background: Minimization with momentum

- The standard minimization problem for loss $\mathcal{L}$ with parameters $\boldsymbol{\theta}$:

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

- The loss gradient at parameters $\boldsymbol{\theta}^j$ denoted by:

$$\boldsymbol{g}^j := \boldsymbol{g}(\boldsymbol{\theta}^j) := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}^j} \tag{2}$$

- We can locally optimize loss using SGD with momentum:

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

Setting up the notation for our method

- Optimization in games generalizes single-objective minimization:

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

Setting up the notation for our method

- Optimization in games generalizes single-objective minimization:

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

- Notation: Concatenate the players' params and grads together

$$\boldsymbol{\omega} := [\boldsymbol{\theta}_A, \boldsymbol{\theta}_B], \quad \hat{\boldsymbol{g}}^j :== [\boldsymbol{g}_A^j, \boldsymbol{g}_B^j] \tag{4}$$

Setting up the notation for our method

- Optimization in games generalizes single-objective minimization:

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

- Notation: Concatenate the players' params and grads together

$$\boldsymbol{\omega} := [\boldsymbol{\theta}_A, \boldsymbol{\theta}_B], \quad \hat{\boldsymbol{g}}^j :== [\boldsymbol{g}_A^j, \boldsymbol{g}_B^j] \tag{4}$$

- Simultaneous SGD is a common gradient-based opt. strategy:

$$\boldsymbol{\theta}_A^{j+1} = \boldsymbol{\theta}_A^j - \alpha \boldsymbol{g}_A^j, \boldsymbol{\theta}_B^{j+1} = \boldsymbol{\theta}_B^j - \alpha \boldsymbol{g}_B^j \iff \boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j - \alpha \hat{\boldsymbol{g}}^j$$

(SimSGD)

Setting up the notation for our method

- Simultaneous SGD is a common gradient-based opt. strategy:

$$\boldsymbol{\theta}_A^{j+1} = \boldsymbol{\theta}_A^j - \alpha \boldsymbol{g}_A^j, \boldsymbol{\theta}_B^{j+1} = \boldsymbol{\theta}_B^j - \alpha \boldsymbol{g}_B^j \iff \boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^j - \alpha \hat{\boldsymbol{g}}^j \quad \text{(SimSGD)}$$

- Or, (simultaneous) SGD with (negative [1]) momentum:

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

Our Method - Complex Momentum (CM)

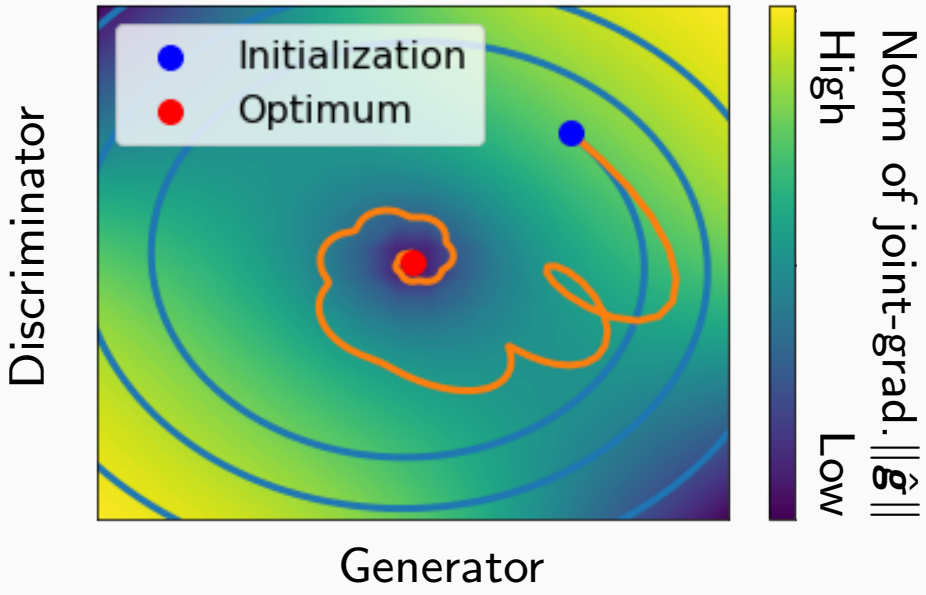
- Or, (simultaneous) SGD with (negative [1]) momentum:

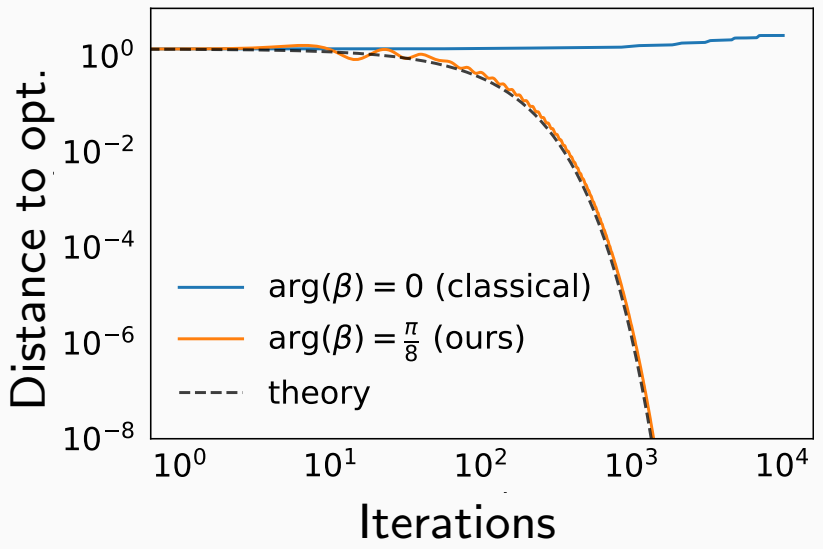
$$\boldsymbol{\mu}^{j+1} = \beta \boldsymbol{\mu}^{j} - \hat{\mathbf{g}}^{j}, \quad \boldsymbol{\omega}^{j+1} = \boldsymbol{\omega}^{j} + \alpha \boldsymbol{\mu}^{j+1} \qquad \text{(SimSGDm)}$$

- We make $\beta \in \mathbb{C}$ instead of $\mathbb{R}$. Then $\boldsymbol{\mu}^{j+1} \in \mathbb{C}$, but need $\mathbb{R}$-valued parameter update. We simply take the real part:

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

Using CM on a Dirac-GAN



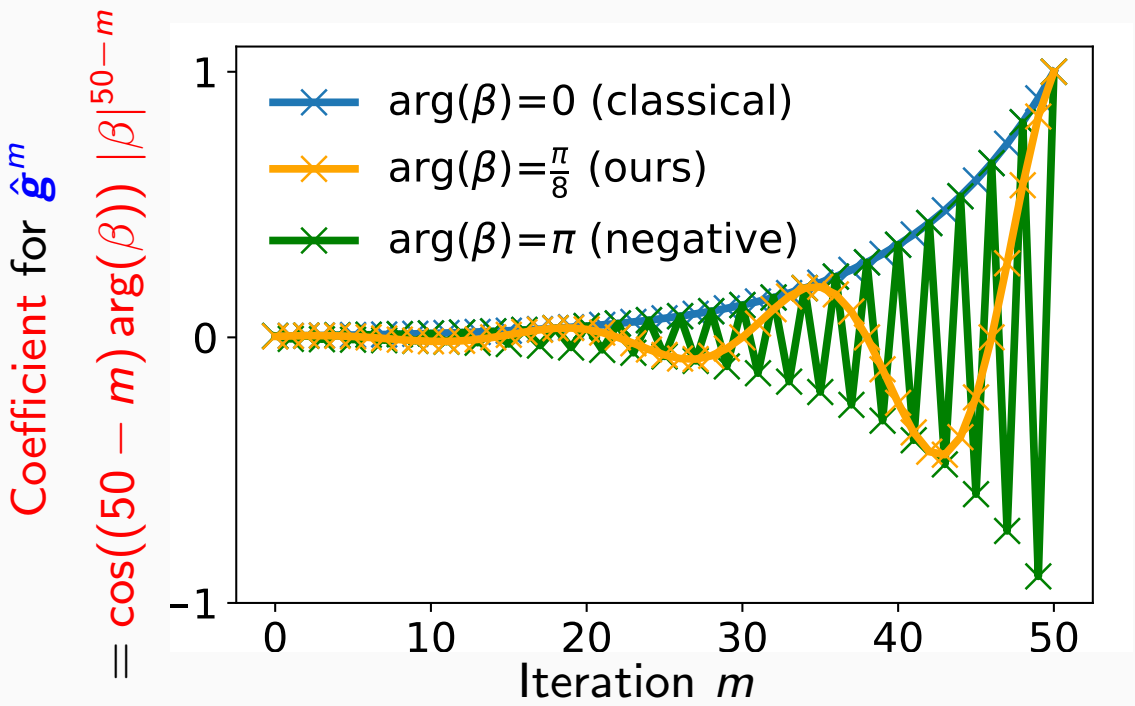


CM connects Classic & Negative Momentum

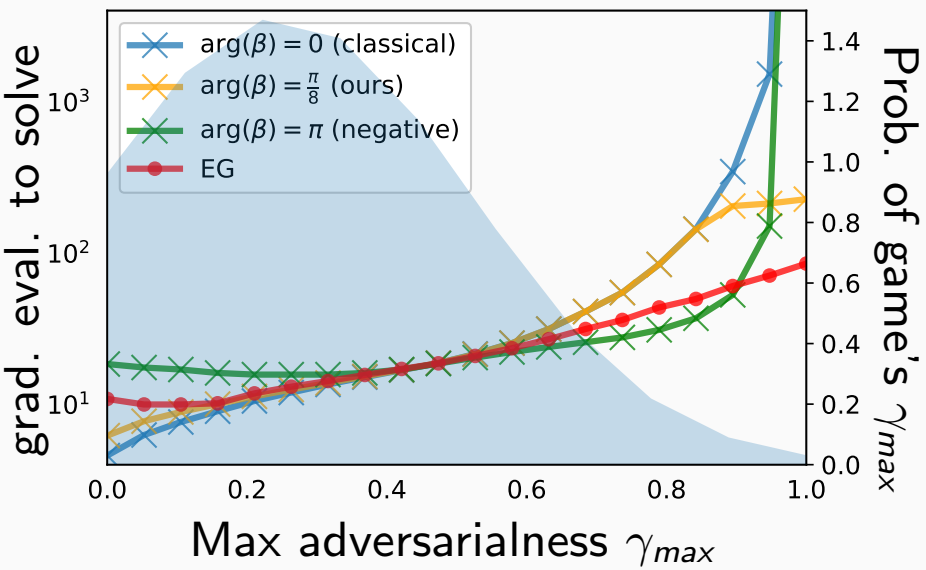
- By combining Euler's Formula and $\boldsymbol{\mu}^{j+1} = -\sum_{k=0}^{k=j} \beta^k \hat{\boldsymbol{g}}^{j-k}$:

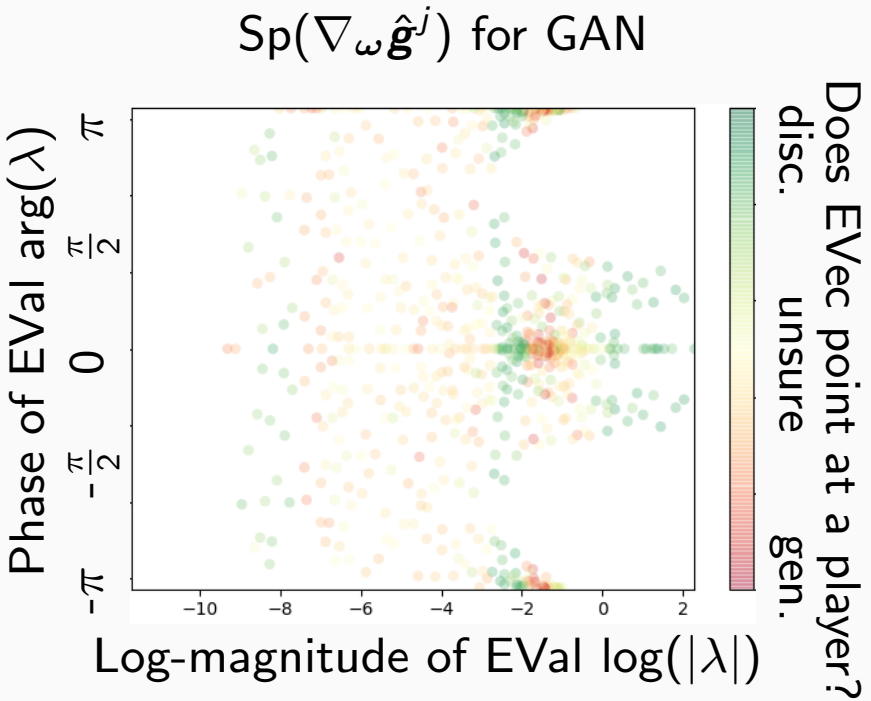
$$\Re(\boldsymbol{\mu}^{j+1}) = -\sum_{k=0}^{k=j} \cos(k \arg(\beta)) \ |\beta|^k \hat{\boldsymbol{g}}^{j-k} \tag{5}$$

CM's $j = 50^{th}$ update's dependence on past grads



Comparing optimization algorithms





Training BigGAN with a Complex Adam

Algorithm 1 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 $\boldsymbol{\omega}^N$

Training BigGAN with a Complex Adam

	CIFAR-10 IS	
Discriminator β_1	Max	Mean
0, BigGAN's [3]	9.10	8.93
$.8\exp(i\pi/8)$, ours	9.25(+.15)	8.97(+.04)
.8, ablation	9.05(−.05)	7.19(−1.7)

Experimental takeaways

- Using an *almost-positive* complex momentum – i.e., $\arg(\beta)$ is near, but not 0 – allows convergence in any setup with 1 grad. eval.

Experimental takeaways

- Using an *almost-positive* complex momentum – i.e., $\arg(\beta)$ is near, but not 0 – allows convergence in any setup with 1 grad. eval.
- Almost-positive momentum approaches classical momentum, gaining similar acceleration properties in *cooperative games*.

Experimental takeaways

- Using an *almost-positive* complex momentum – i.e., $\arg(\beta)$ is near, but not 0 – allows convergence in any setup with 1 grad. eval.
- Almost-positive momentum approaches classical momentum, gaining similar acceleration properties in *cooperative games*.
- A default of $\arg(\beta) = \pi/8$ did well across our experiments.

Experimental takeaways

- Using an *almost-positive* complex momentum – i.e., $\arg(\beta)$ is near, but not 0 – allows convergence in any setup with 1 grad. eval.
- Almost-positive momentum approaches classical momentum, gaining similar acceleration properties in *cooperative games*.
- A default of $\arg(\beta) = \pi/8$ did well across our experiments.
- We were able to extend our method to more sophisticated optimizers, training BigGAN to better inception scores.

Thanks!

Jonathan Lorraine

David Acuna

Paul Vicol

David Duvenaud

References

[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 *AISTATS*, 2019.

[2] 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`.

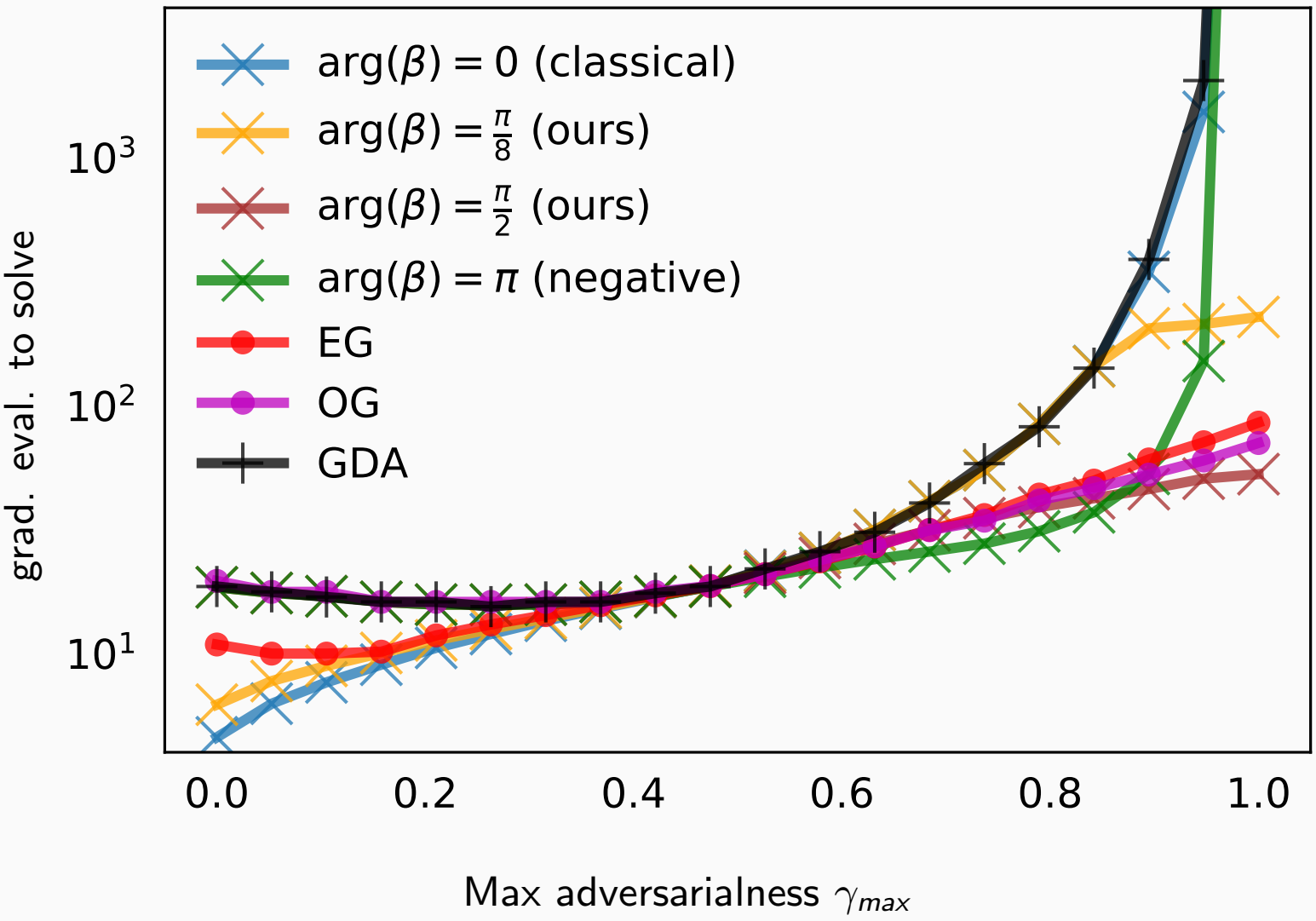
[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

Extra Slides

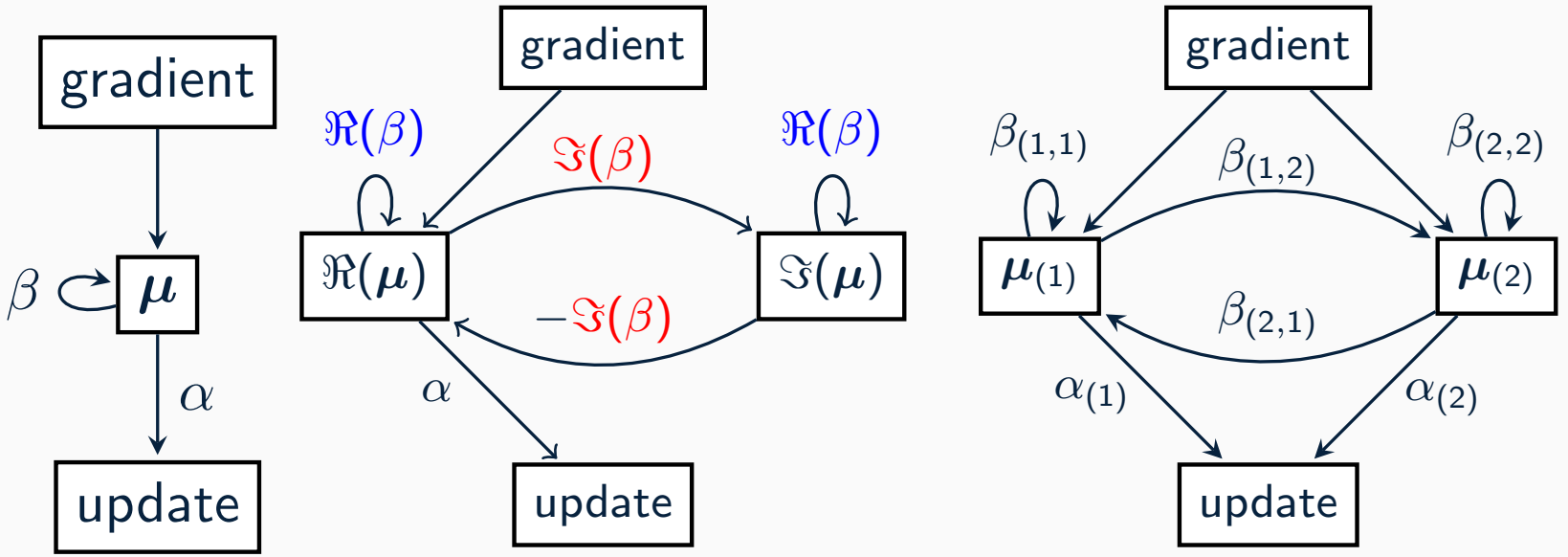
Summary

- Script for associated talk is here.
- Animation of our method is here.

Comparing optimization algorithms



Future Directions



(a) Classical **(b)** Complex (ours) **(c)** Recurrently linked (future)

- How to best extend to Adam?
- What assumptions about game eigenstructure are useful for realistic problems like GANs?