

# Lyapunov Exponents for Diversity in Differentiable Games

International Conference on Autonomous and Multi-Agent Systems (AAMAS) 2022

---

Jonathan Lorraine<sup>1,2</sup>, Paul Vicol<sup>1,2</sup>, Jack Parker-Holder<sup>3</sup>, Tal Kachman<sup>4</sup>, Luke Metz<sup>5</sup>, Jakob Foerster<sup>3</sup>

February 7<sup>th</sup>, 2022

University of Toronto<sup>1</sup>, Vector Institute<sup>2</sup>, University of Oxford<sup>3</sup>, Radboud University<sup>4</sup>, Google Brain<sup>5</sup>

- When minimizing objectives, randomly initializing then optimizing can fail to find different solutions.

# Overview

- When minimizing objectives, randomly initializing then optimizing can fail to find different solutions.
- Ex., in the Iterated Prisoner's Dilemma agents will often learn to "battle" instead of "cooperating" [1].

- When minimizing objectives, randomly initializing then optimizing can fail to find different solutions.
- Ex., in the Iterated Prisoner's Dilemma agents will often learn to "battle" instead of "cooperating" [1].
- We generalize Ridge Rider [2] to differentiable games, providing a method which finds bifurcations and branches the optimization process across them: [animation](#)

- When minimizing objectives, randomly initializing then optimizing can fail to find different solutions.
- Ex., in the Iterated Prisoner's Dilemma agents will often learn to "battle" instead of "cooperating" [1].
- We generalize Ridge Rider [2] to differentiable games, providing a method which finds bifurcations and branches the optimization process across them: [animation](#)
- How do we find bifurcations? With Lyapunov exponent based objectives: [animation](#)

- Why do we want to find different solutions? For example...

- Why do we want to find different solutions? For example...
- In image classification, some generalize better than others - ex., shape vs. texture solutions.

- Why do we want to find different solutions? For example...
- In image classification, some generalize better than others - ex., shape vs. texture solutions.
- In differentiable games, some solutions have much higher social welfare – ex., cooperating vs. battling.



- Why do we want to find different solutions? For example...
- In image classification, some generalize better than others - ex., shape vs. texture solutions.
- In differentiable games, some solutions have much higher social welfare – ex., cooperating vs. battling.
- But, what is a differentiable game?

## Background: Differentiable Games

- Differentiable games generalize single-objective minimization:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*),$$

$$\theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B)$$

## Background: Differentiable Games

- Differentiable games generalize single-objective minimization:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*),$$

$$\theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B)$$

- Games are increasingly important in ML – ex., GANs [3–5], hyperparameter optimization [6–8], meta-learning, self-play, models for RL, adversarial examples, numerous others.

## Background: Differentiable Games

- Differentiable games generalize single-objective minimization:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*),$$

$$\theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B)$$

- Games are increasingly important in ML – ex., GANs [3–5], hyperparameter optimization [6–8], meta-learning, self-play, models for RL, adversarial examples, numerous others.
- Today's example: The Iterated Prisoner's Dilemma (then GANs).

## Background: Differentiable Games

- Differentiable games generalize single-objective minimization:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*),$$

$$\theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B)$$

- Games are increasingly important in ML – ex., GANs [3–5], hyperparameter optimization [6–8], meta-learning, self-play, models for RL, adversarial examples, numerous others.
- Today's example: The Iterated Prisoner's Dilemma (then GANs).
- An infinitely repeated version of the Prisoner's Dilemma, where agents choose to cooperate or defect each round.

## Background: Differentiable Games

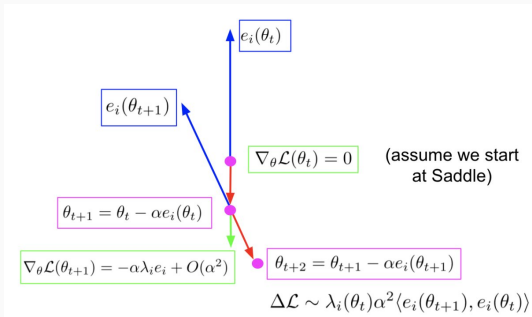
- Differentiable games generalize single-objective minimization:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*),$$

$$\theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B)$$

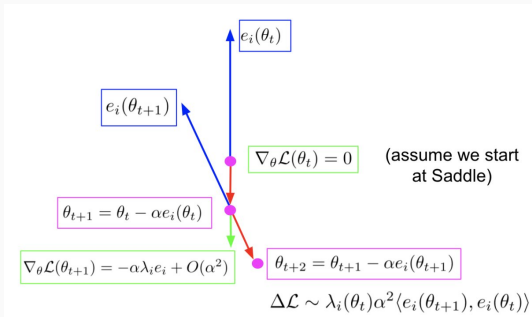
- Games are increasingly important in ML – ex., GANs [3–5], hyperparameter optimization [6–8], meta-learning, self-play, models for RL, adversarial examples, numerous others.
- Today's example: The Iterated Prisoner's Dilemma (then GANs).
- An infinitely repeated version of the Prisoner's Dilemma, where agents choose to cooperate or defect each round.
- Notable solutions: Defect-defect (DD) where agents always defect, and tit-for-tat (TT) which repeats what the opponent did last round allowing for higher welfare via cooperation.

## Background: Ridge Rider



- Ridge rider (RR) [2] finds diverse solutions in single-objective optimization by branching optimization at saddle points.

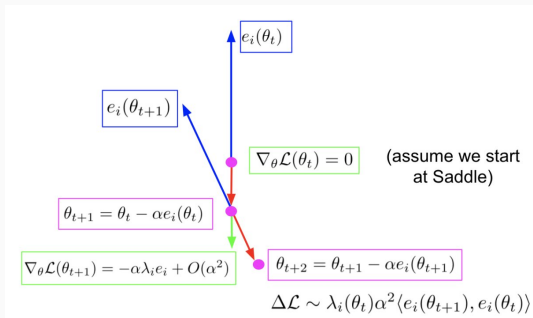
## Background: Ridge Rider



- Ridge rider (RR) [2] finds diverse solutions in single-objective optimization by branching optimization at saddle points.
- Optimization is branched by following/"riding" the most negative eigenvectors of the Hessian.



## Background: Ridge Rider



- Ridge rider (RR) [2] finds diverse solutions in single-objective optimization by branching optimization at saddle points.
- Optimization is branched by following/"riding" the most negative eigenvectors of the Hessian.
- Notable uses: Zero-shot learning, out-of-distribution generalization

## Differentiable Games + Ridge Rider?

- What if we wanted to use Ridge Rider to find solutions in multi-agent setups like the Iterated Prisoner's Dilemma (IPD)?

## Differentiable Games + Ridge Rider?

- What if we wanted to use Ridge Rider to find solutions in multi-agent setups like the Iterated Prisoner's Dilemma (IPD)?
- In differentiable games, there's no single Hessian with eigenvectors to follow. Ridge Rider is not defined!

## Differentiable Games + Ridge Rider?

- What if we wanted to use Ridge Rider to find solutions in multi-agent setups like the Iterated Prisoner's Dilemma (IPD)?
- In differentiable games, there's no single Hessian with eigenvectors to follow. Ridge Rider is not defined!
- Definition: *bifurcations* are where small initial parameter changes cause final solution differences

## Differentiable Games + Ridge Rider?

- What if we wanted to use Ridge Rider to find solutions in multi-agent setups like the Iterated Prisoner's Dilemma (IPD)?
- In differentiable games, there's no single Hessian with eigenvectors to follow. Ridge Rider is not defined!
- Definition: *bifurcations* are where small initial parameter changes cause final solution differences
- **Key insight:** RR was finding saddle point bifurcations and branching across them.

## Differentiable Games + Ridge Rider?

- What if we wanted to use Ridge Rider to find solutions in multi-agent setups like the Iterated Prisoner's Dilemma (IPD)?
- In differentiable games, there's no single Hessian with eigenvectors to follow. Ridge Rider is not defined!
- Definition: *bifurcations* are where small initial parameter changes cause final solution differences
- **Key insight:** RR was finding saddle point bifurcations and branching across them.
- In differentiable games, many more bifurcation types because non-conservative dynamics.

## Differentiable Games + Ridge Rider?

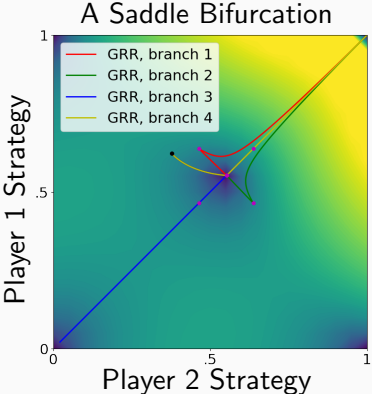
- What if we wanted to use Ridge Rider to find solutions in multi-agent setups like the Iterated Prisoner's Dilemma (IPD)?
- In differentiable games, there's no single Hessian with eigenvectors to follow. Ridge Rider is not defined!
- Definition: *bifurcations* are where small initial parameter changes cause final solution differences
- **Key insight:** RR was finding saddle point bifurcations and branching across them.
- In differentiable games, many more bifurcation types because non-conservative dynamics.
- **Strategy:** can we find bifurcations and branch across them?

## Differentiable Games + Ridge Rider?

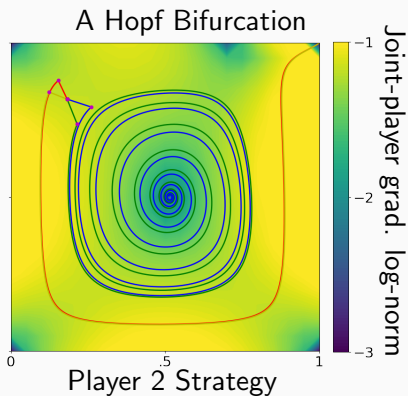
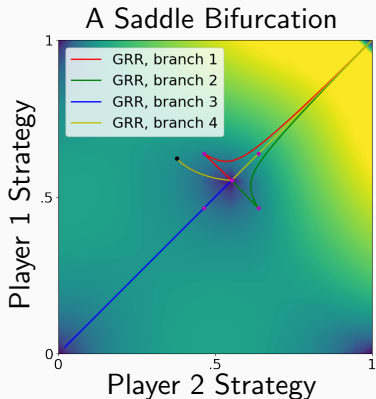
- What if we wanted to use Ridge Rider to find solutions in multi-agent setups like the Iterated Prisoner's Dilemma (IPD)?
- In differentiable games, there's no single Hessian with eigenvectors to follow. Ridge Rider is not defined!
- Definition: *bifurcations* are where small initial parameter changes cause final solution differences
- **Key insight:** RR was finding saddle point bifurcations and branching across them.
- In differentiable games, many more bifurcation types because non-conservative dynamics.
- **Strategy:** can we find bifurcations and branch across them?
- Now, let's look at toys to illustrate the difference:



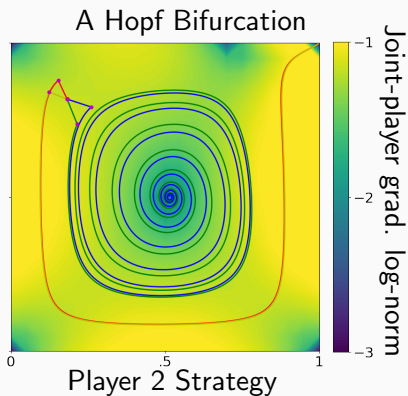
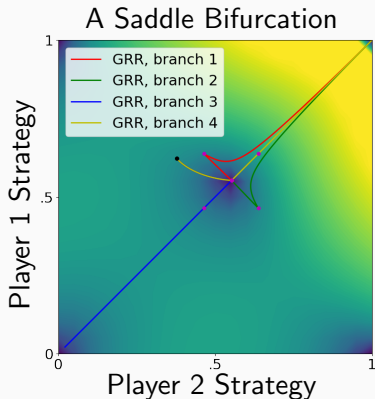
# Showing the Bifurcations



# Showing the Bifurcations

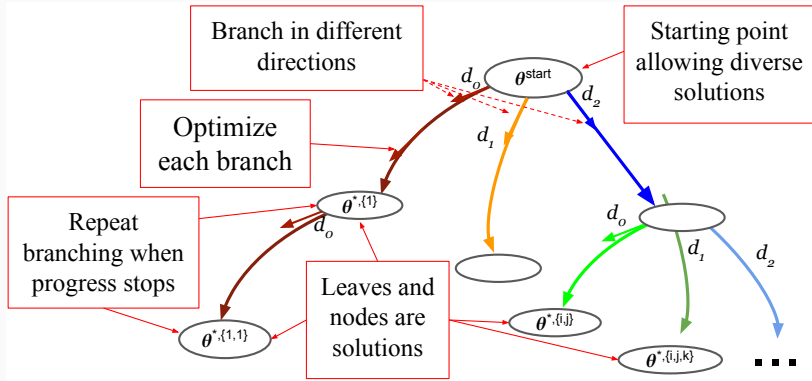


# Showing the Bifurcations



- Now, let's show a framework for finding multiple solutions:

# Branching Optimization Tree Search – animation



- Key parts are (1) Selecting the starting point, (2) creating different branches, (3) optimizing each branch, (4) choosing when to re-branch

## Notation for Finding Bifurcations

Remember our goal:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*), \theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B) \quad (1)$$

## Notation for Finding Bifurcations

Remember our goal:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*), \theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B) \quad (1)$$

For simplicity, concatenate all players' parameters and gradients:

$$\omega := [\theta_A, \theta_B], \quad \hat{\mathbf{g}}^j := [\mathbf{g}_A(\omega^j), \mathbf{g}_B(\omega^j)] \quad (2)$$

## Notation for Finding Bifurcations

Remember our goal:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*), \theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B) \quad (1)$$

For simplicity, concatenate all players' parameters and gradients:

$$\omega := [\theta_A, \theta_B], \quad \hat{\mathbf{g}}^j := [\mathbf{g}_A(\omega^j), \mathbf{g}_B(\omega^j)] \quad (2)$$

We find solutions with an optimizer or fixed-point operator:

$$\omega^{j+1} = \mathbf{F}(\omega^j) \quad (3)$$

## Notation for Finding Bifurcations

Remember our goal:

$$\theta_A^* \in \arg \min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*), \theta_B^* \in \arg \min_{\theta_B} \mathcal{L}_B(\theta_A^*, \theta_B) \quad (1)$$

For simplicity, concatenate all players' parameters and gradients:

$$\omega := [\theta_A, \theta_B], \quad \hat{\mathbf{g}}^j := [\mathbf{g}_A(\omega^j), \mathbf{g}_B(\omega^j)] \quad (2)$$

We find solutions with an optimizer or fixed-point operator:

$$\omega^{j+1} = \mathbf{F}(\omega^j) \quad (3)$$

For example, (simultaneous) gradient descent:

$$\mathbf{F}_{SGD}(\omega^j) = \omega^j - \alpha \hat{\mathbf{g}}^j \quad (4)$$



## Notation for Finding Bifurcations

- Fixed point operator for SGD:

$$\mathbf{F}_{SGD}(\omega^j) = \omega^j - \alpha \hat{\mathbf{g}}^j \quad (5)$$

- Jacobian of the fixed-point op. is useful for analysis of parameter trajectories from op. - ex., convergence rate, ...

$$\mathbf{J}_{SGD} := \nabla_{\omega} \mathbf{F}_{SGD}(\omega) = \mathbf{I} - \alpha \hat{\mathcal{H}} \quad (6)$$

## Notation for Finding Bifurcations

- Fixed point operator for SGD:

$$\mathbf{F}_{SGD}(\omega^j) = \omega^j - \alpha \hat{\mathbf{g}}^j \quad (5)$$

- Jacobian of the fixed-point op. is useful for analysis of parameter trajectories from op. - ex., convergence rate, ...

$$\mathbf{J}_{SGD} := \nabla_{\omega} \mathbf{F}_{SGD}(\omega) = \mathbf{I} - \alpha \hat{\mathcal{H}} \quad (6)$$

- We are interested in finding bifurcations, where trajectories rapidly separate.

## Notation for Finding Bifurcations

- Fixed point operator for SGD:

$$\mathbf{F}_{SGD}(\omega^j) = \omega^j - \alpha \hat{\mathbf{g}}^j \quad (5)$$

- Jacobian of the fixed-point op. is useful for analysis of parameter trajectories from op. - ex., convergence rate, ...

$$\mathbf{J}_{SGD} := \nabla_{\omega} \mathbf{F}_{SGD}(\omega) = \mathbf{I} - \alpha \hat{\mathcal{H}} \quad (6)$$

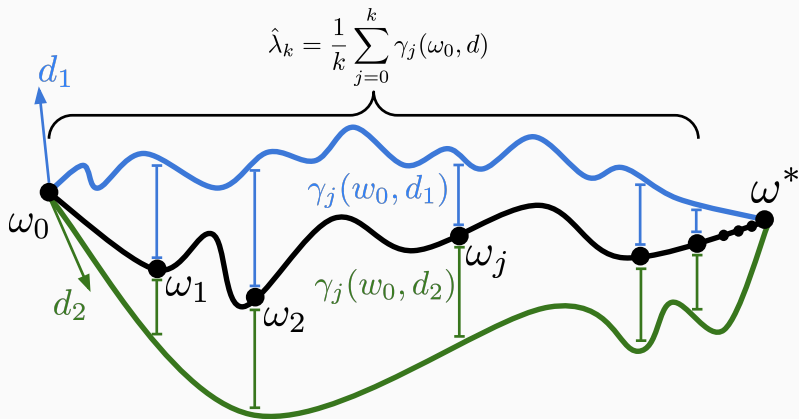
- We are interested in finding bifurcations, where trajectories rapidly separate.
- Idea: Measure the separation rate at some point  $\omega$  in direction  $\mathbf{d}$  via:

$$\mathbf{d}^{\top} (\mathbf{J}^{\top} \mathbf{J})|_{\omega} \mathbf{d} \quad (7)$$

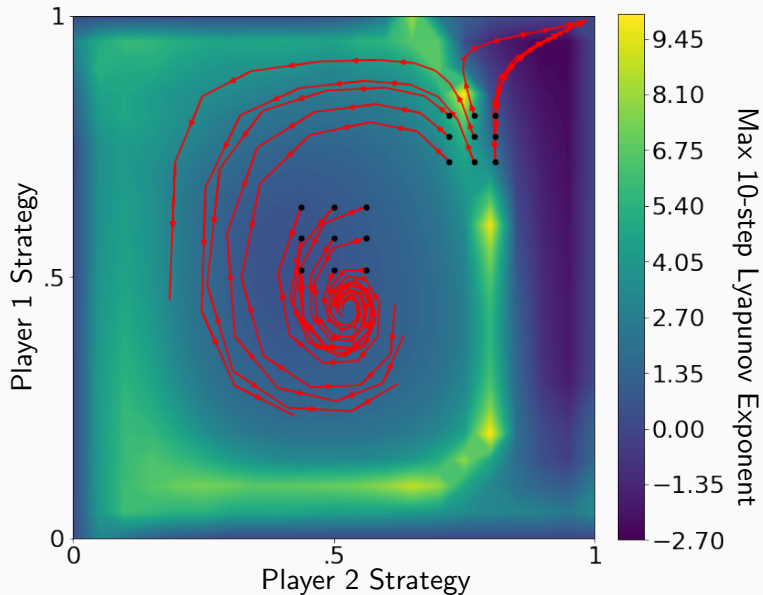
# What are Lyapunov Exponents?

- Intuition: just a (log) EVal integrated over a trajectory.

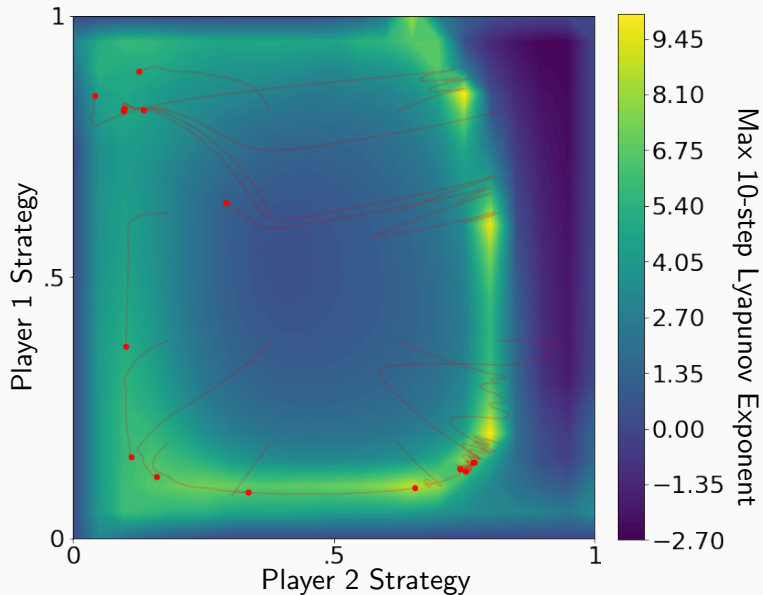
$$\gamma_j(\omega_0, \mathbf{d}) = \log(\mathbf{d}^\top (\mathbf{J}^j(\omega_0))^\top \mathbf{J}^j(\omega_0) \mathbf{d}) \quad (8)$$



# Showing Bifurcations with Lyapunov Exponents - **animation**



# Optimizing Lyapunov Exponents to Find Bifurcations



## Random Subspace Problem

- But, we only looked at 2 bifurcations so far. Can we construct toy problems with more types?

## Random Subspace Problem

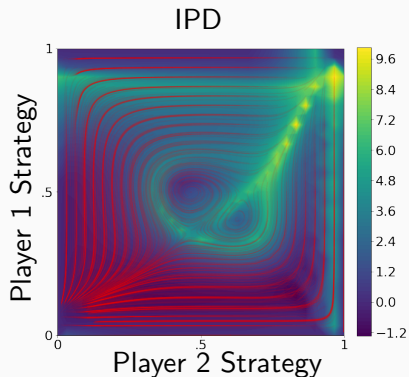
- But, we only looked at 2 bifurcations so far. Can we construct toy problems with more types?
- Idea: take high-dimensional game  $\mathcal{L}_A(\theta_A, \theta_B), \mathcal{L}_B(\theta_A, \theta_B)$  and optimize in a subspace. Ex., GAN, IPD, HO, meta-learning,...



## Random Subspace Problem

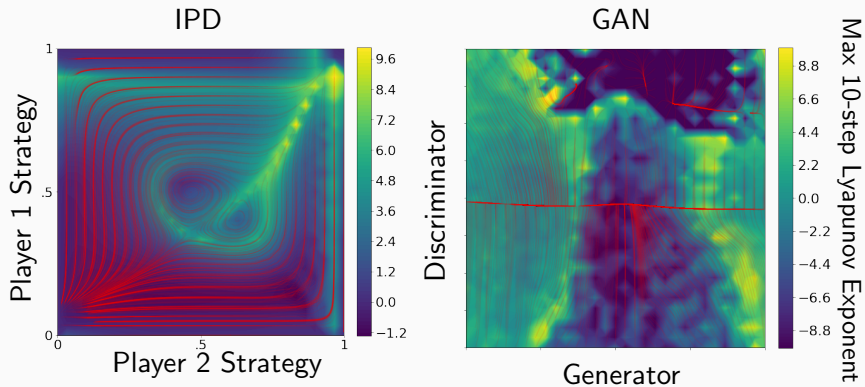
- But, we only looked at 2 bifurcations so far. Can we construct toy problems with more types?
- Idea: take high-dimensional game  $\mathcal{L}_A(\theta_A, \theta_B)$ ,  $\mathcal{L}_B(\theta_A, \theta_B)$  and optimize in a subspace. Ex., GAN, IPD, HO, meta-learning,...
- Specifically, use  $\mathcal{L}_A(\mathbf{v}_{AX} + \mathbf{b}_A, \mathbf{v}_{BY} + \mathbf{b}_B)$ ,  $\mathcal{L}_B(\mathbf{v}_{AX} + \mathbf{b}_A, \mathbf{v}_{BY} + \mathbf{b}_B)$ , where  $\mathbf{v}$  sampled (ex., uniform) randomly, and offset  $\mathbf{b}$  at appropriate value (ex., init., optimal).

# Random Subspace Problem



- Exponent peaks near where trajectories separate, showing we find bifurcations.

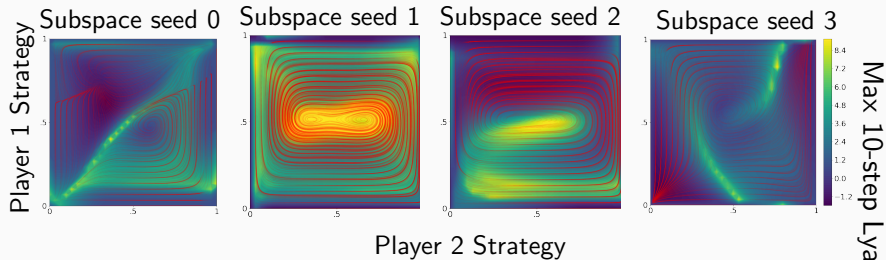
# Random Subspace Problem



- Exponent peaks near where trajectories separate, showing we find bifurcations.

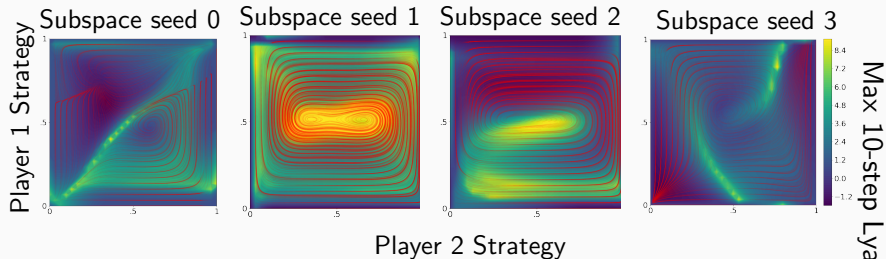
# Random Subspace Problem

## Random subspace IPD

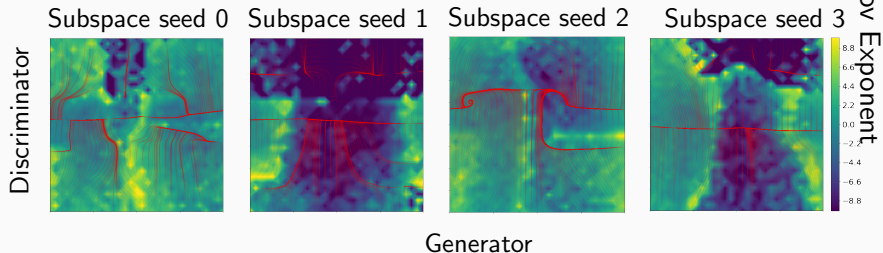


# Random Subspace Problem

## Random subspace IPD



## Random subspace GAN



## Finding Diverse Solutions in the Iterated Prisoners Dilemma

- Now, try the full branching opt. method – Generalized Ridge Rider (GRR) – on the Iterated Prisoners Dilemma (IPD).
- To summarize the parts to our method:

## Finding Diverse Solutions in the Iterated Prisoners Dilemma

- Now, try the full branching opt. method – Generalized Ridge Rider (GRR) – on the Iterated Prisoners Dilemma (IPD).
- To summarize the parts to our method:
  1. Find a suitable *starting point* for our branching process - by maxing a Lyapunov exponent.

## Finding Diverse Solutions in the Iterated Prisoners Dilemma

- Now, try the full branching opt. method – Generalized Ridge Rider (GRR) – on the Iterated Prisoners Dilemma (IPD).
- To summarize the parts to our method:
  1. Find a suitable *starting point* for our branching process - by maxing a Lyapunov exponent.
  2. Select *branching directions* (or perturbations) from a given branching point - by using directions from exponent.



## Finding Diverse Solutions in the Iterated Prisoners Dilemma

- Now, try the full branching opt. method – Generalized Ridge Rider (GRR) – on the Iterated Prisoners Dilemma (IPD).
- To summarize the parts to our method:
  1. Find a suitable *starting point* for our branching process - by maxing a Lyapunov exponent.
  2. Select *branching directions* (or perturbations) from a given branching point - by using directions from exponent.
  3. *Continue the optimization process* along a branch after the initial perturbation.

## Finding Diverse Solutions in the Iterated Prisoners Dilemma

- Now, try the full branching opt. method – Generalized Ridge Rider (GRR) – on the Iterated Prisoners Dilemma (IPD).
- To summarize the parts to our method:
  1. Find a suitable *starting point* for our branching process - by maxing a Lyapunov exponent.
  2. Select *branching directions* (or perturbations) from a given branching point - by using directions from exponent.
  3. *Continue the optimization process* along a branch after the initial perturbation.
- Let's look at these in the following table.

## Finding Diverse Solutions in the Iterated Prisoners Dilemma

Search Strategy	Solution Mode	
	Cooperate	Defect
×20 Random init + LOLA [1]	✓	✗
×20 Random init + GD	✗	✓

- Randomly init. then applying a training method only finds 1 solution mode. Baselines don't find both.

# Finding Diverse Solutions in the Iterated Prisoners Dilemma

Search Strategy	Solution Mode	
	Cooperate	Defect
×20 Random init + LOLA [1]	✓	✗
×20 Random init + GD	✗	✓
<b>GRR:</b> tune max Lyap + top EVec branch + GD	✓	✓
<b>GRR:</b> tune max Lyap + top EVec branch + LOLA	✓	✓

- Randomly init. then applying a training method only finds 1 solution mode. Baselines don't find both.
- Our method finds both solution modes (with *any* opt.).

# Finding Diverse Solutions in the Iterated Prisoners Dilemma

Search Strategy	Solution Mode	
	Cooperate	Defect
×20 Random init + LOLA [1]	✓	✗
×20 Random init + GD	✗	✓
<b>GRR:</b> tune max Lyap + top EVec branch + GD	✓	✓
<b>GRR:</b> tune max Lyap + top EVec branch + LOLA	✓	✓
×20 Random init + top EVec branch + GD	✗	✓

- Randomly init. then applying a training method only finds 1 solution mode. Baselines don't find both.
- Our method finds both solution modes (with *any* opt.).
- If we don't tune the Lyapunov exponent, then branching doesn't affect the soln. Evidence we are near a bifurcation.

## Scaling up to GANs

Init scale, step size	Max Lyap Coeff	Ensemble log-prob
0.001, 1.0	$0.952 \pm 0.834$	$-16\,342 \pm 817$
0.1, 1.0	$6.485 \pm 1.155$	$-13\,691 \pm 1317$
10.0, 1.0	$0.053 \pm 0.128$	$-46\,659 \pm 26\,793$
0.001, 0.1	$0.849 \pm 0.765$	$-12\,321 \pm 126$
0.1, 0.1	$6.571 \pm 0.953$	$-10\,846 \pm 256$
10.0, 0.1	$-0.012 \pm 0.014$	$-23\,459 \pm 12\,693$

- We used our method to train ensembles of GANs by branching optimization.
- Our method may find ensembles of GANs that cover more modes, mitigating *mode dropping* problem.
- Takeaway: Exponent calculation is scalable to larger problems.

# Thanks!

Jonathan Lorraine



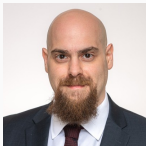
Paul Vicol



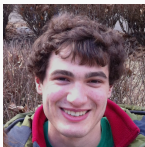
Jack Parker-Holder



Tal Kachman



Luke Metz



Jakob Foerster



# References

---



- [1] 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.
- [2] Jack Parker-Holder, Luke Metz, Cinjon Resnick, Hengyuan Hu, Adam Lerer, Alistair Letcher, Alexander Peysakhovich, Aldo Pacchiano, and Jakob Foerster. Ridge rider: Finding diverse solutions by following eigenvectors of the hessian. In *Advances in Neural Information Processing Systems*, volume 33, pages 753–765, 2020.
- [3] 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.

- [4] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.
- [5] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.
- [6] Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.
- [7] 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.

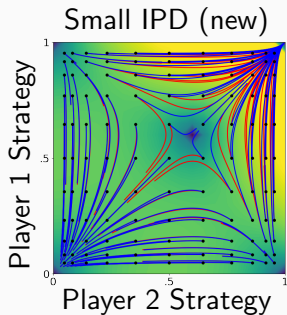
- [8] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1540–1552, 2020.

## Extra Slides

---

# Illustrative Toy Problems

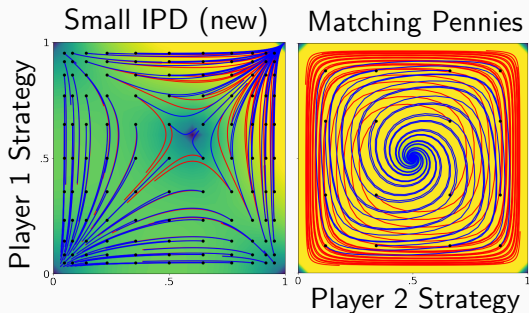
Following the gradient in red, LOLA [1] in blue



- Small IPD is a 2 param. Iterated Prisoner's Dilemma with TT and DD solutions, but only real EVals.

# Illustrative Toy Problems

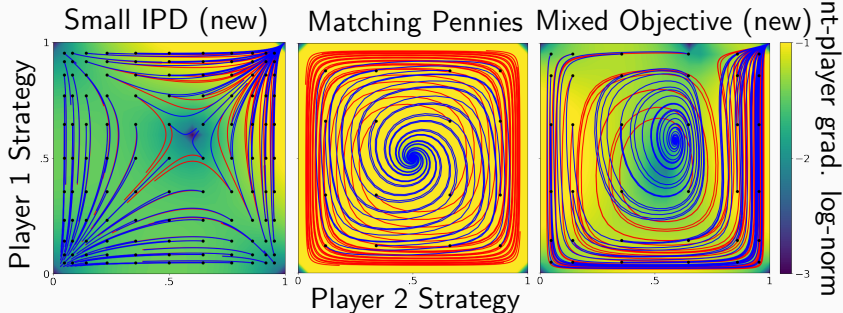
Following the gradient in red, LOLA [1] in blue



- Small IPD is a 2 param. Iterated Prisoner's Dilemma with TT and DD solutions, but only real EVals.
- Matching Pennies is a 2 param. rock-paper-scissors with imaginary EVals, but only 1 solution.

# Illustrative Toy Problems

Following the gradient in red, LOLA [1] in blue



- Small IPD is a 2 param. Iterated Prisoner's Dilemma with TT and DD solutions, but only real EVals.
- Matching Pennies is a 2 param. rock-paper-scissors with imaginary EVals, but only 1 solution.
- Mixing these gives a 2 param. problem like the full IPD with multiple solutions, complex EVals, and a Hopf bifurcation.

## Notation for Finding Bifurcations

- Interested in finding bifurcations, where trajectories rapidly separate.
- Idea: Measure the spread at some point  $\omega$  in direction  $\mathbf{d}$  via:

$$\mathbf{d}^\top (\mathbf{J}^\top \mathbf{J})|_{\omega} \mathbf{d} \quad (9)$$

- Interested in the spread at iterate  $\omega^j$  from the fixed-point op. from  $\omega_0$ . So, define Jacobian there  $\mathbf{J}^j(\omega_0)$



## Notation for Finding Bifurcations

- Interested in finding bifurcations, where trajectories rapidly separate.
- Idea: Measure the spread at some point  $\omega$  in direction  $\mathbf{d}$  via:

$$\mathbf{d}^\top (\mathbf{J}^\top \mathbf{J})|_\omega \mathbf{d} \quad (9)$$

- Interested in the spread at iterate  $\omega^j$  from the fixed-point op. from  $\omega_0$ . So, define Jacobian there  $\mathbf{J}^j(\omega_0)$
- If we take log, then + when diverge and - when converge:

$$\gamma_j(\omega_0, \mathbf{d}) = \log(\mathbf{d}^\top (\mathbf{J}^j(\omega_0))^\top \mathbf{J}^j(\omega_0) \mathbf{d}) \quad (10)$$

## Notation for Finding Bifurcations

- Interested in finding bifurcations, where trajectories rapidly separate.
- Idea: Measure the spread at some point  $\omega$  in direction  $\mathbf{d}$  via:

$$\mathbf{d}^\top (\mathbf{J}^\top \mathbf{J})|_{\omega} \mathbf{d} \quad (9)$$

- Interested in the spread at iterate  $\omega^j$  from the fixed-point op. from  $\omega_0$ . So, define Jacobian there  $\mathbf{J}^j(\omega_0)$
- If we take log, then + when diverge and - when converge:

$$\gamma_j(\omega_0, \mathbf{d}) = \log(\mathbf{d}^\top (\mathbf{J}^j(\omega_0))^\top \mathbf{J}^j(\omega_0) \mathbf{d}) \quad (10)$$

- Aggregate these values over opt. trajectory via average:

$$\hat{\lambda}_k(\omega_0, \mathbf{d}) = \frac{1}{k} \sum_{j=0}^k \gamma_j(\omega_0, \mathbf{d}) \quad (11)$$

## What are Lyapunov Exponents?

$$\gamma_j(\boldsymbol{\omega}_0, \mathbf{d}) = \log(\mathbf{d}^\top (\mathbf{J}^j(\boldsymbol{\omega}_0))^\top \mathbf{J}^j(\boldsymbol{\omega}_0) \mathbf{d}) \quad (12)$$

- Aggregate these values over opt. trajectory via average:

$$\hat{\lambda}_k(\boldsymbol{\omega}_0, \mathbf{d}) = \frac{1}{k} \sum_{j=0}^k \gamma_j(\boldsymbol{\omega}_0, \mathbf{d}) \quad (13)$$

## What are Lyapunov Exponents?

$$\gamma_j(\boldsymbol{\omega}_0, \mathbf{d}) = \log(\mathbf{d}^\top (\mathbf{J}^j(\boldsymbol{\omega}_0))^\top \mathbf{J}^j(\boldsymbol{\omega}_0) \mathbf{d}) \quad (12)$$

- Aggregate these values over opt. trajectory via average:

$$\hat{\lambda}_k(\boldsymbol{\omega}_0, \mathbf{d}) = \frac{1}{k} \sum_{j=0}^k \gamma_j(\boldsymbol{\omega}_0, \mathbf{d}) \quad (13)$$

- Take limit at opt. horizon  $k \rightarrow \infty$ . We get the (global) Lyapunov exponent in direction  $\mathbf{d}$  at point  $\boldsymbol{\omega}_0$ .
- Often interested in the max Lyapunov exponent:

$$\hat{\lambda}_k^{max}(\boldsymbol{\omega}_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \hat{\lambda}_k(\boldsymbol{\omega}_0, \mathbf{d}) \quad (14)$$

## What are Lyapunov Exponents?

- Often interested in the max Lyapunov exponent:

$$\hat{\lambda}_k^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \hat{\lambda}_k(\omega_0, \mathbf{d}) \quad (15)$$

- Ex., the (max, global) Lyapunov exponent is negative inside a basin of attraction to a fixed point (because traj. converge).

$$\hat{\lambda}^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \lim_{k \rightarrow \infty} \hat{\lambda}_k(\omega_0, \mathbf{d}) < 0 \quad (16)$$

## What are Lyapunov Exponents?

- Often interested in the max Lyapunov exponent:

$$\hat{\lambda}_k^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \hat{\lambda}_k(\omega_0, \mathbf{d}) \quad (15)$$

- Ex., the (max, global) Lyapunov exponent is negative inside a basin of attraction to a fixed point (because traj. converge).

$$\hat{\lambda}^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \lim_{k \rightarrow \infty} \hat{\lambda}_k(\omega_0, \mathbf{d}) < 0 \quad (16)$$

- Consequently, between basins of attraction exponent is max (at 0) and the max direction points along separatrix.

## What are Lyapunov Exponents?

- Often interested in the max Lyapunov exponent:

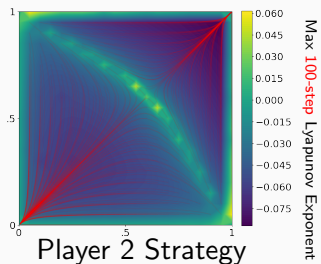
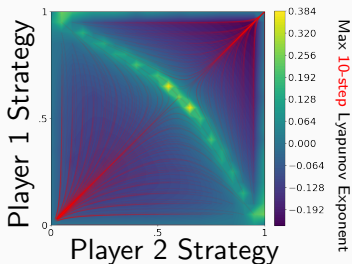
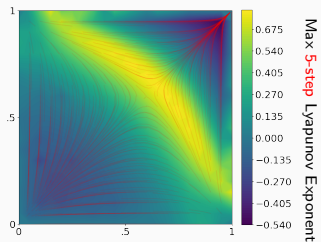
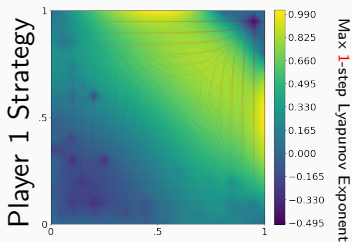
$$\hat{\lambda}_k^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \hat{\lambda}_k(\omega_0, \mathbf{d}) \quad (15)$$

- Ex., the (max, global) Lyapunov exponent is negative inside a basin of attraction to a fixed point (because traj. converge).

$$\hat{\lambda}^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \lim_{k \rightarrow \infty} \hat{\lambda}_k(\omega_0, \mathbf{d}) < 0 \quad (16)$$

- Consequently, between basins of attraction exponent is max (at 0) and the max direction points along separatrix.
- Can find bifurcations between basins by maxing exponent!

# Impact of Optimization Horizon on Exponent - animation





## Truncated Lyapunov Exponents

- We look at the truncated Lyapunov exponent – i.e., a finite optimization horizon  $k$  – which has desirable properties:

# Truncated Lyapunov Exponents

- We look at the truncated Lyapunov exponent – i.e., a finite optimization horizon  $k$  – which has desirable properties:

1. Computationally tractability

# Truncated Lyapunov Exponents

- We look at the truncated Lyapunov exponent – i.e., a finite optimization horizon  $k$  – which has desirable properties:
  1. Computationally tractability
  2. Non-zero gradient signals for finding bifurcations

# Truncated Lyapunov Exponents

- We look at the truncated Lyapunov exponent – i.e., a finite optimization horizon  $k$  – which has desirable properties:
  1. Computationally tractability
  2. Non-zero gradient signals for finding bifurcations
  3. A better separation rate description for the finite trajectories used in practice

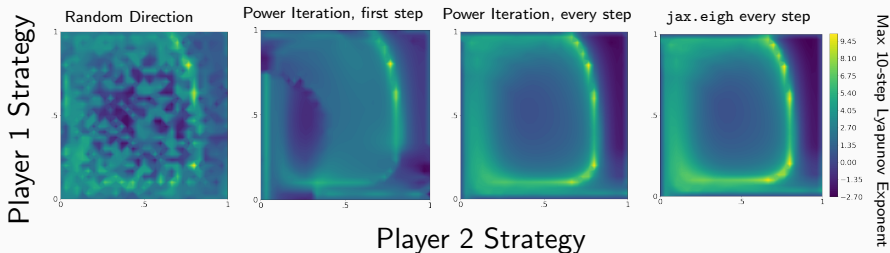
# Truncated Lyapunov Exponents

- We look at the truncated Lyapunov exponent – i.e., a finite optimization horizon  $k$  – which has desirable properties:
  1. Computationally tractability
  2. Non-zero gradient signals for finding bifurcations
  3. A better separation rate description for the finite trajectories used in practice
- But, we lose many of the theoretical results.

# Truncated Lyapunov Exponents

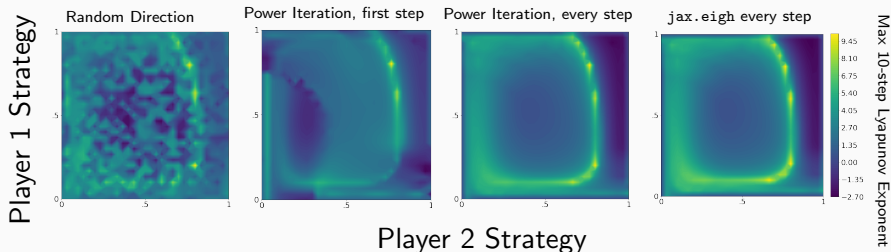
- We look at the truncated Lyapunov exponent – i.e., a finite optimization horizon  $k$  – which has desirable properties:
  1. Computationally tractability
  2. Non-zero gradient signals for finding bifurcations
  3. A better separation rate description for the finite trajectories used in practice
- But, we lose many of the theoretical results.
- The next slides help to build an intuition for the exponent of visualizable toy problems.

# Impact of Direction Choice on Showing Bifurcations



- Re-estimating the top EVecs at each iteration performs best, but is most expensive, and diverges from theory.

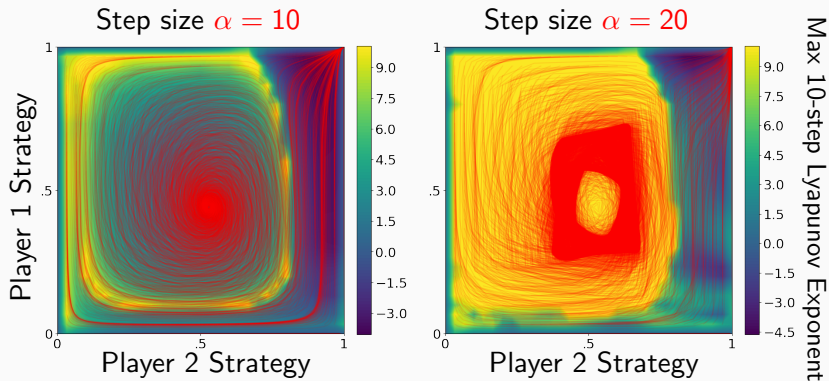
# Impact of Direction Choice on Showing Bifurcations



- Re-estimating the top EVecs at each iteration performs best, but is most expensive, and diverges from theory.
- In the optimization limit  $k \rightarrow \infty$  the choice of direction almost certainly doesn't matter.

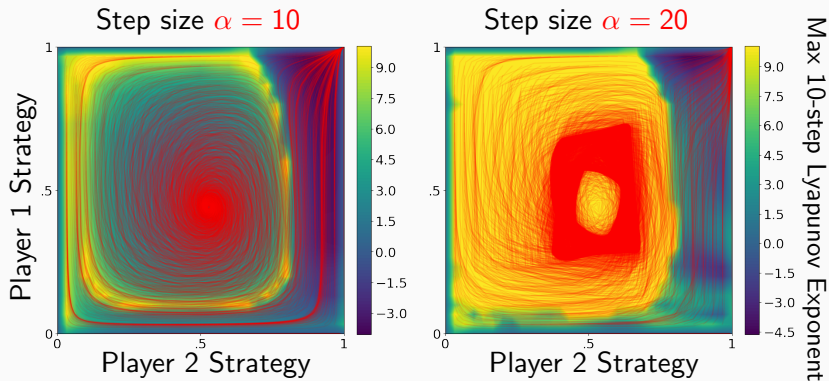


# What if the Optimizer doesn't Converge?



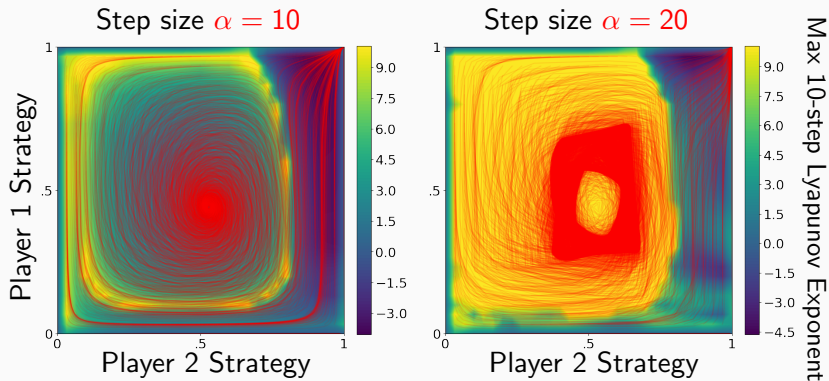
- Make sure step size small, so optimizer converges.

# What if the Optimizer doesn't Converge?



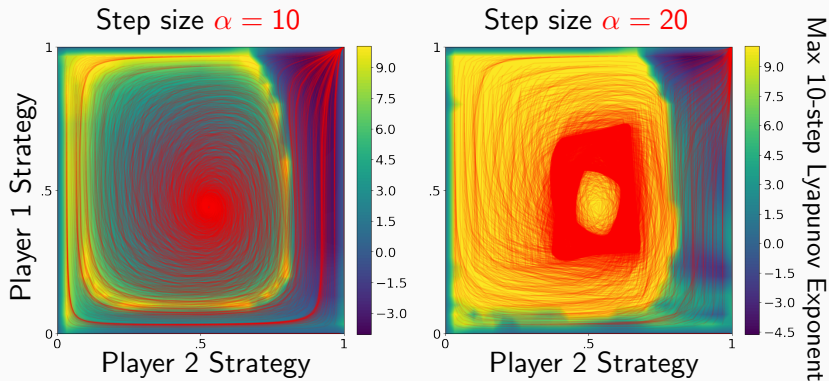
- Make sure step size small, so optimizer converges.
- If not, exponent maybe max where trajectories don't find soln.

# What if the Optimizer doesn't Converge?



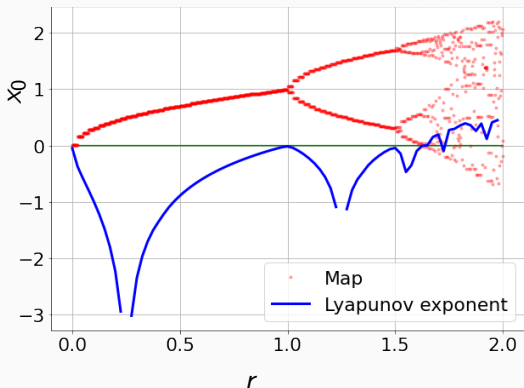
- Make sure step size small, so optimizer converges.
- If not, exponent maybe max where trajectories don't find soln.
- Can get fractal bifurcation for critical step size: [video](#)

# What if the Optimizer doesn't Converge?



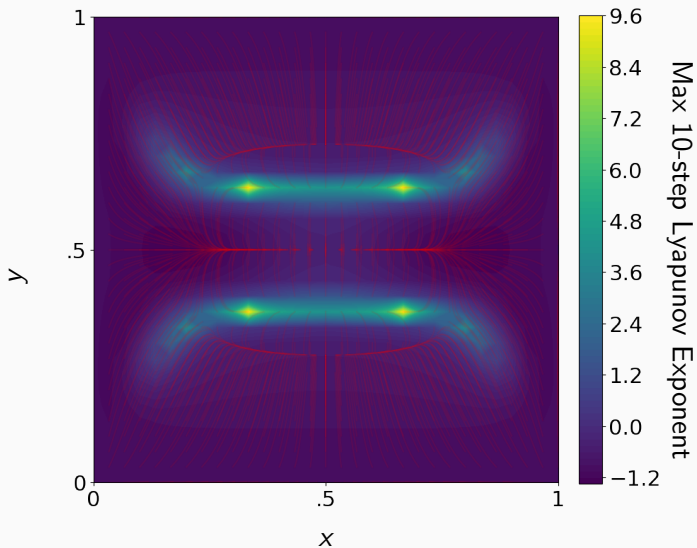
- Make sure step size small, so optimizer converges.
- If not, exponent maybe max where trajectories don't find soln.
- Can get fractal bifurcation for critical step size: [video](#)
- We could use more complicated toy problems...

# The Logistic Map Example

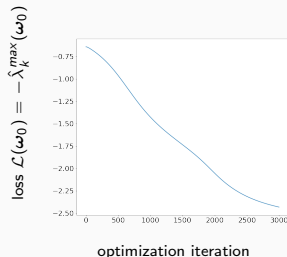


- A canonical 1-dimensional example for bifurcations:  
$$x(t+1) = rx(t)(1-x(t))$$

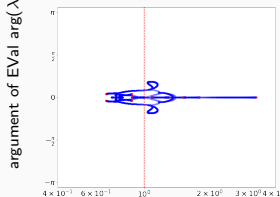
# Single-objective Optimization Example



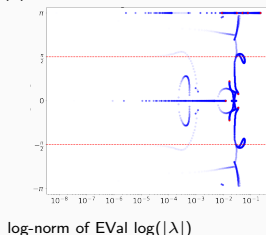
# Analyzing the IPD Optimization



EVals of Jac. of fixed point operator  $\text{Sp}(\mathbf{J})$



EVals of game Hessian  $\text{Sp}(\mathcal{H})$



- **Takeaway:** We effectively reduce our loss and correspondingly raise the max EVal of  $\mathbf{J}$ .
- What if we need separation in more than 1 direction?

## Connections to Kolmogorov-Sinai Entropy

- Max exponent  $\hat{\lambda}_k^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \hat{\lambda}_k(\omega_0, \mathbf{d})$  only guarantees separation in 1 direction.



## Connections to Kolmogorov-Sinai Entropy

- Max exponent  $\hat{\lambda}_k^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \hat{\lambda}_k(\omega_0, \mathbf{d})$  only guarantees separation in 1 direction.
- What if we want spread in multiple directions?

$$\mathcal{L}_n^{\text{sum}}(\omega_0) = - \max_{\mathbf{d}_1, \dots, \mathbf{d}_n} \sum_{l=1}^n \hat{\lambda}_k(\omega_0, \mathbf{d}_l),$$

such that  $\|\mathbf{d}_l\| = 1, \mathbf{d}_l^\top \mathbf{d}_m = 0$  for all  $l, m \in 1, \dots, n, l \neq m$

## Connections to Kolmogorov-Sinai Entropy

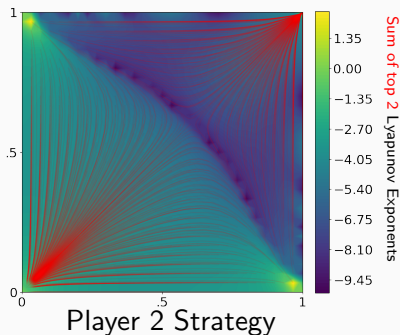
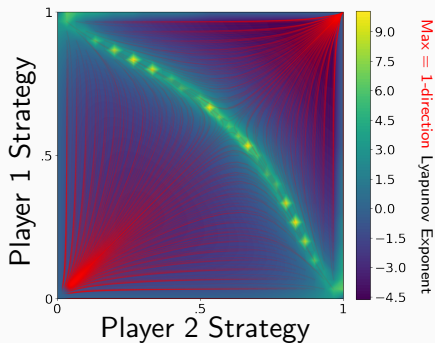
- Max exponent  $\hat{\lambda}_k^{max}(\omega_0) = \max_{\mathbf{d}, \|\mathbf{d}\|=1} \hat{\lambda}_k(\omega_0, \mathbf{d})$  only guarantees separation in 1 direction.
- What if we want spread in multiple directions?

$$\mathcal{L}_n^{\text{sum}}(\omega_0) = - \max_{\mathbf{d}_1, \dots, \mathbf{d}_n} \sum_{l=1}^n \hat{\lambda}_k(\omega_0, \mathbf{d}_l),$$

such that  $\|\mathbf{d}_l\| = 1, \mathbf{d}_l^\top \mathbf{d}_m = 0$  for all  $l, m \in 1, \dots, n, l \neq m$

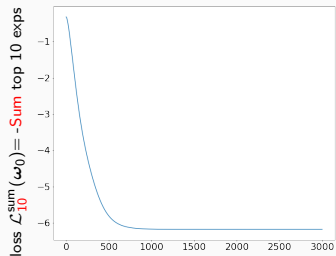
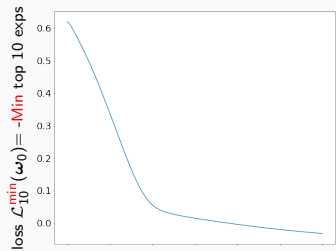
- Cool connection: Kolmogorov-Sinai entropy is  $\approx \#$  symbols for optimal coding of the particle trajectory. This is  $\leq$  sum of positive exponents.

# Connections to Kolmogorov-Sinai Entropy



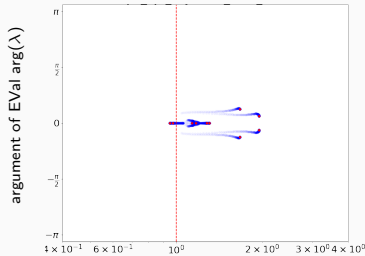
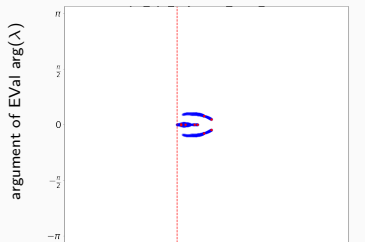
- Local maxima – not saddles – allow trajectory separation in all directions here.

# Trajectory Separation in Multiple Directions



optimization iteration

EVals of Jac. of fixed point operator  $\text{Sp}(J)$



log-norm of Eval  $\log(|\lambda|)$

## Estimating EVecs in Single Objective

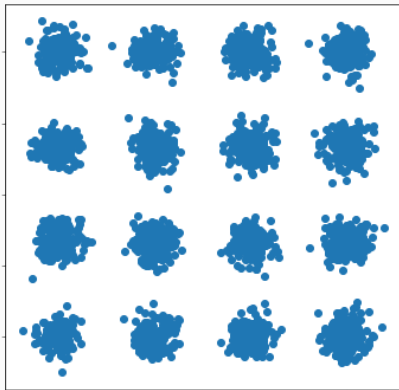
# HVP Evaluations	MNIST Accuracy	
	Our method	Method from RR
10 000	19%(+8%)	11%
100 000	89%(+6%)	83%
1 000 000	93%(+2%)	91%

- How many HVP evaluations to reach different MNIST classifier accuracies by following EVecs, repeating the exp. in RR's Fig. 4.
- Not designed to train a single strong classifier! But, to test our ability to efficiently follow negative EVecs.
- Takeaway: Estimate largest EVecs of Jacobian of fixed-point op. is an efficient way to estimate most negative EVecs of Hessian, and generalizes to other setups.

## Branching Optimization Tree Search— RR/GRR changes in red

- 1: Select optimization parameters  $\alpha$
- 2: Find starting parameters  $\omega^{start} = \text{FindStartingPoint}(\alpha)$
- 3: Initialize a branch  $\psi^{init} = \text{InitBranch}(\omega^{start}, \alpha)$
- 4: Initialize the set of branches  $\mathcal{B} = \text{SplitBranch}(\psi^{init})$
- 5: Initialize the set of solutions  $\mathcal{S} = \emptyset$
- 6: **while** Branches  $\mathcal{B}$  non-empty **do**
- 7:    $\psi, \mathcal{B} = \text{ChooseBranch}(\mathcal{B})$
- 8:    $\omega^* = \text{Optimize}(\psi.\omega, \psi.\alpha) \#$  Optimize our parameters
- 9:   **if**  $\text{VerifySolution}(\omega^*)$  **then**
- 10:      $\mathcal{S} = \mathcal{S} \cup \{\omega^*\}$
- 11:   Make new branch to split  $\psi' = \text{copy}(\psi)$
- 12:   Store the optimized parameters  $\psi'.parameters = \omega^*$
- 13:   **if**  $\text{ContinueBranching}(\psi')$  **then**
- 14:      $\mathcal{B} = \mathcal{B} \cup \text{SplitBranch}(\psi')$
- 15: **return**  $\mathcal{S}$

# GAN Samples



- Ground truth samples for our GAN Mixture of Gaussian experiment.
- Designed to test mode dropping.