Lyapunov Exponents for Diversity in Differentiable Games

Conference on Autonomous and Multi-agent Systems (AAMAS) 2022

arXiv link

Jonathan Lorraine[1,2], Paul Vicol[1,2], Jack Parker-Holder[3], Tal Kachman[4], Luke Metz[5], Jakob Foerster[3]

February 7^{th}, 2022

University of Toronto[1], Vector Institute[2], University of Oxford[3], Radboud University[4], Google Brain[5]

Overview

- 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].

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].
- We generalize Ridge Rider [2] to differentiable games, providing a method which finds bifurcations and branches the optimization process across them: animation

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].
- 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

Motivation

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

Motivation

- When minimizing objectives, randomly initializing and optimizing can fail to find different solutions.
- Why do we want to find different solutions? For example...

Motivation

- When minimizing objectives, randomly initializing and optimizing can fail to find different solutions.
- Why do we want to find different solutions? For example...
- In image classification, some generalize better than others - ex., shape vs. texture solutions.

Motivation

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

Motivation

- When minimizing objectives, randomly initializing and optimizing can fail to find different 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.
- But, what is a differentiable game?

Background: Differentiable Games

- Differentiable games generalize single-objective minimization:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*),$$
$$\boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B)$$

Background: Differentiable Games

- Differentiable games generalize single-objective minimization:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*),$$
$$\boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B)$$

- 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:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*),$$
$$\boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B)$$

- 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. A 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:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*),$$
$$\boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B)$$

- 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. A 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.

Hyperparameter Optimization as a Differentiable Game

- Hyperparameter optimization and many meta-learning problems can be formulated as a differentiable game.

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})),$$
$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w})$$

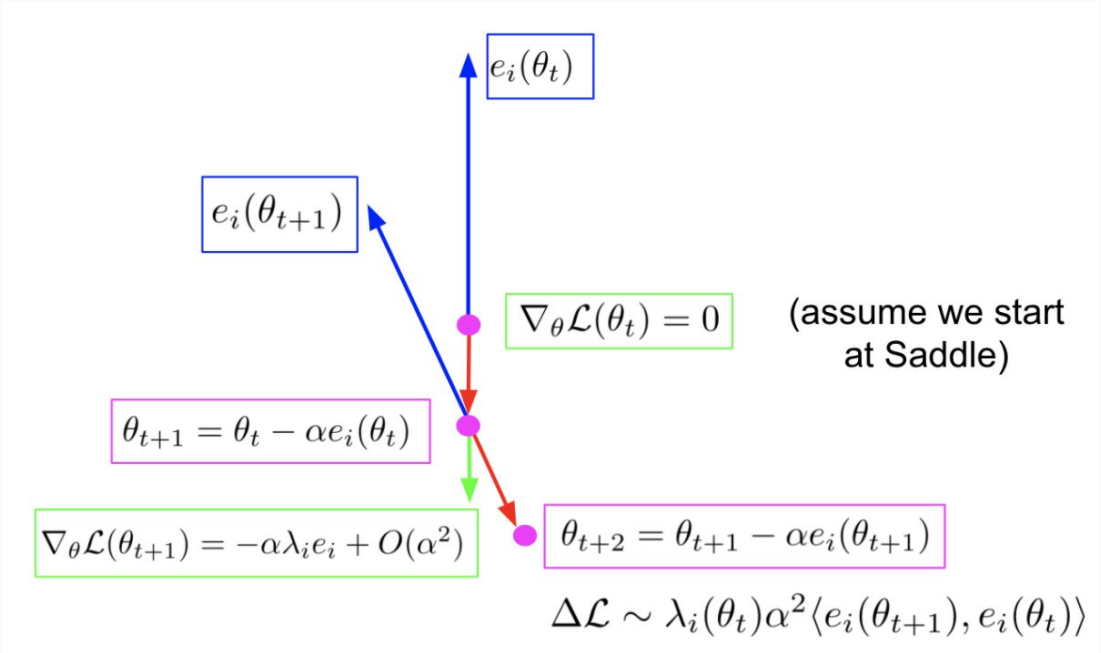
Hyperparameter Optimization as a Differentiable Game

- Hyperparameter optimization and many meta-learning problems can be formulated as a differentiable game.

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})),$$
$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w})$$

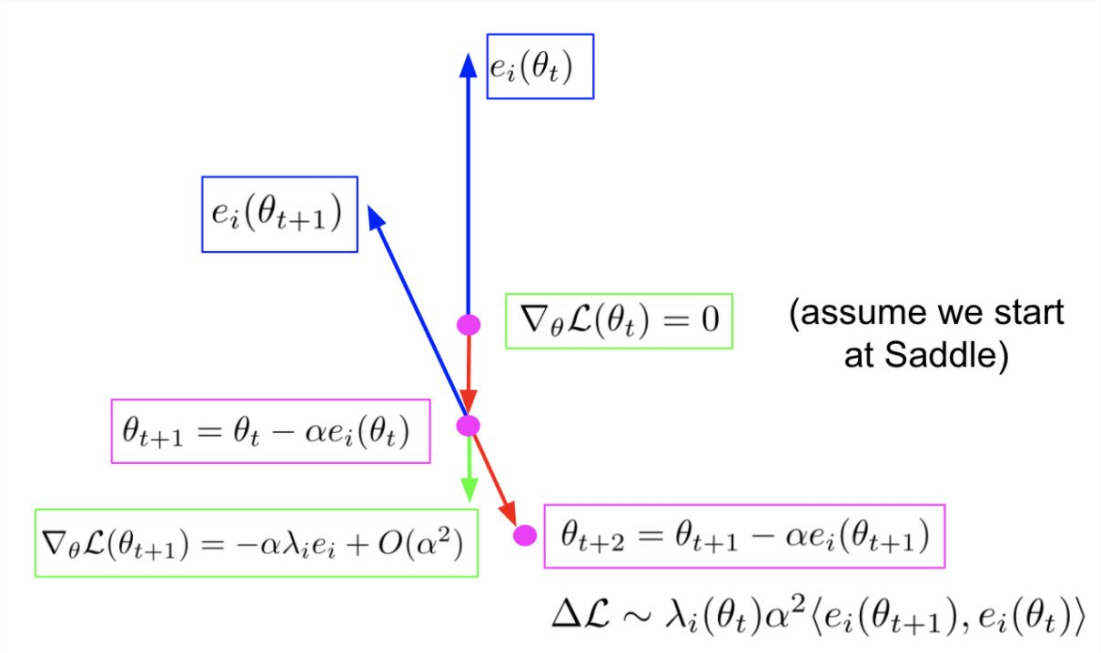
- I have various papers on this - some slides with connections at the end.

Background: Ridge Rider



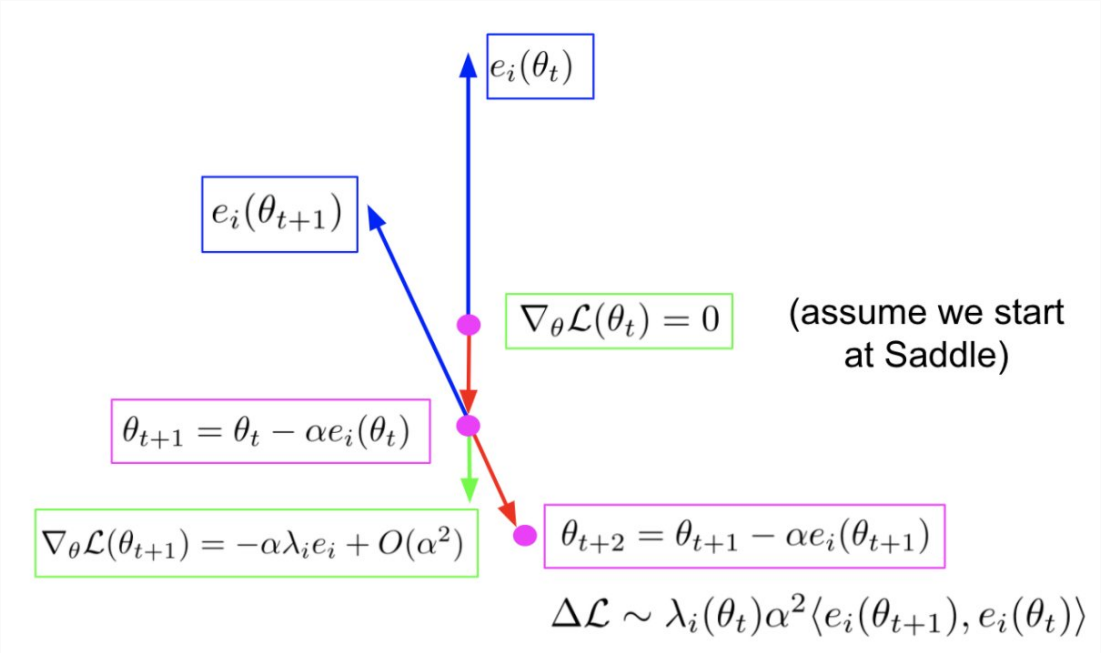
- Ridge rider (RR) [2] finds diverse solutions in single-objective optimization by branching optimization at saddle points.

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

RR for Differentiable Games?

- But, what if we wanted to use Ridge Rider to find multiple 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!
- Generalizations of the Hessian's for games – i.e., the *Game Hessian* – may have complex EVals from lack of symmetry.

$$\overbrace{\hat{\mathcal{H}}}^{\text{Game Hessian}} = \begin{bmatrix} \text{Player } A \text{ Hessian} \nabla^2_{\theta_A} \mathcal{L}_A & \nabla_{\theta_A} \nabla_{\theta_B} \mathcal{L}_A \\ \nabla_{\theta_B} \nabla_{\theta_A} \mathcal{L}_B{}^\top & \text{Player } B \text{ Hessian} \nabla^2_{\theta_B} \mathcal{L}_B \end{bmatrix}$$

Another viewpoint

- Following the gradient for single-objective optimization forms a conservative vector field. Vector field is just gradient of loss.

Another viewpoint

- Following the gradient for single-objective optimization forms a conservative vector field. Vector field is just gradient of loss.
- Bifurcations are where small initial parameter changes cause final solution differences.

Another viewpoint

- Following the gradient for single-objective optimization forms a conservative vector field. Vector field is just gradient of loss.
- Bifurcations are where small initial parameter changes cause final solution differences.
- Saddles are a key bifurcation in conservative systems from following gradients.

Another viewpoint

- Following the gradient for single-objective optimization forms a conservative vector field. Vector field is just gradient of loss.
- Bifurcations are where small initial parameter changes cause final solution differences.
- Saddles are a key bifurcation in conservative systems from following gradients.
- Following the simultaneous gradient for differentiable games can form a non-conservative vector field. No single loss for vector-field to be gradient of!

Another viewpoint

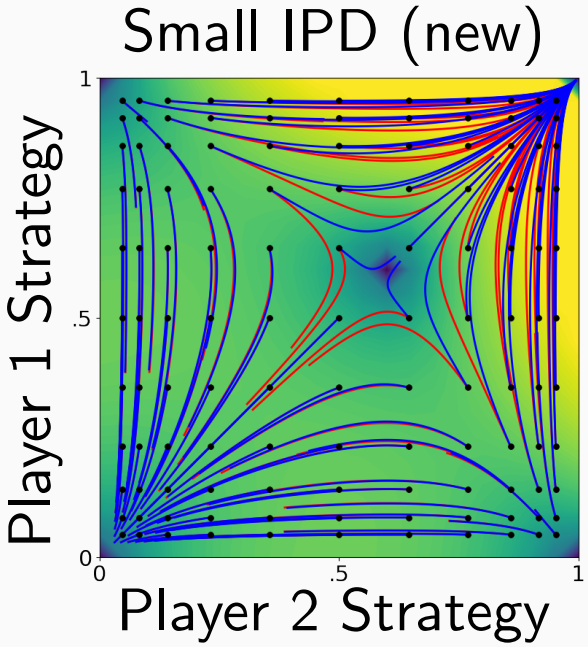
- Following the gradient for single-objective optimization forms a conservative vector field. Vector field is just gradient of loss.
- Bifurcations are where small initial parameter changes cause final solution differences.
- Saddles are a key bifurcation in conservative systems from following gradients.
- Following the simultaneous gradient for differentiable games can form a non-conservative vector field. No single loss for vector-field to be gradient of!
- In non-conservative fields, many more bifurcation types.

Another viewpoint

- Following the gradient for single-objective optimization forms a conservative vector field. Vector field is just gradient of loss.
- Bifurcations are where small initial parameter changes cause final solution differences.
- Saddles are a key bifurcation in conservative systems from following gradients.
- Following the simultaneous gradient for differentiable games can form a non-conservative vector field. No single loss for vector-field to be gradient of!
- In non-conservative fields, many more bifurcation types.
- Now, let's look at toys to illustrate the difference:

Illustrative Toy Problems

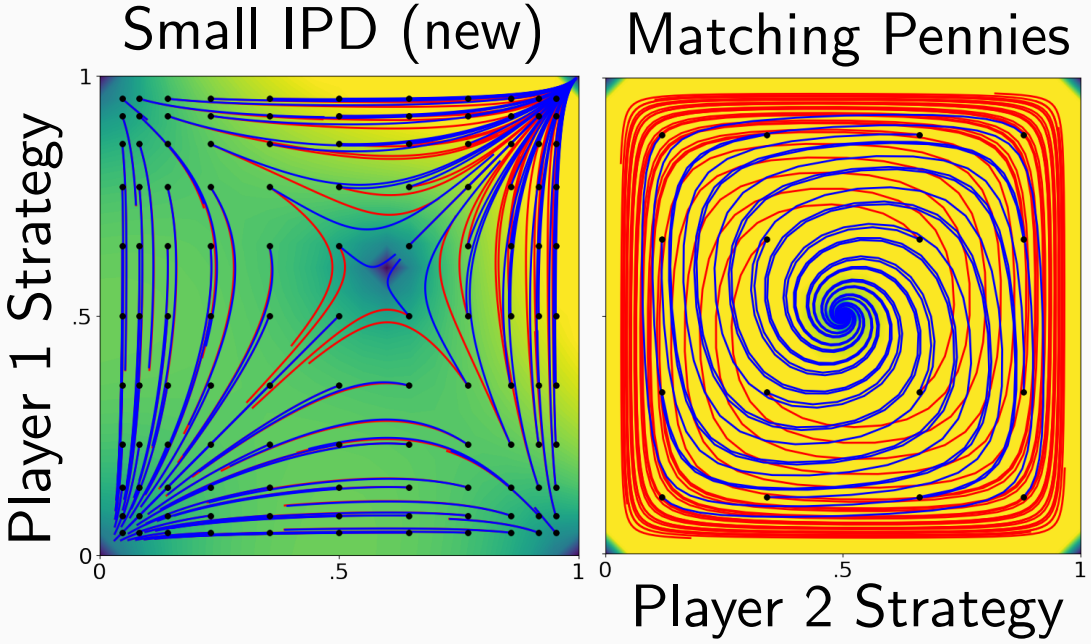
Following the gradient in red, LOLA [1] in blue



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

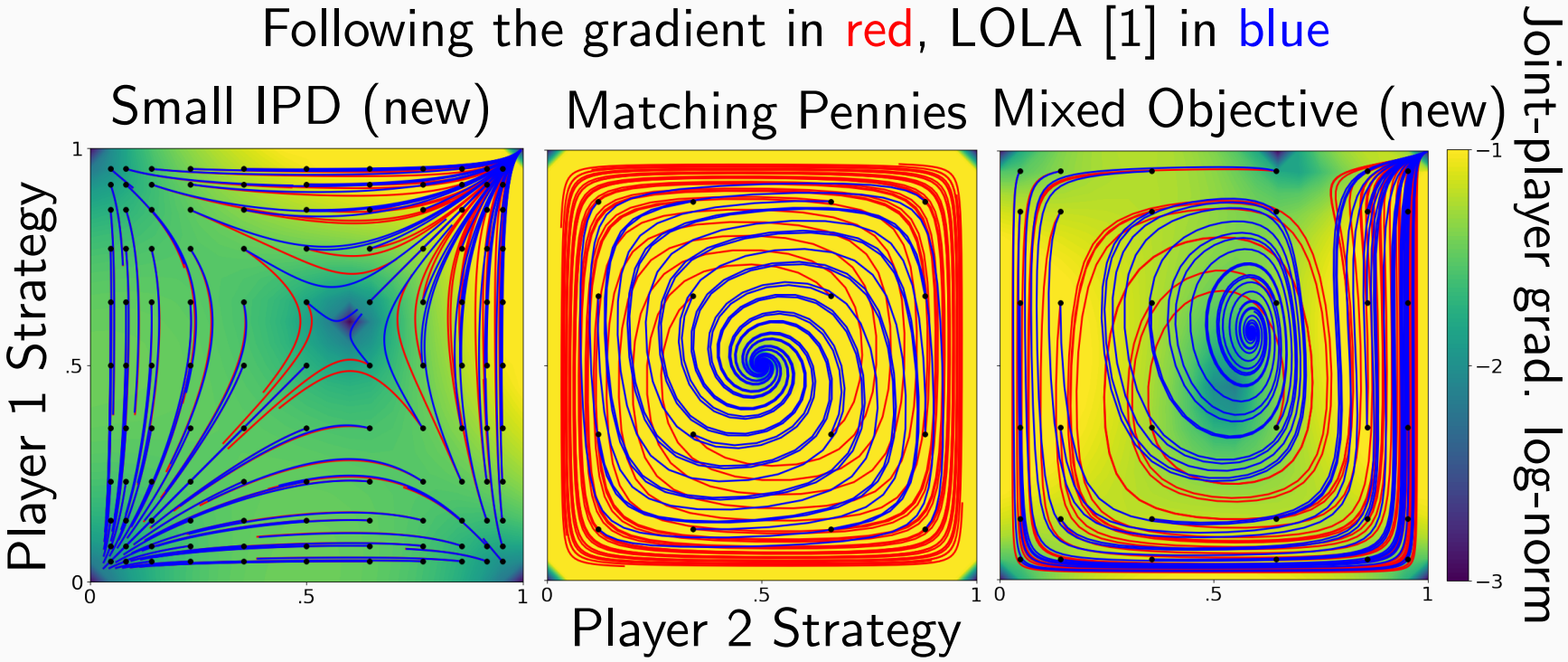
Illustrative Toy Problems

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



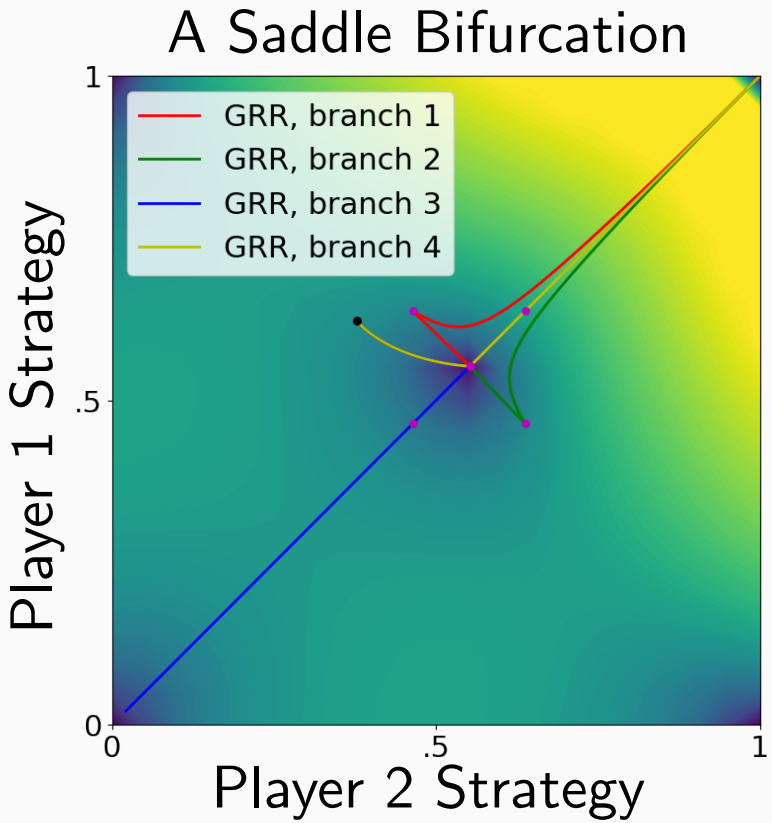
- Small IPD is a 2 param. Iterated Prisoner's Dilemma with TT and DD solutions, but only real EVals.
- Matching Pennies is a 2 param. rock-paper-scissors with imaginary EVals, but only 1 solution.

Illustrative Toy Problems

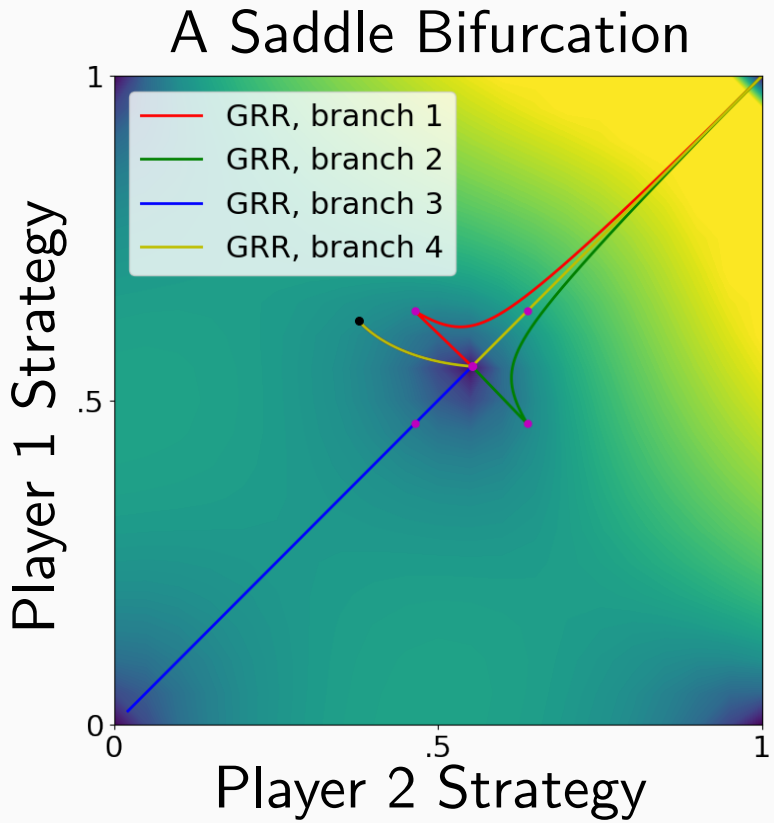


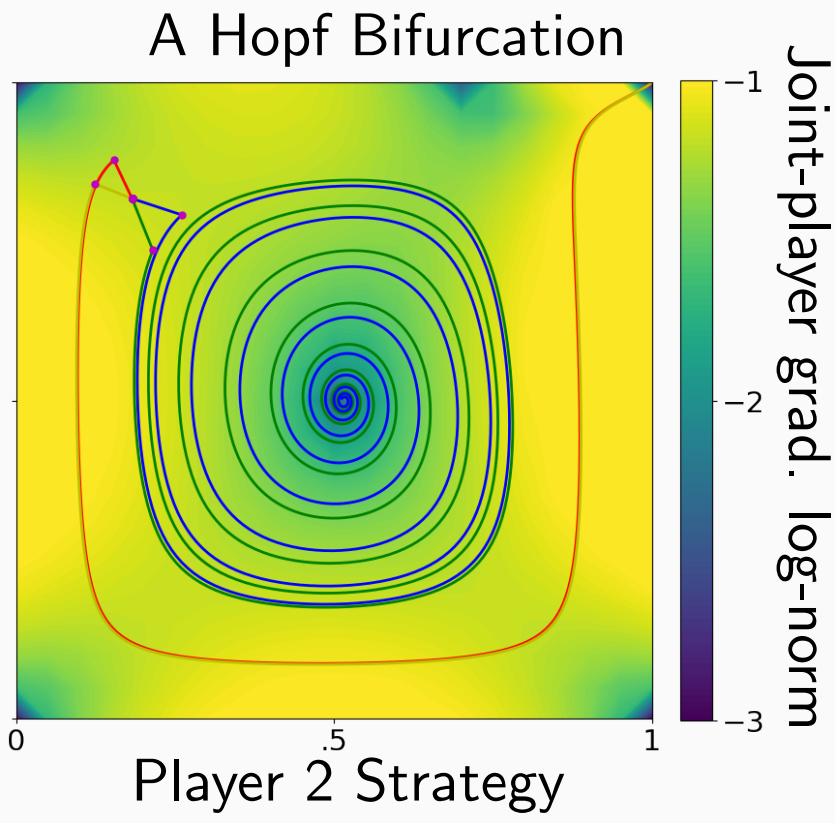
- Small IPD is a 2 param. Iterated Prisoner's Dilemma with TT and DD solutions, but only real EVals.
- Matching Pennies is a 2 param. rock-paper-scissors with imaginary EVals, but only 1 solution.
- Mixing these gives a 2 param. problem like the full IPD with multiple solutions, complex EVals, and a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

Showing the Bifurcations

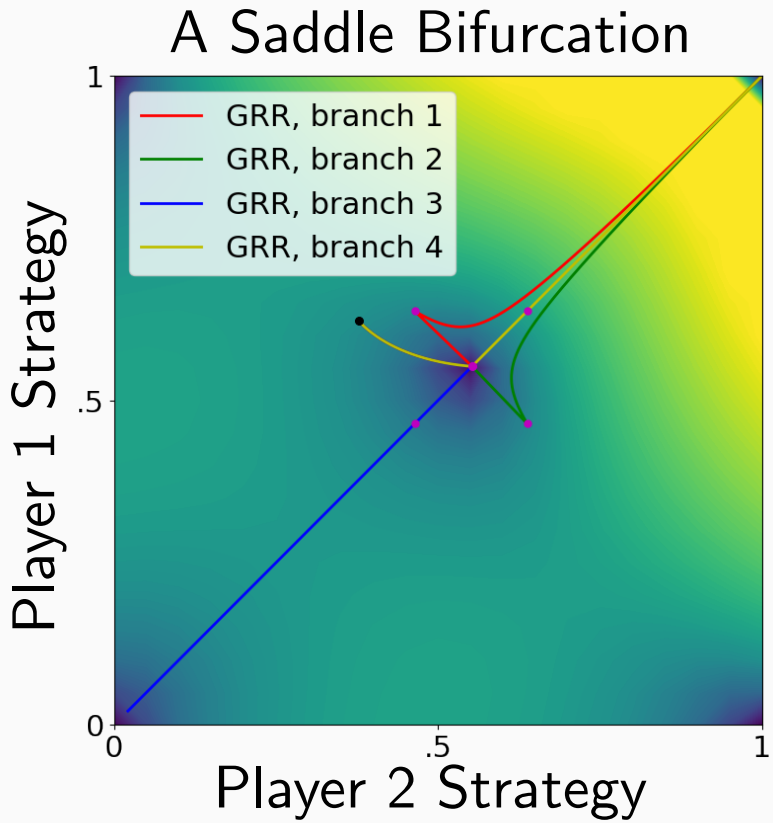


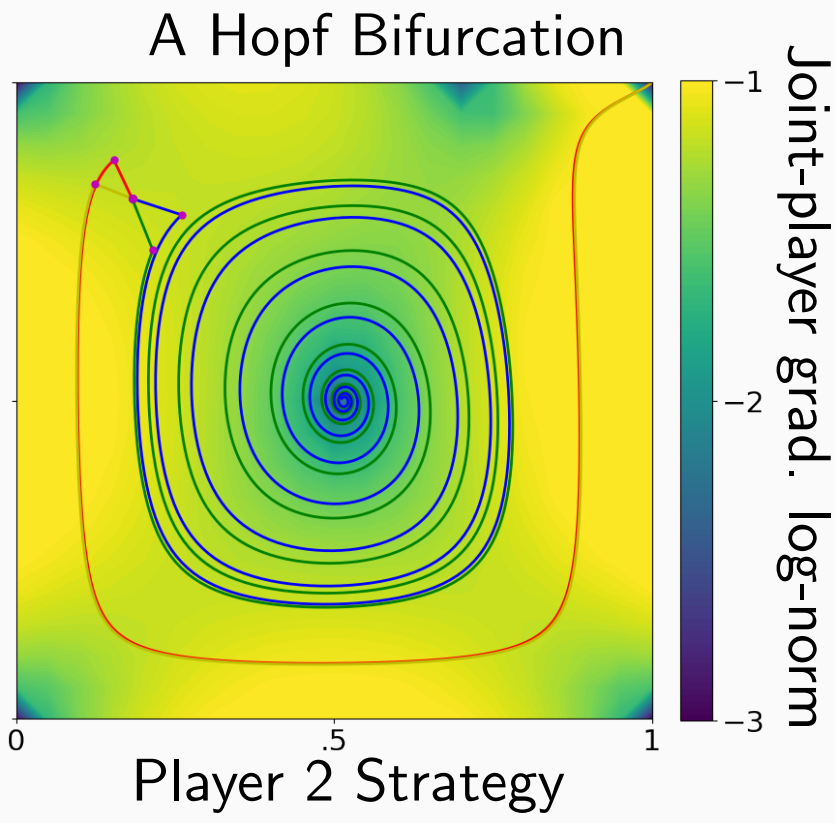
Showing the Bifurcations





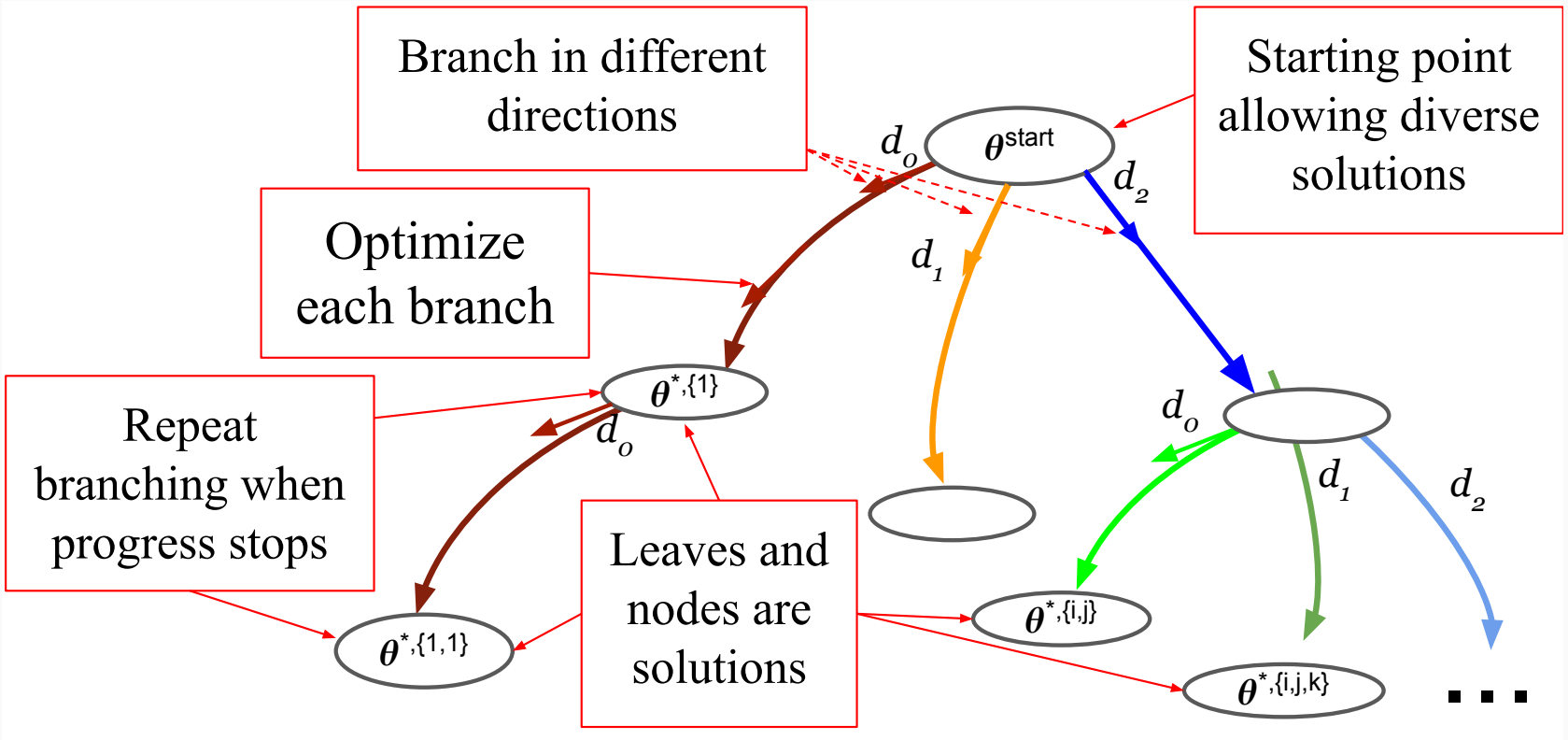
Showing the Bifurcations





- Now, lets show a framework for finding multiple solutions:

Branching Optimization Tree Search



- 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:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*), \boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B) \tag{1}$$

Notation for Finding Bifurcations

Remember our goal:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*), \boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B) \tag{1}$$

For simplicity, concatenate all players' parameters:

$$\boldsymbol{\omega} := [\boldsymbol{\theta}_A, \boldsymbol{\theta}_B] \tag{2}$$

Notation for Finding Bifurcations

Remember our goal:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*), \boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B) \tag{1}$$

For simplicity, concatenate all players' parameters:

$$\boldsymbol{\omega} := [\boldsymbol{\theta}_A, \boldsymbol{\theta}_B] \tag{2}$$

We are trying to find solutions with some optimizer or fixed-point operator:

$$\boldsymbol{\omega}^{j+1} = \boldsymbol{F}(\boldsymbol{\omega}^j) \tag{3}$$

Notation for Finding Bifurcations

Remember our goal:

$$\boldsymbol{\theta}_A^* \in \arg\min_{\boldsymbol{\theta}_A} \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*), \boldsymbol{\theta}_B^* \in \arg\min_{\boldsymbol{\theta}_B} \mathcal{L}_B(\boldsymbol{\theta}_A^*, \boldsymbol{\theta}_B) \tag{1}$$

For simplicity, concatenate all players' parameters:

$$\boldsymbol{\omega} := [\boldsymbol{\theta}_A, \boldsymbol{\theta}_B] \tag{2}$$

We are trying to find solutions with some optimizer or fixed-point operator:

$$\boldsymbol{\omega}^{j+1} = \boldsymbol{F}(\boldsymbol{\omega}^j) \tag{3}$$

For example, SGD or LOLA:

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

Notation for Finding Bifurcations

- Fixed point operator for SGD:

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

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

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

Notation for Finding Bifurcations

- Fixed point operator for SGD:

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

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

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

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

Notation for Finding Bifurcations

- Fixed point operator for SGD:

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

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

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

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

$$\boldsymbol{d}^\top (\boldsymbol{J}^\top \boldsymbol{J})|_{\boldsymbol{\omega}} \boldsymbol{d} \tag{7}$$

Notation for Finding Bifurcations

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

$$\boldsymbol{d}^\top (\boldsymbol{J}^\top \boldsymbol{J})|_{\boldsymbol{\omega}} \boldsymbol{d} \tag{8}$$

- Interested in the spread at iterate $\boldsymbol{\omega}^j$ from the fixed-point op. from $\boldsymbol{\omega}_0$. So, define Jacobian there $\boldsymbol{J}^j(\boldsymbol{\omega}_0)$

Notation for Finding Bifurcations

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

$$\boldsymbol{d}^\top (\boldsymbol{J}^\top \boldsymbol{J})|_{\boldsymbol{\omega}} \boldsymbol{d} \tag{8}$$

- Interested in the spread at iterate $\boldsymbol{\omega}^j$ from the fixed-point op. from $\boldsymbol{\omega}_0$. So, define Jacobian there $\boldsymbol{J}^j(\boldsymbol{\omega}_0)$
- If we take log, then $+$ when diverge and $-$ when converge:

$$\gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) = \log(\boldsymbol{d}^\top (\boldsymbol{J}^j(\boldsymbol{\omega}_0))^\top \boldsymbol{J}^j(\boldsymbol{\omega}_0) \boldsymbol{d}) \tag{9}$$

Notation for Finding Bifurcations

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

$$\boldsymbol{d}^\top (\boldsymbol{J}^\top \boldsymbol{J})|_{\boldsymbol{\omega}} \boldsymbol{d} \tag{8}$$

- Interested in the spread at iterate $\boldsymbol{\omega}^j$ from the fixed-point op. from $\boldsymbol{\omega}_0$. So, define Jacobian there $\boldsymbol{J}^j(\boldsymbol{\omega}_0)$
- If we take log, then $+$ when diverge and $-$ when converge:

$$\gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) = \log(\boldsymbol{d}^\top (\boldsymbol{J}^j(\boldsymbol{\omega}_0))^\top \boldsymbol{J}^j(\boldsymbol{\omega}_0)\boldsymbol{d}) \tag{9}$$

- Aggregate these values over opt. trajectory via average:

$$\hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}) = \frac{1}{k}\sum_{j=0}^{k} \gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) \tag{10}$$

What are Lyapunov Exponents?

$$\gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) = \log(\boldsymbol{d}^\top (\boldsymbol{J}^j(\boldsymbol{\omega}_0))^\top \boldsymbol{J}^j(\boldsymbol{\omega}_0)\boldsymbol{d}) \tag{11}$$

- Aggregate these values over opt. trajectory via average:

$$\hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}) = \frac{1}{k}\sum_{j=0}^{k}\gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) \tag{12}$$

What are Lyapunov Exponents?

$$\gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) = \log(\boldsymbol{d}^\top (\boldsymbol{J}^j(\boldsymbol{\omega}_0))^\top \boldsymbol{J}^j(\boldsymbol{\omega}_0)\boldsymbol{d}) \tag{11}$$

- Aggregate these values over opt. trajectory via average:

$$\hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}) = \frac{1}{k}\sum_{j=0}^{k}\gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) \tag{12}$$

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

$$\hat{\lambda}_k^{max}(\boldsymbol{\omega}_0) = \max_{\boldsymbol{d}, \|\boldsymbol{d}\|=1} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}) \tag{13}$$

What are Lyapunov Exponents?

- Often interested in the max Lyapunov exponent:

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

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

$$\hat{\lambda}^{max}(\boldsymbol{\omega}_0) = \max_{\boldsymbol{d}, \|\boldsymbol{d}\|=1} \lim_{k \to \infty} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}) < 0 \tag{15}$$

What are Lyapunov Exponents?

- Often interested in the max Lyapunov exponent:

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

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

$$\hat{\lambda}^{max}(\boldsymbol{\omega}_0) = \max_{\boldsymbol{d}, \|\boldsymbol{d}\|=1} \lim_{k \to \infty} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}) < 0 \tag{15}$$

- 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:

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

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

$$\hat{\lambda}^{max}(\boldsymbol{\omega}_0) = \max_{\boldsymbol{d}, \|\boldsymbol{d}\|=1} \lim_{k \to \infty} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}) < 0 \tag{15}$$

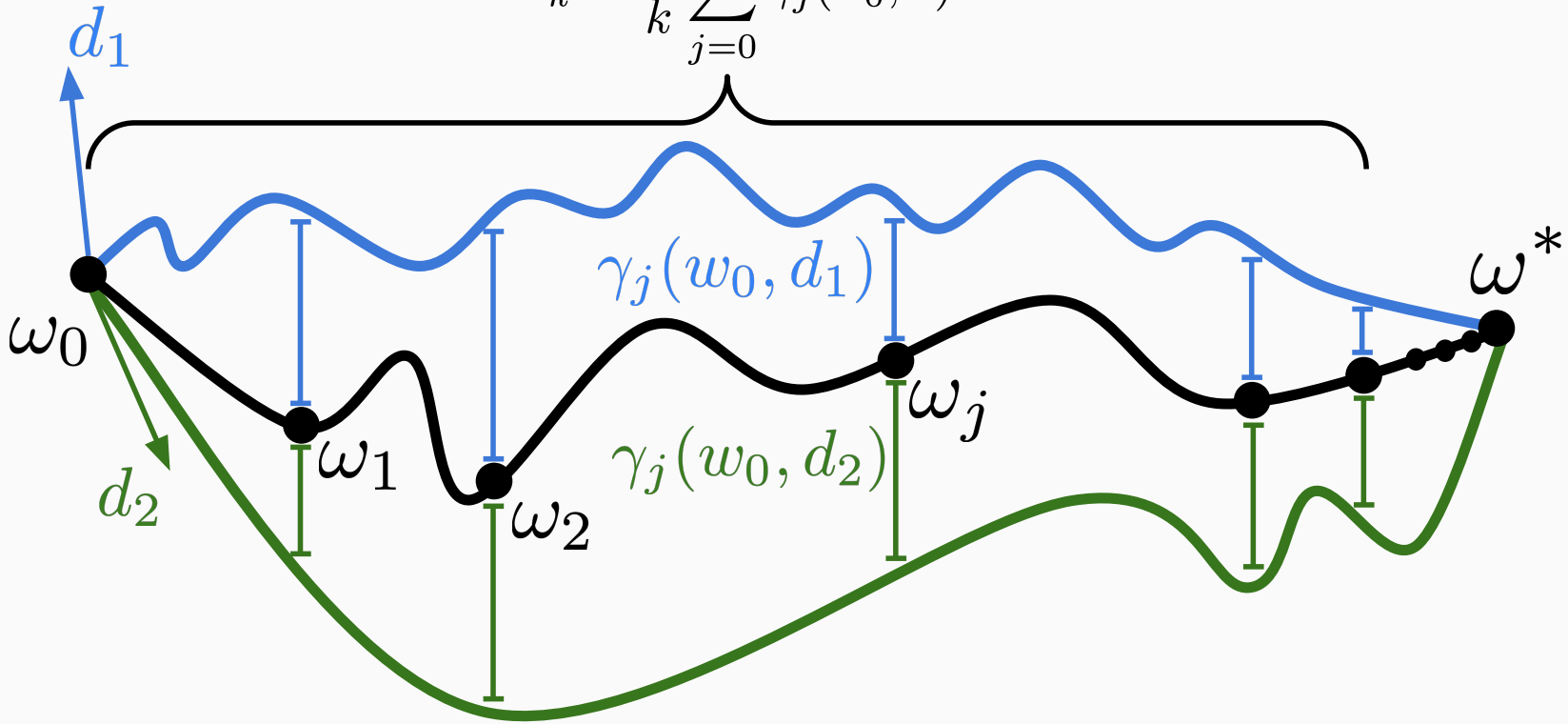
- Consequently, between basins of attraction exponent is max (at 0) and the max direction points along separatrix.
- Can find bifurcations between basins by maxing exponent!

What are Lyapunov Exponents?

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

$$\gamma_j(\boldsymbol{\omega}_0, \boldsymbol{d}) = \log(\boldsymbol{d}^\top (\boldsymbol{J}^j(\boldsymbol{\omega}_0))^\top \boldsymbol{J}^j(\boldsymbol{\omega}_0)\boldsymbol{d}) \tag{16}$$

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



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.

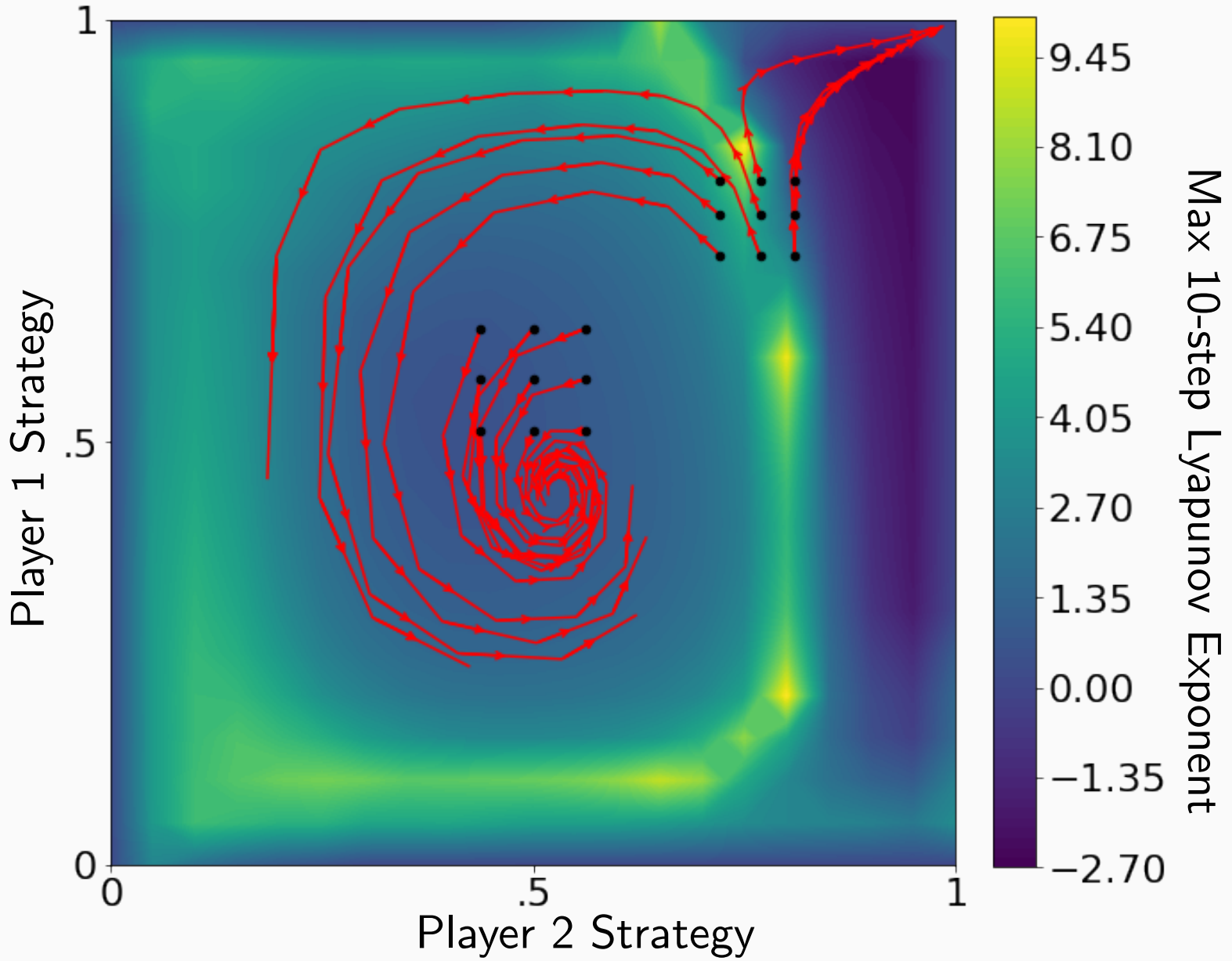
Truncated Lyapunov Exponents

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

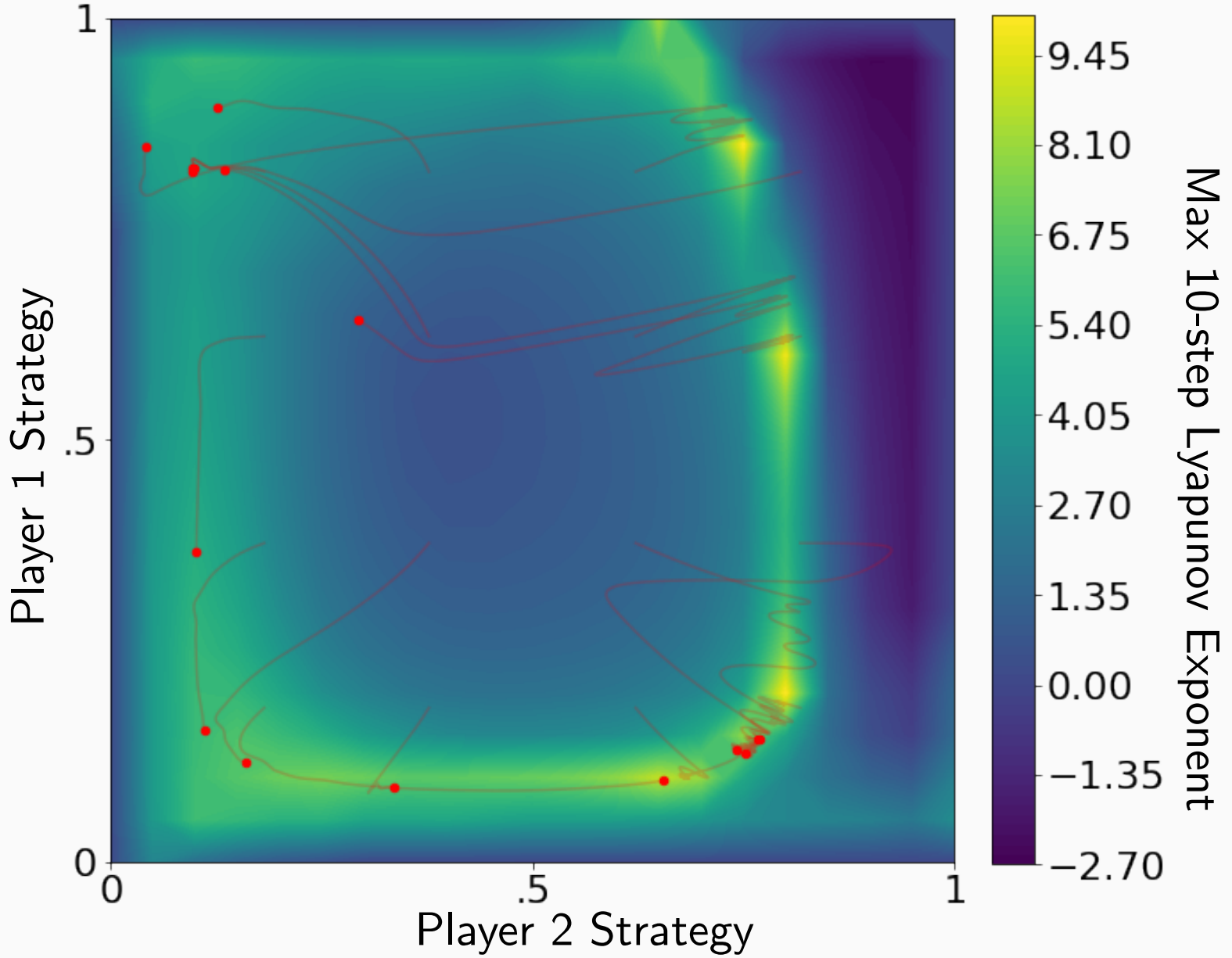
1. Computationally tractability
2. Non-zero gradient signals for finding bifurcations
3. A better separation rate description for the finite trajectories used in practice

- But, we lose many of the theoretical results.
- The next slides help to build an intuition for the exponent of visualizable toy problems.

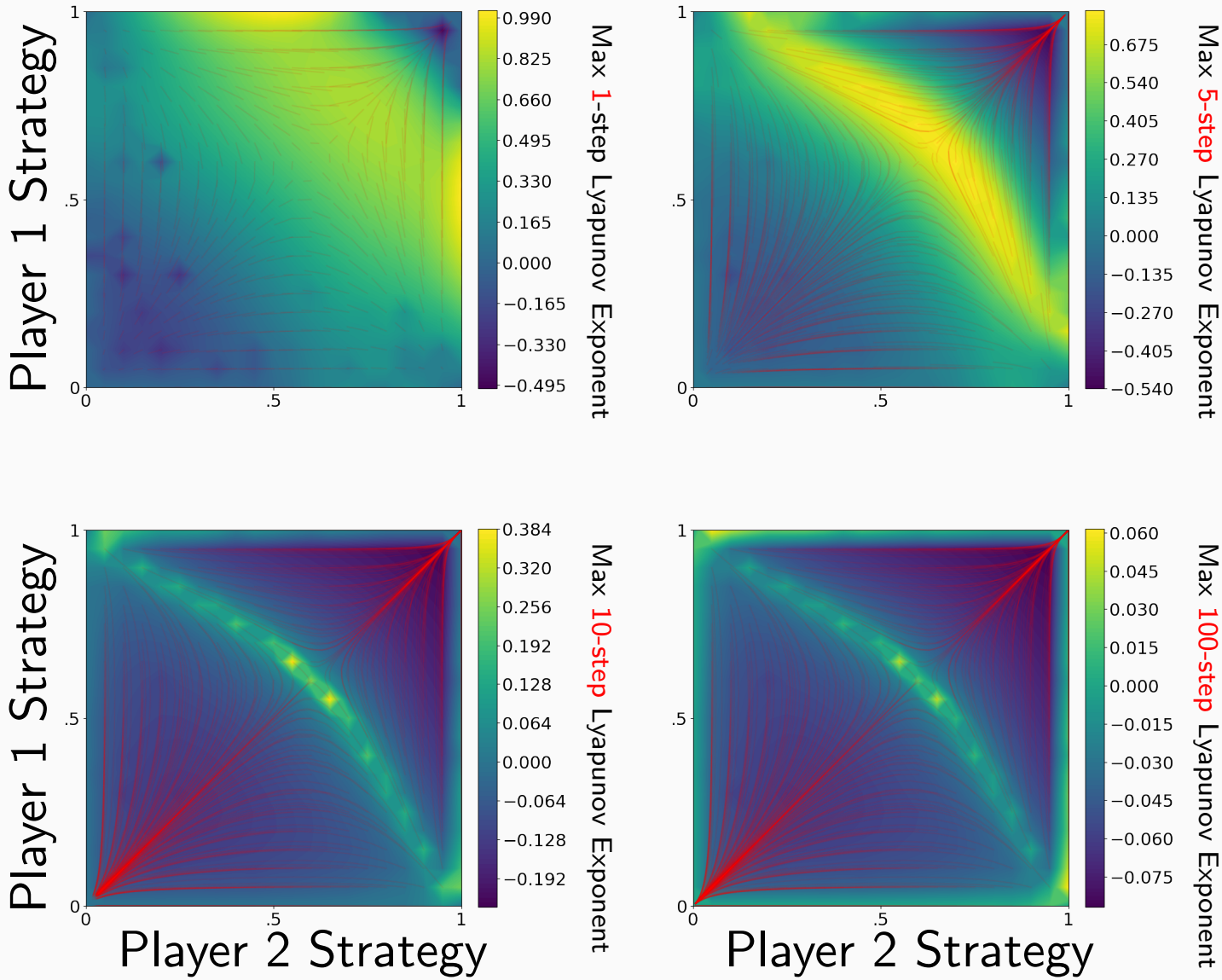
Showing Bifurcations with Lyapunov Exponents - animation



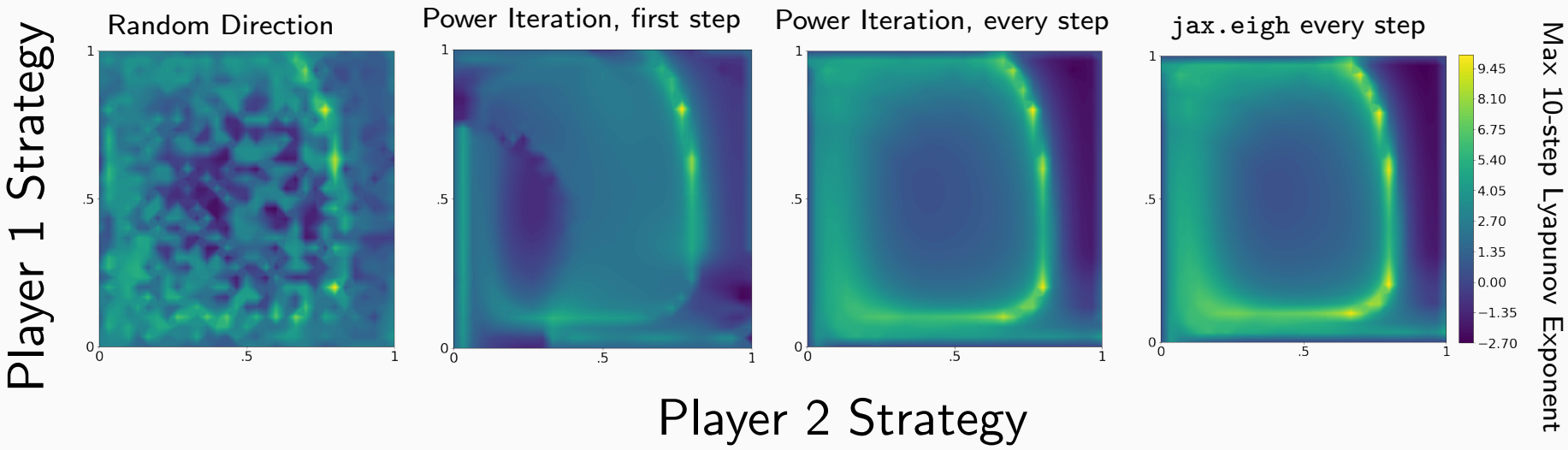
Optimizing Lyapunov Exponents to Find Bifurcations



Impact of Optimization Horizon on Exponent - animation

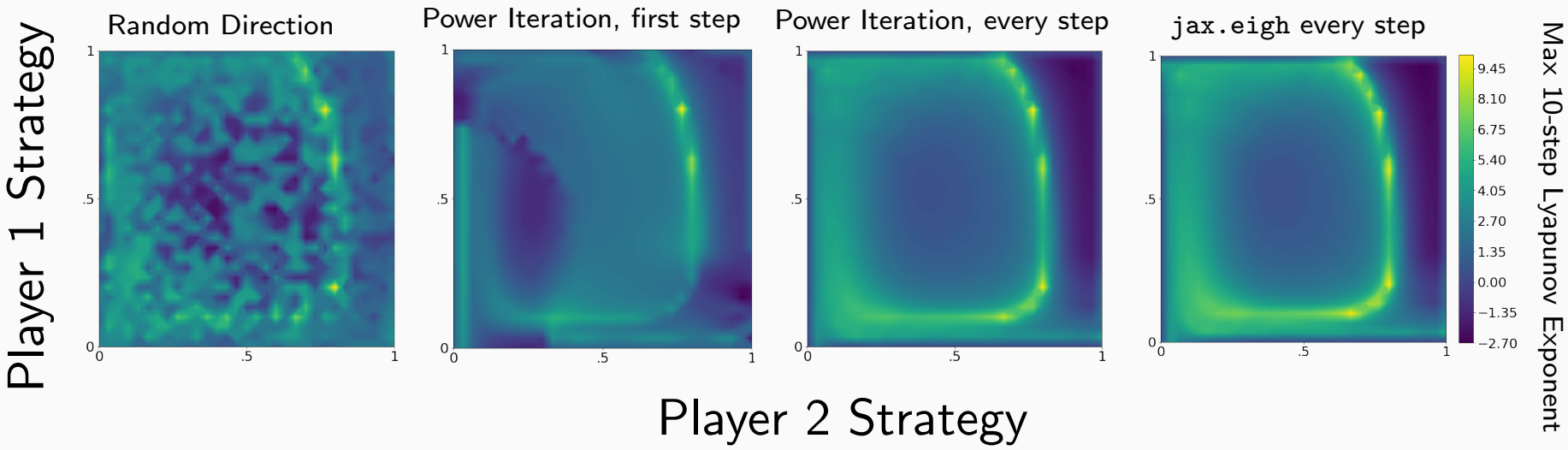


Impact of Direction Choice on Showing Bifurcations



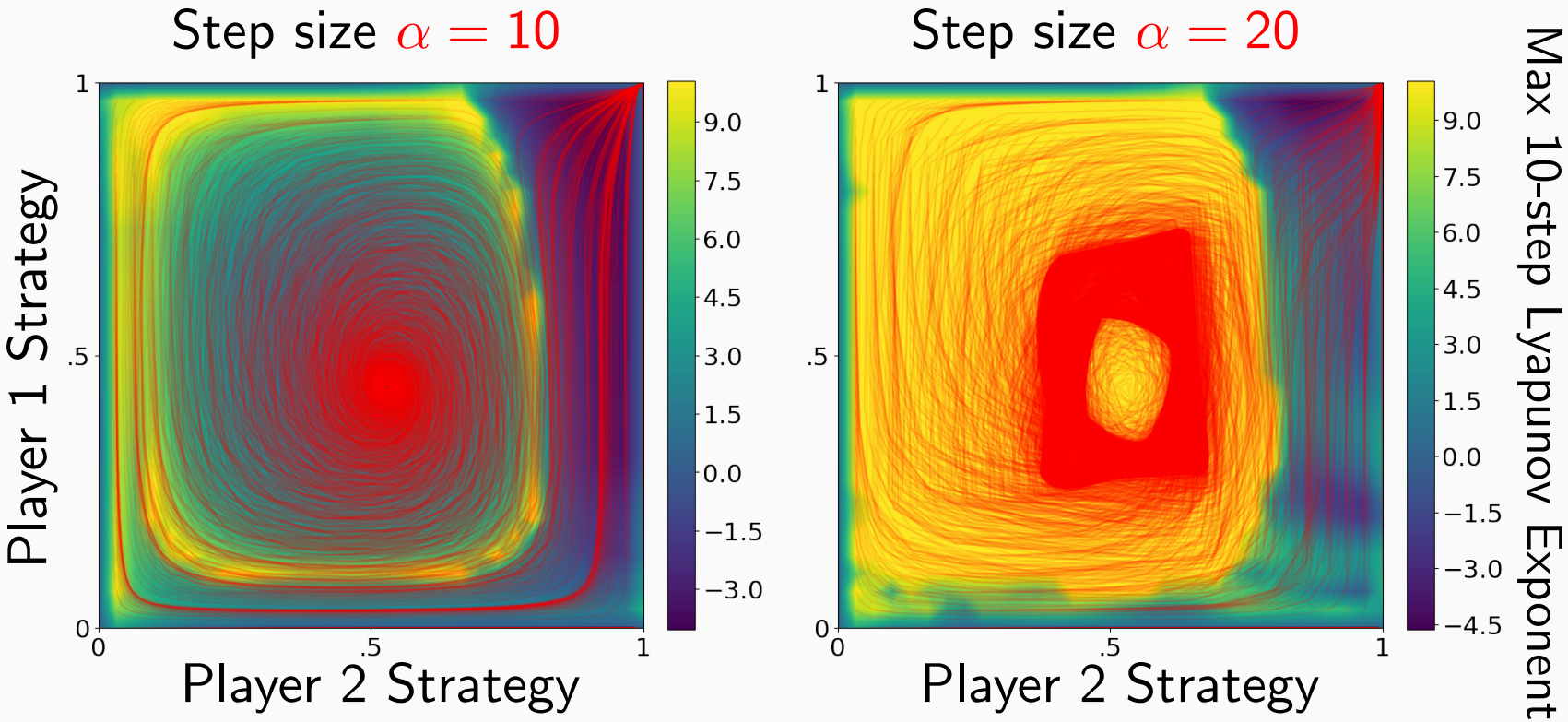
- Re-estimating the top EVecs at each iteration performs best, but is most expensive, and diverges from theory.

Impact of Direction Choice on Showing Bifurcations



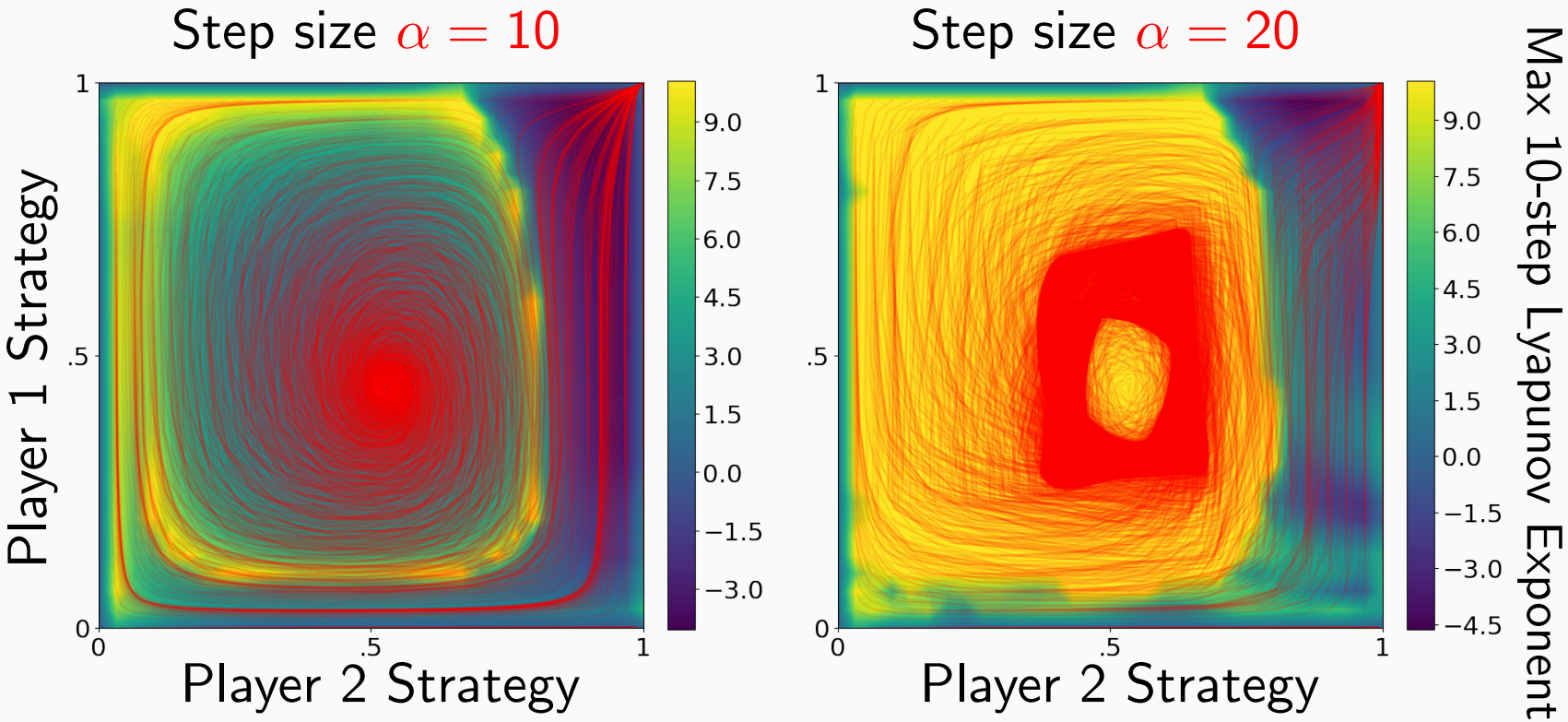
- Re-estimating the top EVecs at each iteration performs best, but is most expensive, and diverges from theory.
- In the optimization limit $k \to \infty$ the choice of direction almost certainly doesn't matter.

What if the Optimizer doesn't Converge?



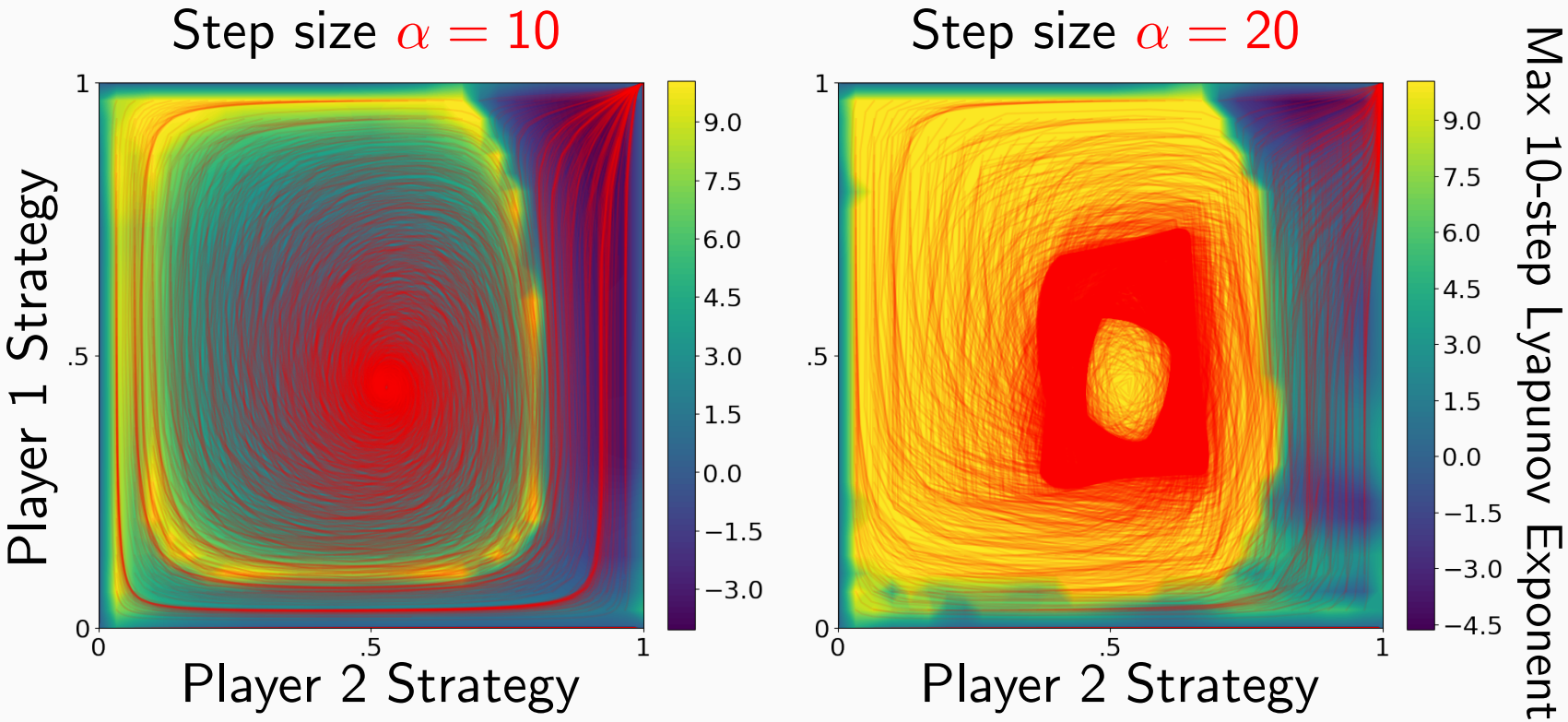
- Make sure step size small, so optimizer converges.

What if the Optimizer doesn't Converge?



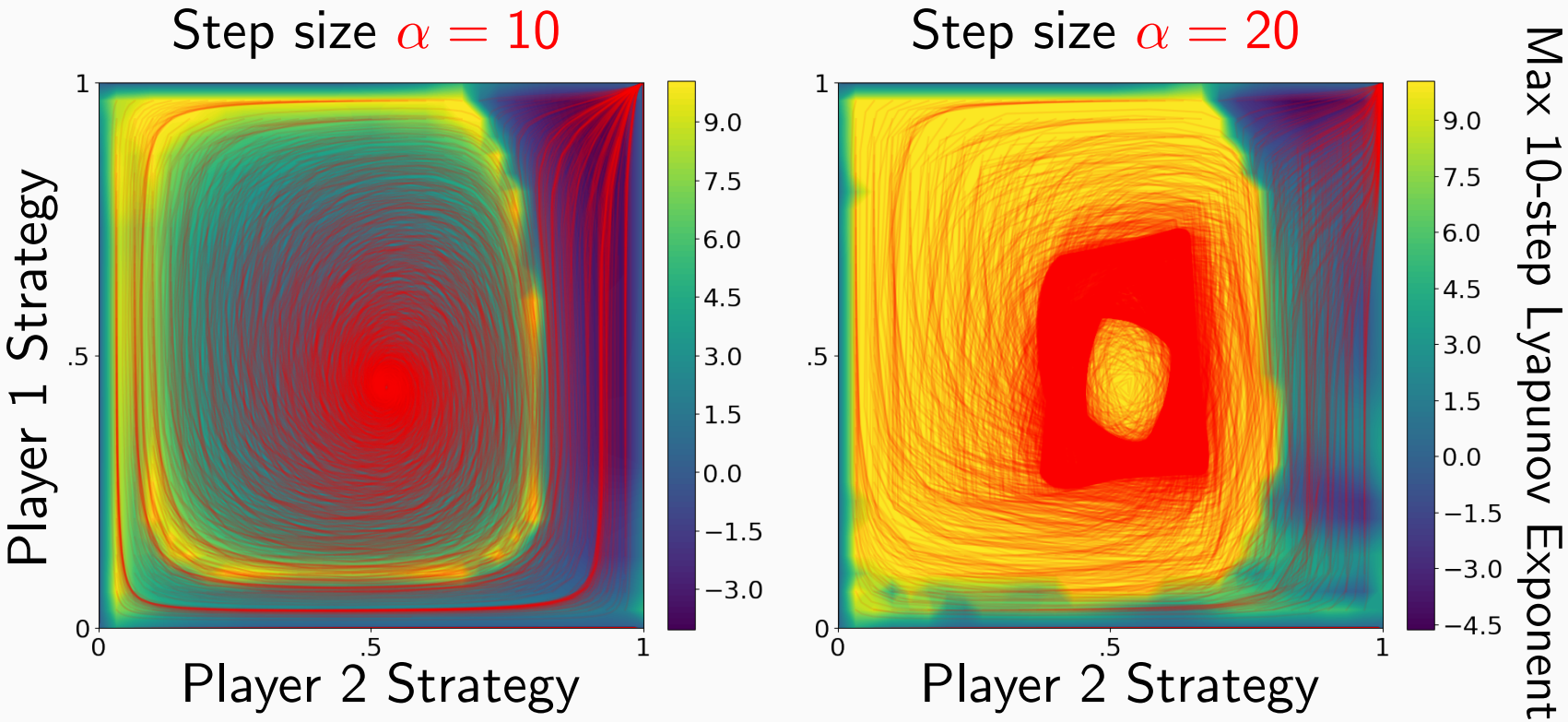
- Make sure step size small, so optimizer converges.
- If not, exponent maybe max where trajectories don't find soln.

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...

Random Subspace Problem

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

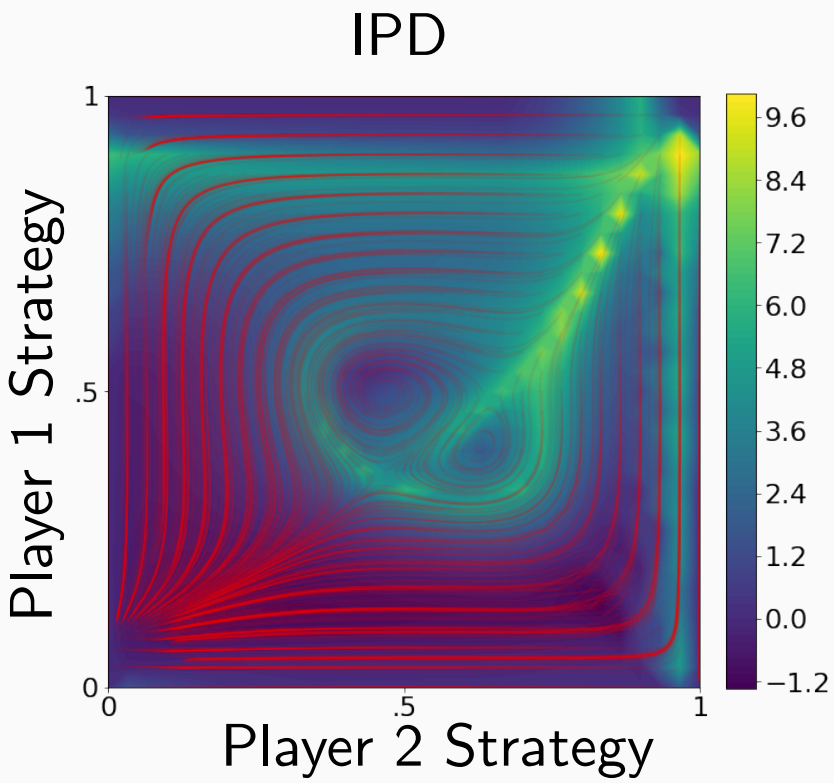
Random Subspace Problem

- 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

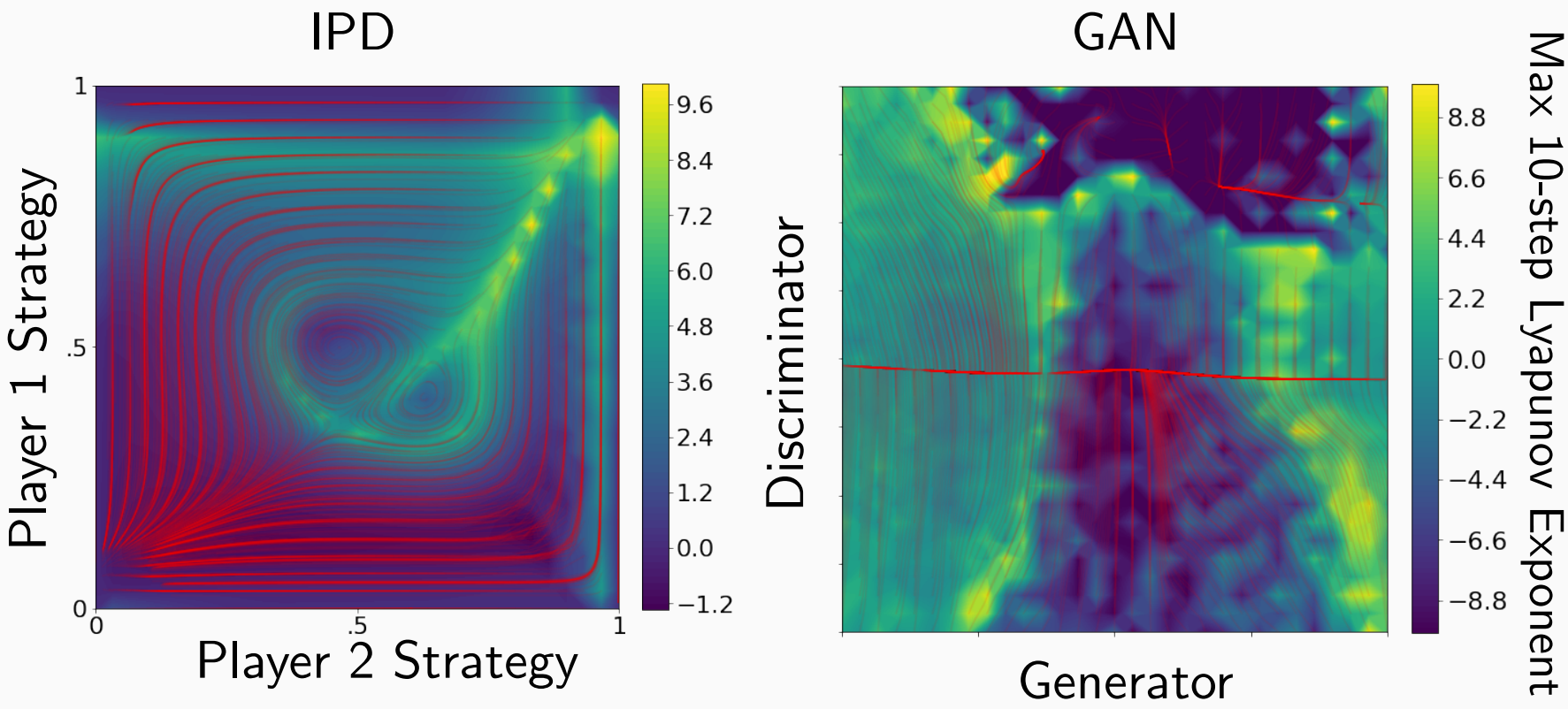
- But, we only looked at 2 bifurcations so far. Can we construct toy problems with more types?
- Idea: take high-dimensional game $\mathcal{L}_A(\theta_A, \theta_B), \mathcal{L}_B(\theta_A, \theta_B)$ and optimize in a subspace. Ex., GAN, IPD, HO, meta-learning,...
- Specifically, use $\mathcal{L}_A(\mathbf{v}_A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

Random Subspace Problem



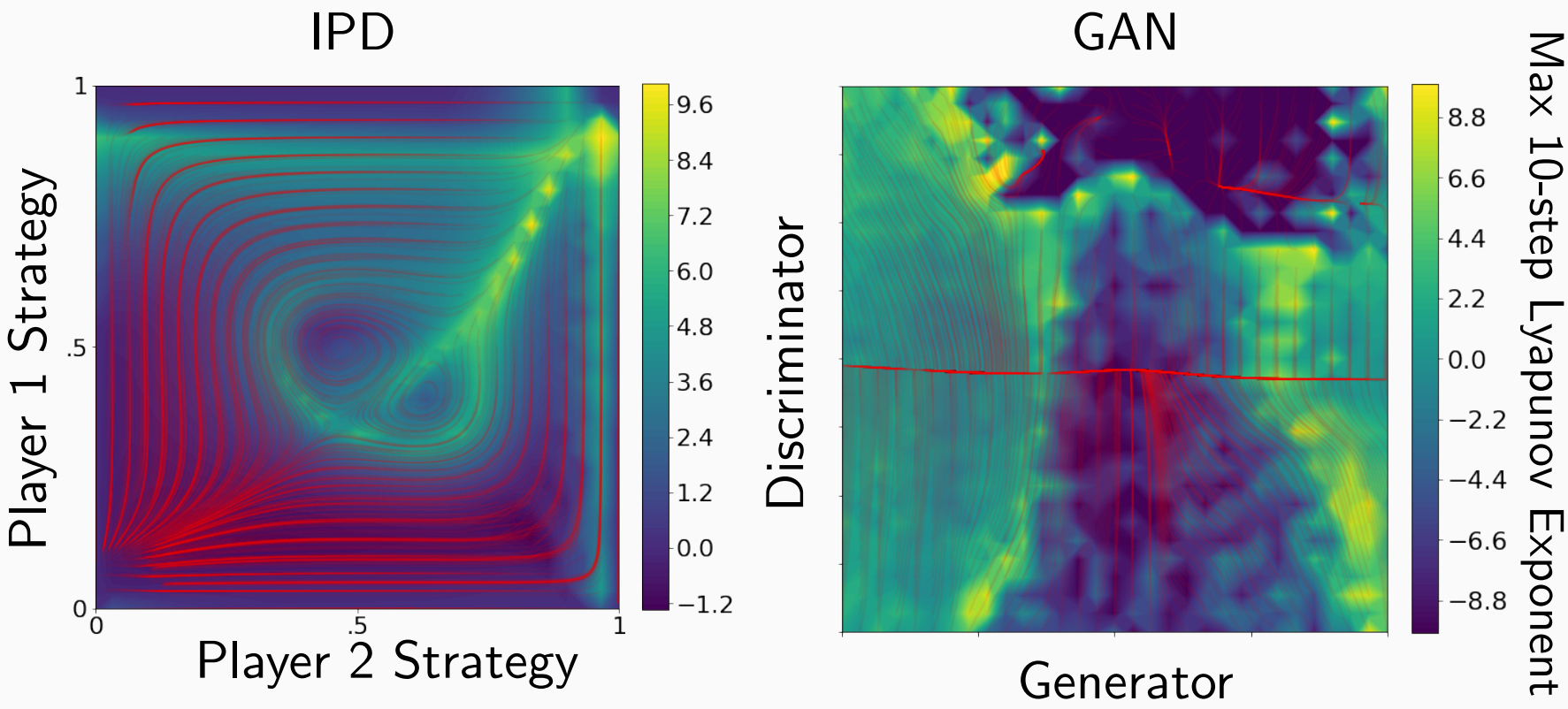
- Exponent peaks near where trajectories separate, showing we find bifurcations.
- Are some bifurcations more prevalent in some games?

Random Subspace Problem



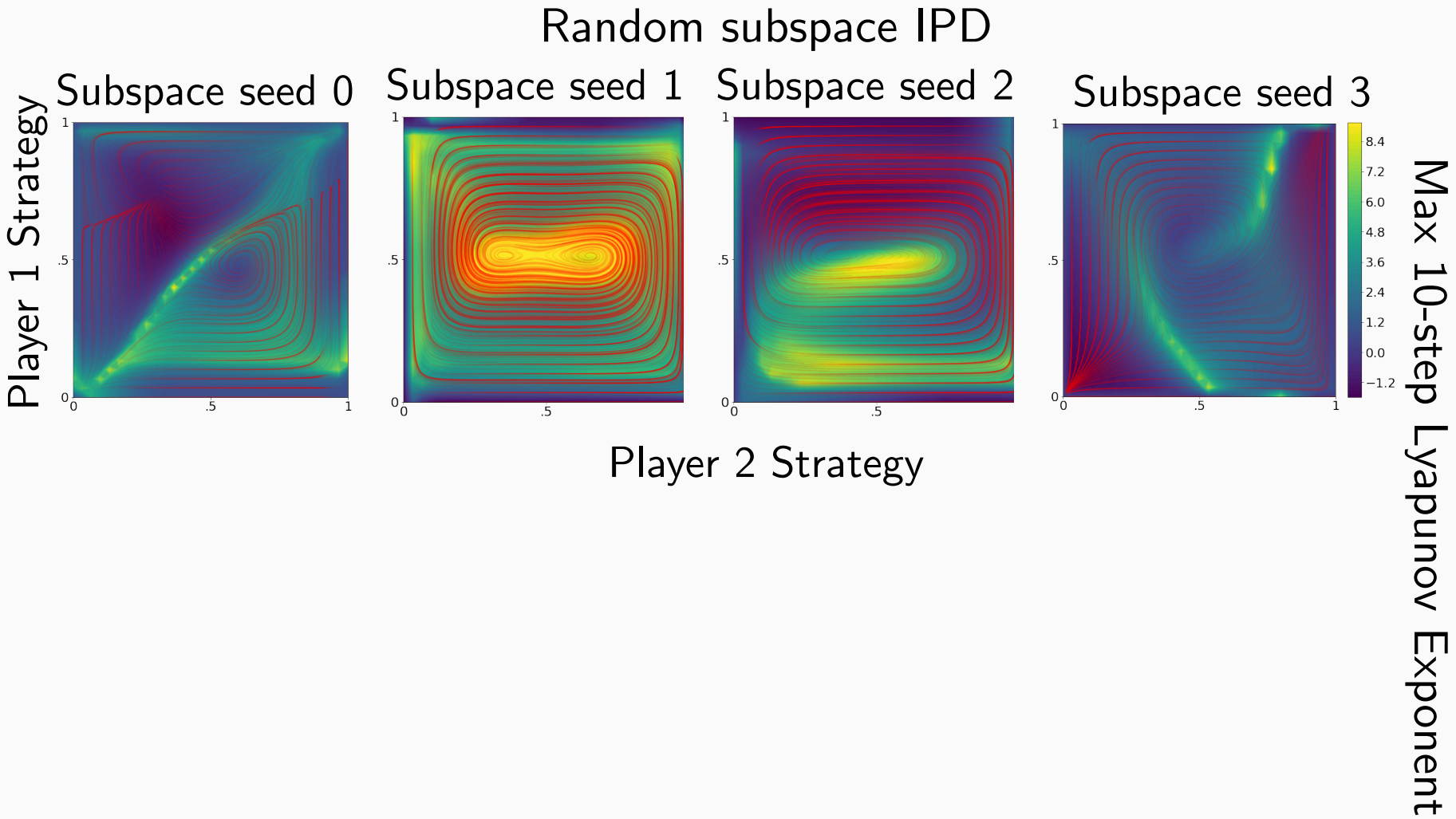
- Exponent peaks near where trajectories separate, showing we find bifurcations.

Random Subspace Problem

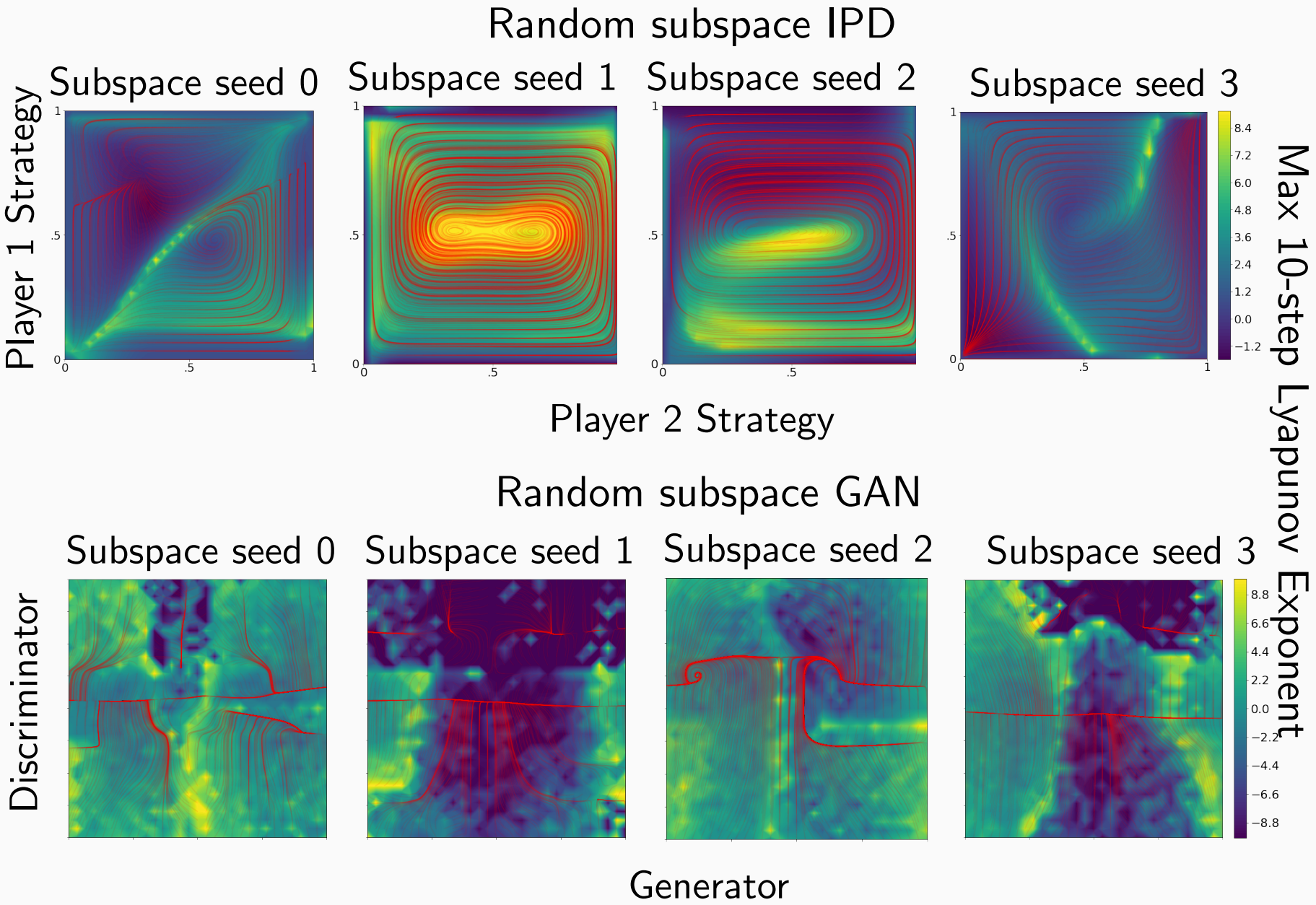


- Exponent peaks near where trajectories separate, showing we find bifurcations.
- Are some bifurcations more prevalent in some games?

Random Subspace Problem



Random Subspace Problem



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).

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.

- We look at these in the following table.

Finding Diverse Solutions in the Iterated Prisoners Dilemma

Search Strategy	Solution Mode: Cooperate	Solution Mode: Defect
$\times 20$ Random init + LOLA [1]	✓	×
$\times 20$ 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	Solution Mode: Defect
×20 Random init + LOLA [1]	✓	×
×20 Random init + GD	×	✓
GRR: tune max Lyap + top EVec branch + GD	✓	✓
GRR: tune max Lyap + top EVec branch + LOLA	✓	✓

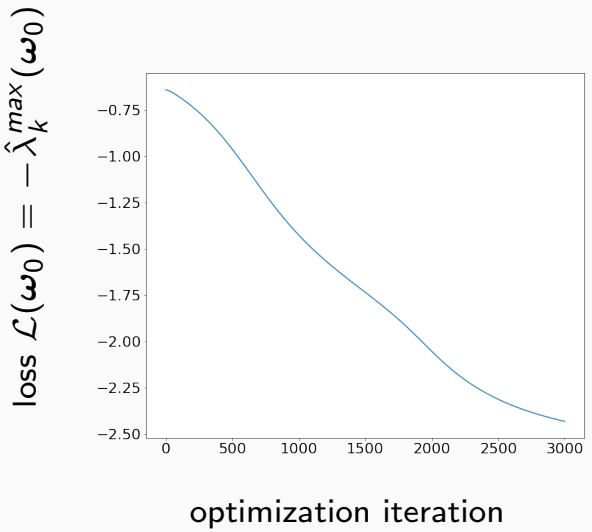
- Randomly init. then applying a training method only finds 1 solution mode. Baselines don't find both.
- Our method finds both solution modes (with *any* opt.).

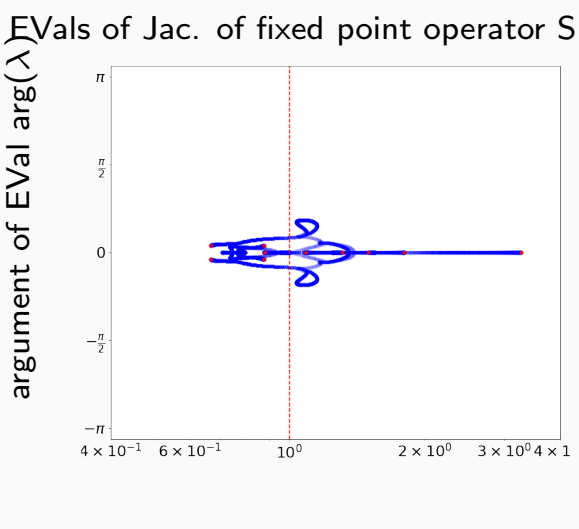
Finding Diverse Solutions in the Iterated Prisoners Dilemma

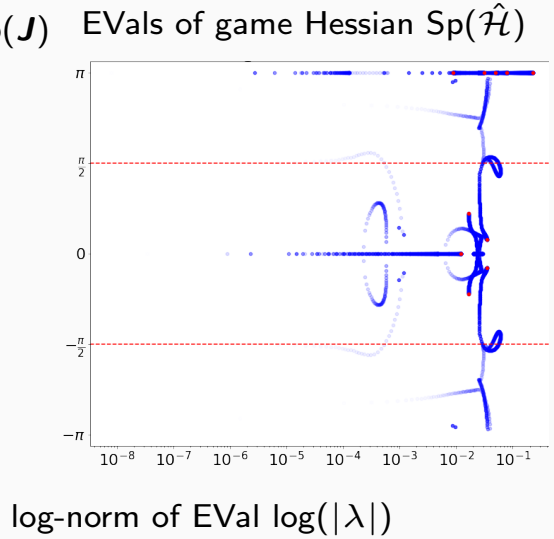
Search Strategy	Solution Mode: Cooperate	Solution Mode: Defect
×20 Random init + LOLA [1]	✓	×
×20 Random init + GD	×	✓
GRR: tune max Lyap + top EVec branch + GD	✓	✓
GRR: tune max Lyap + top EVec branch + LOLA	✓	✓
×20 Random init + top EVec branch + GD	×	✓

- 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.

Analyzing the IPD Optimization







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

Connections to Kolmogorov-Sinai Entropy

- Max exponent $\hat{\lambda}_k^{max}(\boldsymbol{\omega}_0) = \max_{\boldsymbol{d}, \|\boldsymbol{d}\|=1} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d})$ only guarantees separation in 1 direction.

Connections to Kolmogorov-Sinai Entropy

- Max exponent $\hat{\lambda}_k^{max}(\boldsymbol{\omega}_0) = \max_{\boldsymbol{d}, \|\boldsymbol{d}\|=1} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d})$ only guarantees separation in 1 direction.
- What if we want spread in multiple directions?

$$\mathcal{L}_n^{\text{sum}}(\boldsymbol{\omega}_0) = - \max_{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n} \sum_{l=1}^{n} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}_l),$$

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

Connections to Kolmogorov-Sinai Entropy

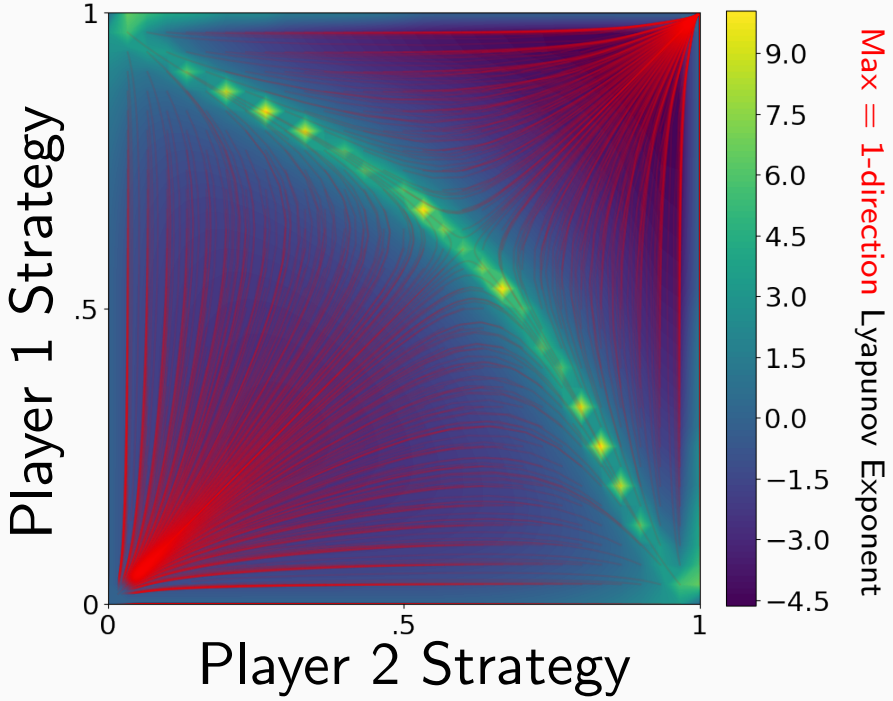
- Max exponent $\hat{\lambda}_k^{max}(\boldsymbol{\omega}_0) = \max_{\boldsymbol{d}, \|\boldsymbol{d}\|=1} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d})$ only guarantees separation in 1 direction.
- What if we want spread in multiple directions?

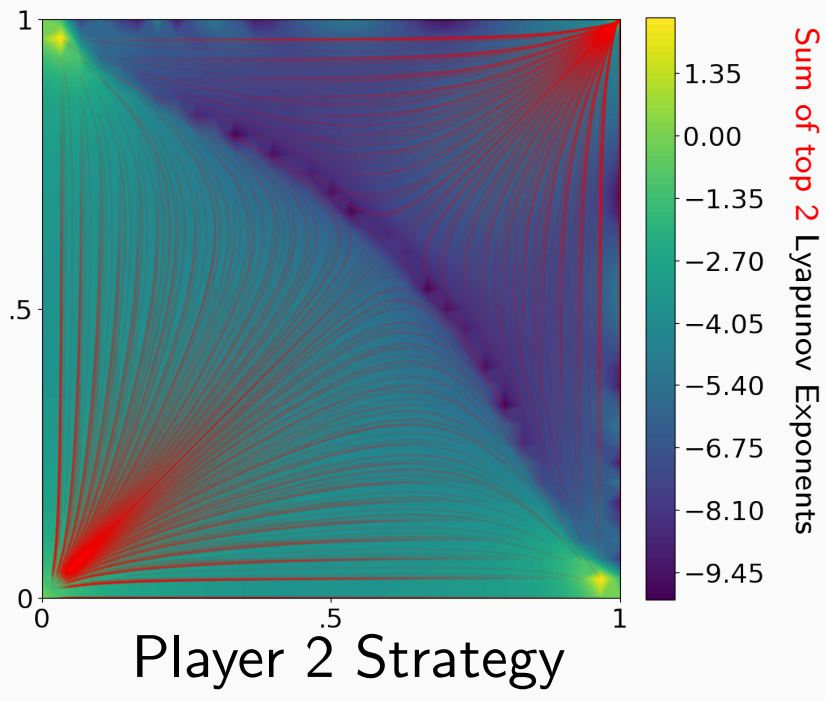
$$\mathcal{L}_n^{\text{sum}}(\boldsymbol{\omega}_0) = - \max_{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n} \sum_{l=1}^{n} \hat{\lambda}_k(\boldsymbol{\omega}_0, \boldsymbol{d}_l),$$

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

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

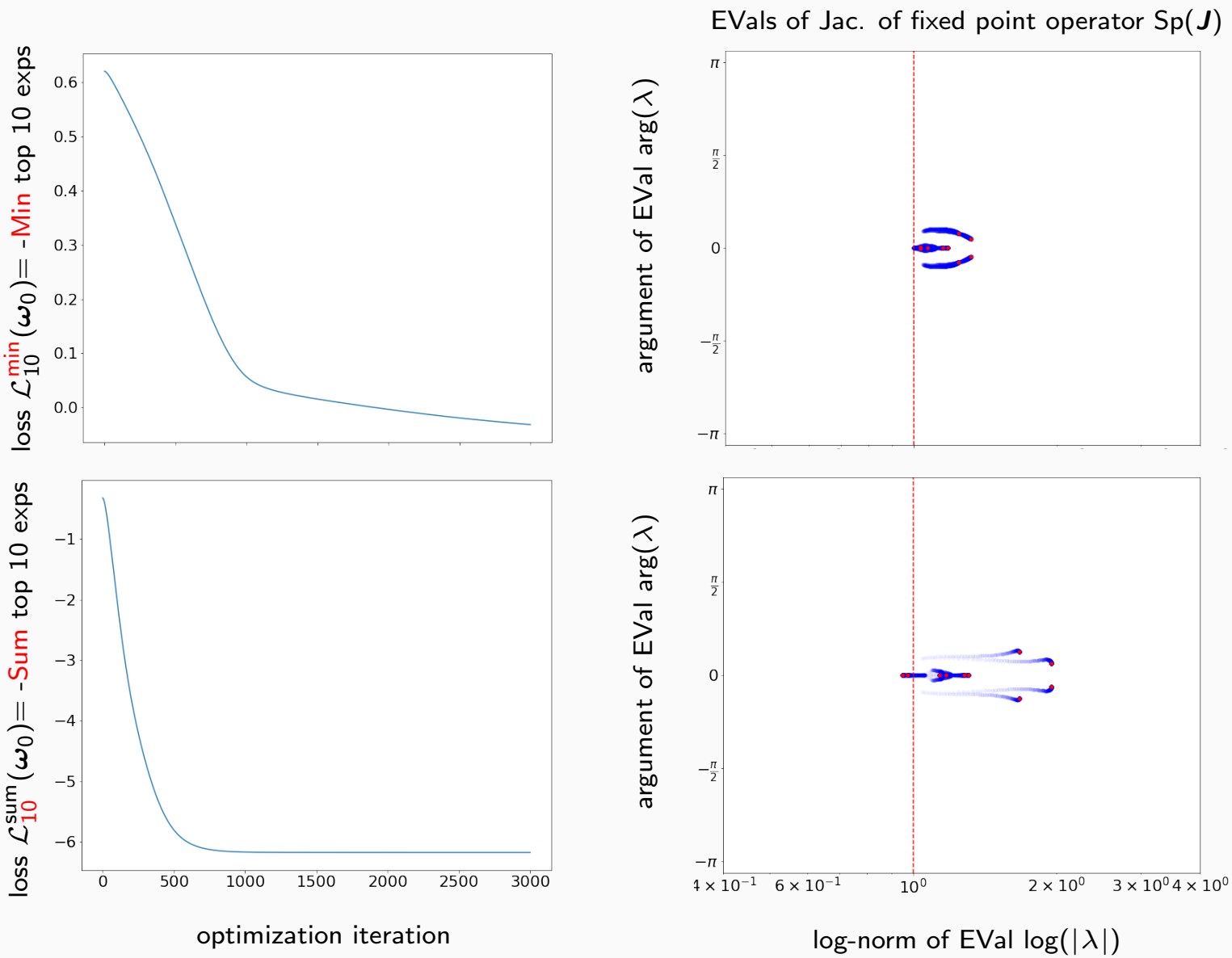
Connections to Kolmogorov-Sinai Entropy





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

Trajectory Separation in Multiple Directions



Scaling up to GANs

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

- Goal: Exponent calculation is scalable to larger problems.
- Mean and std. dev. (over 10 runs) of the max 10-step exponent and the log-prob. of an ensemble of 5 GANs branching in the top 5 directions at the init.
- Higher exponent then better ensemble performance?
- Each branch's GAN may be learning a different part of the data distribution.

Estimating EVecs in Single Objective

	MNIST Accuracy	
# HVP Evaluations	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 idra to other setups.

Thanks!

Jonathan Lorraine

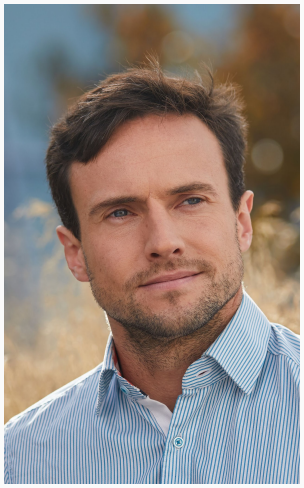
Paul Vicol

Jack Parker-Holder

Tal Kachman

Luke Metz

Jakob Foerster



Connecting to my other work

Hyperparameter Optimization is Nested Optimization

- $\mathcal{L}_T$ is training loss.
- $\mathcal{L}_V$ is validation loss.
- $\mathbf{w}$ are (elementary or NN) parameters.
- $\boldsymbol{\lambda}$ are hyperparameters.

Hyperparameter Optimization is Nested Optimization

- $\mathcal{L}_T$ is training loss.
- $\mathcal{L}_V$ is validation loss.
- $\mathbf{w}$ are (elementary or NN) parameters.
- $\boldsymbol{\lambda}$ are hyperparameters.
- $\mathbf{w}^*(\boldsymbol{\lambda})$ are the best parameters on the train loss given the hyperparameters:

$$\mathbf{w}^*(\boldsymbol{\lambda}) := \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w})$$

Hyperparameter Optimization is Nested Optimization

- $\mathcal{L}_T$ is training loss.
- $\mathcal{L}_V$ is validation loss.
- $\mathbf{w}$ are (elementary or NN) parameters.
- $\boldsymbol{\lambda}$ are hyperparameters.
- $\mathbf{w}^*(\boldsymbol{\lambda})$ are the best parameters on the train loss given the hyperparameters:

$$\mathbf{w}^*(\boldsymbol{\lambda}) := \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w})$$

- Want to optimize validation loss using optimal parameters:

$$\mathcal{L}_V^*(\boldsymbol{\lambda}) := \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))$$

Hypergradient Decomposition

- The gradient is difficult to compute because we may need the **Jacobian of the best-response**, which could require differentiating through optimization:

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \frac{\partial \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})} \frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$$

Hypergradient Decomposition

- The gradient is difficult to compute because we may need the **Jacobian of the best-response**, which could require differentiating through optimization:

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \frac{\partial \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})} \frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$$

$$\frac{\partial \mathcal{L}_V^*}{\partial \boldsymbol{\lambda}} = \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}$$

IFT Hypergradient [8]

Theorem (Implicit Function Theorem)

If $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'} = 0$ *for some* $(\boldsymbol{\lambda}', \mathbf{w}')$ *and regularity conditions are satisfied, then surrounding* $(\boldsymbol{\lambda}', \mathbf{w}')$ *there exists a function* $\mathbf{w}^*(\boldsymbol{\lambda})$ *s.t.* $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} = 0$ *and*

$$\left.\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}'} = -\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \left.\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}', \mathbf{w}^*(\boldsymbol{\lambda}')}$$

IFT Hypergradient [8]

Theorem (Implicit Function Theorem)

If $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'} = 0$ *for some* $(\boldsymbol{\lambda}', \mathbf{w}')$ *and regularity conditions are satisfied, then surrounding* $(\boldsymbol{\lambda}', \mathbf{w}')$ *there exists a function* $\mathbf{w}^*(\boldsymbol{\lambda})$ *s.t.* $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} = 0$ *and*

$$\left.\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}'} = -\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \left.\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}', \mathbf{w}^*(\boldsymbol{\lambda}')}$$

So,

$$\begin{aligned}\left.\frac{\partial \mathcal{L}_V^*}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}'} &= \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}} \\ &= -\frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1} \left.\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}', \mathbf{w}^*(\boldsymbol{\lambda}')}\end{aligned}$$

Learned Data Augmentation

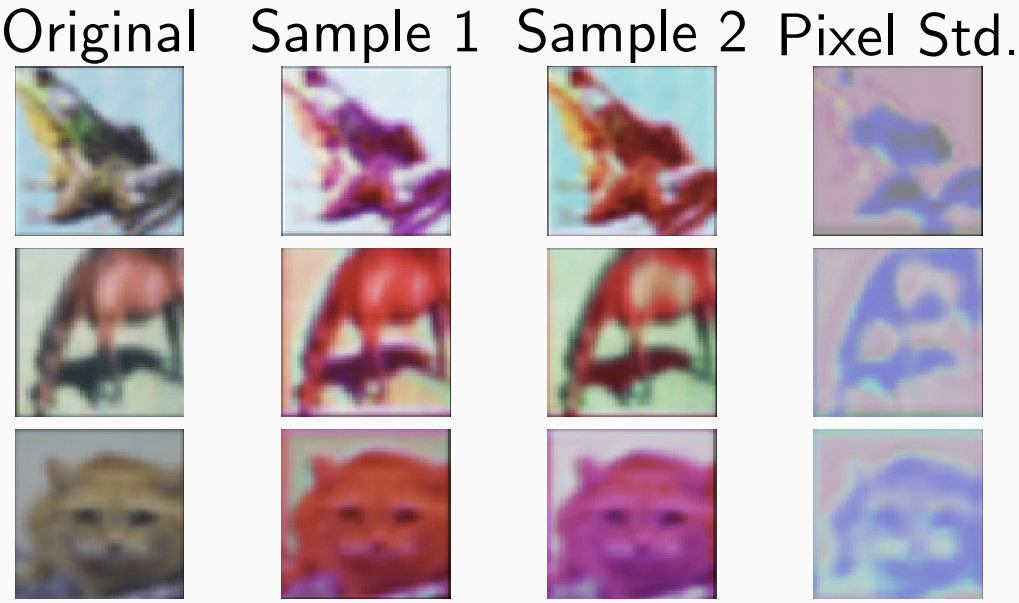


Figure 2: The original image is on the left, followed by two augmented samples and the standard deviation of the pixel intensities from the augmentation distribution.

The hyperparameters are weights in a U-Net [9], which learns a stochastic data augmentation: $\mathbf{x}' = U_{\boldsymbol{\lambda}}(\mathbf{x}, \epsilon), \epsilon \sim \mathcal{N}(0, I), \mathbf{x} \sim \mathcal{D}$.

IFT for meta-learning/pre-training [10]

- What if we want to tune pre-training parameters? Or have a meta-learning setup with more than 2 levels?
- Well, we can use the IFT for each level.

Hyperparameter Optimization through Hypernetworks [6, 7]

- Remember that:

$$\mathbf{w}^*(\boldsymbol{\lambda}) = \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w})$$

- Idea: approximate the response with a neural network. In this case, a hypernetwork with parameters ϕ:

$$\mathbf{w}^*(\boldsymbol{\lambda}) \approx \hat{\boldsymbol{w}}_\phi(\boldsymbol{\lambda})$$

1st Order Optimization in Differentiable Games

- The **direct gradient** is easy to compute.
- The gradient is difficult to compute because we may need the **Jacobian of the best-response**, which could require differentiating through optimization:

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\text{hyperparameter direct grad.}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparameter indirect grad.}}$$

1st Order Optimization in Differentiable Games

- The **direct gradient** is easy to compute.
- The gradient is difficult to compute because we may need the **Jacobian of the best-response**, which could require differentiating through optimization:

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\text{hyperparameter direct grad.}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparameter indirect grad.}}$$

$$\frac{\partial \mathcal{L}_V^*}{\partial \boldsymbol{\lambda}} = \left(\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}} \right) \Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})}$$

1st Order Optimization in Differentiable Games

- The **direct gradient** is often identically 0 for hyperparameter optimization.
- If the **direct gradient** available, we can simple use first-order methods.
- These can be much simpler to implement, compute, and analyze.
- Minimax games, $\mathcal{L}_A = -\mathcal{L}_B$, always have a **direct gradient** – ex., GANs.

Complex Momentum [11] - animation

Actual JAX implementation: changes in green

```
mass = .7 + .2j

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
  ...
```

- Gradient descent in differentiable games (like GANs) rotates around solutions.
- We solve this with a simple trick: complex momentum damps the oscillations.

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.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

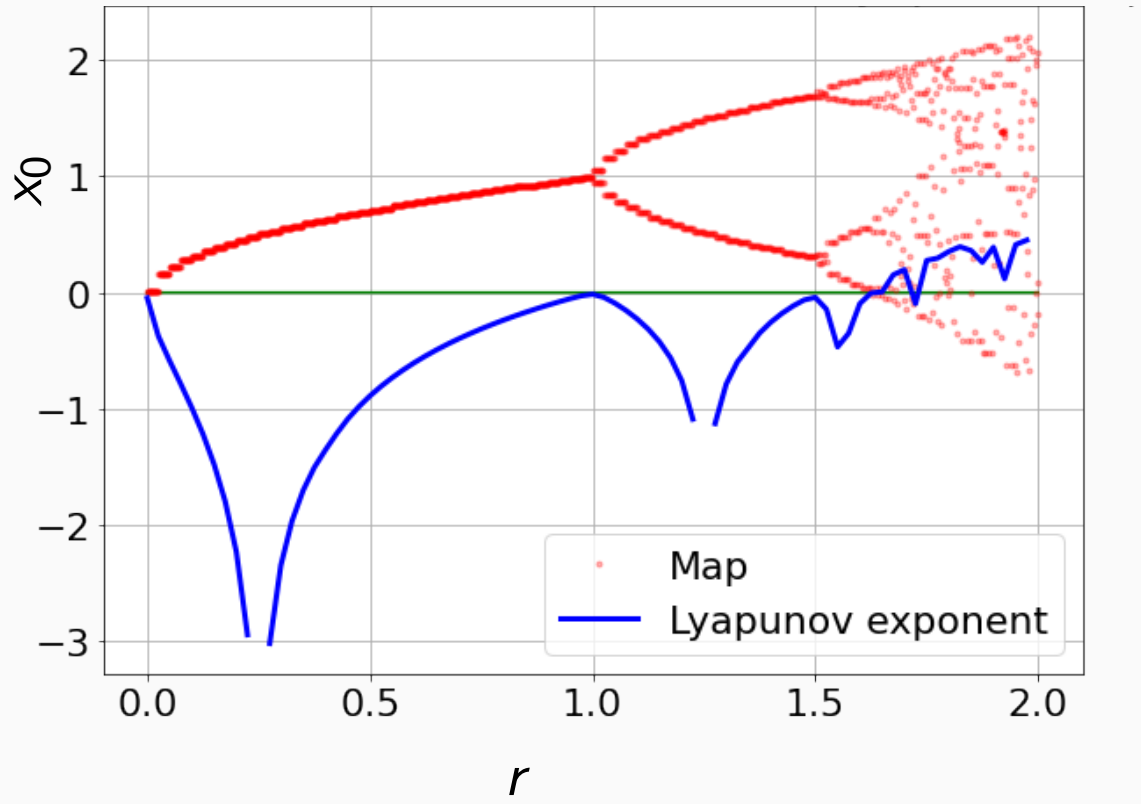
[10] Aniruddh Raghu, Jonathan Lorraine, Simon Kornblith, Matthew McDermott, and David K Duvenaud. Meta-learning to improve pre-training. *Advances in Neural Information*

Processing Systems, 34, 2021.

[11] Jonathan Lorraine, David Acuna, Paul Vicol, and David Duvenaud. Complex momentum for learning in games. *arXiv e-prints*, pages arXiv–2102, 2021.

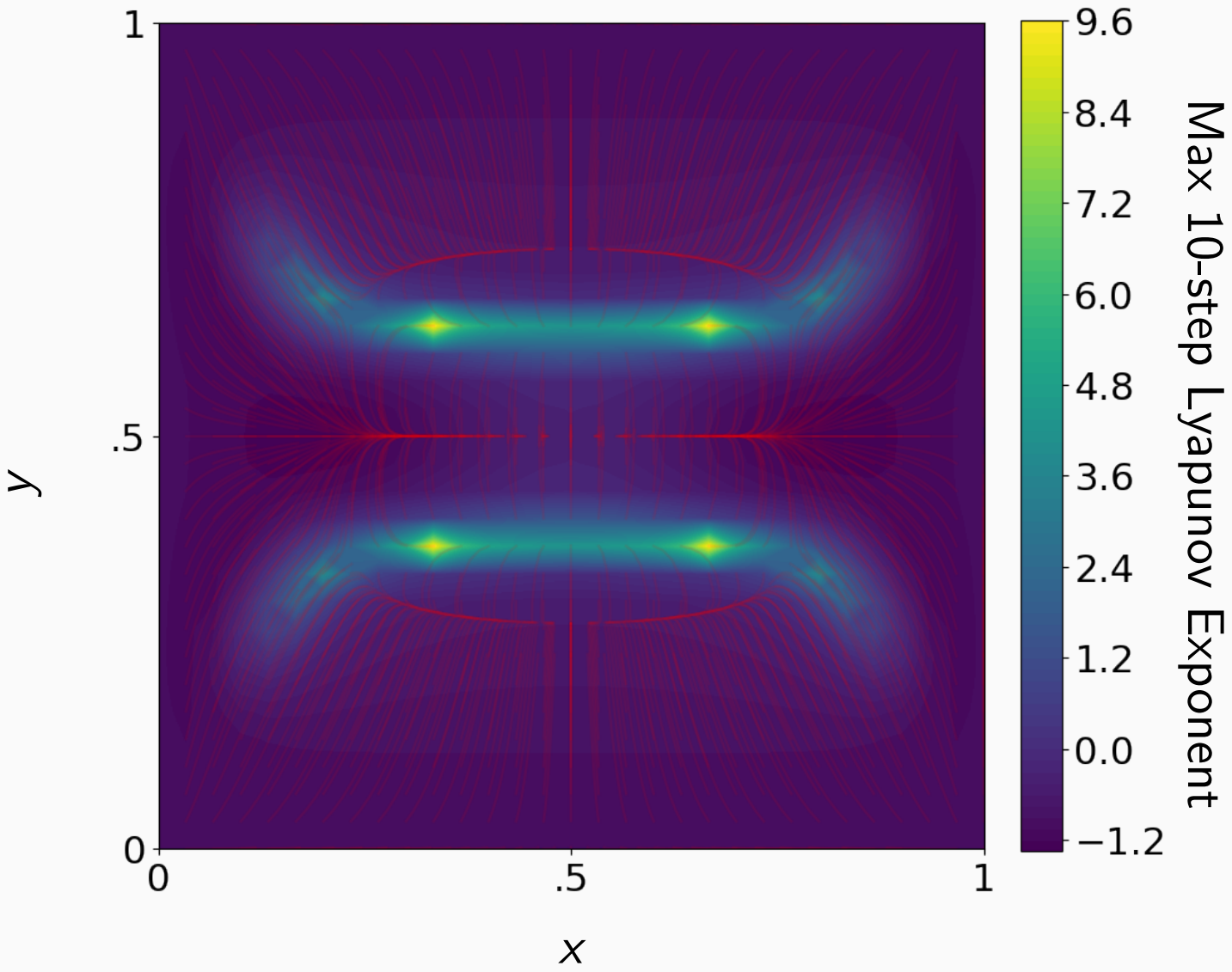
Extra Slides

The Logistic Map Example



- A canonical 1-dimensional example for bifurcations:
  $x(t+1) = x(t) + r + x(t)^2$

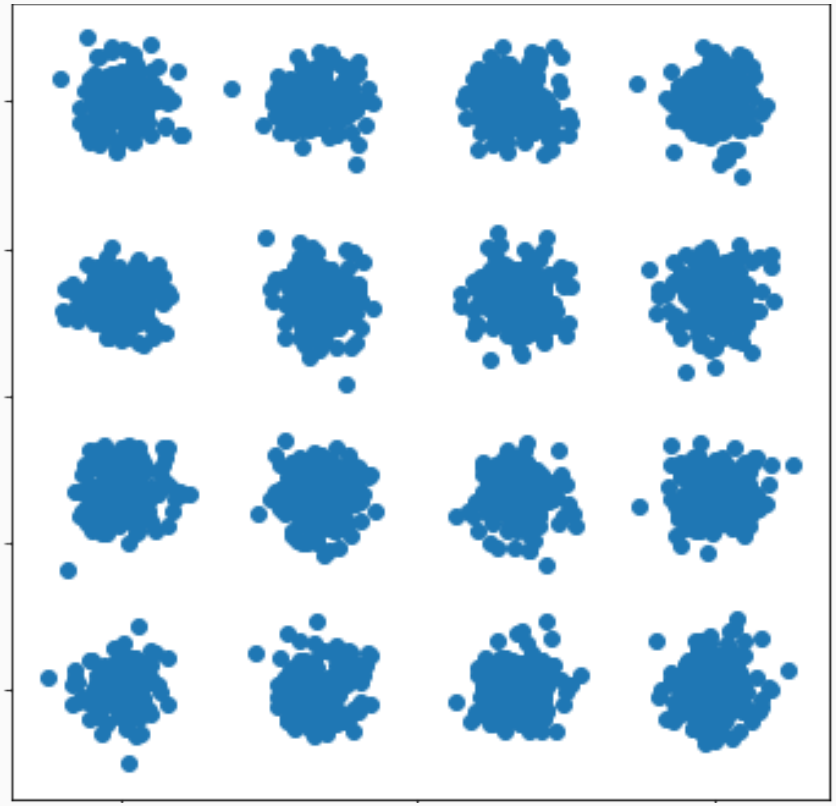
Single-objective Optimization Example



Branching Optimization Tree Search– RR/GRR changes in red

1: Select optimization parameters α
2: Find starting parameters $\boldsymbol{\omega}^{start} = \text{FindStartingPoint}(\alpha)$
3: Initialize a branch $\psi^{init} = \text{InitBranch}(\boldsymbol{\omega}^{start}, \alpha)$
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: $\quad \psi, \mathcal{B} = \text{ChooseBranch}(\mathcal{B})$
8: $\quad \boldsymbol{\omega}^* = \text{Optimize}(\psi.\boldsymbol{\omega}, \psi.\alpha)$ # Optimize our parameters
9: $\quad$ **if** $\text{VerifySolution}(\boldsymbol{\omega}^*)$ **then**
10: $\quad\quad \mathcal{S} = \mathcal{S} \cup \{\boldsymbol{\omega}^*\}$
11: $\quad$ Make new branch to split $\psi' = \text{copy}(\psi)$
12: $\quad$ Store the optimized parameters $\psi'.\textit{parameters} = \boldsymbol{\omega}^*$
13: $\quad$ **if** $\text{ContinueBranching}(\psi')$ **then**
14: $\quad\quad \mathcal{B} = \mathcal{B} \cup \text{SplitBranch}(\psi')$
15: **return** $\mathcal{S}$

GAN Samples



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