Model-Based RL and Policy Learning

CS 294-112: Deep Reinforcement Learning

Sergey Levine

Class Notes

1. Homework 3 due next Wednesday
2. Accept CMT peer review invitations
   - These are required (part of your final project grade)
   - If you have not received/cannot find invitation, email Kate Rakelly!
3. Project proposal feedback from TAs will be out shortly, please read it carefully!

Overview

1. Last time: learning models of system dynamics and using optimal control to choose actions
   - Global models and model-based RL
   - Local models and model-based RL with *constraints*
   - Uncertainty estimation
   - Models for complex observations, like images
2. What if we want a *policy*?
   - Much quicker to evaluate actions at runtime
   - Potentially better generalization
3. Can we just backpropagate into the policy?

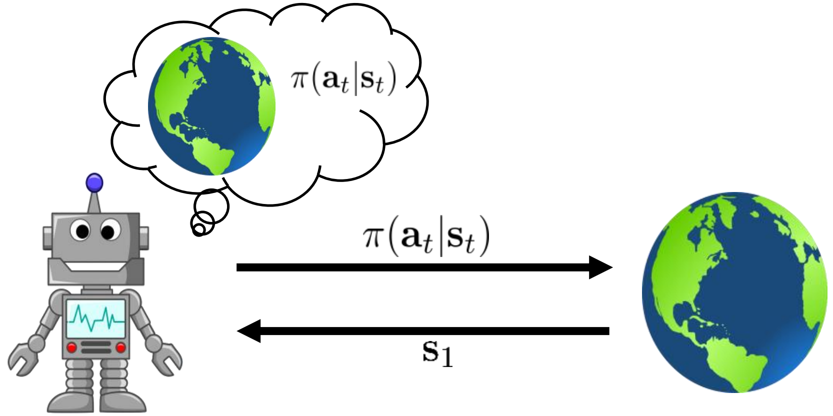
Today’s Lecture

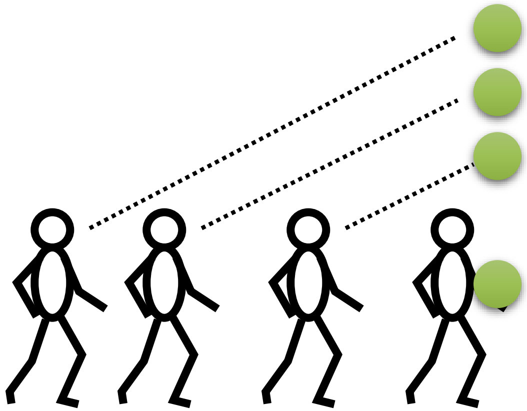
1. Backpropagating into a policy with learned models
2. How this becomes equivalent to *imitating* optimal control
3. The guided policy search algorithm
4. Imitating optimal control with DAgger
5. Model-based vs. model-free RL tradeoffs

- Goals
  - Understand how to train policies guided by control/planning
  - Understand tradeoffs between various methods
  - Get a high-level overview of recent research work on policy learning with model-based RL

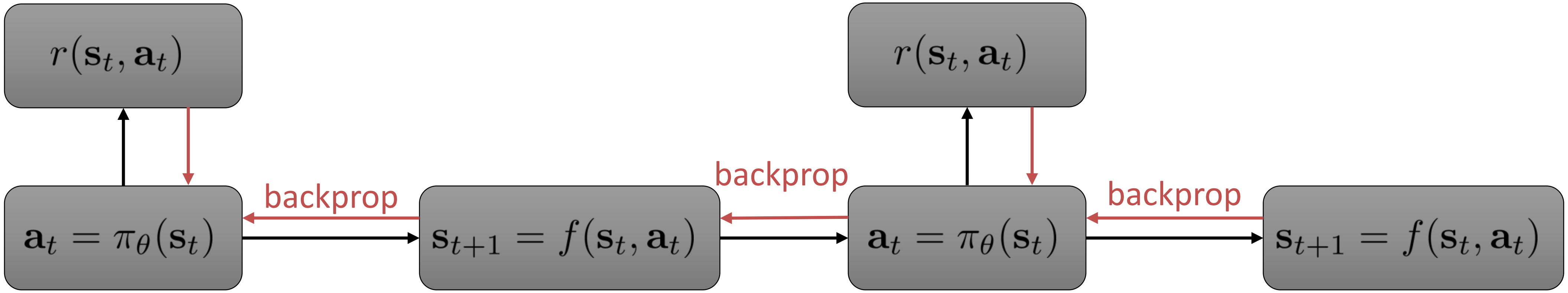
So how can we train policies?

- So far we saw how we can…
  - Train global models (e.g. GPs, neural networks)
  - Train local models (e.g. linear models)
- But what if we want a policy?
  - Don't need to replan (faster)
  - Potentially better generalization
  - Closed loop control!





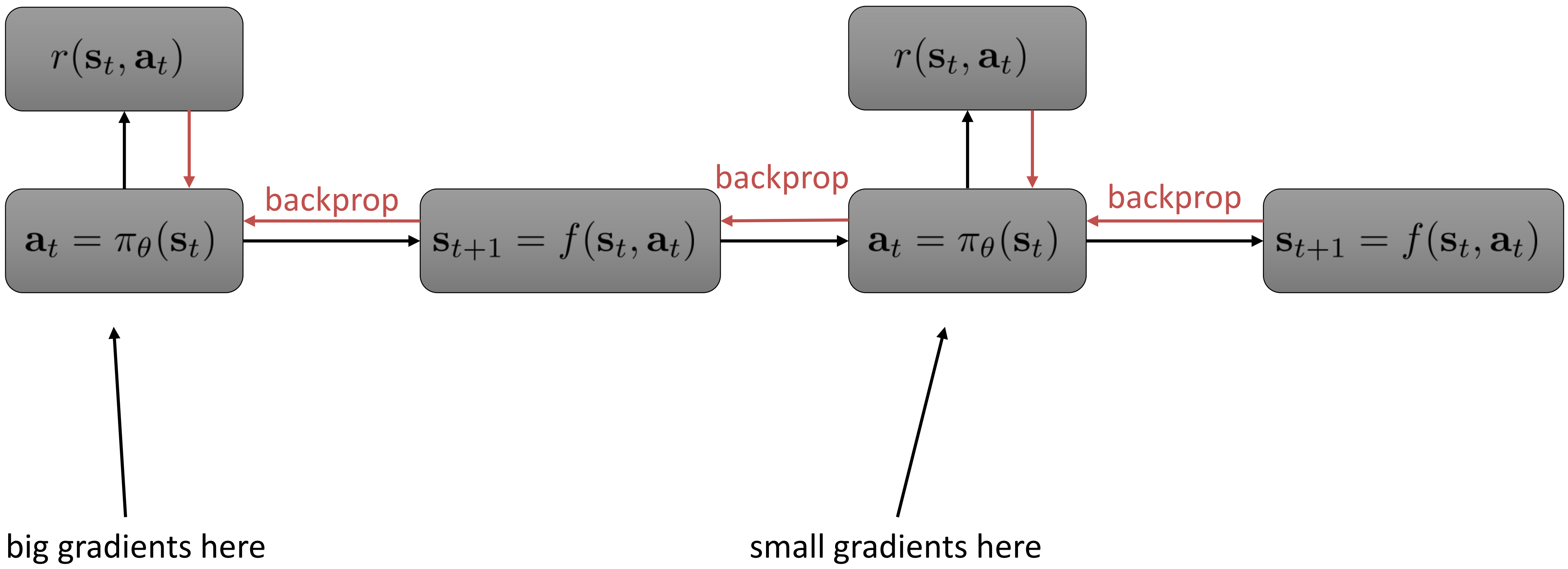
Backpropagate directly into the policy?



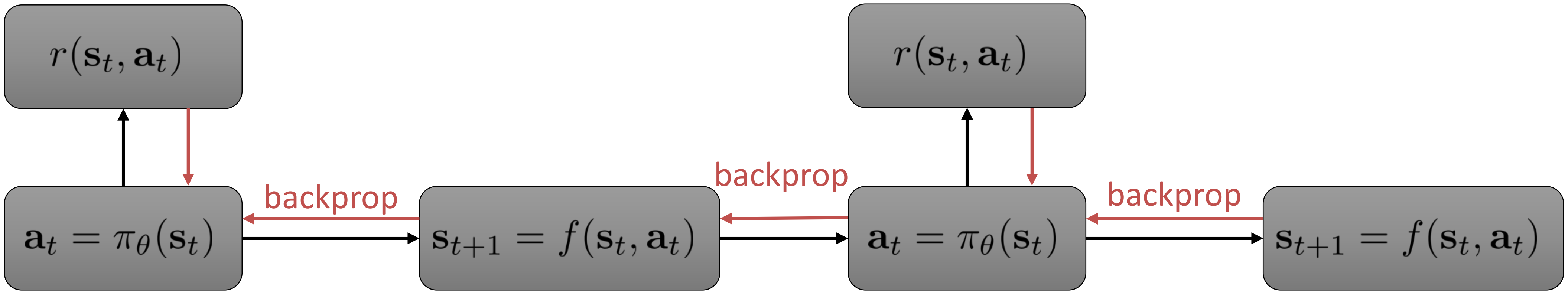
model-based reinforcement learning version 2.0:

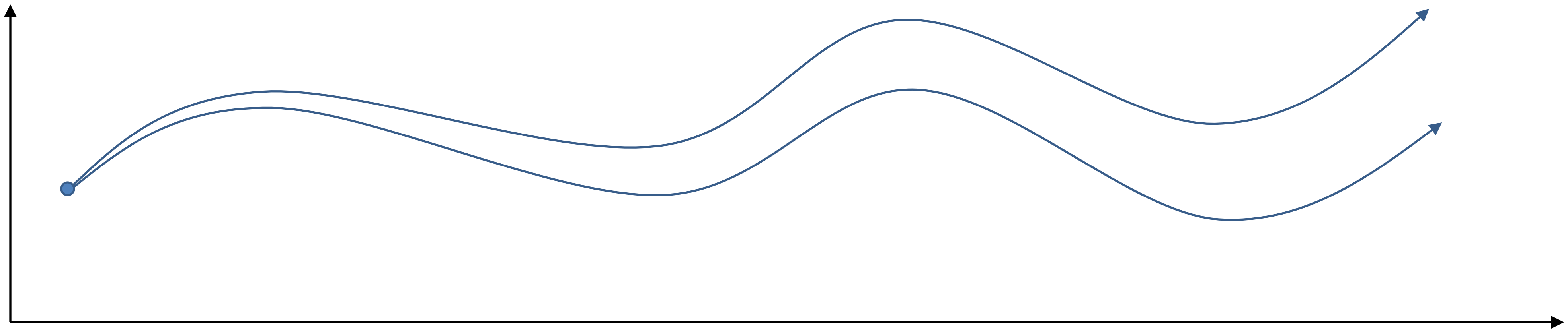
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. backpropagate through $f(\mathbf{s}, \mathbf{a})$ into the policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

What's the problem with backprop into policy?

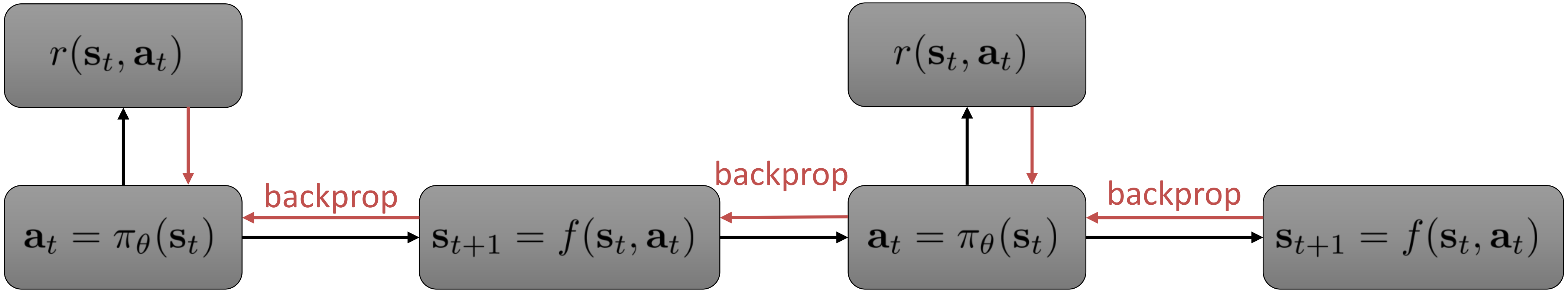


What’s the problem?





What's the problem?

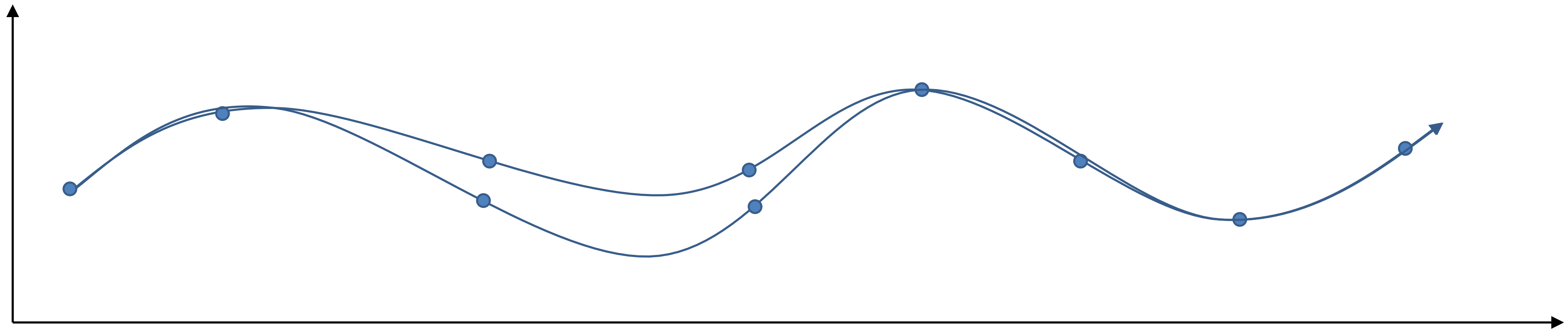


- Similar parameter sensitivity problems as shooting methods
  - But no longer have convenient second order LQR-like method, because policy parameters couple all the time steps, so no dynamic programming
- Similar problems to training long RNNs with BPTT
  - Vanishing and exploding gradients
  - Unlike LSTM, we can't just "choose" a simple dynamics, dynamics are chosen by nature

What's the problem?

- What about collocation methods?

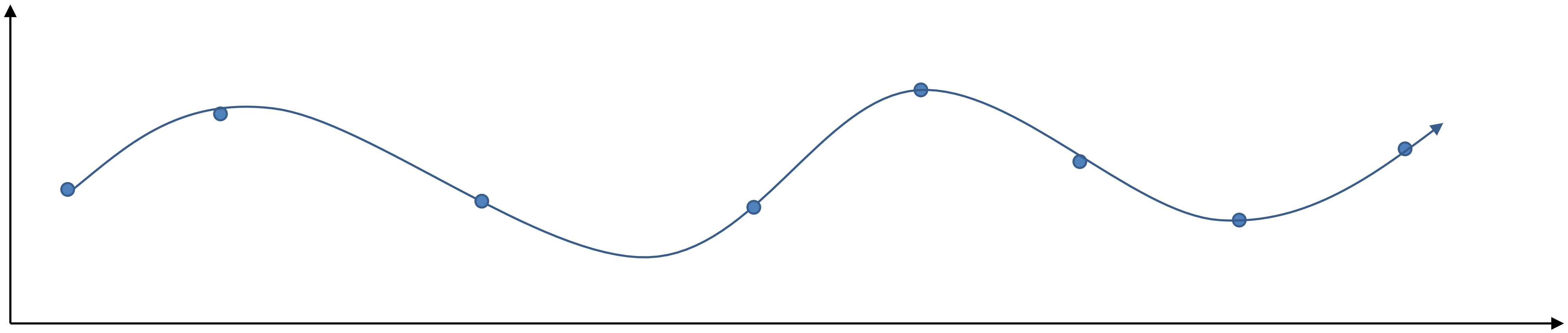
$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$



What's the problem?

- What about collocation methods?

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T,\theta} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}), \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$



Even simpler…

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T,\mathbf{x}_1,\dots,\mathbf{x}_T,\theta} \sum_{t=1}^{T} c(\mathbf{x}_t,\mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1},\mathbf{u}_{t-1})$$

generic trajectory optimization, solve however you want

$$\text{s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

- How can we impose constraints on trajectory optimization?

Review: dual gradient descent

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$g(\lambda) = \mathcal{L}(\mathbf{x}^\star(\lambda), \lambda)$$

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$$

$$\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$$

1. Find $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda)$
2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(\mathbf{x}^\star, \lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

A small tweak to DGD: augmented Lagrangian

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } C(\mathbf{x}) = 0$$

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x})$$

$$\bar{\mathcal{L}}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda C(\mathbf{x}) + \rho \|C(\mathbf{x})\|^2$$

- Still converges to correct solution
- When far from solution, quadratic term tends to improve stability
- Closely related to alternating direction method of multipliers (ADMM)

1. Find $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x}} \bar{\mathcal{L}}(\mathbf{x}, \lambda)$
2. Compute $\frac{dg}{d\lambda} = \frac{d\bar{\mathcal{L}}}{d\lambda}(\mathbf{x}^\star, \lambda)$
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

Constraining trajectory optimization with dual gradient descent

$$\min_{\tau,\theta} c(\tau) \text{ s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

$$\mathcal{L}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)$$

$$\bar{\mathcal{L}}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t) + \sum_{t=1}^{T} \rho_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)^2$$

Constraining trajectory optimization with dual gradient descent

$$\min_{\tau,\theta} c(\tau) \text{ s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

$$\bar{\mathcal{L}}(\tau,\theta,\lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t) + \sum_{t=1}^{T} \rho_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)^2$$

1. Find $\tau \leftarrow \arg\min_\tau \bar{\mathcal{L}}(\tau,\theta,\lambda)$ (e.g. via iLQR)
2. Find $\theta \leftarrow \arg\min_\theta \bar{\mathcal{L}}(\tau,\theta,\lambda)$ (e.g. via SGD)
3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

Guided policy search discussion

1. Find $\tau \leftarrow \arg\min_\tau \bar{\mathcal{L}}(\tau, \theta, \lambda)$ (e.g. via iLQR)

2. Find $\theta \leftarrow \arg\min_\theta \bar{\mathcal{L}}(\tau, \theta, \lambda)$ (e.g. via SGD)

3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$

- Can be interpreted as *constrained* trajectory optimization method
- Can be interpreted as imitation of an optimal control expert, since step 2 is just supervised learning
- The optimal control "teacher" adapts to the learner, and avoids actions that the learner can't mimic

General guided policy search scheme

1. Optimize $p(\tau)$ with respect to some surrogate $\tilde{c}(\mathbf{x}_t, \mathbf{u}_t)$
2. Optimize θ with respect to some supervised objective
3. Increment or modify dual variables λ

Need to choose:

form of $p(\tau)$ or τ (if deterministic)

optimization method for $p(\tau)$ or τ

surrogate $\tilde{c}(\mathbf{x}_t, \mathbf{u}_t)$

supervised objective for $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

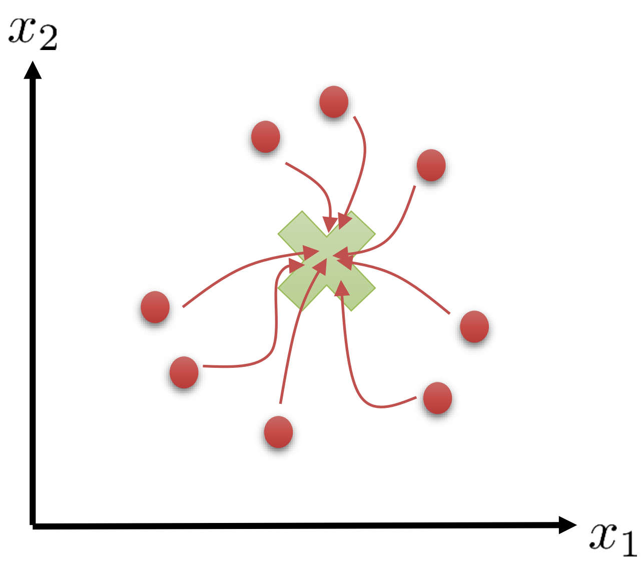
Deterministic case

$$\min_{\tau,\theta} c(\tau) \text{ s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

$$\bar{\mathcal{L}}(\tau,\theta,\lambda) = \underbrace{c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t) + \sum_{t=1}^{T} \rho_t(\pi_\theta(\mathbf{x}_t) - \mathbf{u}_t)^2}_{\tilde{c}(\tau)}$$

1. Optimize τ with respect to surrogate $\tilde{c}(\tau)$
2. Optimize θ with respect to supervised objective
3. Increment or modify dual variables λ

Learning with multiple trajectories



$$\min_{\tau_1,\ldots,\tau_N,\theta} \sum_{i=1}^{N} c(\tau_i) \text{ s.t. } \mathbf{u}_{t,i} = \pi_\theta(\mathbf{x}_{t,i})\ \forall i\ \forall t$$

1. Optimize each τ_i *in parallel* with respect to $\tilde{c}(\tau_i)$
2. Optimize θ with respect to supervised objective
3. Increment or modify dual variables λ

Case study: learning locomotion skills

Interactive Control of Diverse Complex Characters with Neural Networks

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, Emanuel Todorov
Department of Computer Science, University of Washington
{mordatch,lowrey,galen,zoran,todorov}@cs.washington.edu

Interactive Control of Diverse Complex Characters with Neural Networks

Submitted to NIPS 2015

Stochastic (Gaussian) GPS

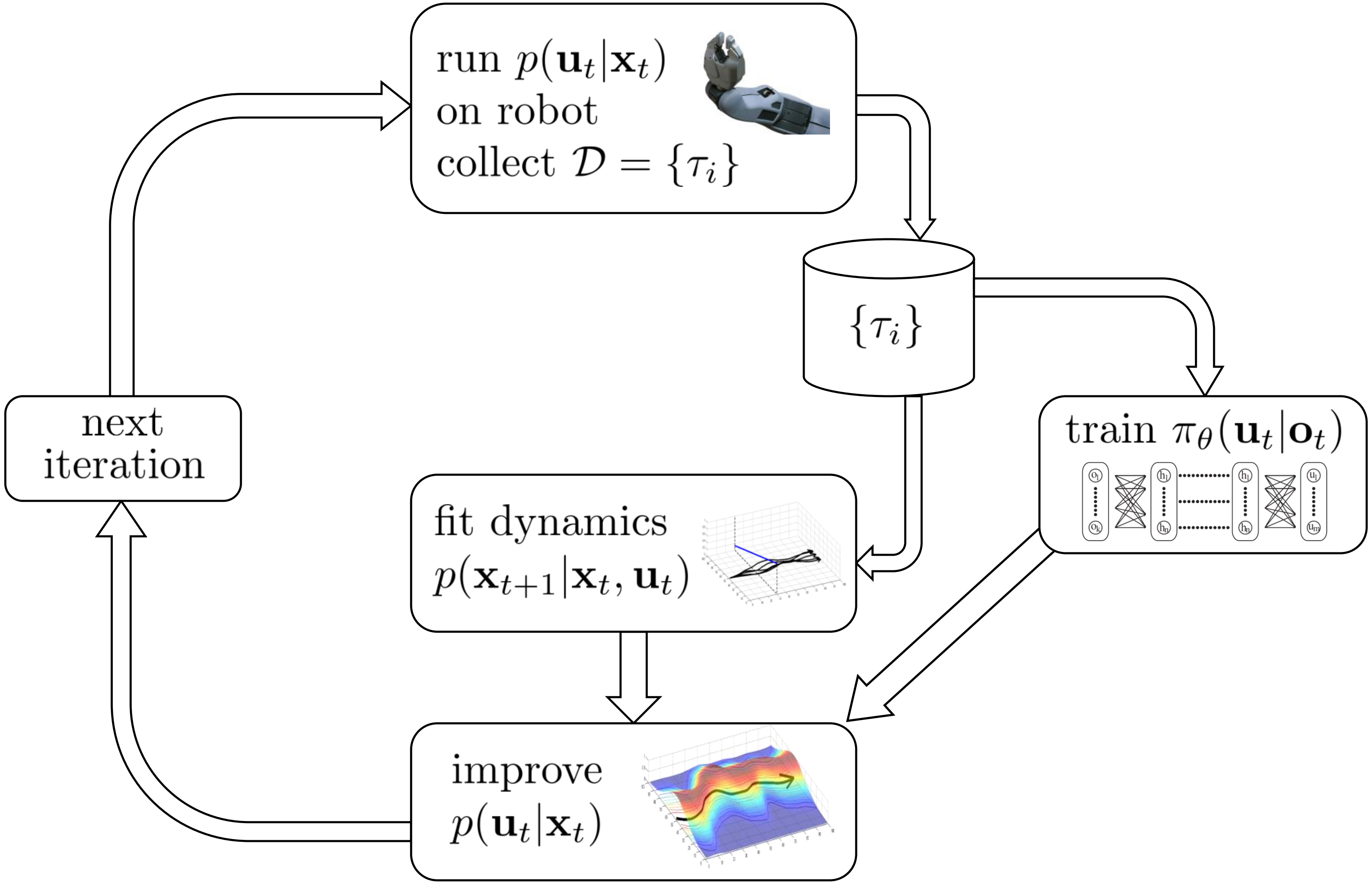
$$\min_{p,\theta} E_{\tau \sim p(\tau)}[c(\tau)] \text{ s.t. } p(\mathbf{u}_t|\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$$

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

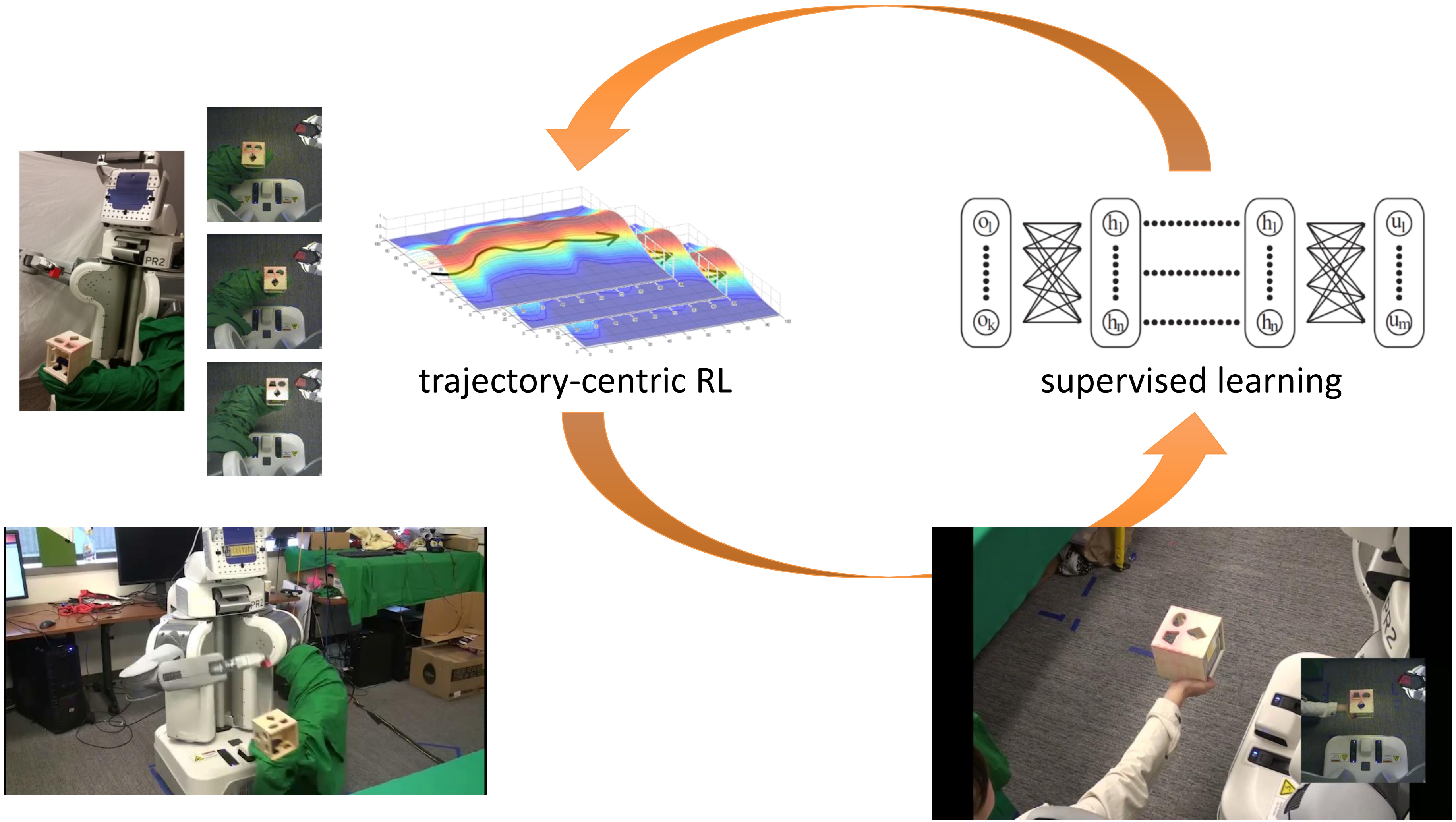
$$\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\tilde{c}(\mathbf{x}_t,\mathbf{u}_t)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon$$

1. Optimize $p(\tau)$ with respect to some surrogate $\tilde{c}(\mathbf{x}_t, \mathbf{u}_t)$
2. Optimize θ with respect to some supervised objective
3. Increment or modify dual variables λ

Stochastic (Gaussian) GPS with local models



Robotics Example

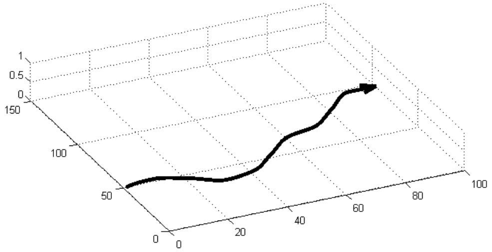


Input Remapping Trick

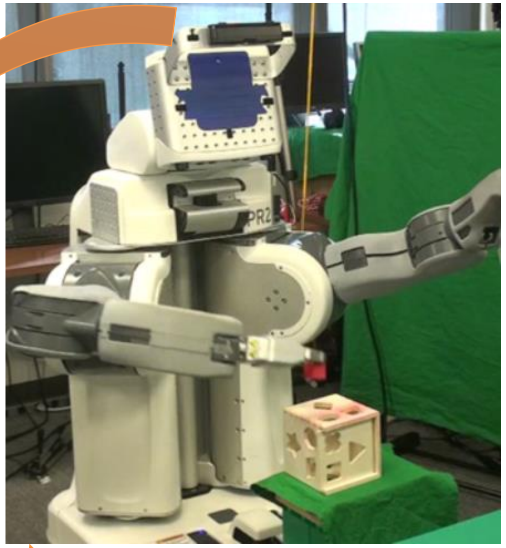
$$\min_{p,\theta} E_{\tau \sim p(\tau)}[c(\tau)] \text{ s.t. } p(\mathbf{u}_t|\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\cancel{\mathbf{x}}\mathbf{o}_t)$$

training time

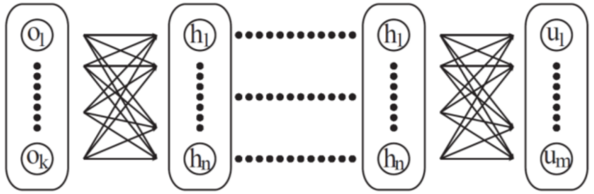
$$\mathbf{x}_t \rightarrow \mathbf{u}_t$$



test time



$$\mathbf{o}_t \rightarrow \mathbf{u}_t$$



Case study: vision-based control with GPS

“End-to-End Training of Deep Visuomotor Policies”

Break

Imitating optimal control with DAgger

Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning

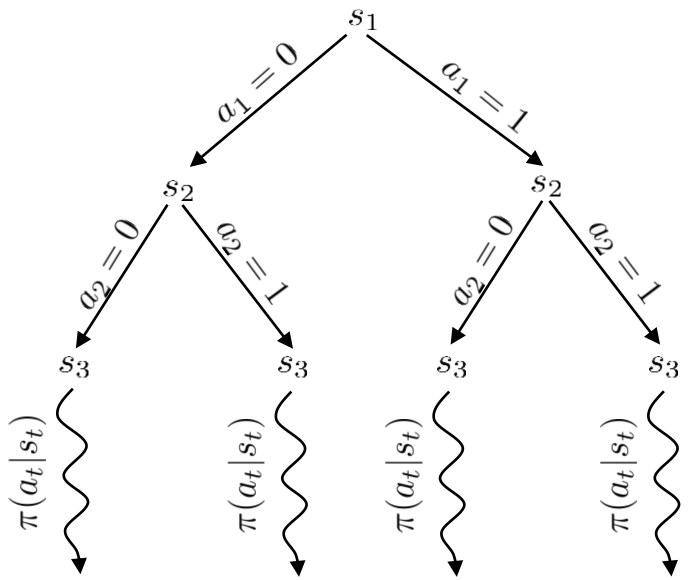
Xiaoxiao Guo
Computer Science and Eng.
University of Michigan
guoxiao@umich.edu

Satinder Singh
Computer Science and Eng.
University of Michigan
baveja@umich.edu

Honglak Lee
Computer Science and Eng.
University of Michigan
honglak@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Xiaoshi Wang
Computer Science and Eng.
University of Michigan
xiaoshiw@umich.edu

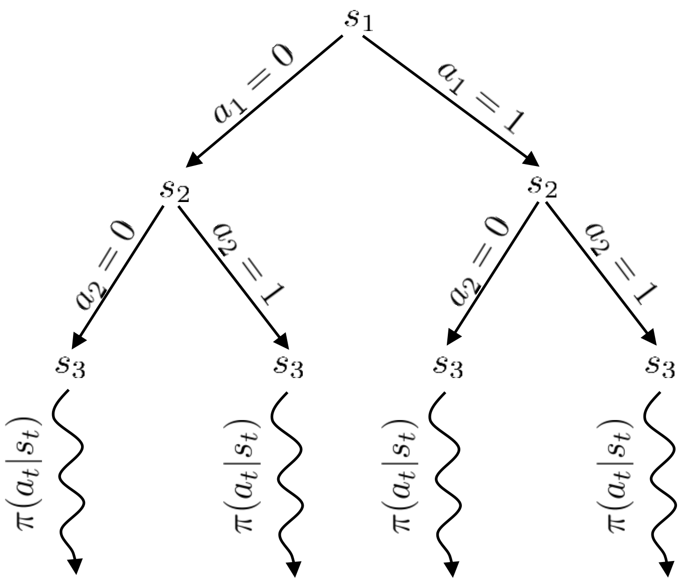


Imitating optimal control with DAgger

Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning

every N steps

1. from current state s_t, run MCTS to get $a_t, a_{t+1}, \ldots$
2. add (s_t, a_t) to dataset $\mathcal{D}$
3. execute action $a_t \sim \pi(a_t|s_t)$ (*not* MCTS action!)
4. update the policy by training on $\mathcal{D}$



A problem with DAgger

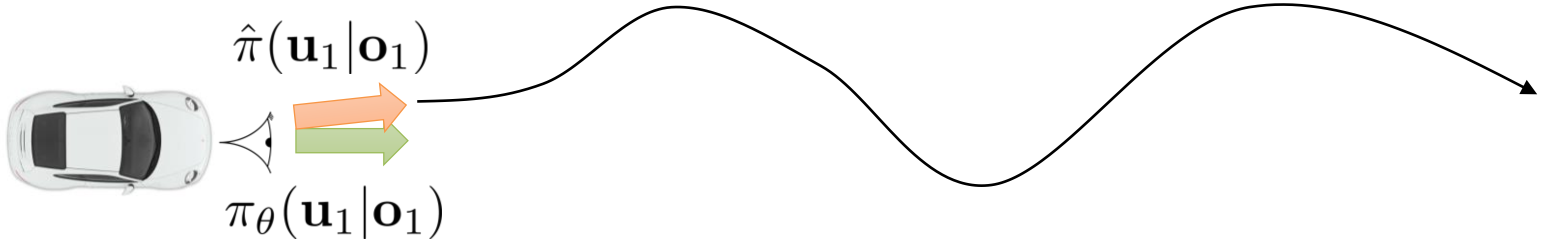
1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Imitating MPC: PLATO algorithm

1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\hat{\pi}(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask computer to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

simple stochastic policy: $\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_{\mathbf{u}_t})$

$$\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(\mathbf{x}_{t'}, \mathbf{u}_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) \| \pi_\theta(\mathbf{u}_t|\mathbf{o}_t))$$



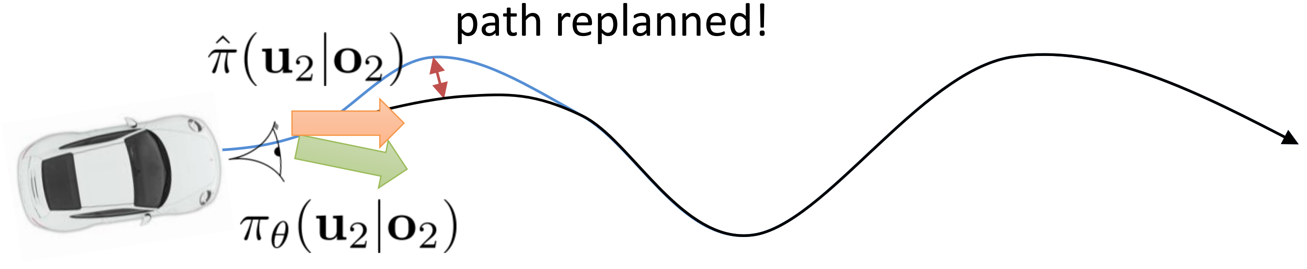
Kahn, Zhang, Levine, Abbeel ‘16

Imitating MPC: PLATO algorithm

1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\hat{\pi}(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask computer to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

simple stochastic policy: $\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_{\mathbf{u}_t})$

$$\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(\mathbf{x}_{t'}, \mathbf{u}_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) \| \pi_\theta(\mathbf{u}_t|\mathbf{o}_t))$$

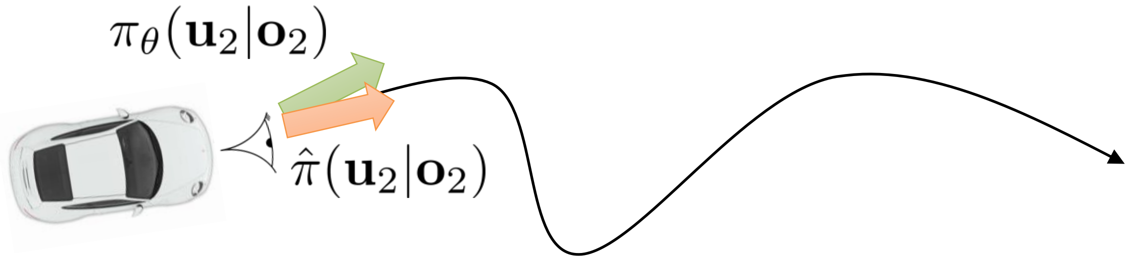


Imitating MPC: PLATO algorithm

1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\hat{\pi}(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask computer to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

simple stochastic policy: $\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_{\mathbf{u}_t})$

$$\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(\mathbf{x}_{t'}, \mathbf{u}_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) \| \pi_\theta(\mathbf{u}_t|\mathbf{o}_t))$$

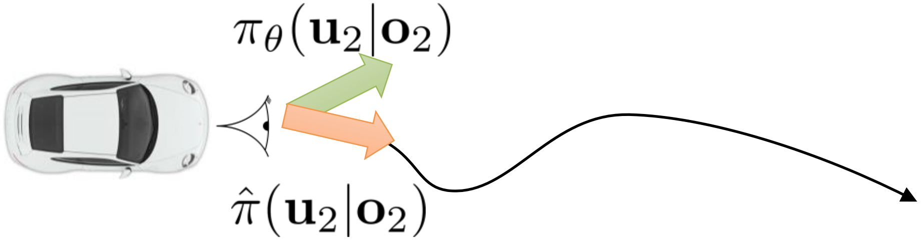


Imitating MPC: PLATO algorithm

1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\hat{\pi}(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask computer to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

simple stochastic policy: $\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_{\mathbf{u}_t})$

$$\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(\mathbf{x}_{t'}, \mathbf{u}_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) \| \pi_\theta(\mathbf{u}_t|\mathbf{o}_t))$$

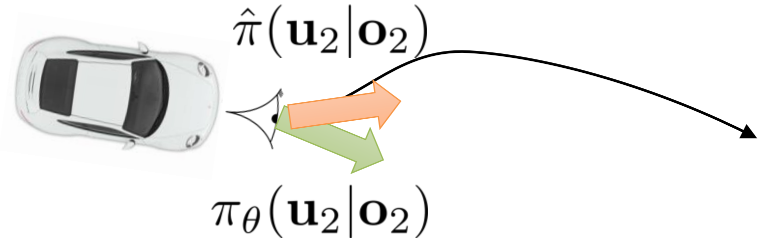


Imitating MPC: PLATO algorithm

1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\hat{\pi}(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask computer to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

simple stochastic policy: $\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t, \Sigma_{\mathbf{u}_t})$

$$\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(\mathbf{x}_{t'}, \mathbf{u}_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) \| \pi_\theta(\mathbf{u}_t|\mathbf{o}_t))$$

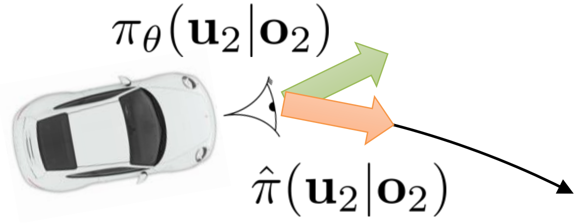


Imitating MPC: PLATO algorithm

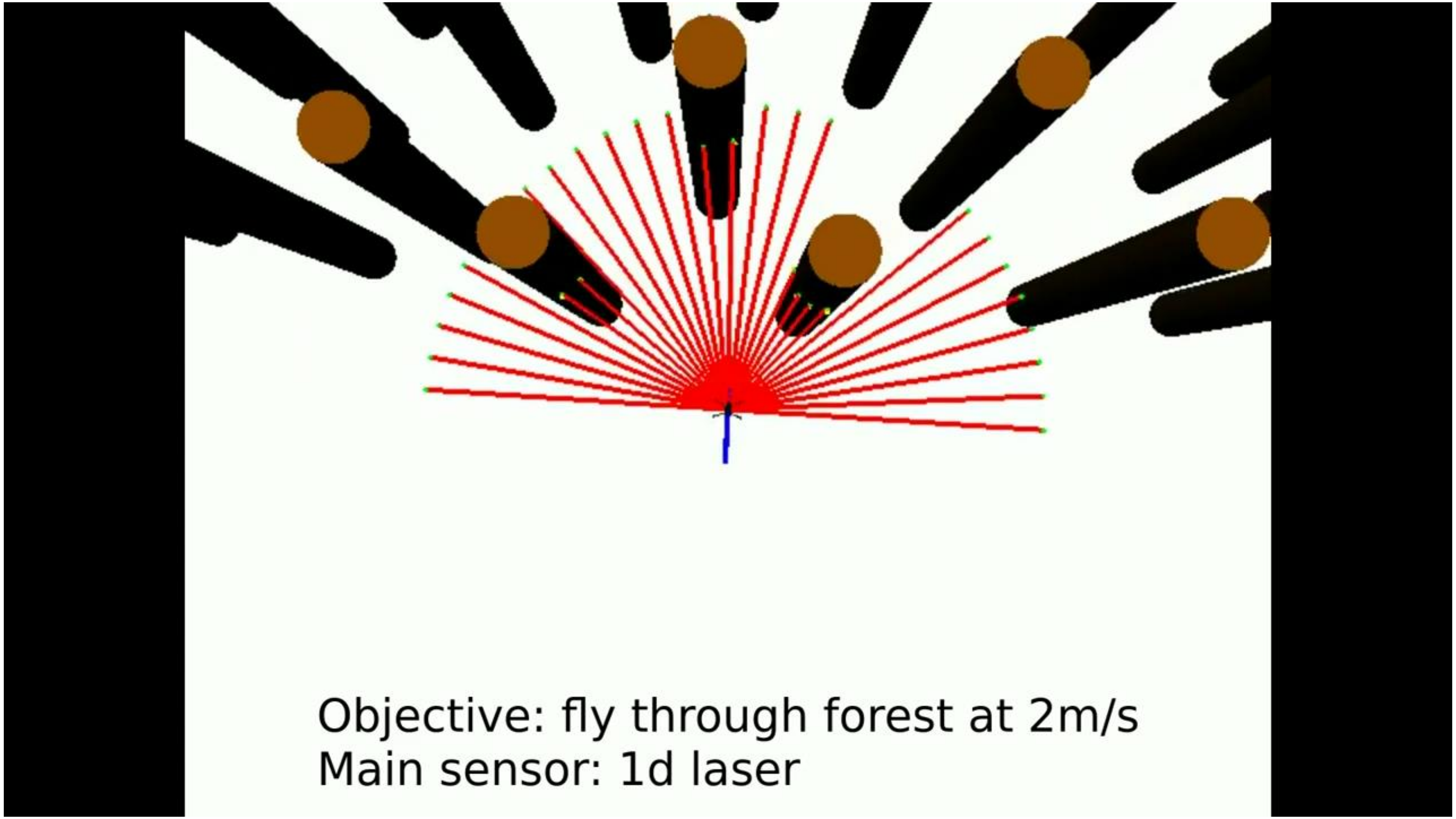
1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\hat{\pi}(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask computer to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

simple stochastic policy: $\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_{\mathbf{u}_t})$

$$\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(\mathbf{x}_{t'}, \mathbf{u}_{t'})] + \lambda D_{\mathrm{KL}}(\hat{\pi}(\mathbf{u}_t|\mathbf{x}_t) \| \pi_\theta(\mathbf{u}_t|\mathbf{o}_t))$$



Imitating MPC: PLATO algorithm



DAgger vs GPS

- DAgger does not require an adaptive expert
  - Any expert will do, so long as states from learned policy can be labeled
  - Assumes it is possible to match expert’s behavior up to bounded loss
    - Not always possible (e.g. partially observed domains)
- GPS adapts the “expert” behavior
  - Does not require bounded loss on initial expert (expert will change)

Why imitate?

- Relatively stable and easy to use
  - Supervised learning works very well
  - Control/planning (usually) works very well
  - The combination of the two (usually) works very well
- Input remapping trick: can exploit availability of additional information at training time to learn policy from raw observations
- Overcomes optimization challenges of backpropagating into policy directly
- Usually sample-efficient and viable for real physical systems

Model-free optimization with a model

- Just use policy gradient (or another model-free RL method) even though you have a model
- Sometimes better than using the gradients!
- See a recent analysis here:
  - Parmas et al. ‘18: PIPP: Flexible Model-Based Policy Search Robust to the Curse of Chaos

Model-free optimization with a model

Dyna

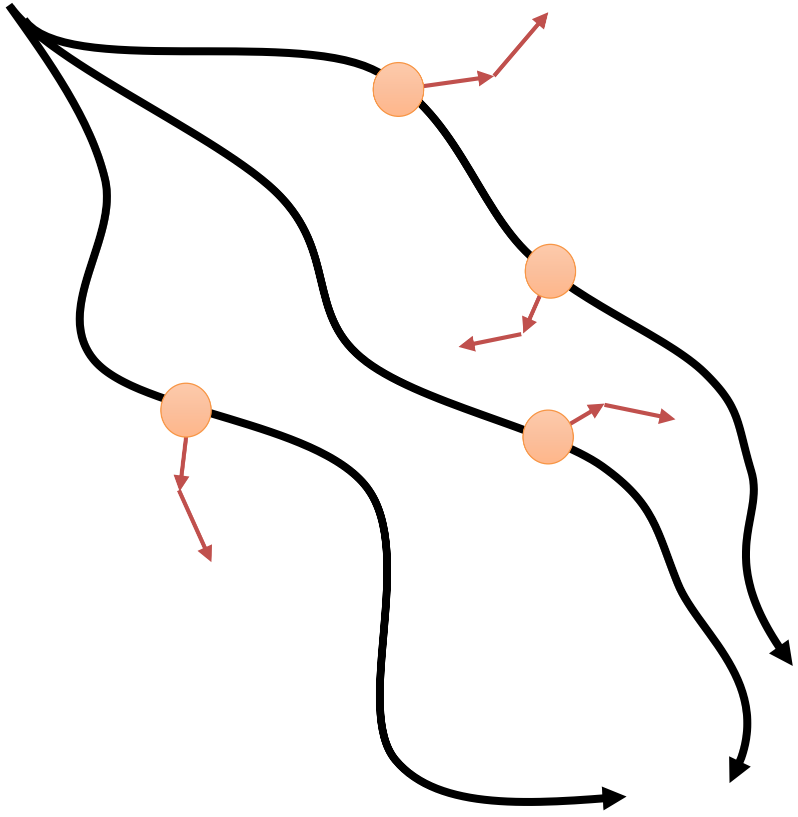
online Q-learning algorithm that performs model-free RL with a model

1. given state s, pick action a using exploration policy
2. observe s' and r, to get transition (s, a, s', r)
3. update model $\hat{p}(s'|s, a)$ and $\hat{r}(s, a)$ using (s, a, s')
4. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$
5. repeat K times:
   6. sample $(s, a) \sim \mathcal{B}$ from buffer of past states and actions
   7. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming.

General "Dyna-style" model-based RL recipe

1. collect some data, consisting of transitions (s, a, s', r)
2. learn model $\hat{p}(s'|s, a)$ (and optionally, $\hat{r}(s, a)$)
3. repeat K times:
   4. sample $s \sim \mathcal{B}$ from buffer
   5. choose action a (from $\mathcal{B}$, from π, or random)
   6. simulate $s' \sim \hat{p}(s'|s, a)$ (and $r = \hat{r}(s, a)$)
   7. train on (s, a, s', r) with model-free RL
   8. (optional) take N more model-based steps

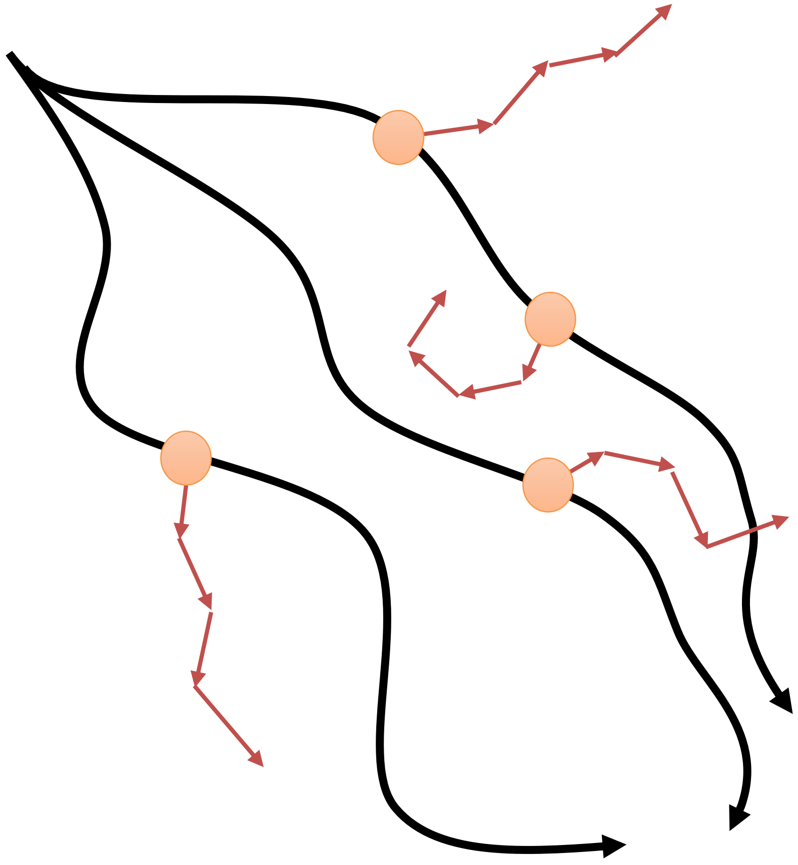


+ only requires short (as few as one step) rollouts from model

+ still sees diverse states

Model-Based Acceleration (MBA) & Model-Based Value Expansion (MVE)

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. use $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j\}$ to update model $\hat{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$
4. sample $\{\mathbf{s}_j\}$ from $\mathcal{B}$
5. for each $\mathbf{s}_j$, perform model-based rollout with $\mathbf{a} = \pi(\mathbf{s})$
6. use all transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ along rollout to update Q-function



Gu et al. Continuous deep Q-learning with model-based acceleration. '16
Feinberg et al. Model-based value expansion. '18

Model-based RL algorithms summary

THIS WILL BE ON HW4!

- Learn model and plan (without policy)
  - Iteratively collect more data to overcome distribution mismatch
  - Replan every time step (MPC) to mitigate small model errors
- Learn policy
  - Backpropagate into policy (e.g., PILCO) – simple but potentially unstable
  - Imitate optimal control in a constrained optimization framework (e.g., GPS)
  - Imitate optimal control via DAgger-like process (e.g., PLATO)
  - Use model-free algorithm with a model (Dyna, etc.)

Limitations of model-based RL

- Need some kind of model
  - Not always available
  - Sometimes harder to learn than the policy
- Learning the model takes time & data
  - Sometimes expressive model classes (neural nets) are not fast
  - Sometimes fast model classes (linear models) are not expressive
- Some kind of additional assumptions
  - Linearizability/continuity
  - Ability to reset the system (for local linear models)
  - Smoothness (for GP-style global models)
  - Etc.

So… which algorithm do I use?

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, SAC, etc.)

10x

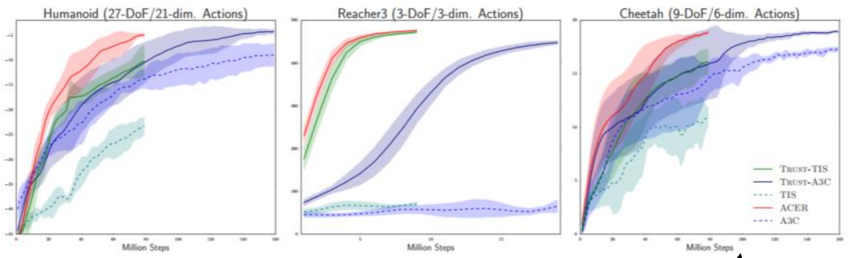
model-based deep RL
(e.g. PETS, guided policy search)

10x

model-based "shallow" RL
(e.g. PILCO)

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

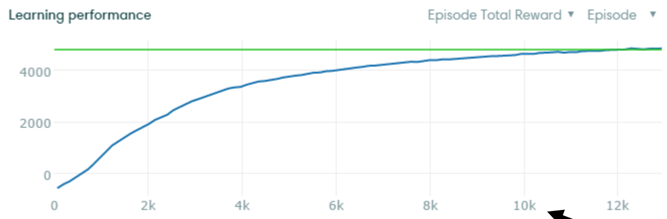
Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]



Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

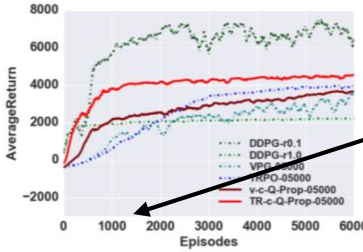
half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

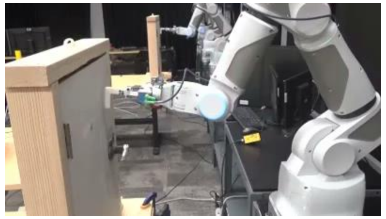
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

half-cheetah

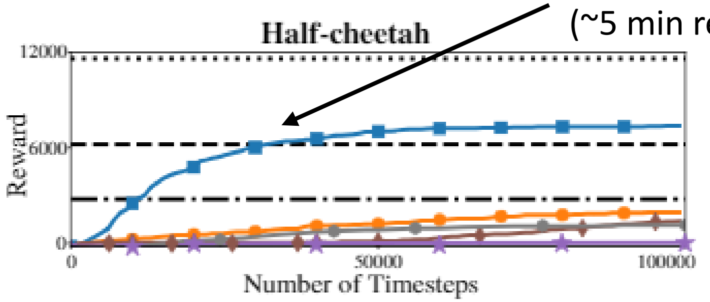


Gu et al. '16

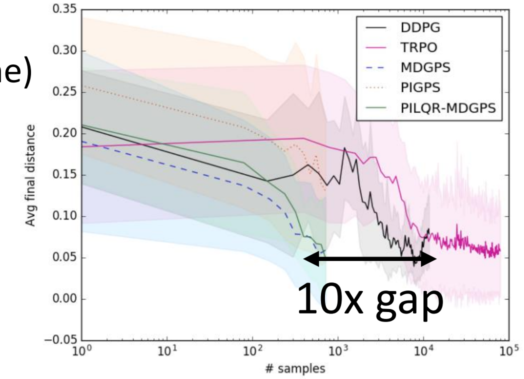
1,000,000 steps
(1,000 episodes)
(~3 hours real time)



30,000 steps
(30 episodes)
(~5 min real time)



Chua et a. '18: Deep Reinforcement Learning in a Handful of Trials



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

Which RL algorithm to use?

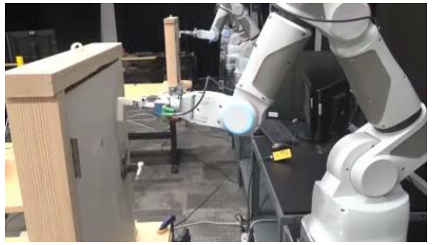
are you learning in a simulator?

no → how patient are you?

- impatient → model-based RL (GPS, PETS)
- I can wait → DDPG, NAF, SQL, SAC, TD3

yes → is simulation cost negligible compared to training cost?

- no → DDPG, NAF, SQL, SAC, TD3
- yes → TRPO, PPO, A3C



Lillicrap et al. "Continuous control…"
Gu et al. "Continuous deep Q-learning…"
Haarnoja et al. "Reinforcement learning with deep energy-based…"
Haarnoja et al. "Soft actor-critic"
Fujimoto et al. "Addressing function approximation error…"

BUT: if you have a simulator, you can compute gradients through it – do you need model-free RL?