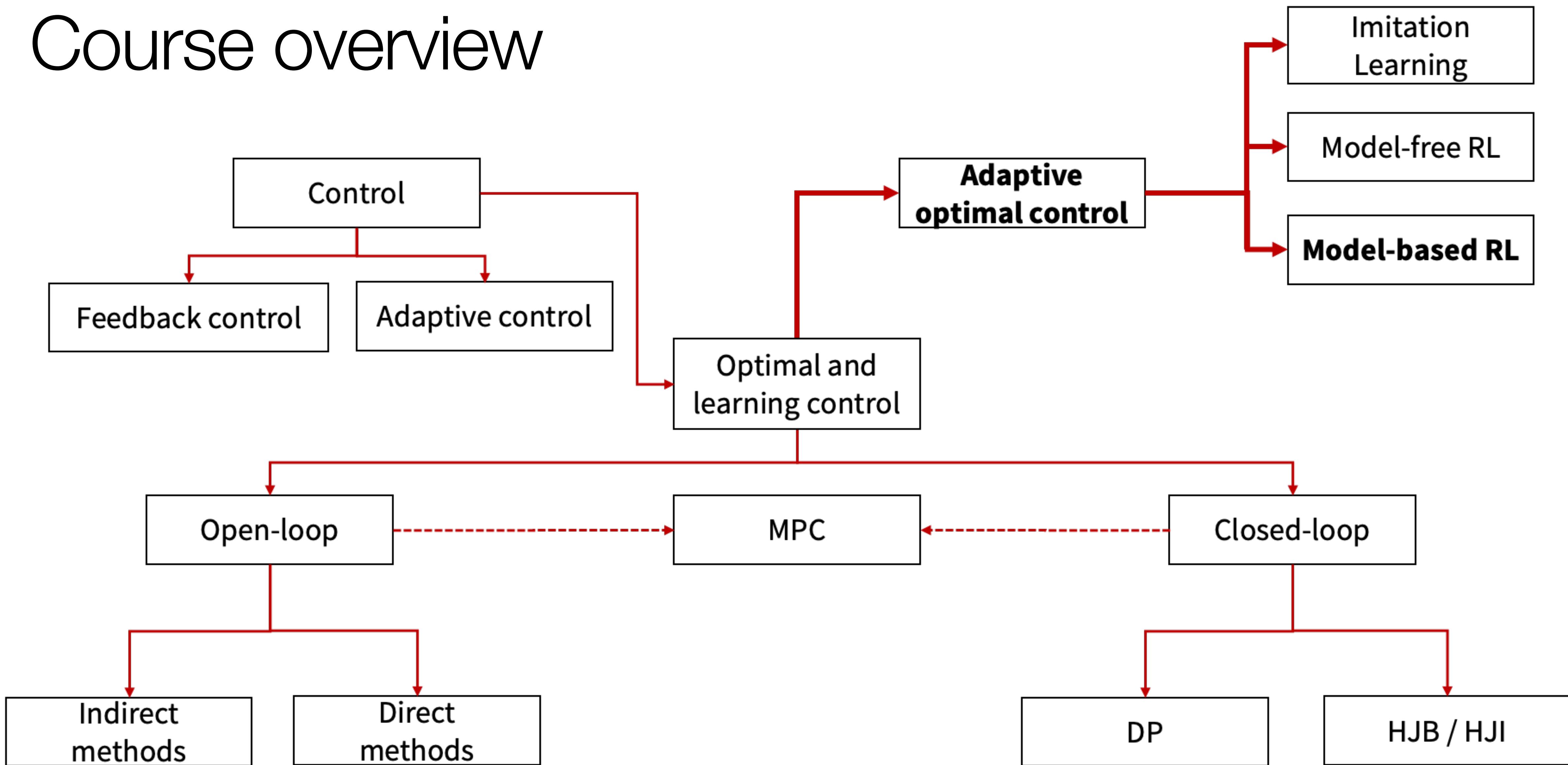


AA203
Optimal and Learning-based Control
Model-based Reinforcement Learning

Course overview



Last time: Outline

Intro to policy gradients

- REINFORCE algorithm
- Reducing variance of policy gradient

Actor-Critic methods

- Advantage
- Architecture design

Deep RL Algorithms & Applications

Reducing RL to optimization

- Much of modern ML entails reducing learning to a numerical optimization problem
 - Supervised learning as *training error minimization*
- This is different from what we have seen so far in RL:
 - Q-learning: fixed-point iteration → can (in principle) include all transitions seen so far, however, it optimizes for the wrong objective
 - Policy gradient: yes, stochastic gradients of the RL objective, but no optimization problem
- We'll discuss approaches that define an optimization problem that allows us to do a small update to policy π , based on data sampled from π

Defining the objective

- We discussed how, in PO, we want to compute the following gradient $\nabla_{\theta} J(\theta) = \mathbb{E} [\nabla_{\theta} \log \pi_{\theta}(u_t | x_t) A(x_t, u_t)]$
- To implement this using modern auto-diff tools (e.g., Torch, Jax, Tensorflow), this means writing the following loss function:

$$L^{PG}(\theta) = \mathbb{E} [\log \pi_{\theta}(u_t | x_t) A(x_t, u_t)]$$

- But we don't want to optimize it too far, since we are not working with the *true* advantage, rather with a noisy estimate
- Equivalently differentiate

$$L^{IS}(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\frac{\pi_{\theta}(u_t | x_t)}{\pi_{\theta_{old}}(u_t | x_t)} A(\tau) \right]$$

- If we take the derivative of L^{IS} and evaluate at $\theta = \theta_{old}$, we get the same gradient

$$\nabla_{\theta} \log f(\theta) \Big|_{\theta_{old}} = \frac{\nabla_{\theta} f(\theta) \Big|_{\theta_{old}}}{f(\theta_{old})} = \nabla_{\theta} \left(\frac{f(\theta)}{f(\theta_{old})} \right) \Big|_{\theta_{old}}$$

Trust Region Policy Optimization (TRPO)

$$\begin{aligned} & \underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(u_t | x_t)}{\pi_{\theta_{old}}(u_t | x_t)} \hat{A}_t \right] \\ & \text{subject to } \hat{\mathbb{E}}_t \left[\text{KL}[\pi_{\theta_{old}}(\cdot | x_t), \pi_{\theta}(\cdot | x_t)] \right] \leq \delta \end{aligned}$$

- Main idea: use trust region to constrain change **in distribution space** (opposed to e.g., parameter space)
- Hard to use with architectures with multiple outputs, e.g., policy and value function
- Empirically performs poorly on tasks requiring deep nets, e.g., deep CNNs, RNNs
- Conjugate gradient makes implementation more complicated

Proximal Policy Optimization (PPO)

- Can we solve the problem defined in TRPO without second-order optimization?

PPO v1 - Surrogate loss with Lagrange multipliers

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(u_t | x_t)}{\pi_{\theta_{old}}(u_t | x_t)} \hat{A}_t \right] + \beta \left(\hat{\mathbb{E}}_t [\text{KL}[\pi_{\theta_{old}}(\cdot | x_t), \pi_{\theta}(\cdot | x_t)] - \delta \right)$$

- Run SGD on the above objective
- Do dual descent update for β

PPO v2 - Clipped surrogate loss

$$r(\theta) = \frac{\pi_{\theta}(u_t | x_t)}{\pi_{\theta_{old}}(u_t | x_t)}, \quad r(\theta_{old}) = 1$$

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t [\min(r(\theta)A(\tau), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)A(\tau)]$$

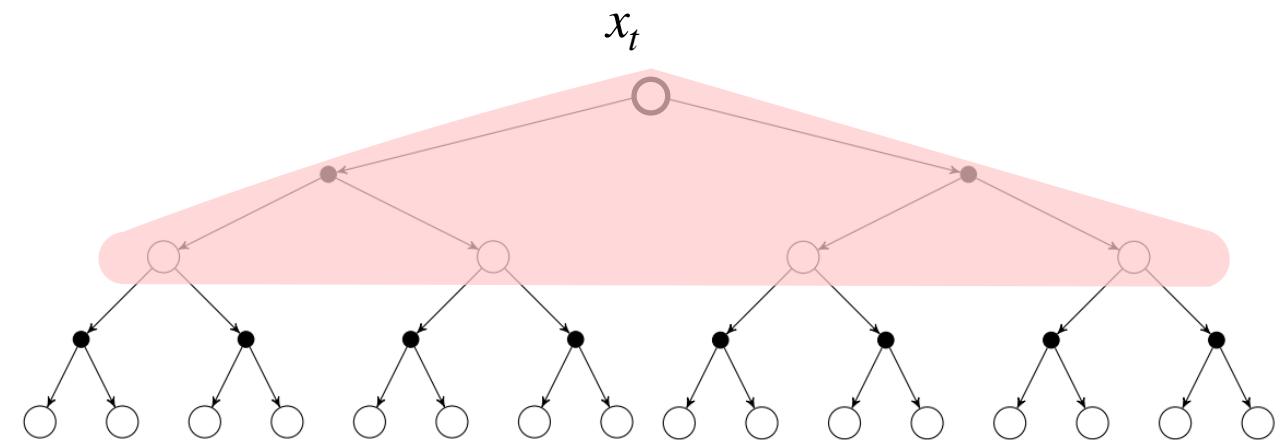
- Heuristically replicates constraint in the objective
- One of the (if not the) most popular PO algorithm

Summary: Model-free RL

- We discussed different ways to estimate value functions

Dynamic Programming

$$\hat{V}(x_t) \leftarrow \mathbb{E} [R_t + \gamma \hat{V}(x_{t+1})]$$

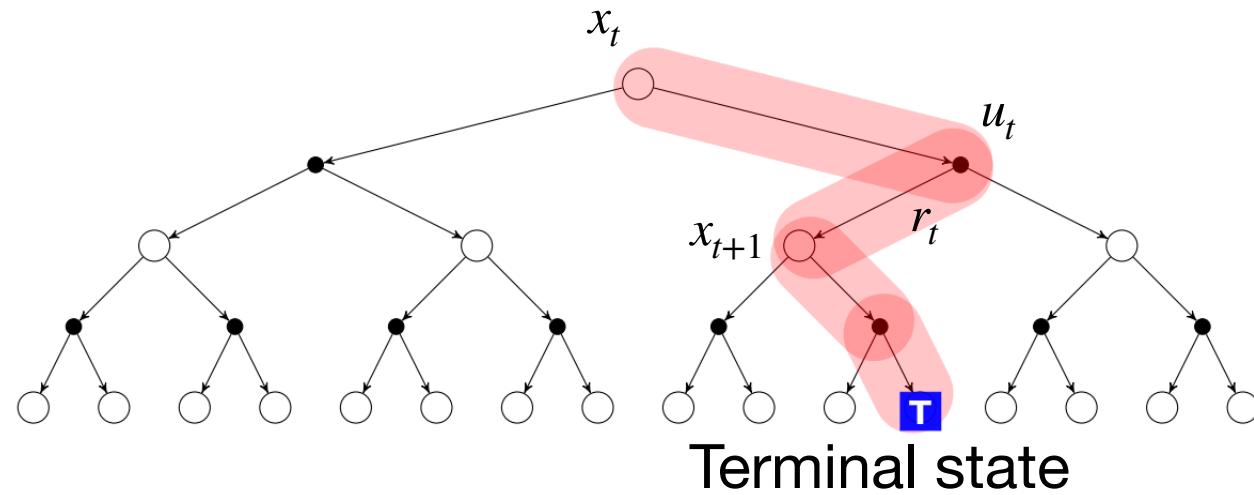


$$\hat{Q}(x_t, u_t) \leftarrow \mathbb{E} [R_t + \gamma \hat{Q}(x_{t+1}, u_{t+1})]$$

Exact
Requires
knowledge
of MDP

Monte Carlo

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha (G_t - \hat{V}(x_t))$$



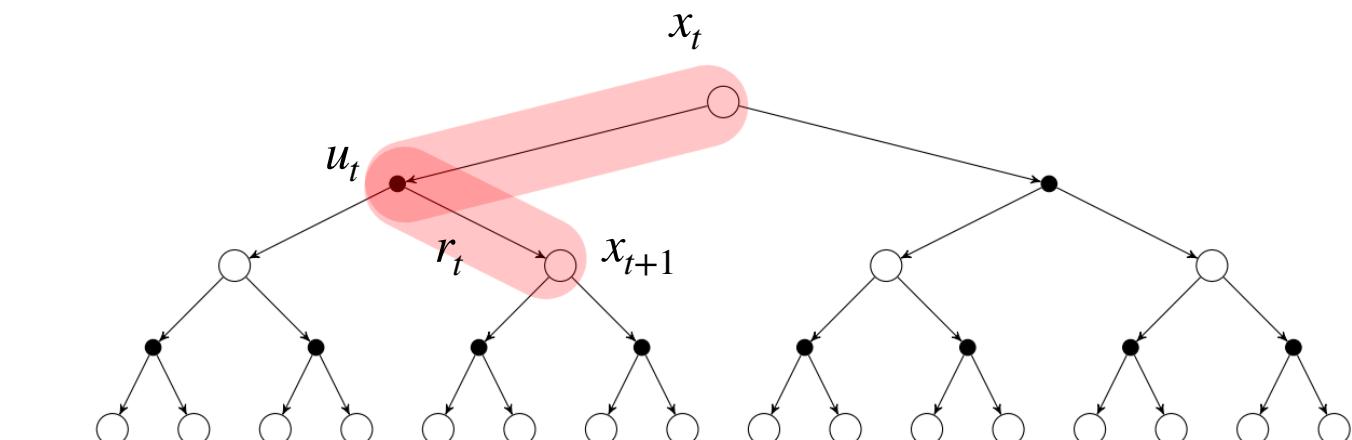
$$\hat{Q}(x_t, u_t) \leftarrow \hat{Q}(x_t, u_t) + \alpha (G_t - \hat{Q}(x_t, u_t))$$

Unbiased
High variance;
must reach
terminal state

Low variance; can learn online
Biased

Temporal-Difference

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha (R_t + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t))$$



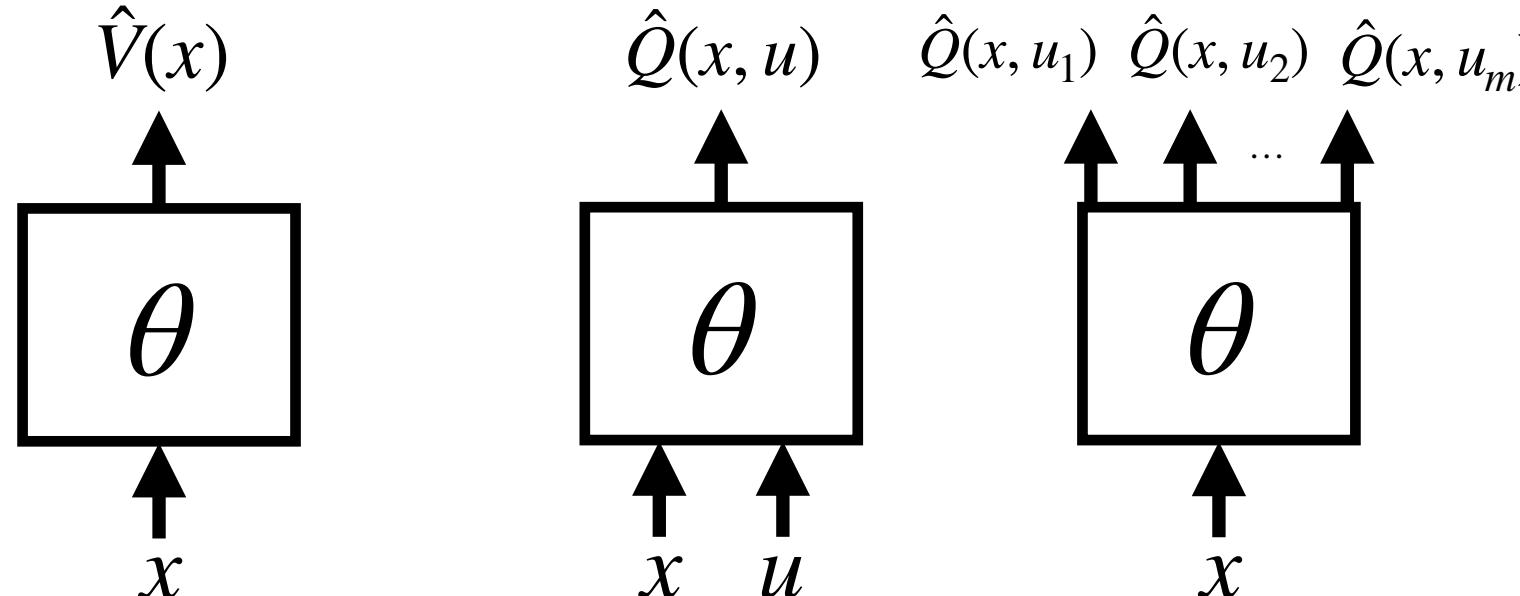
$$\hat{Q}(x_t, u_t) \leftarrow \hat{Q}(x_t, u_t) + \alpha (R_t + \gamma \hat{Q}(x_{t+1}, u_{t+1}) - \hat{Q}(x_t, u_t))$$

- And how to scale these ideas through function approximation

Tabular representation:

$$\hat{V}(x) = \begin{bmatrix} \hat{V}(x_1) \\ \hat{V}(x_2) \\ \vdots \\ \hat{V}(x_n) \end{bmatrix} \quad \hat{Q}(x, u) = \begin{bmatrix} \hat{Q}(x_1, u_1) & \hat{Q}(x_1, u_2) & \dots & \hat{Q}(x_1, u_m) \\ \hat{Q}(x_2, u_1) & \hat{Q}(x_2, u_2) & \dots & \hat{Q}(x_2, u_m) \\ \vdots \\ \hat{Q}(x_n, u_1) & \hat{Q}(x_n, u_2) & \dots & \hat{Q}(x_n, u_m) \end{bmatrix}$$

Function approximation:



$$\Delta\theta = \alpha (G_t - \hat{V}_\theta(x_t)) \nabla_\theta \hat{V}_\theta(x_t)$$

MC update

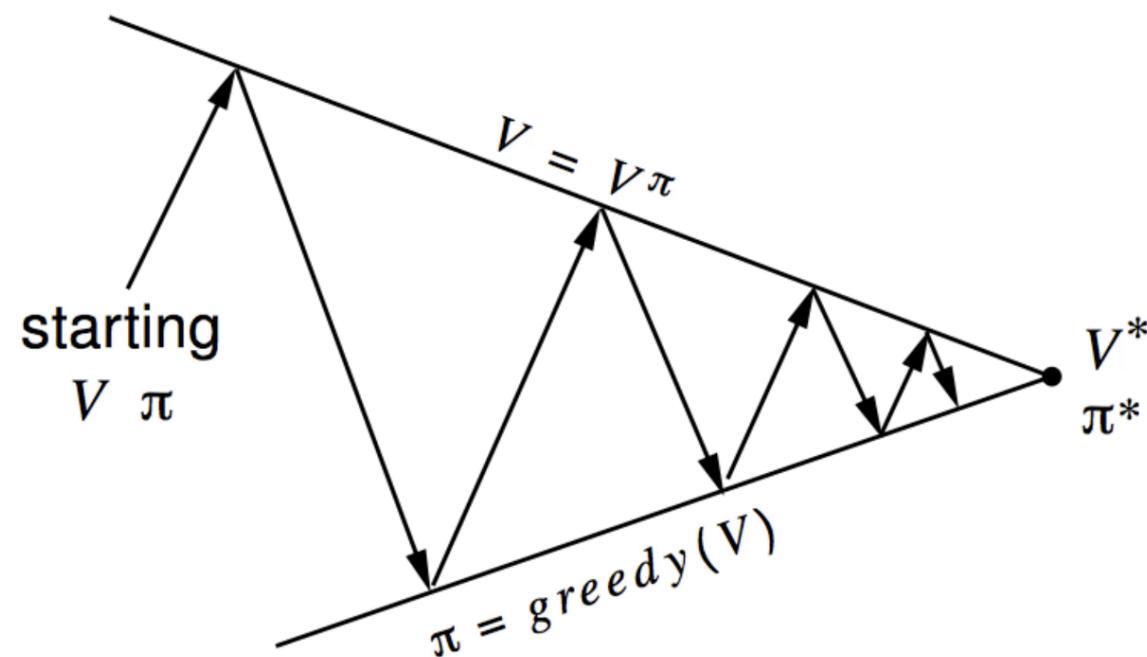
$$\Delta\theta = \alpha (r_t + \gamma \hat{V}_\theta(x_{t+1}) - \hat{V}_\theta(x_t)) \nabla_\theta \hat{V}_\theta(x_t)$$

TD update

Summary: Model-free RL

Value-based methods

- Generalized Policy Iteration



- Sarsa & Q-learning

SARSA: on-policy

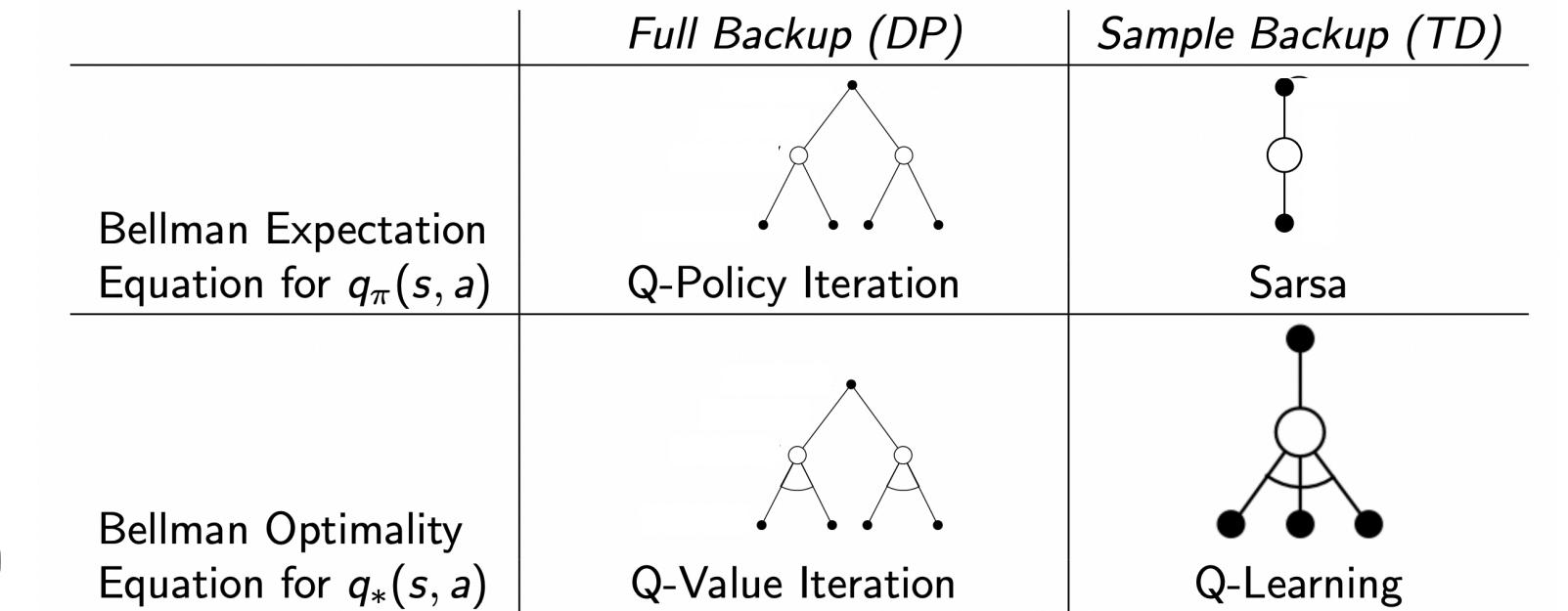
$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha (r_t + \gamma Q(x_{t+1}, u_{t+1}) - Q(x_t, u_t))$$

Q-learning: off-policy

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left(r_t + \gamma \max_{u'_{t+1}} Q(x_{t+1}, u'_{t+1}) - Q(x_t, u_t) \right)$$

On-policy: evaluate or improve the policy that is used to make decisions

Off-policy: evaluate or improve a policy different from that used to generate the data



- Deep RL:

- (1) Use **deep neural nets** to represent Q_θ
- (2) Uses **experience replay** and **fixed Q-targets**

- In policy optimization, we care about learning an (explicit) parametric policy π_θ , with parameters θ to directly maximize:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \underbrace{\left[\sum_{t \geq 0} \gamma^t R(x_t, u_t) \right]}_{J(\theta)} \quad \begin{array}{l} (1) \text{ estimate its gradient } \nabla_{\theta} J(\theta) \\ (2) \text{ do approximate gradient ascent on } J(\theta): \theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \end{array}$$

Policy gradient: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(u_{i,t} | x_{i,t}) \right) \left(\sum_{t=1}^T R(x_{i,t}, u_{i,t}) \right) \right]$

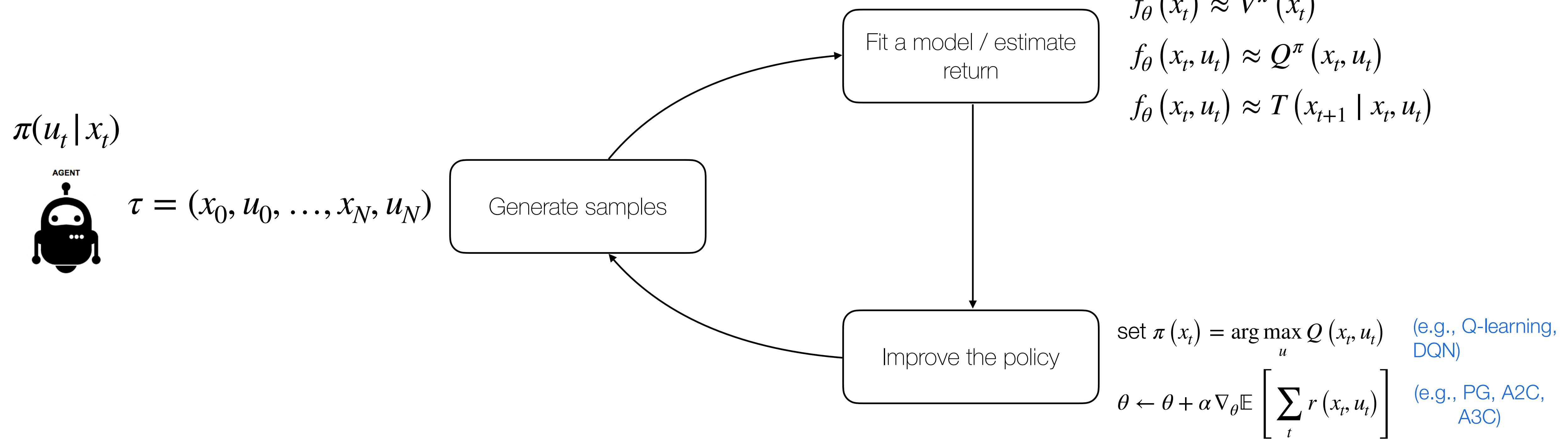
Maximum Likelihood: $\nabla_{\theta} J_{MLE}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(u_{i,t} | x_{i,t}) \right) \right] \quad \text{"Change parameters } \theta \text{ s.t. trajectories with higher reward have higher probability"}$

Problem: high variance of PG

Solution: baselines, “critics”

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(u_{i,t} | x_{i,t}) Q_{\phi}(x_t, u_t)$$

A bird's eye view on the RL algorithms covered so far

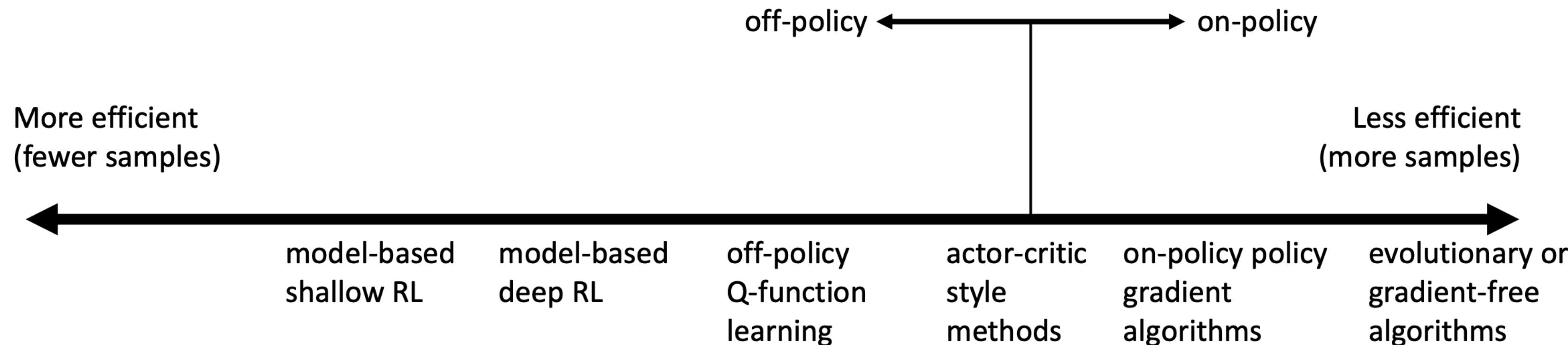


Revisiting the discussion on trade-offs

- **Different tradeoffs:**
 - Sample efficiency
 - Stability & easy of use
- **Different assumptions:**
 - Stochastic or deterministic
 - Continuous or discrete
 - Episodic or infinite horizon
- **Different things are easy or hard in different settings:**
 - Easier to represent the policy?
 - Easier to represent the model?

Comparison: sample efficiency

- Sample efficiency = how many samples do we need to get a good policy?
- Crucial question: is the algorithm *on-* or *off-policy*?
 - **Off policy**: able to improve the policy without generating new samples from the current policy
 - **On policy**: each time the policy is changed, even a little bit, we need to generate new samples



Why even bother using less efficient algorithms? Wall-clock time is not the same as efficiency!

Outline

Basics of model-based RL

- A basic recipe (and its limitations)
- Learning with high-capacity models: distributional shift

Uncertainty quantification in model-based RL

- Gaussian Processes
- Bootstrap Ensembles

Examples & Applications (e.g., PETs)

General recipe

- If we knew the dynamics $T(x_{t+1} | x_t, u_t)$, we could use tools from optimal control
- **Main idea:** learn a model $f_\theta(x_t, u_t) \approx T(x_{t+1} | x_t, u_t)$ from data (or $p(x_{t+1} | x_t, u_t)$ in the stochastic case)

At a high-level, we could apply the following strategy:

1. Run base policy $\pi_0(u_t | x_t)$ in the environment (e.g., random policy, exploration policy) and collect dataset of transitions
 $\mathcal{D} = \{(x_t, u_t, x_{t+1})_i\}$
2. Fit dynamics model to data to minimize error (or equivalently, maximize (log) likelihood)

$$\theta^* = \arg \min_{\theta} \sum_i \|f_\theta(x_t, u_t) - x_{t+1}\|^2$$

3. Use the learned model to plan a sequence of actions

Will this work?

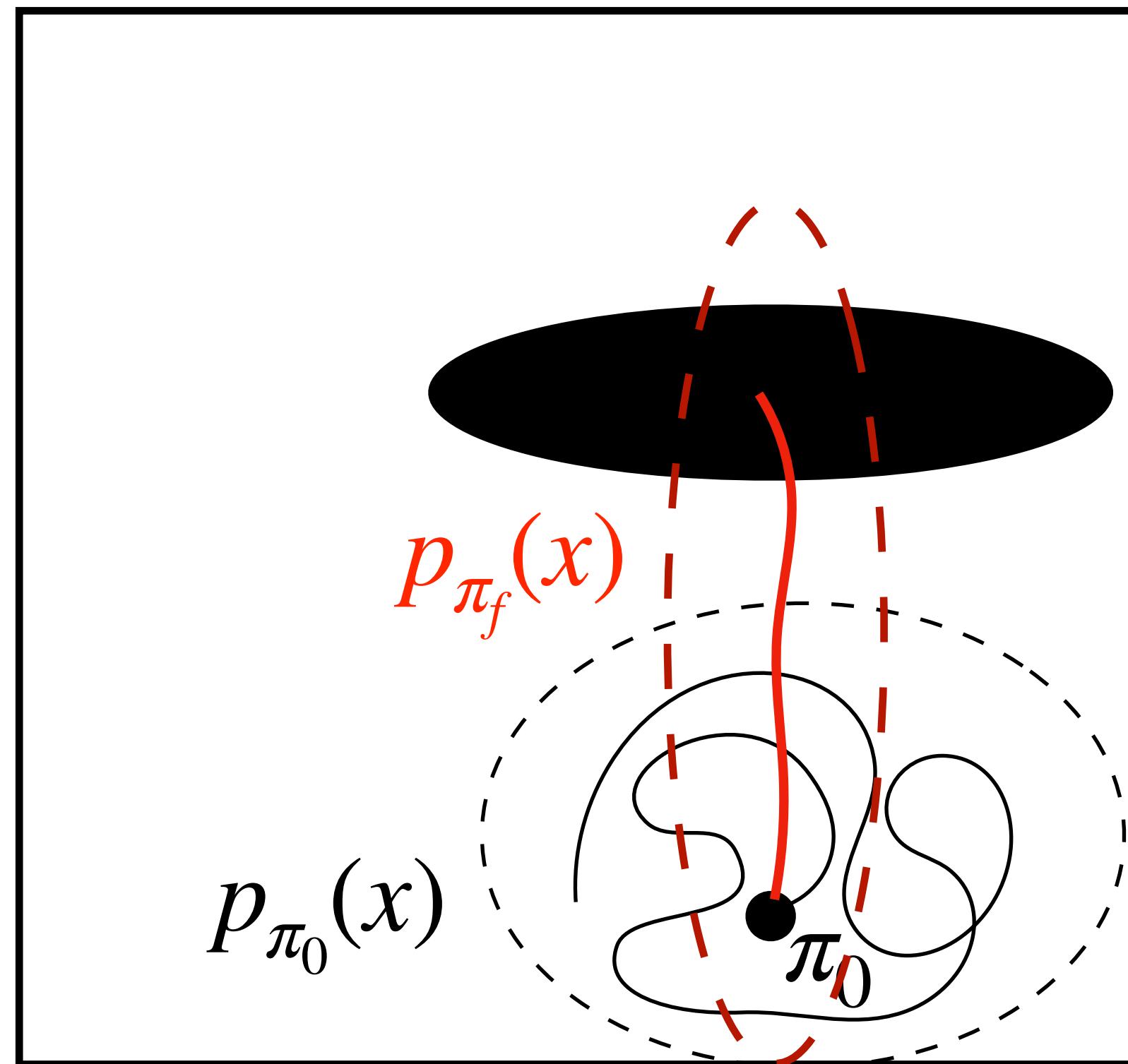
YES

- In cases with e.g., linear-time invariant dynamics, this tends to work pretty well
- Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters
 - If the dataset is generated with sufficient excitation, it gives **global** knowledge (i.e., some care should be taken to design a good base policy)
- This is essentially how system identification works

NO

- If we're dealing with non-linear dynamics (and high-capacity models! e.g., neural networks) **extrapolation is difficult and can be misleading**

Motivating example



- The goal is to go as further north as possible
- The base policy defines state distribution (under π_0)
- When planning under the model we observe a different state distribution, i.e., $p_{\pi_f}(x)$

The more (i) the dynamics are complex, (ii) we use high-capacity models, the easier it is incur in distribution mismatch

A simple improvement

- We can leverage ideas from adaptive and receding-horizon control:
 1. Run base policy $\pi_0(u_t | x_t)$ in the environment (e.g., random policy, exploration policy) and collect dataset of transitions
 $\mathcal{D} = \{(x_t, u_t, x_{t+1})_i\}$
 2. Fit dynamics model to data to minimize error (or equivalently, maximize (log) likelihood)
$$\theta^* = \arg \min_{\theta} \sum_i \| f_{\theta}(x_t, u_t) - x_{t+1} \|^2$$
 3. Use the learned model to plan a sequence of actions
 4. Execute only the first action and measure the new state x_{t+1} (**i.e., MPC**)
 5. Add the observed transition (x_t, u_t, x_{t+1}) to the dataset \mathcal{D} and update model (**i.e., gradually closing the gap between $p_{\pi_0}(x)$ and $p_{\pi_f}(x)$**)

Outline

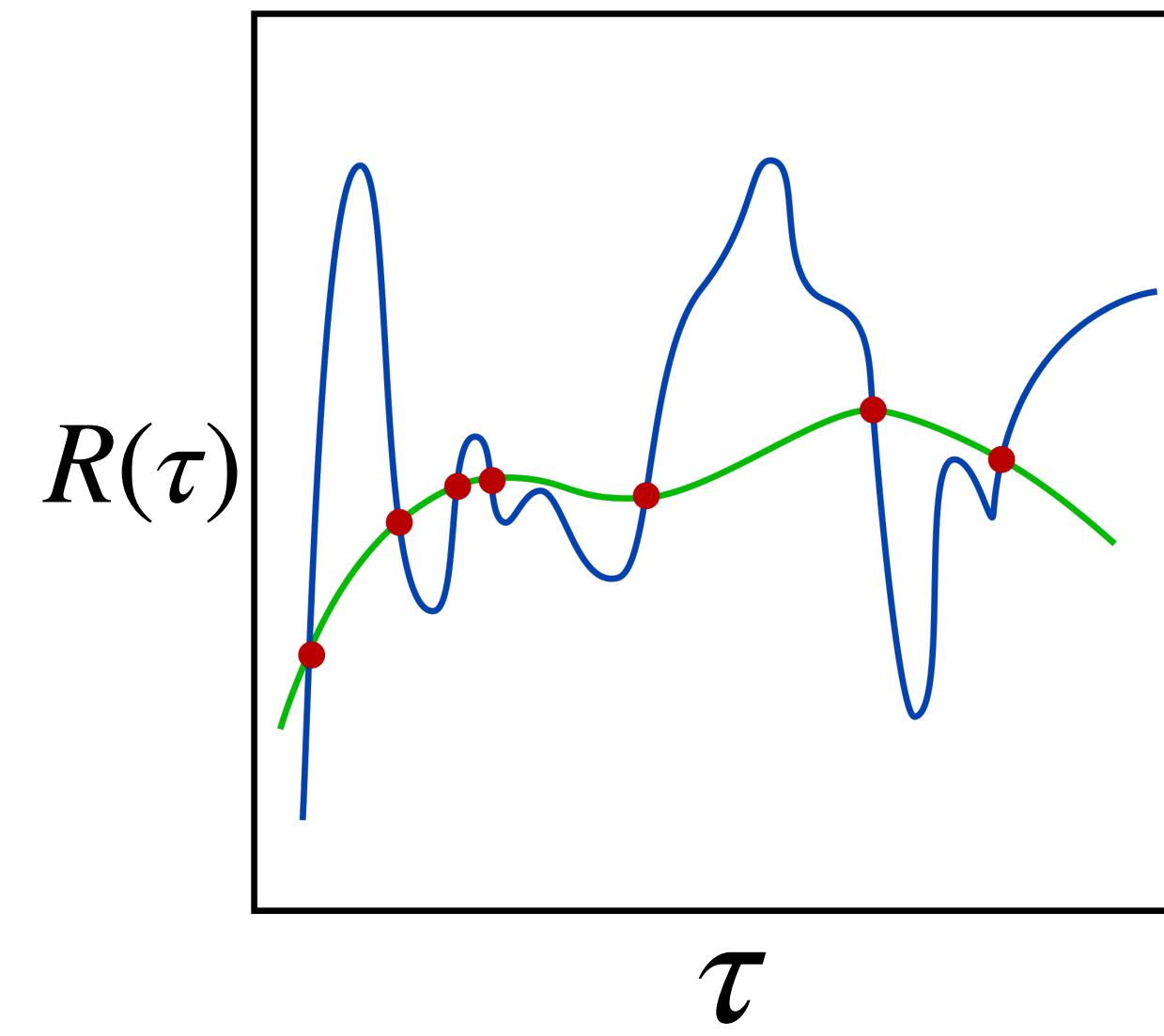
Uncertainty quantification in model-based RL

- Gaussian Processes
- Bootstrap Ensembles

The main challenge in MBRL

- Ideally, we'd want our model to:
 - Have high-capacity to represent complex dynamics in the high-data regime
 - Not overfit to observed data in the low-data regime
- For example, consider the case where we fit our model to observed data and use it to plan, according to the previous scheme

1. Run base policy $\pi_0(u_t | x_t)$ in the environment (e.g., random policy, exploration policy) and collect dataset of transitions
 $\mathcal{D} = \{(x_t, u_t, x_{t+1})_i\}$
2. Fit dynamics model to data to minimize error (or equivalently, maximize (log) likelihood)
$$\theta^* = \arg \min_{\theta} \sum_i \| f_{\theta}(x_t, u_t) - x_{t+1} \|^2$$
3. Use the learned model to plan a sequence of actions
4. Execute only the first action and measure the new state x_{t+1} (**i.e., MPC**)
5. Add the observed transition (x_t, u_t, x_{t+1}) to the dataset \mathcal{D} and update model (**i.e., gradually closing the gap between $p_{\pi_0}(x)$ and $p_{\pi_f}(x)$**)

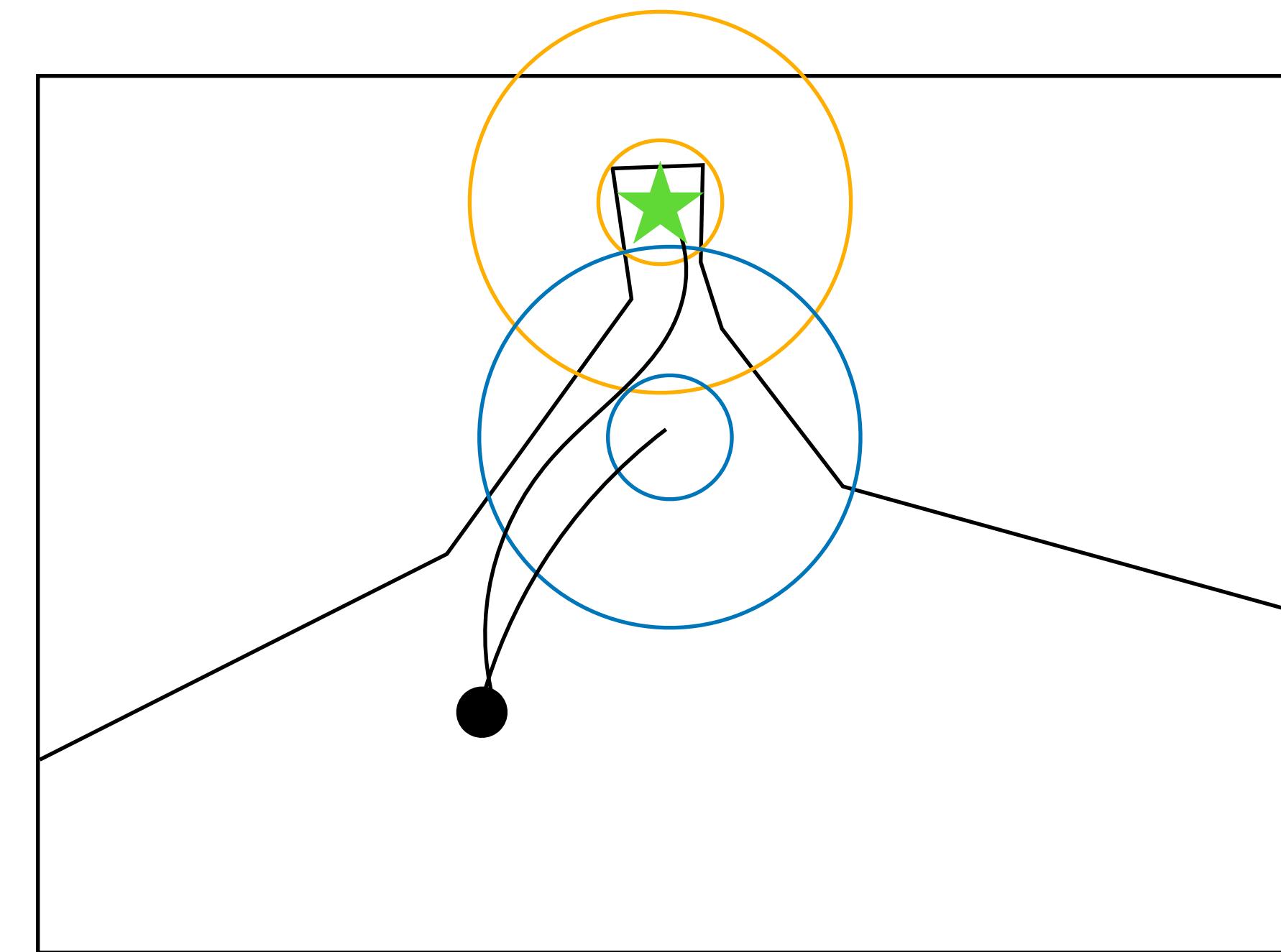
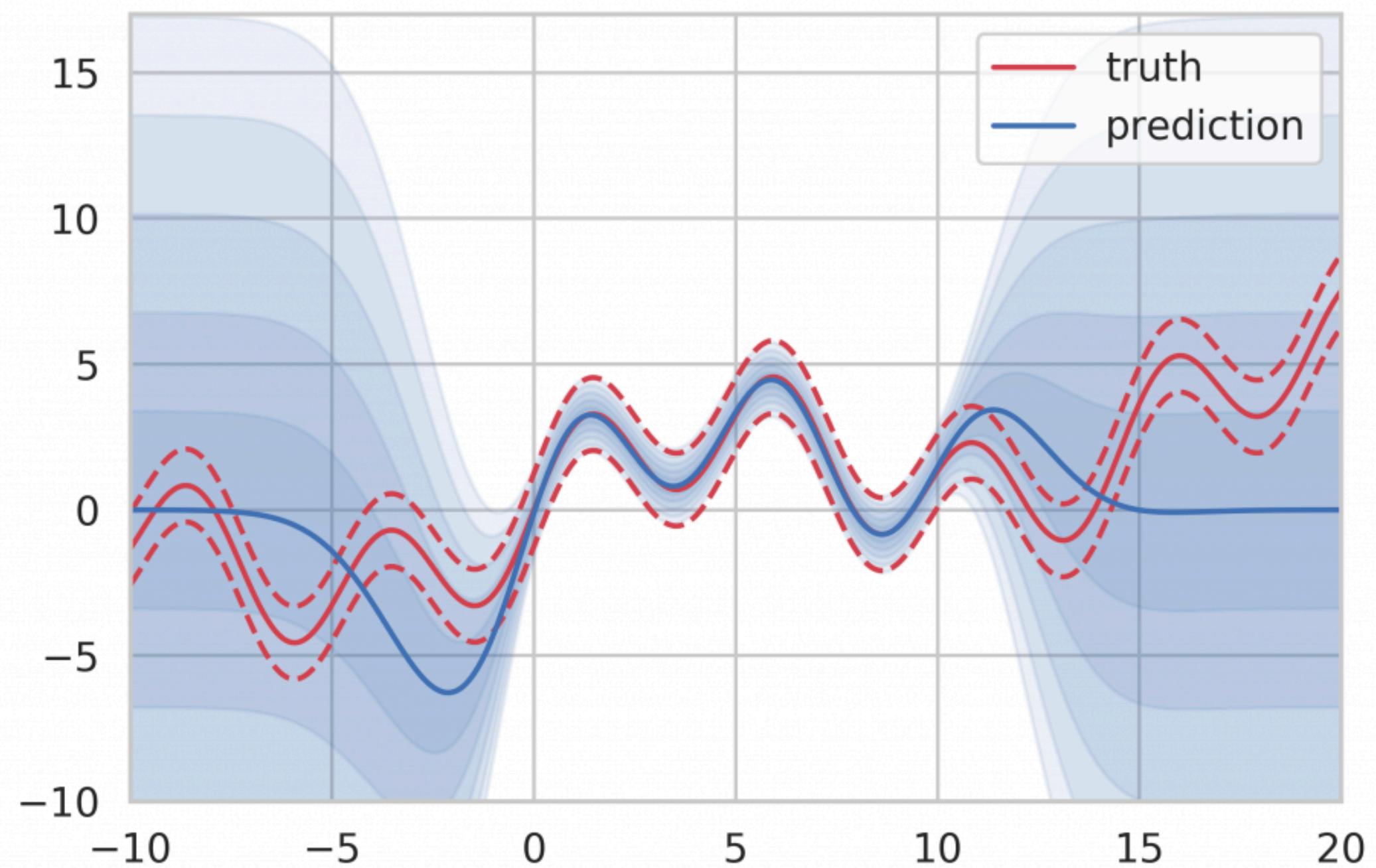


Problem: we'll likely erroneously exploit our model where it is less knowledgeable

(Possible) Solution: consider how "certain" we are about the prediction

The role of uncertainty estimation

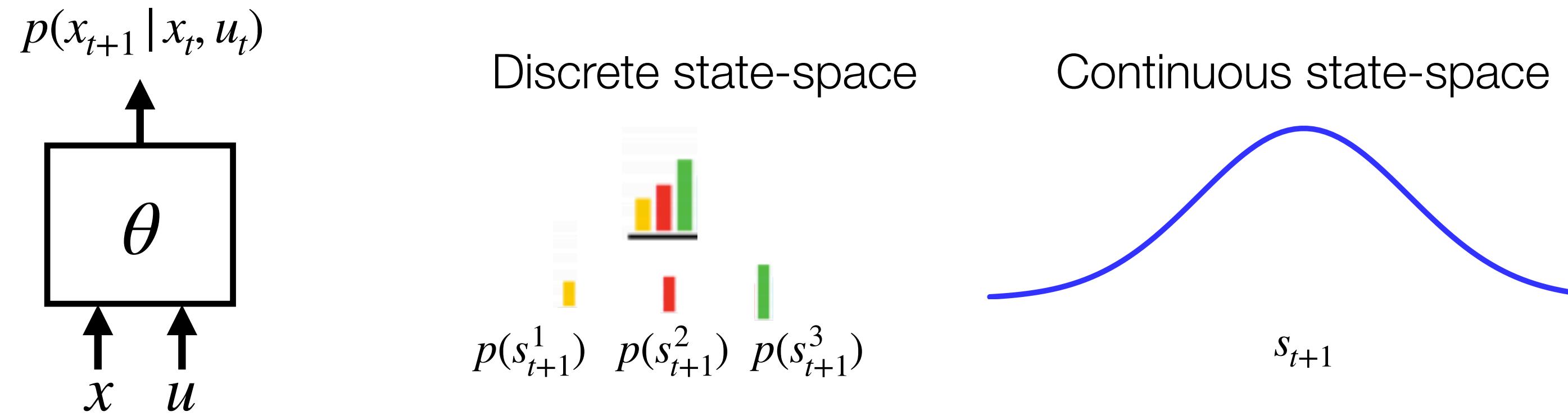
- Specifically, by uncertainty on our predictions, we mean an expression of a *distribution over possible outcomes*
- This allows us to reason in terms of expectations under our model



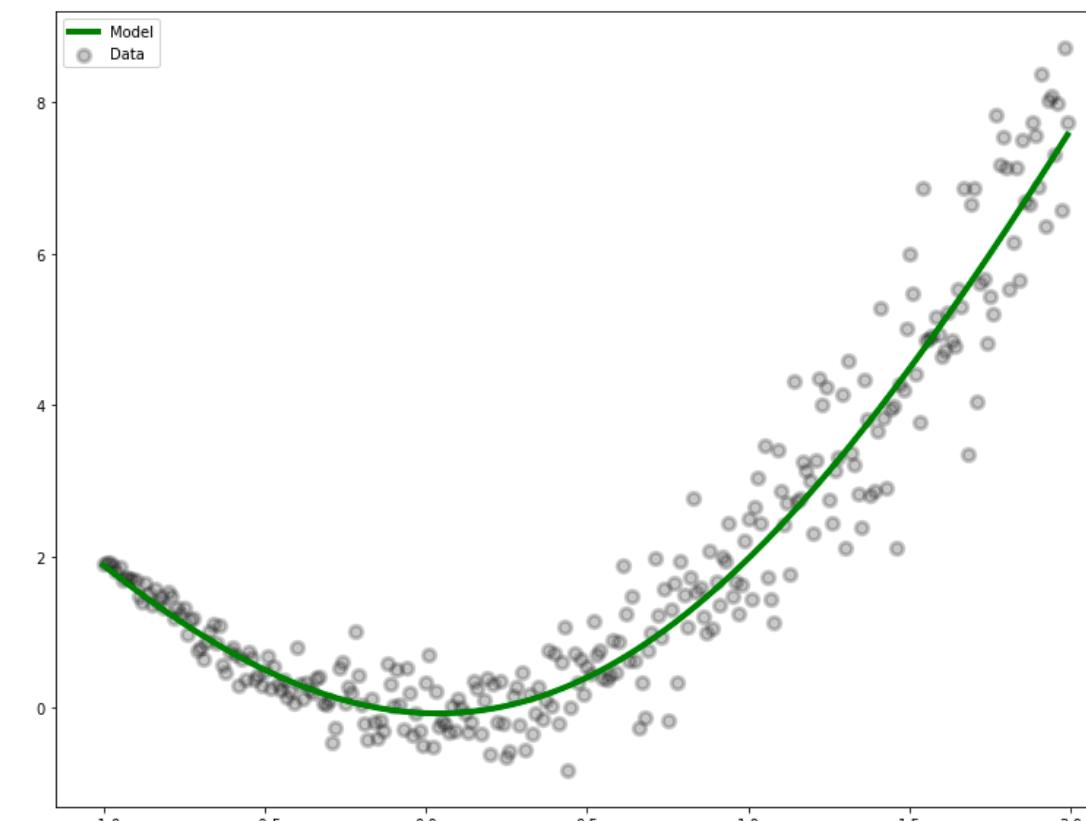
Expected reward under high-variance prediction is low

How can we model uncertainty?

- **Idea 1:** use output entropy
- Suppose we estimated a model, why not use its entropy?

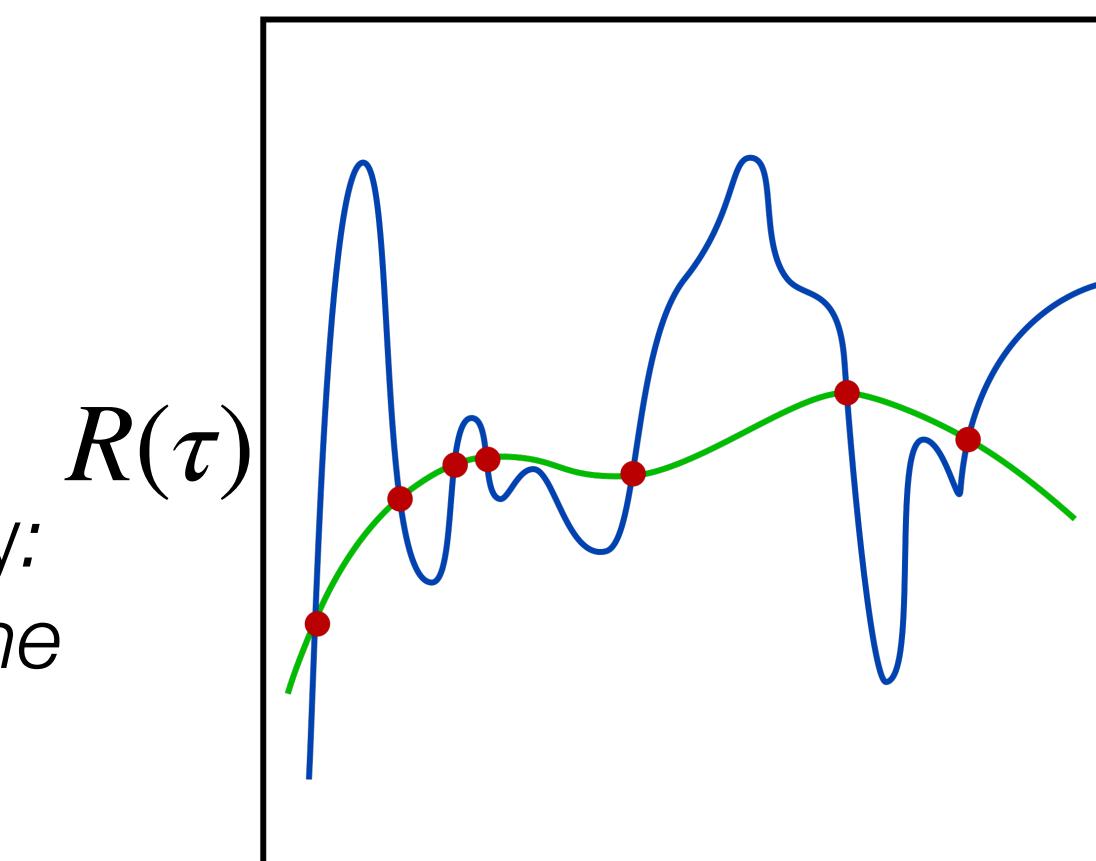


- Doing so will not take *epistemic* uncertainty into account



Aleatoric uncertainty:
“The process is
intrinsically noisy”

Epistemic uncertainty:
“Uncertainty about the
model”



How can we model uncertainty?

- **Idea 2:** estimate model uncertainty

$$p(x_{t+1} | x_t, u_t)$$

```

graph TD
    x[x] --> theta[θ]
    u[u] --> theta
    theta --> px["p(x_{t+1} | x_t, u_t)"]
  
```

- Typically, given a dataset \mathcal{D} , we estimate:

$$\arg \max_{\theta} \log p(\mathcal{D} | \theta)$$

- To express model uncertainty means estimating:

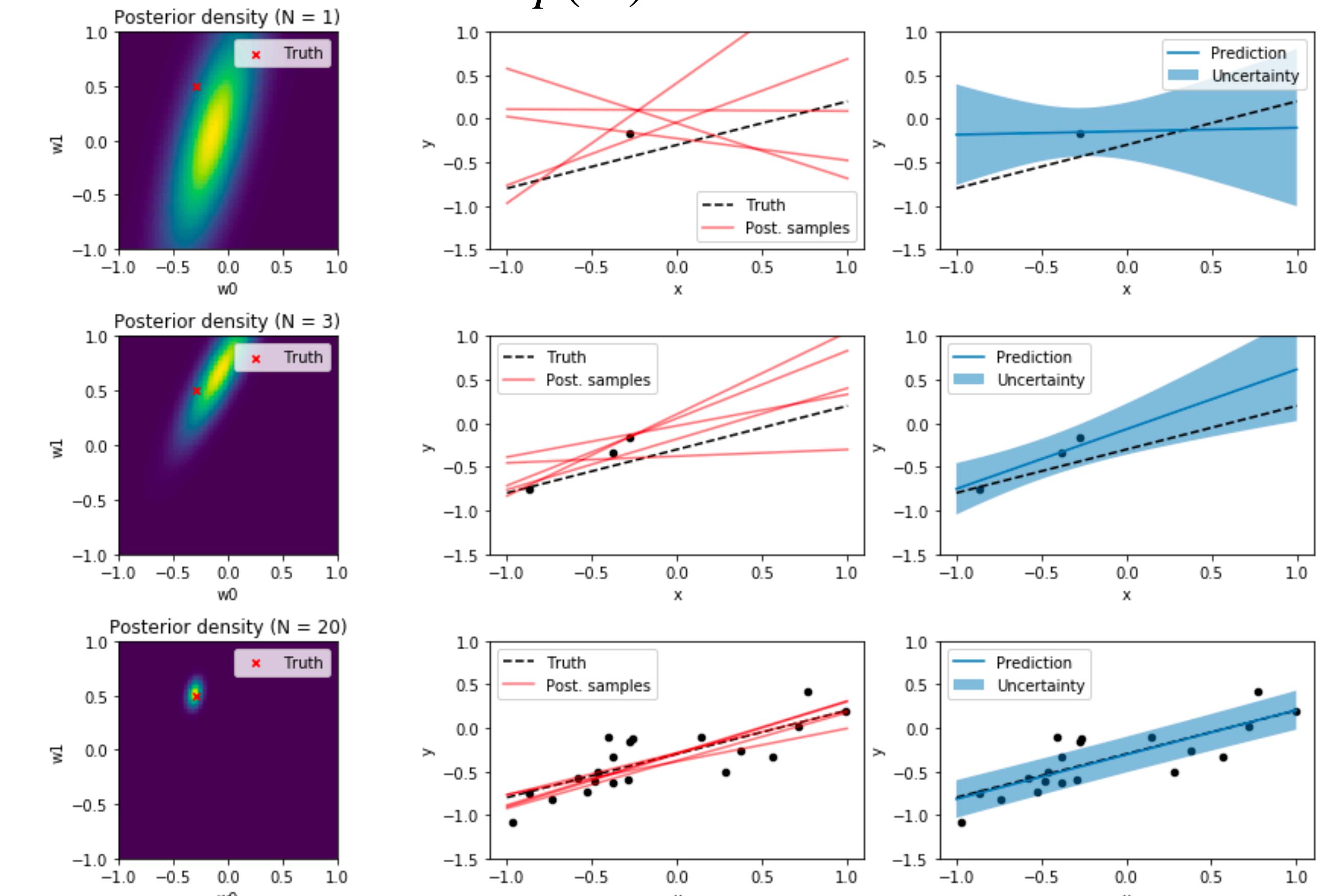
$$p(\theta | \mathcal{D})$$

and predict according to the *predictive posterior distribution*

$$\int p(x_{t+1} | x_t, u_t, \theta) p(\theta | \mathcal{D}) d\theta$$

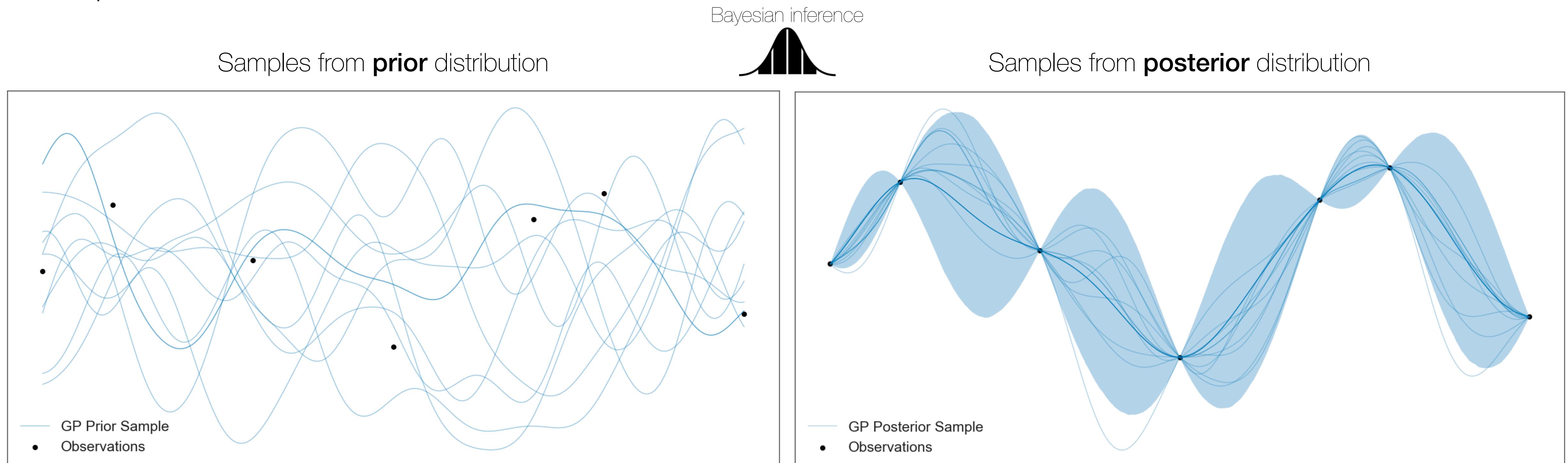
Prior: $p(\theta)$, Likelihood: $p(\mathcal{D} | \theta)$, Posterior $p(\theta | \mathcal{D})$

$$\text{Bayes' Theorem } p(\theta | \mathcal{D}) = \frac{p(\mathcal{D} | \theta)p(\theta)}{p(\mathcal{D})}$$



(1) Gaussian Processes

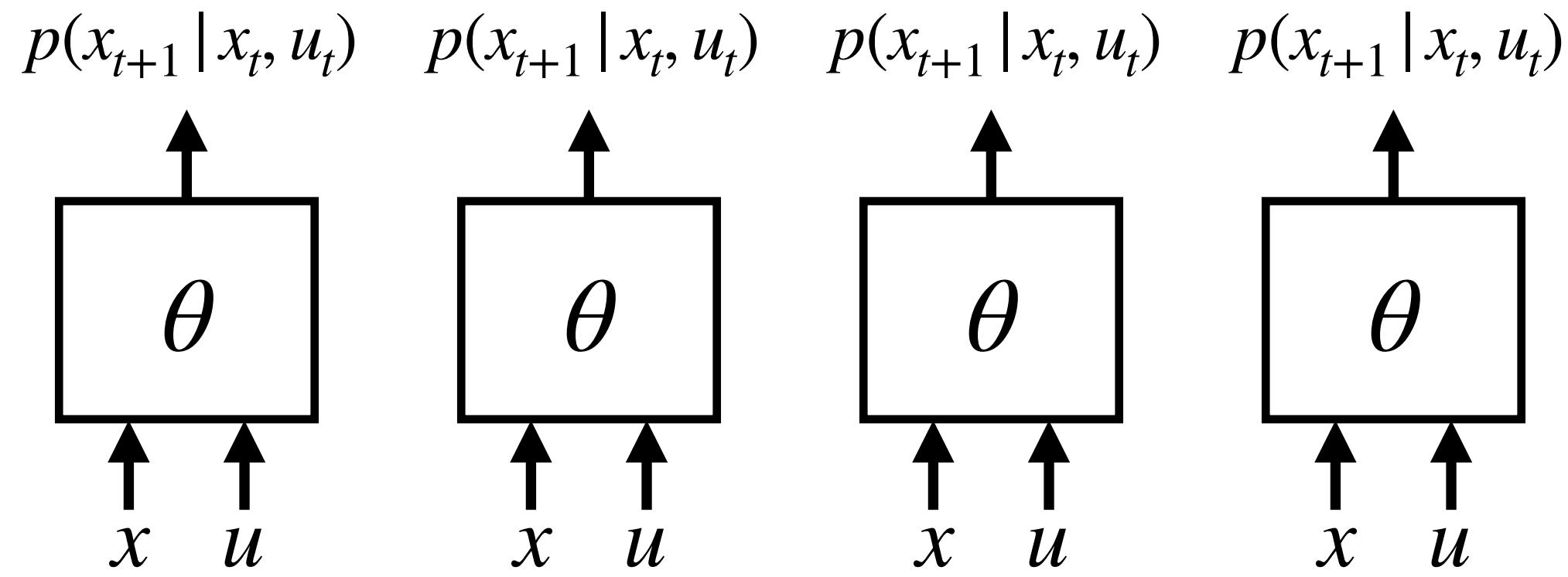
- Represent “distribution over functions”



- Strengths
 - Data efficient
 - Exact posterior
- Weaknesses
 - High computational complexity
 - Cannot learn expressive features

(2) Bootstrap ensembles

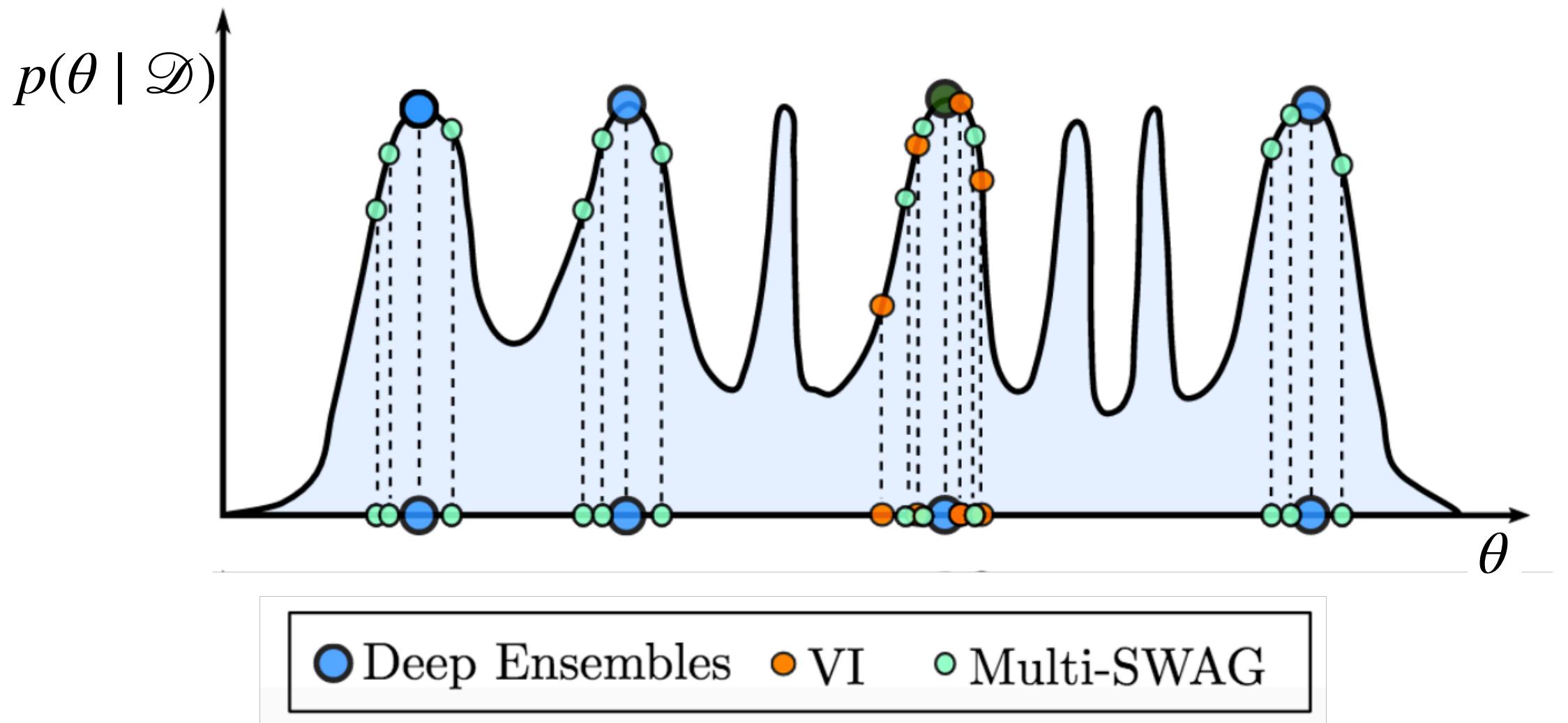
- High level idea: “train multiple models and see if they agree”
 - Different models will likely agree in regions where we have data and disagree where we do not



- Formally, we approximate the posterior with a mixture of Dirac distributions:

$$p(\theta | \mathcal{D}) \approx \frac{1}{N} \sum_i \delta(\theta_i)$$

$$\int p(x_{t+1} | x_t, u_t, \theta) p(\theta | \mathcal{D}) d\theta \approx \frac{1}{N} \sum_i p(x_{t+1} | x_t, u_t, \theta_i)$$



- Usually, no need for resampling or independent datasets: SGD and random initialization make the models sufficiently independent

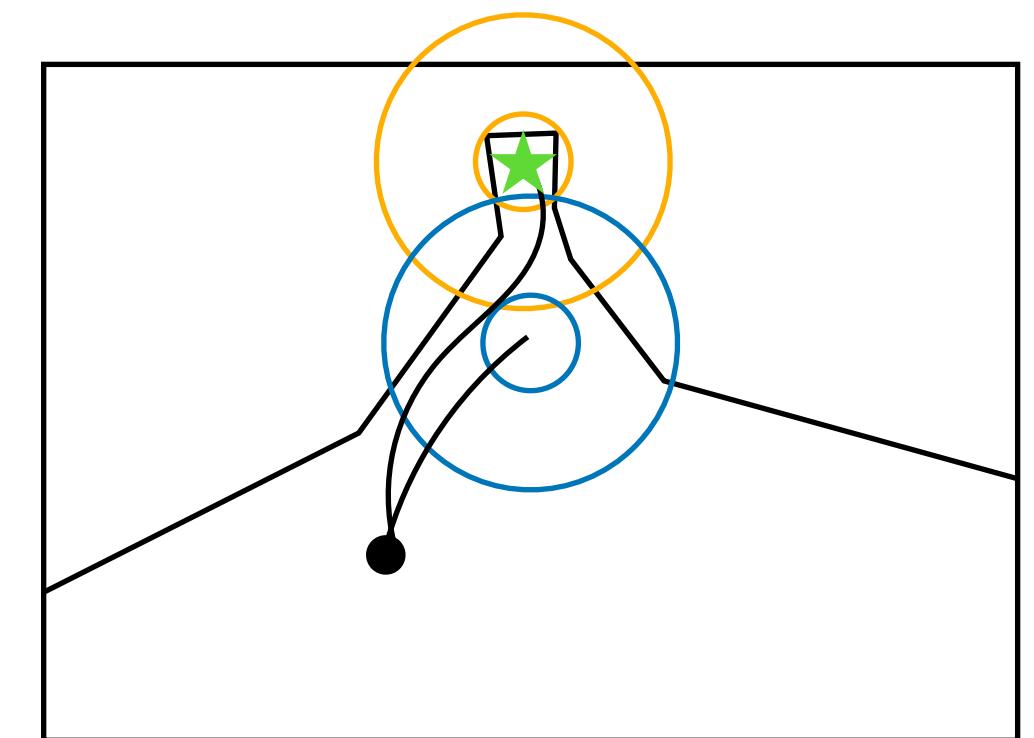
Outline

Examples & Applications (e.g., PETS)

Planning with uncertainty

- How can we use this additional knowledge in planning?
- Given a candidate action sequence u_1, \dots, u_T :
 1. Sample $\theta_i \sim p(\theta | \mathcal{D})$ (in the case of ensembles, this is equivalent to choosing one among the models)
 2. Propagate forward the learned dynamics according to $x_{t+1} \sim p_{\theta_i}(x_{t+1} | x_t, u_t)$, for all t
 3. Compute (predicted) rewards $\sum_t r(x_t, u_t)$
 4. Repeat steps 1-3 and compute the average reward

$$J(u_1, \dots, u_T) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^H r(x_{t,i}, u_t), \text{ where } x_{t+1,i} \sim p_{\theta_i}(x_{t+1,i} | x_{t,i}, u_t)$$



- Caveat: this is only a choice, one could think of other ways to approximate the posterior predictive distribution.
 - The general idea is that, when planning, we want to evaluate the expected reward under our model

Case study: PETS

- Probabilistic Ensembles with Trajectory Sampling
- Key idea:
 - **Model:** Use ensemble of NNs to approximate posterior over model
 - **Propagation:** sample different models and use them to generate predictions of different “futures”
 - **Planning:** apply MPC (compute action sequence via sampling, i.e., cross-entropy method (CEM))

Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

Kurtland Chua

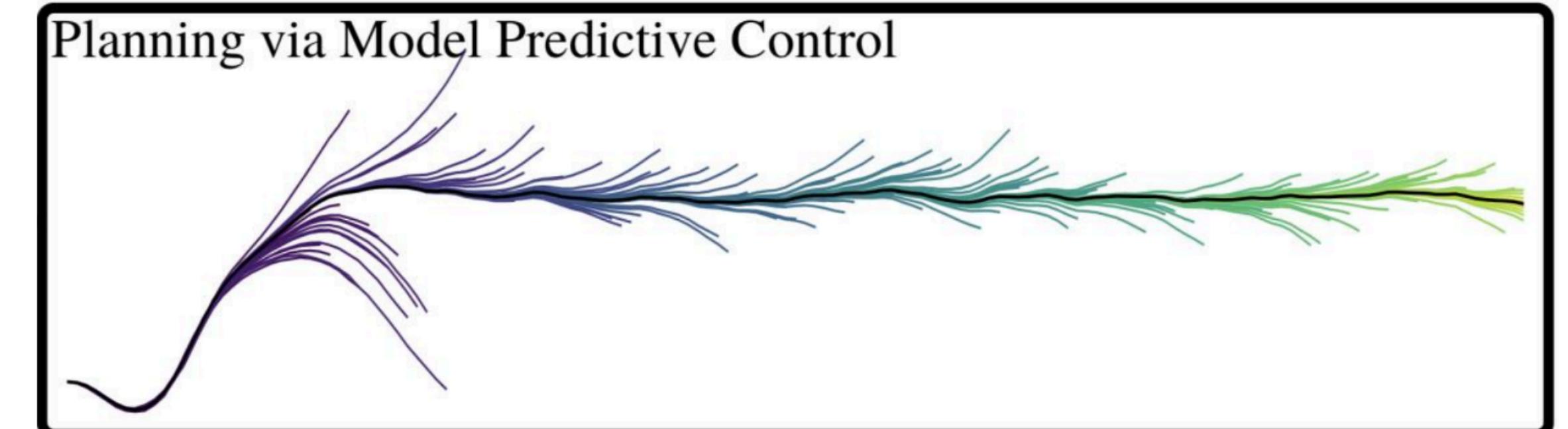
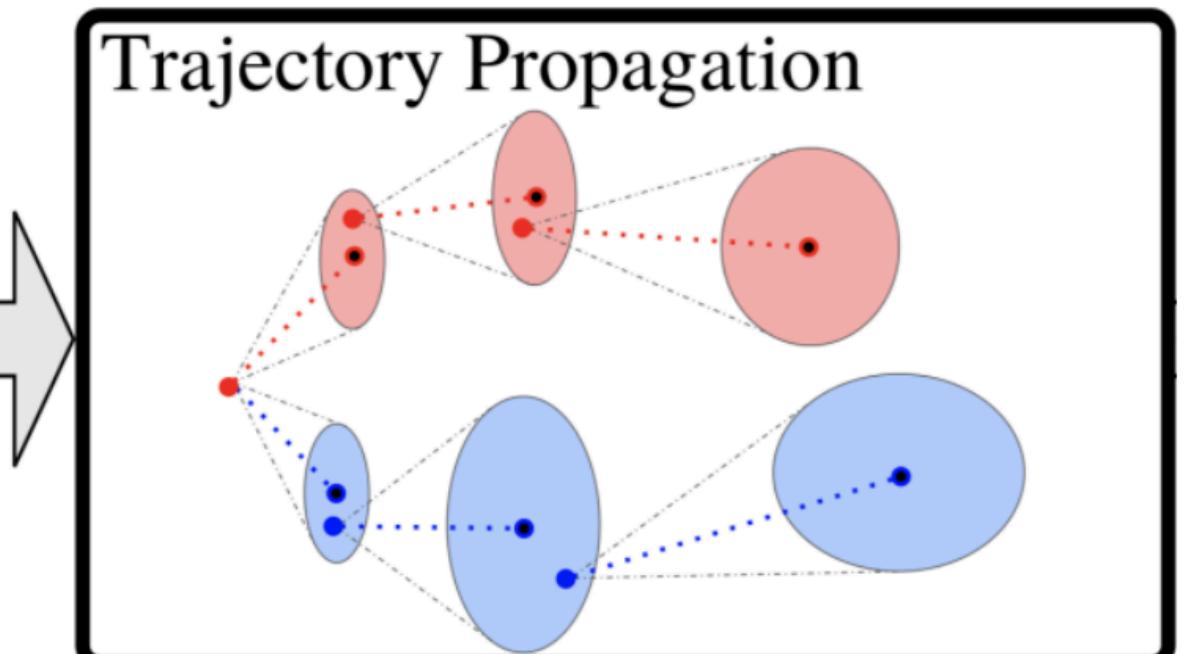
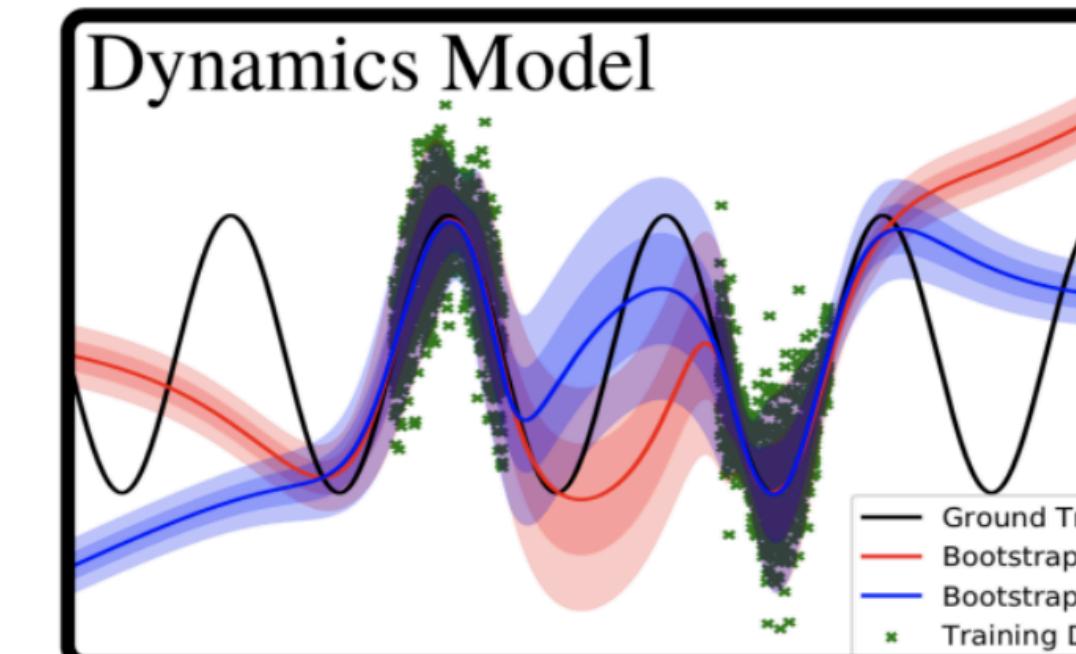
Roberto Calandra

Rowan McAllister

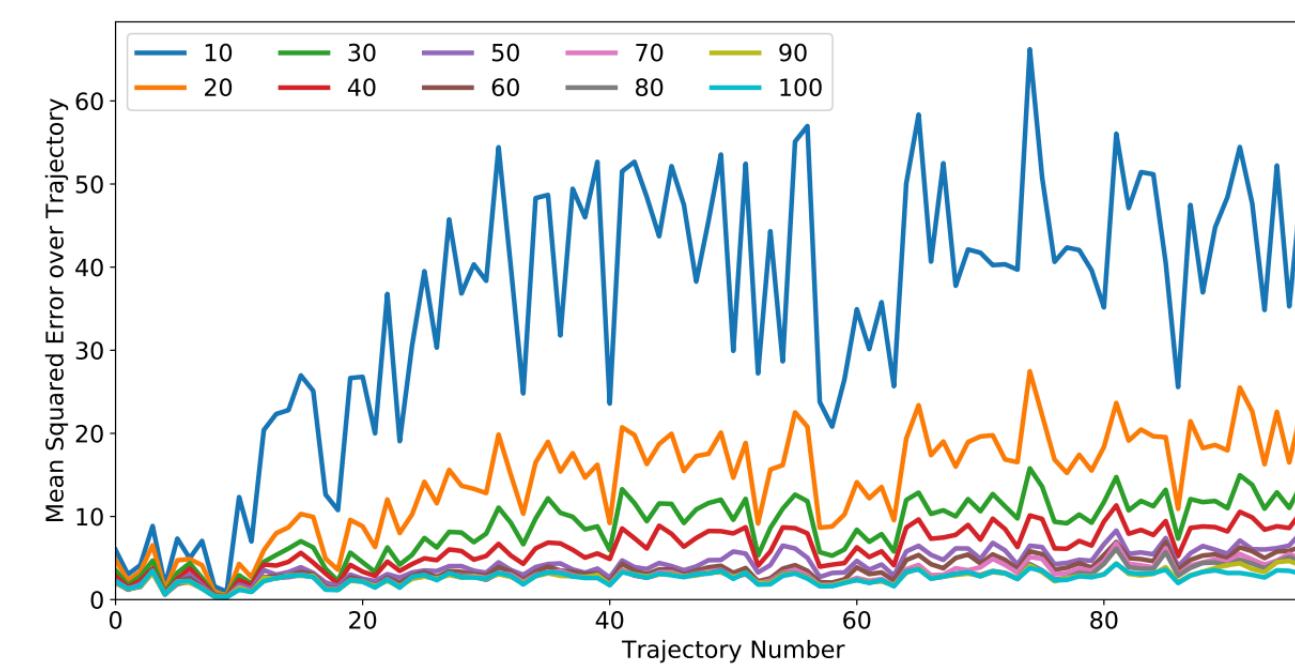
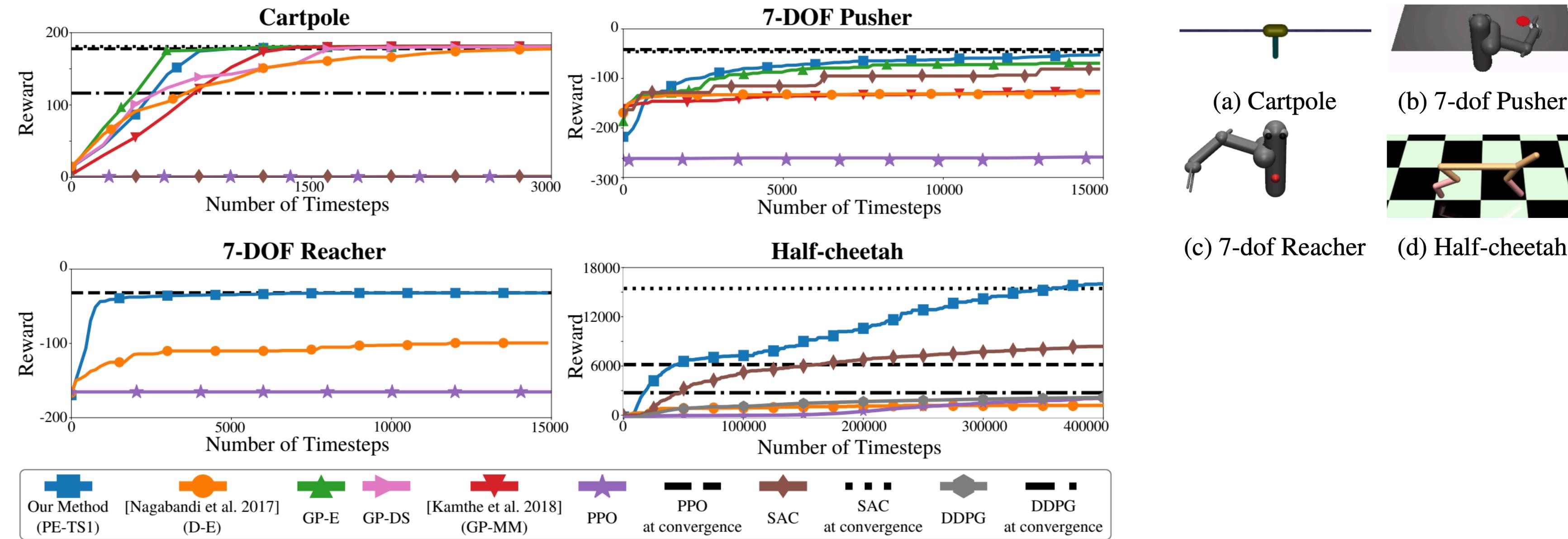
Sergey Levine

Berkeley Artificial Intelligence Research
University of California, Berkeley

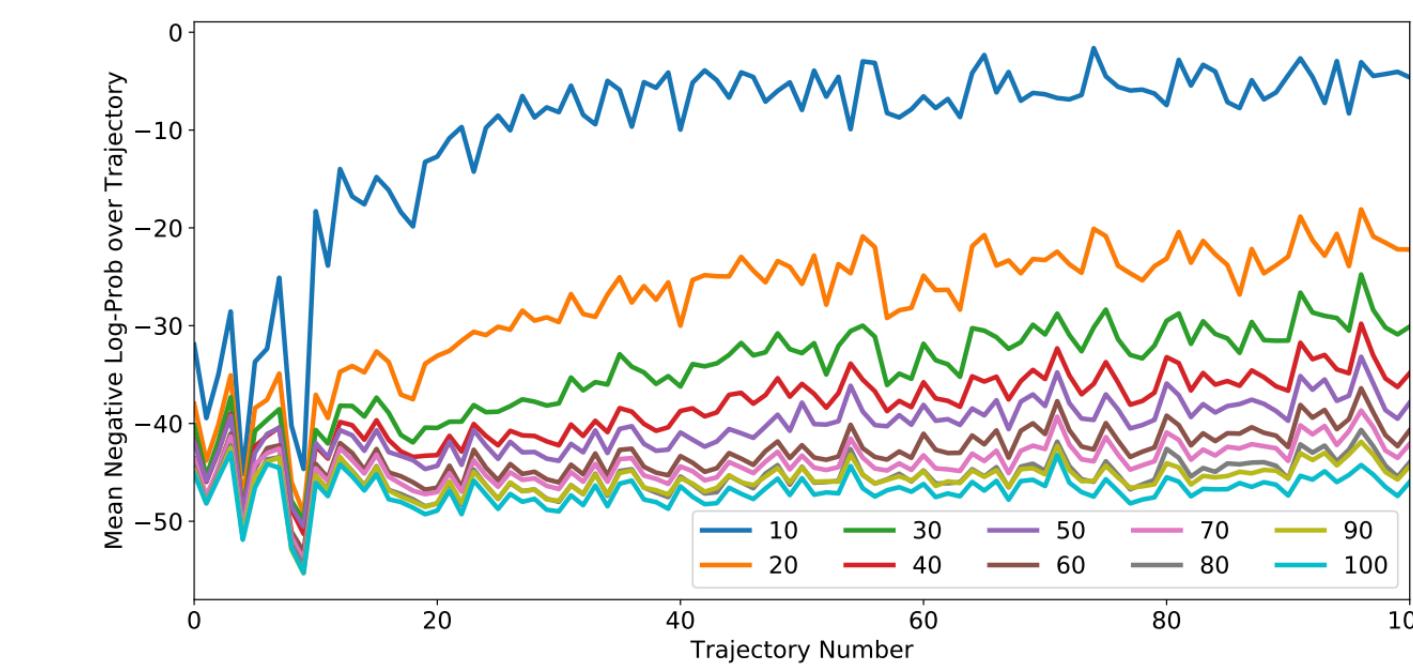
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu



Case study: PETS



(a) Mean squared error.



(b) Negative log likelihood.

Concluding thoughts

