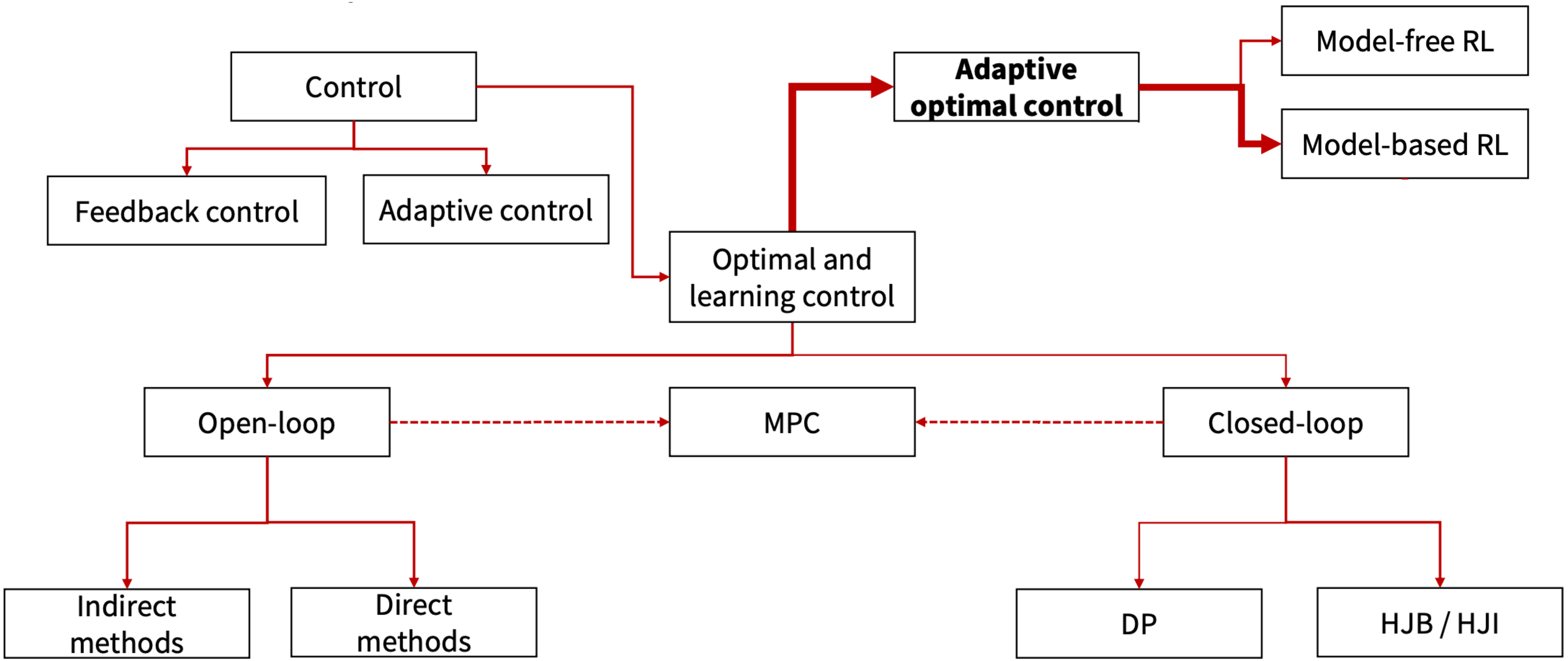
AA203 Optimal and Learning-based Control
Lecture 17
Model-based RL

Autonomous Systems Laboratory
Daniele Gammelli

Stanford University

Autonomous Systems Laboratory
Stanford Aeronautics & Astronautics

Roadmap


Control
Feedback control
Adaptive control
Optimal and learning control
Adaptive optimal control
Model-free RL
Model-based RL
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
DP
HJB / HJI

Recap: Model-free RL

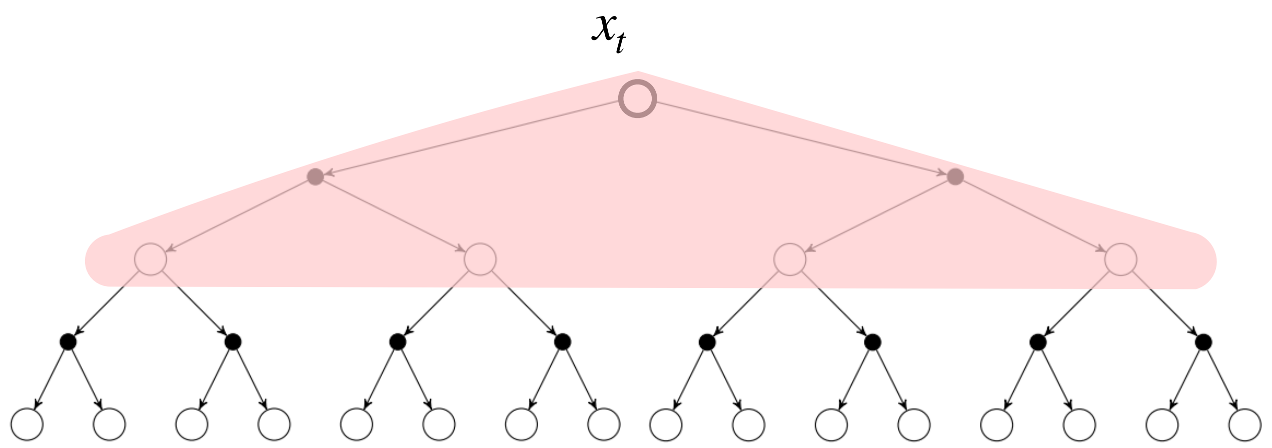
- We discussed different ways to estimate value functions

Dynamic Programming

Exact

Requires knowledge of MDP

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



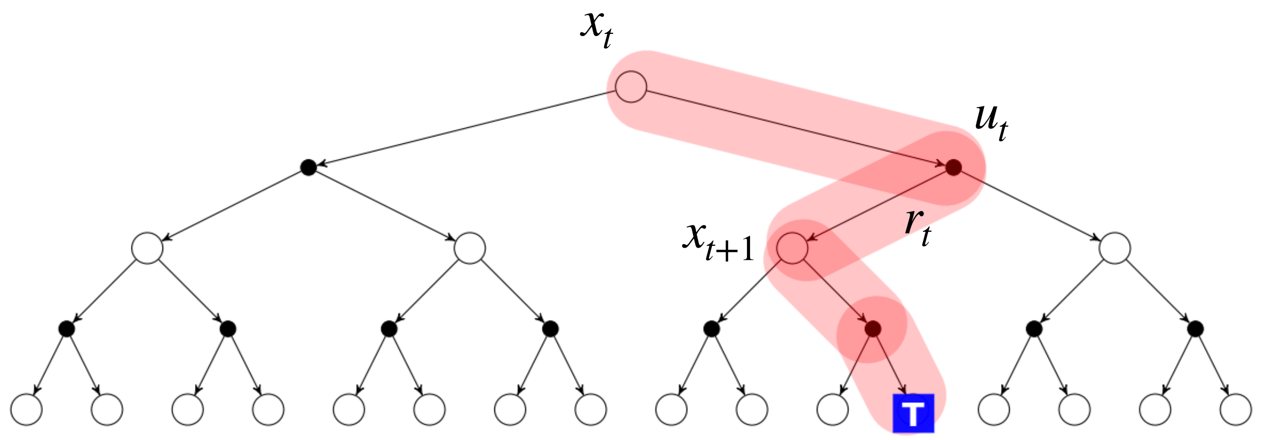
$$\hat{Q}(x_t, u_t) \leftarrow \mathbb{E}\left[R_t + \gamma \hat{Q}(x_{t+1}, u_{t+1})\right]$$

Monte Carlo

Unbiased

High variance; must reach terminal state

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



Terminal state

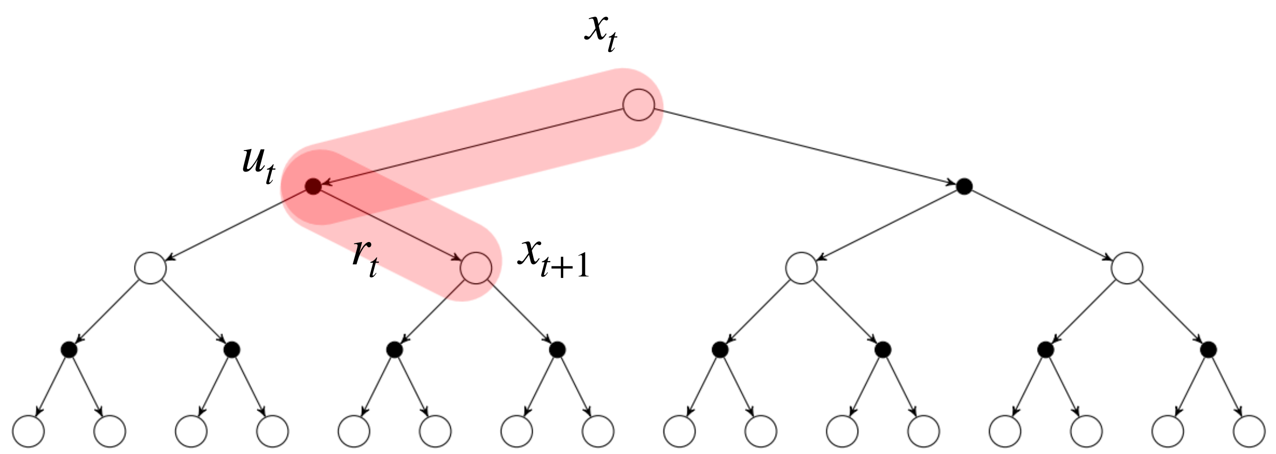
$$\hat{Q}(x_t, u_t) \leftarrow \hat{Q}(x_t, u_t) + \alpha\left(G_t - \hat{Q}(x_t, u_t)\right)$$

Temporal-Difference

Low variance; can learn online

Biased

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



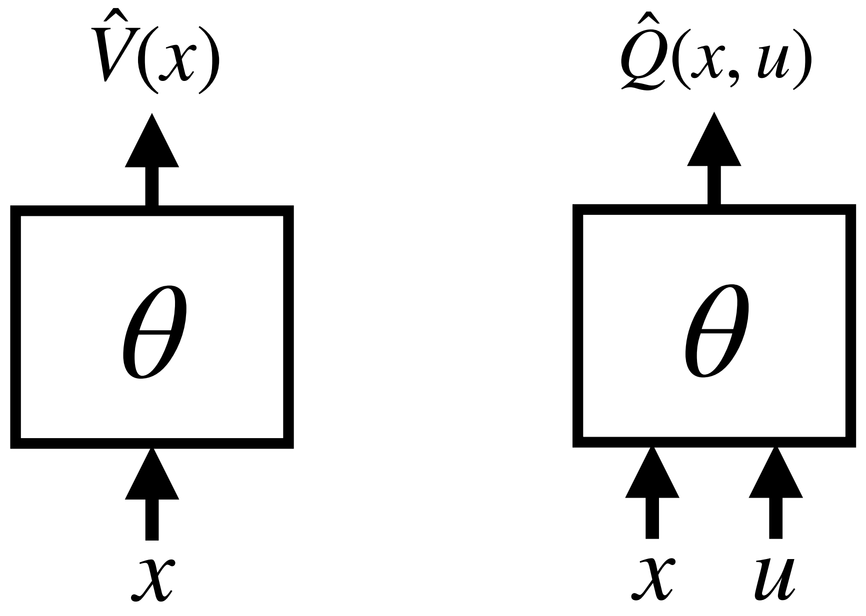
$$\hat{Q}(x_t, u_t) \leftarrow \hat{Q}(x_t, u_t) + \alpha\left(R_t + \gamma \hat{Q}(x_{t+1}, u_{t+1}) - \hat{Q}(x_t, u_t)\right)$$

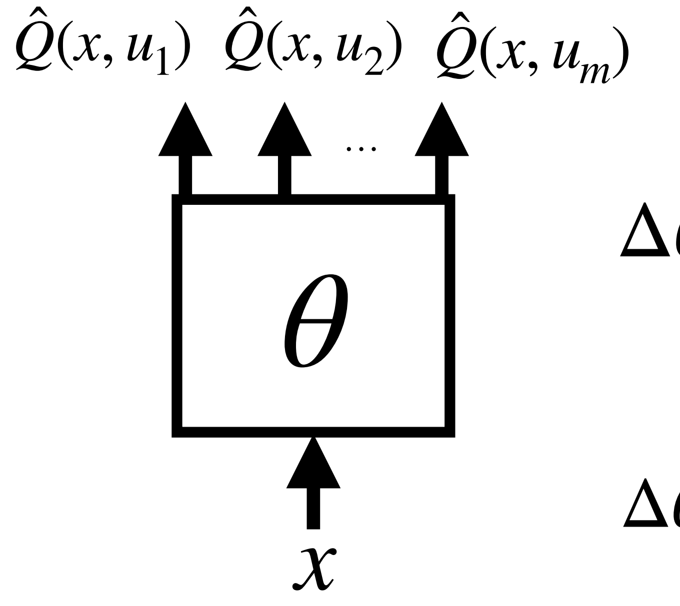
- And how to scale these ideas through function approximation

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:





MC update

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

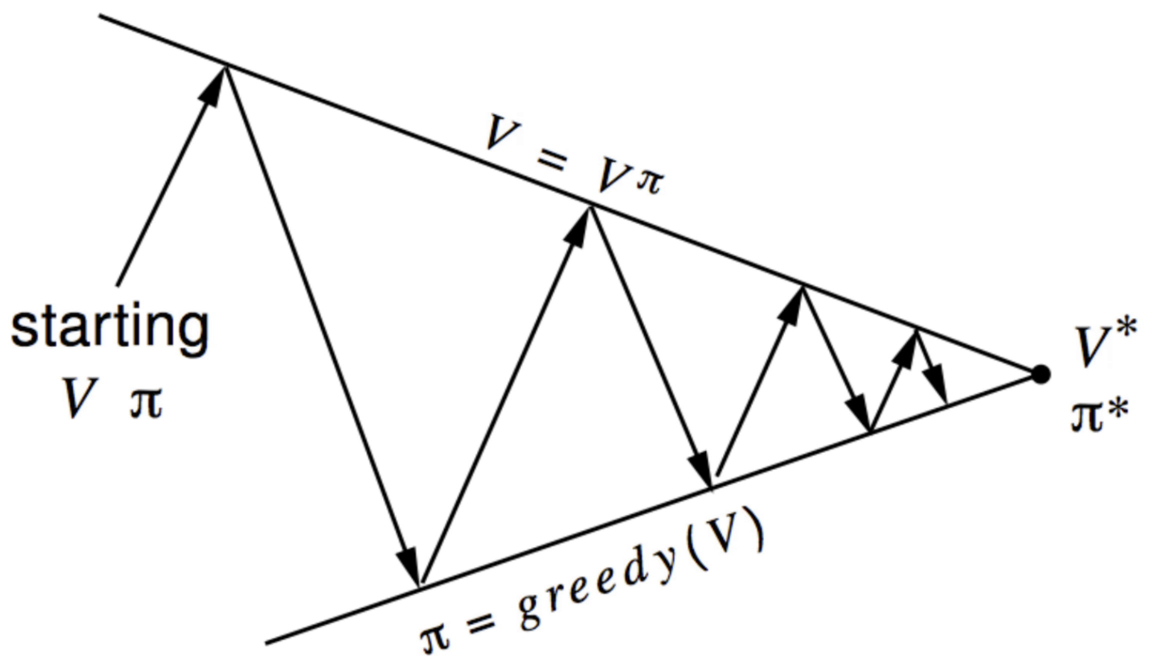
TD update

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

Recap: Model-free RL

Value-based methods

- Generalized Policy Iteration



- Sarsa & Q-learning

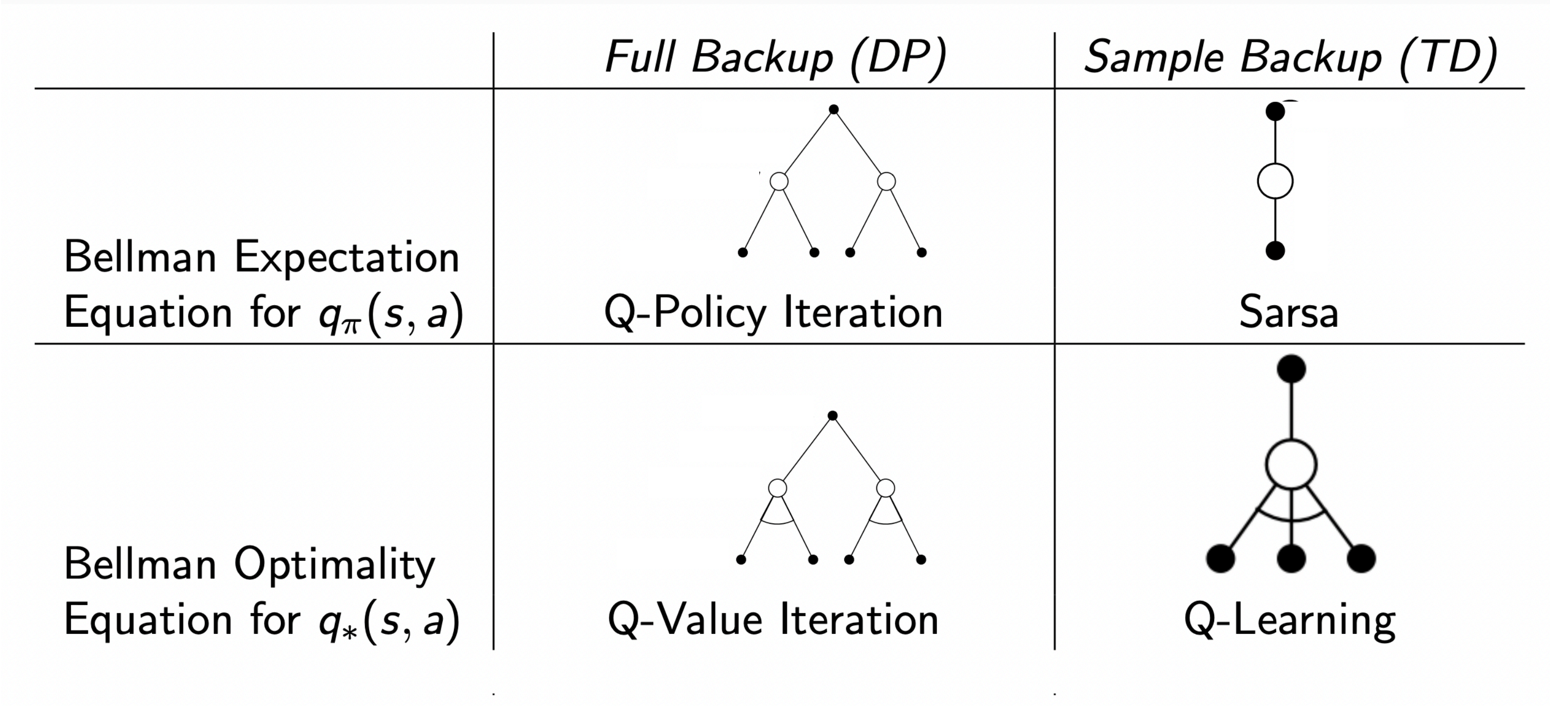
SARSA: on-policy

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

Q-learning: off-policy

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

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

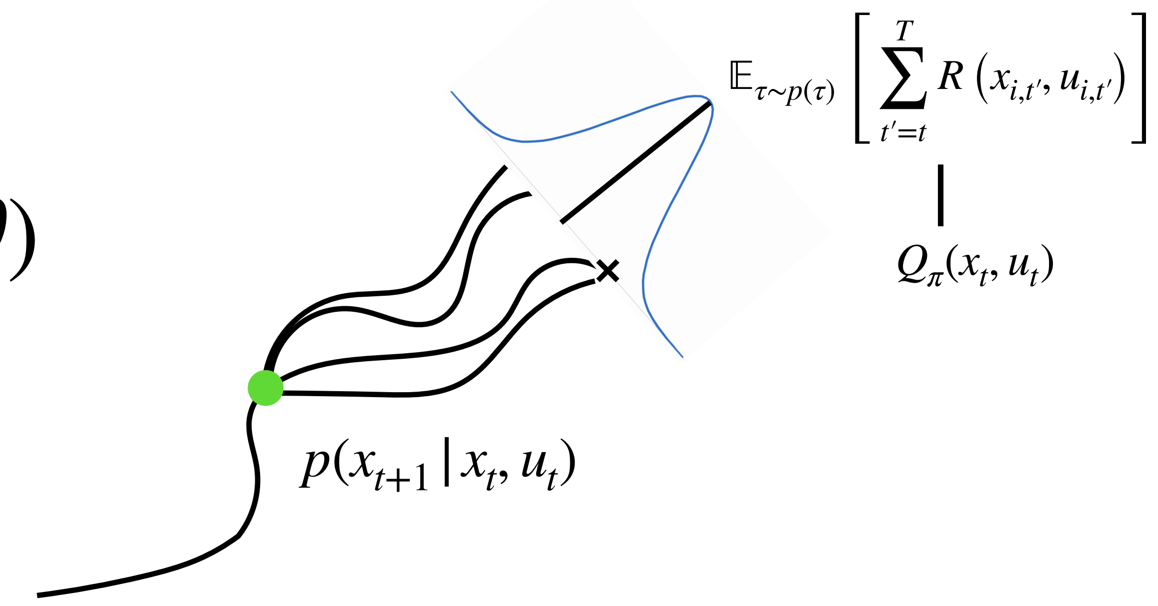


- Deep RL:
  (1) Use **deep neural nets** to represent Q_θ
  (2) Uses **experience replay** and **fixed Q-targets**

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

$$\theta^* = \underbrace{\arg\max_{\pi} \mathbb{E}_{\tau\sim p(\tau)} \left[\sum_{t\geq 0} \gamma^t R\left(x_t, u_t\right) \right]}_{J(\theta)}$$

(1) estimate its gradient $\nabla_\theta J(\theta)$
(2) do approximate gradient ascent on $J(\theta)$: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Problem: high variance of PG

Solution: baselines, "critics"

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

Maximum Likelihood: $\nabla_\theta J_{MLE}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(u_{i,t} \mid x_{i,t}\right)\right)\right]$

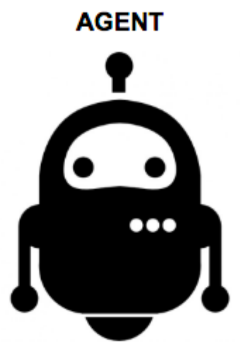
"Change parameters θ s.t. trajectories with higher reward have higher probability"

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

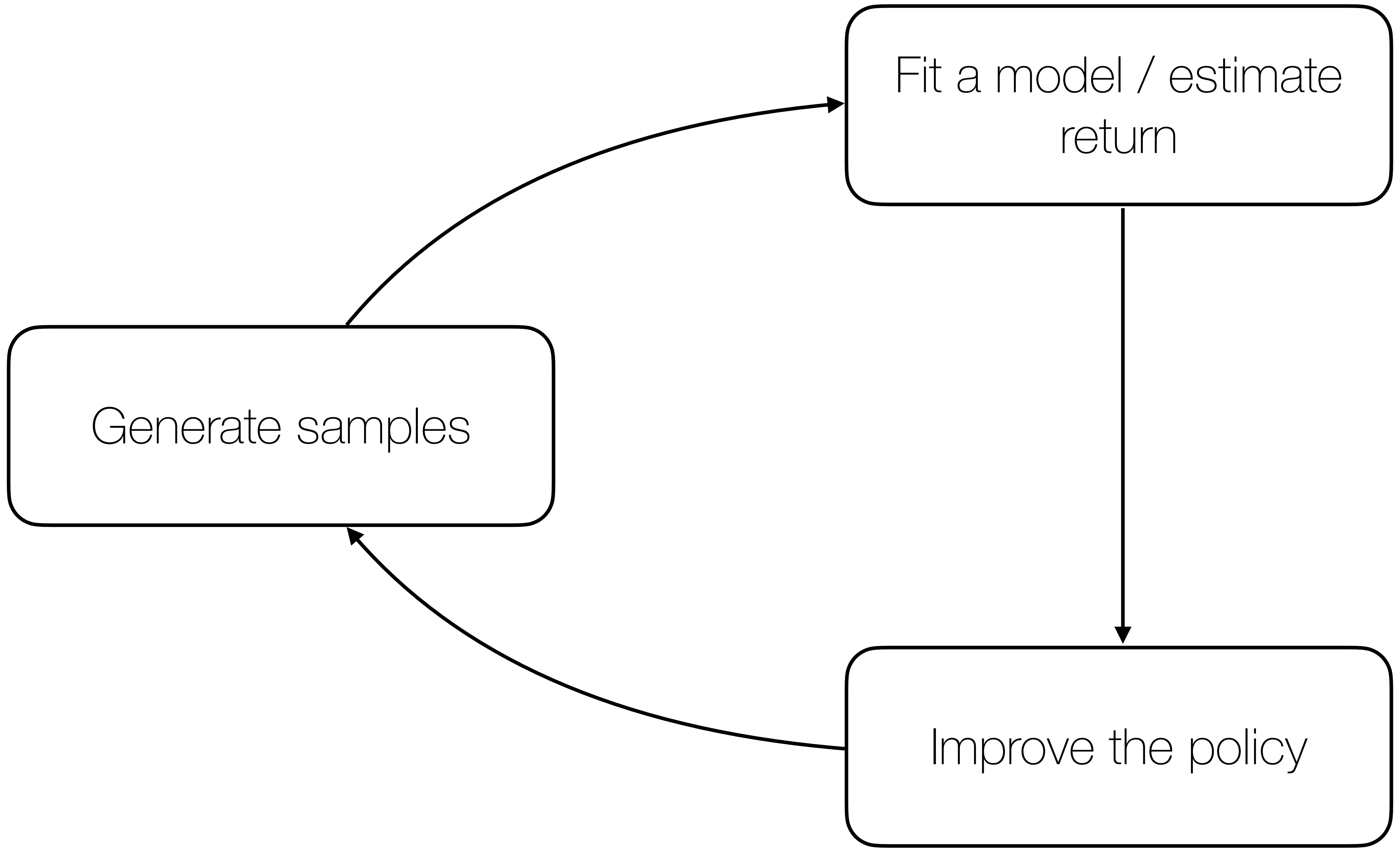
Policy Optimization

Recap: The skeleton of an RL algorithm

$\pi(u_t \mid x_t)$



$\tau = (x_0, u_0, \ldots, x_N, u_N)$



$f_\theta\left(x_t\right) \approx V^\pi\left(x_t\right)$

$f_\theta\left(x_t, u_t\right) \approx Q^\pi\left(x_t, u_t\right)$

$f_\theta\left(x_t, u_t\right) \approx P\left(x_{t+1} \mid x_t, u_t\right)$

set $\pi\left(x_t\right) = \arg\max_a Q\left(x_t, u_t\right)$ (e.g., Q-learning, DQN)

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}\left[\sum_t r\left(x_t, u_t\right)\right]$ (e.g., PG, A2C, A3C)

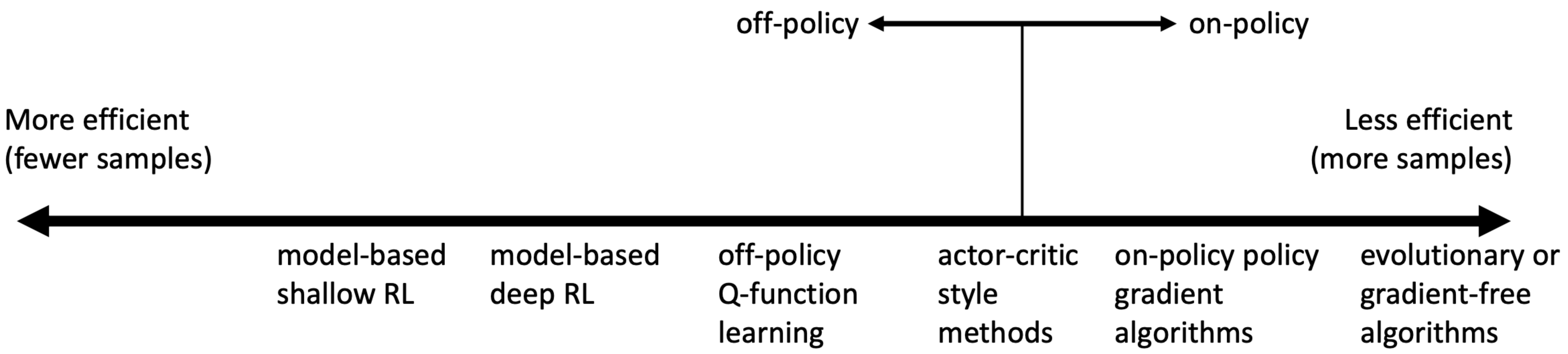
Recap: Why so many RL algorithms?

- **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?

Recap: Comparison: sample efficiency

- Sample efficiency = how many samples do we need to get a good policy?

- Crucial question: is the algorithm *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!

Recap: stability and ease of use

- Does it converge?
- And if it does, to what?
- Does it *always* converge?

- Supervised learning: almost always gradient descent
- Reinforcement learning: often not gradient descent
    - Q-learning: fixed point iteration
    - Model-based RL: model estimator is not optimized for expected reward
    - Policy gradient actually is gradient descent (but can be sample inefficient)

Outline (from last week)

Intro to policy gradients

- REINFORCE algorithm
- Reducing variance of policy gradient

Actor-Critic methods

- Advantage
- Architecture design

Deep RL Algorithms & Applications

Practical implementation (and alternative formulation)

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

$$L^{PG}(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\log p_\theta(\tau) A(\tau) \right]$$

- But we don't want to optimize it too far, since we are not working with the *true* advantage, rather with a noisy estimate
- Let's define an alternative loss

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

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

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

Trust Region Policy Optimization (TRPO)

$$\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(u_t \mid x_t\right)}{\pi_{\theta_{old}}\left(u_t \mid x_t\right)} \hat{A}_t \right]$$

$$\text{subject to } \hat{\mathbb{E}}_t \left[\mathrm{KL}[\pi_{\theta_{old}}\left(\cdot \mid x_t\right), \pi_\theta\left(\cdot \mid x_t\right) \right] \leq \delta$$

- 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 CNNs and 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

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

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

PPO v2 - Clipped surrogate loss

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

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

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

A taxonomy of RL

RL Algorithms

Model-free

do not use dynamics $T(x_{t+1} \mid x_t, u_t)$

Policy optimization

directly maximize the RL objective

$$\mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t r\left(x_t, u_t\right) \right]$$

Value-based

policy implicitly defined via

$V(x)$ or $Q(x, u)$

set $\pi\left(\mathbf{x}_t\right) = \arg\max_u Q\left(\mathbf{x}_t, \mathbf{u}_t\right)$

Model-based

use dynamics $T(x_{t+1} \mid x_t, u_t)$

Learn the model

estimate

$f_\theta \approx T(x_{t+1} \mid x_t, u_t)$

Given the model

$T(x_{t+1} \mid x_t, u_t)$ is known

A taxonomy of RL

RL Algorithms

- Model-free — do not use dynamics $T(x_{t+1} \mid x_t, u_t)$
  - Policy optimization — directly maximize the RL objective $\mathbb{E}_{\tau \sim p_\pi(\tau)}\left[\sum_{t=0}^{H} \gamma^t r\left(x_t, u_t\right)\right]$
  - Value-based — policy implicitly defined via $V(x)$ or $Q(x, u)$; set $\pi\left(\mathbf{x}_t\right) = \arg\max_u Q\left(\mathbf{x}_t, \mathbf{u}_t\right)$
- Model-based — use dynamics $T(x_{t+1} \mid x_t, u_t)$
  - Learn the model — estimate $f_\theta \approx T(x_{t+1} \mid x_t, u_t)$
  - Given the model — $T(x_{t+1} \mid x_t, u_t)$ is known

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)

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)

<u>Approach 1:</u>
“Learn a model and use it to plan”

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)

<u>Approach 1:</u>
"Learn a model and use it to plan"

Next week

<u>Approach 2:</u>
"Learn a model and improve model-free learning"

General recipe

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

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

1. Run base policy $\pi_0(u_t \mid x_t)$ 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)

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

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

Will this work?

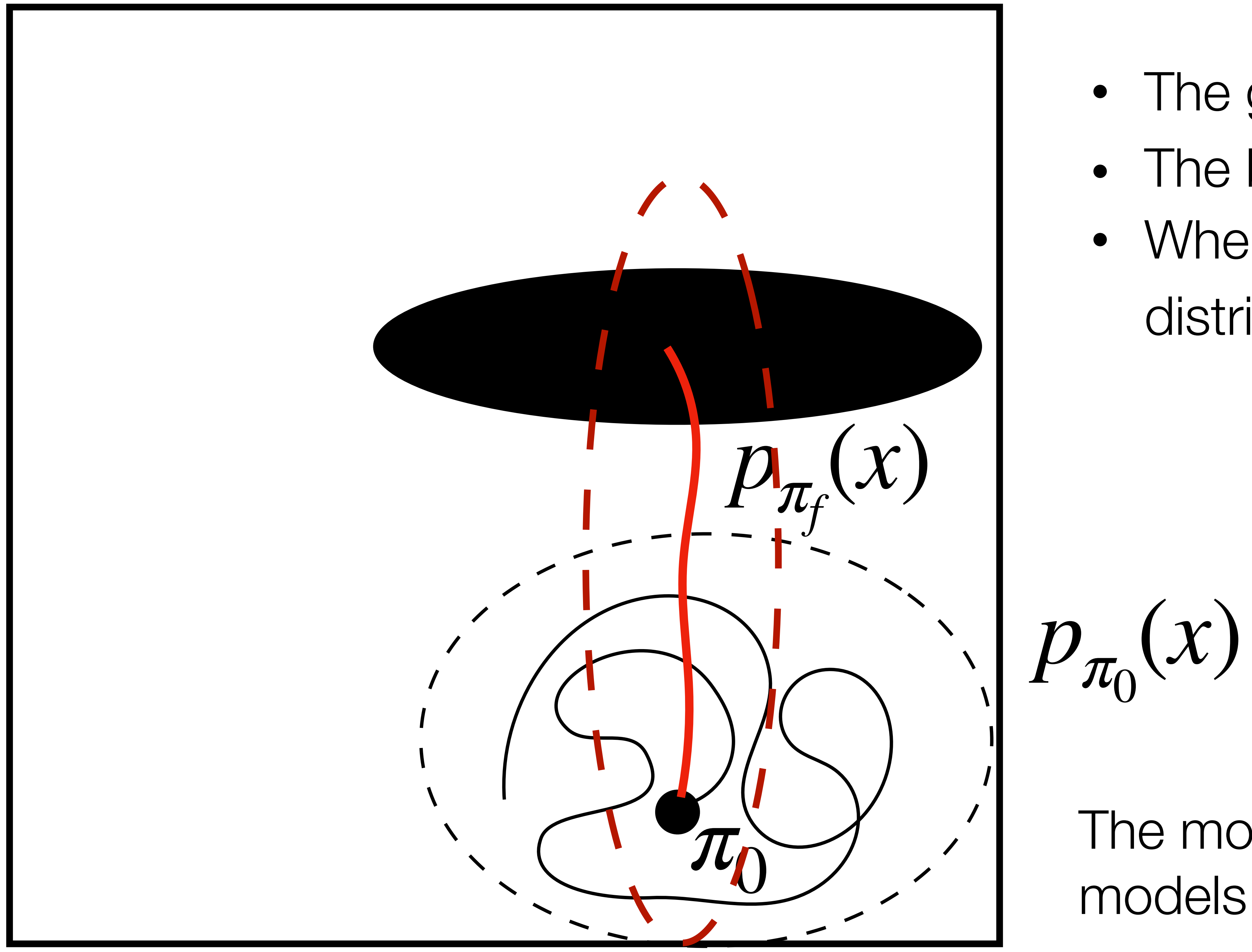
YES

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

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

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 $\mathscr{D}$ and update model **(i.e., gradually closing the gap between $p_{\pi_0}(x)$ and $p_{\pi_f}(x)$)**

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

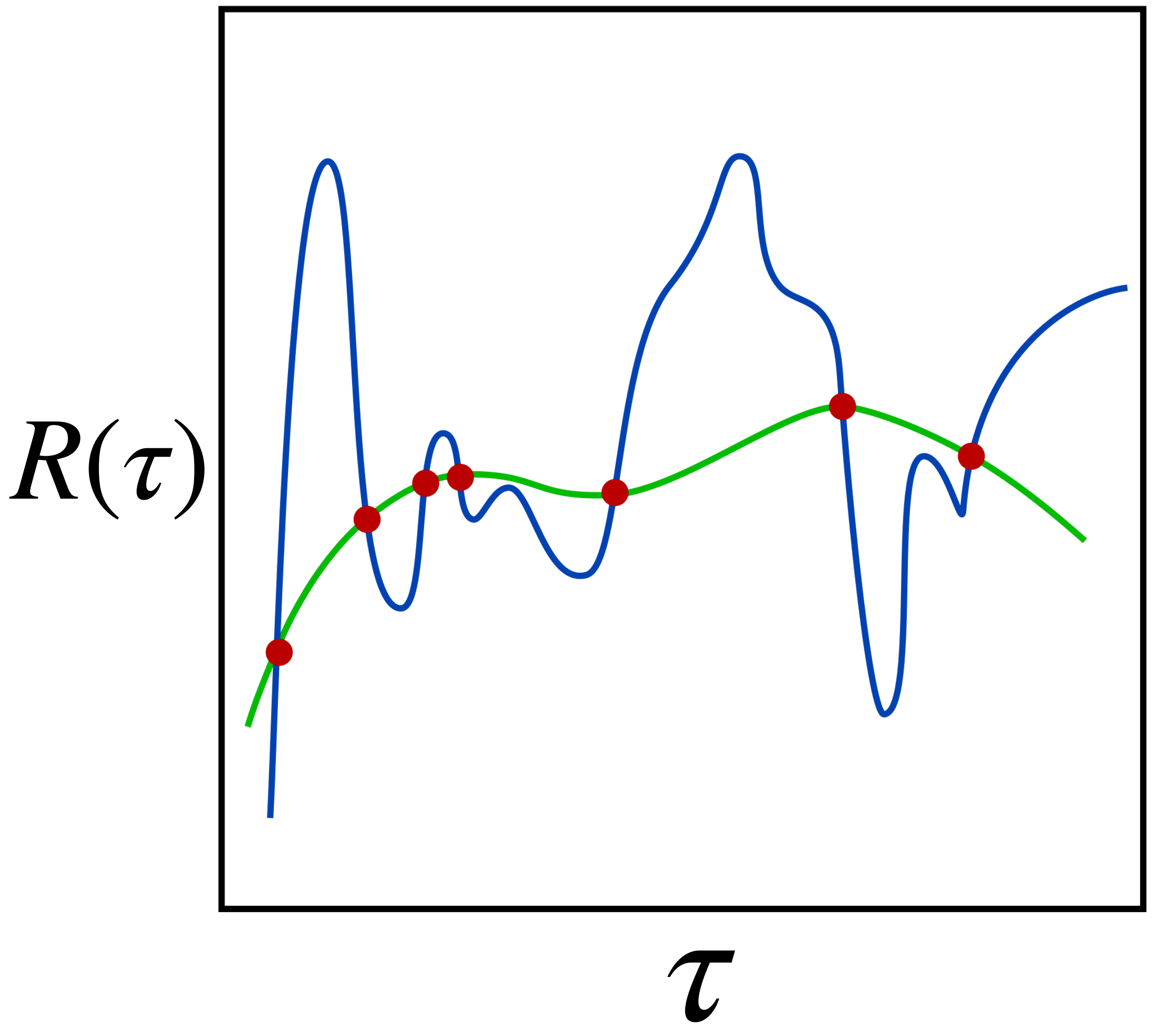
Examples & Applications (e.g., PETS)

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 $\pi_0(u_t \mid x_t)$ 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)
$$\theta^* = \arg\min_{\theta} \sum_i \left\| f_\theta\left(x_t, u_t\right) - x_{t+1} \right\|^2$$
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 $p_{\pi_f}(x)$)**

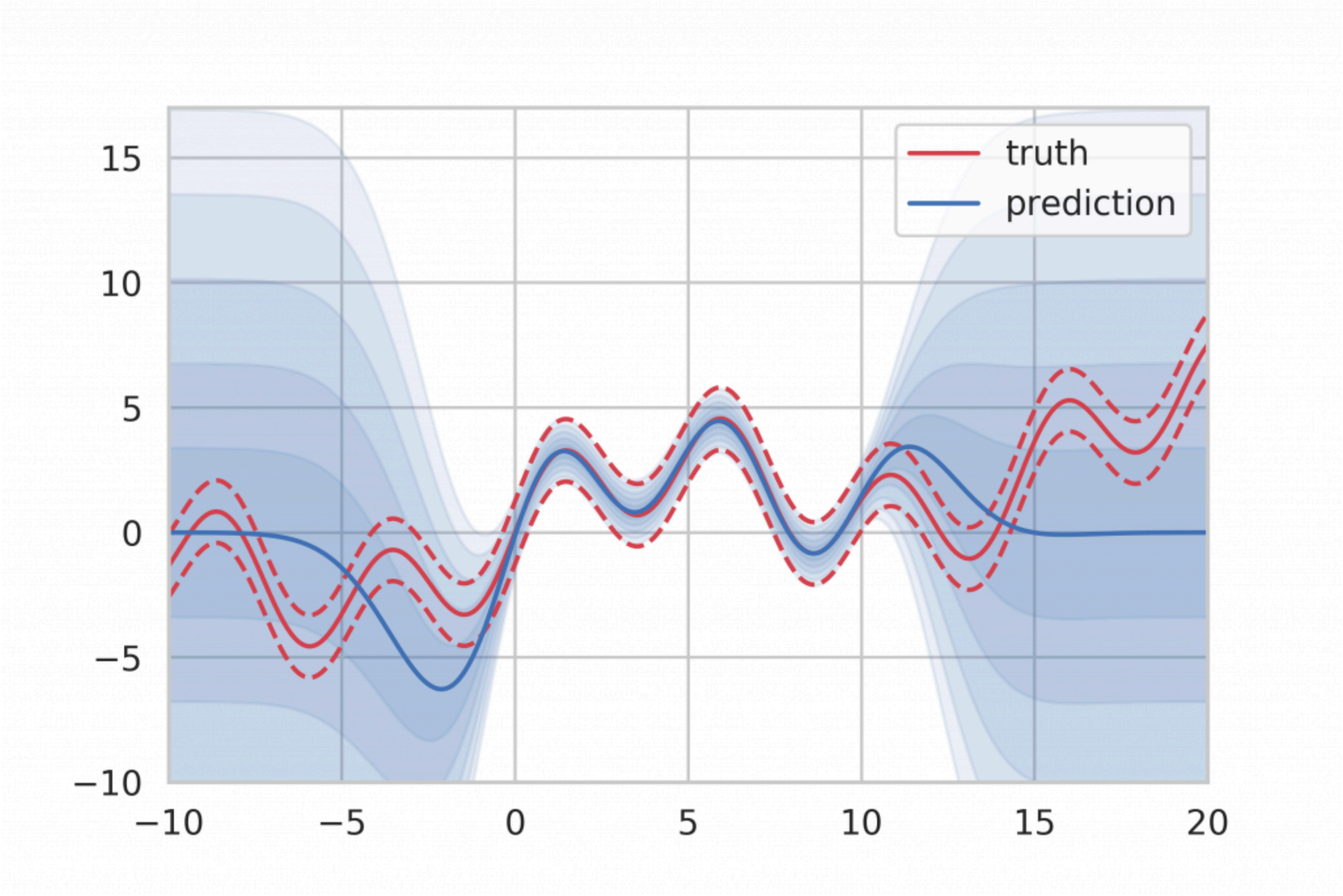


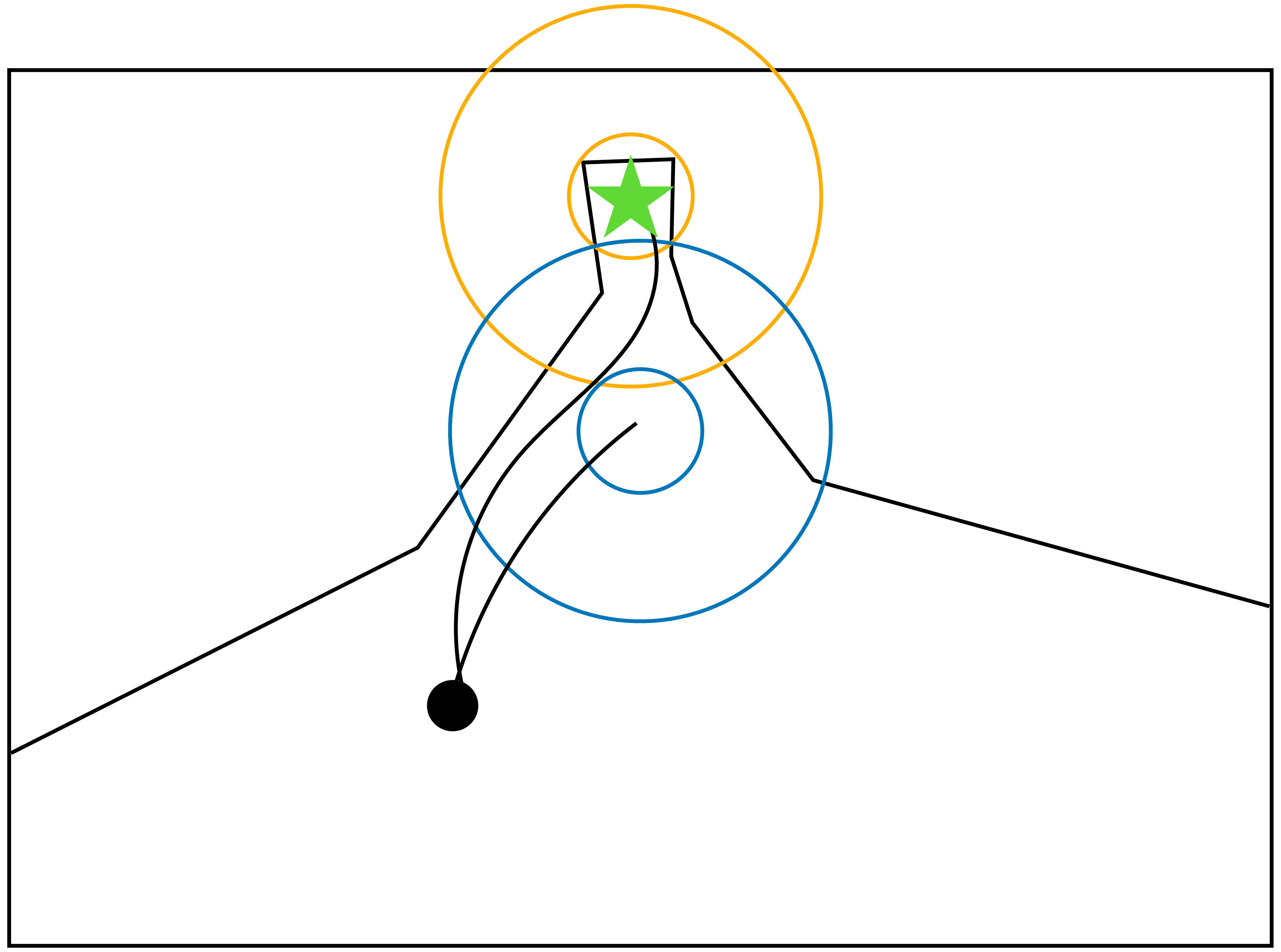
Problem: we'll likely *erroneously* exploit our model where it is less knowledgeable

(Possible) Solution: consider how "certain" we our 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

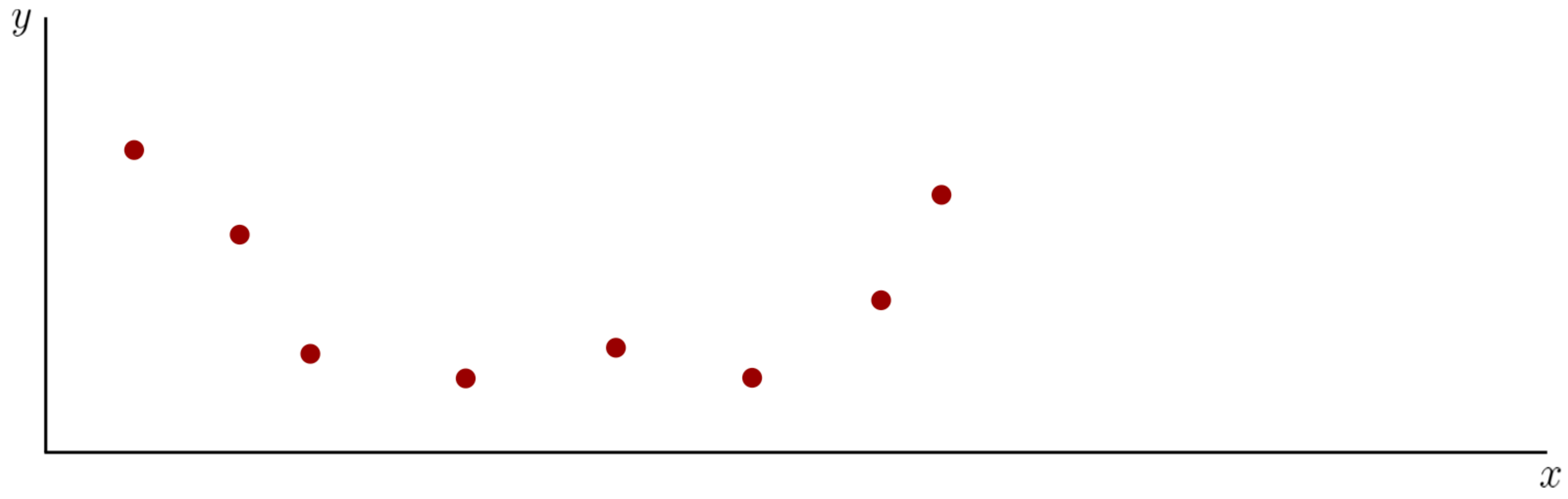




Expected reward under high-variance prediction is low

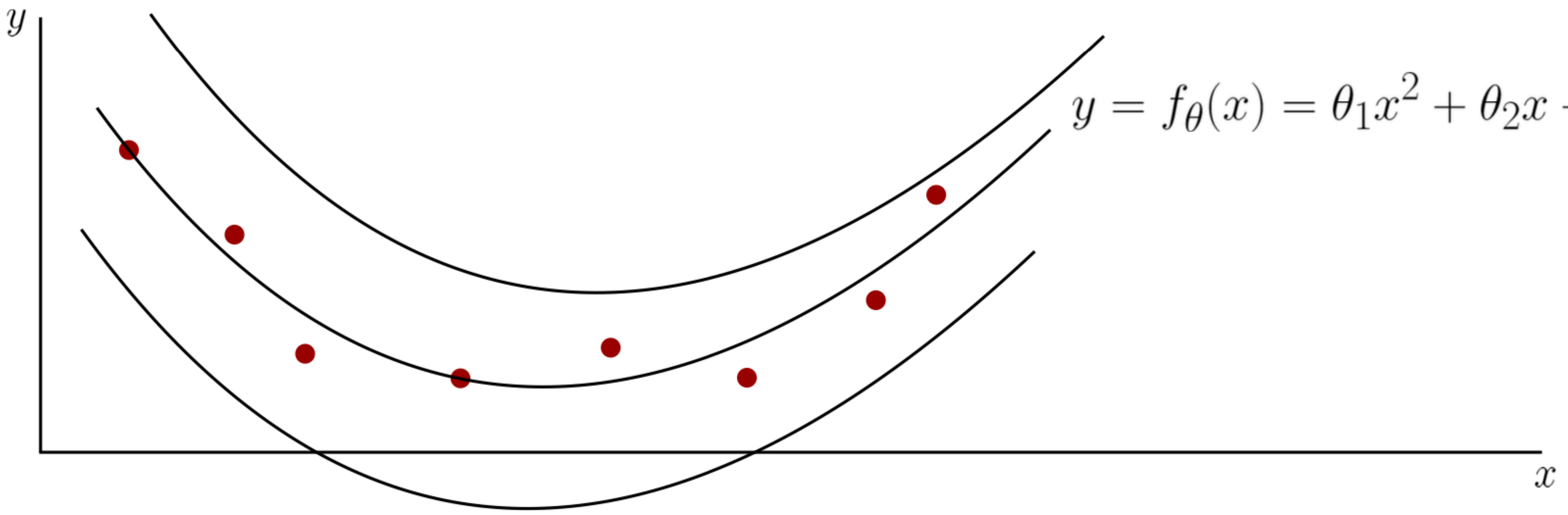
Learning from a probabilistic standpoint

- Let's consider regression as an example:



Learning from a probabilistic standpoint

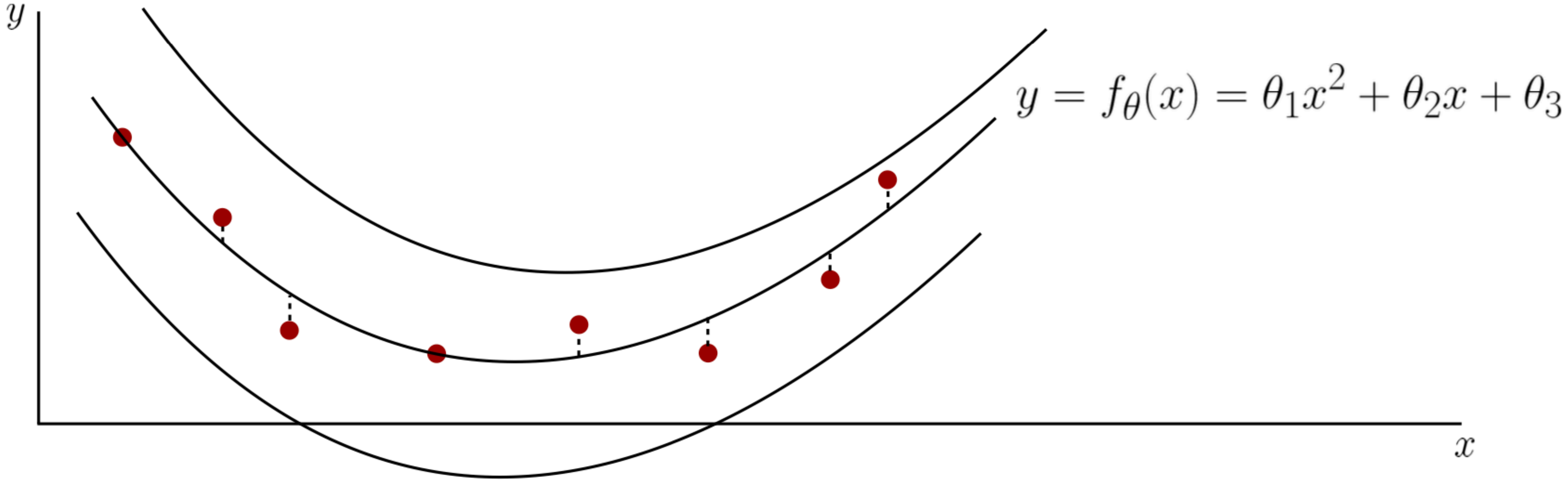
- Let's consider regression as an example:



$$y = f_\theta(x) = \theta_1 x^2 + \theta_2 x + \theta_3$$

Learning from a probabilistic standpoint

- Let's consider regression as an example:

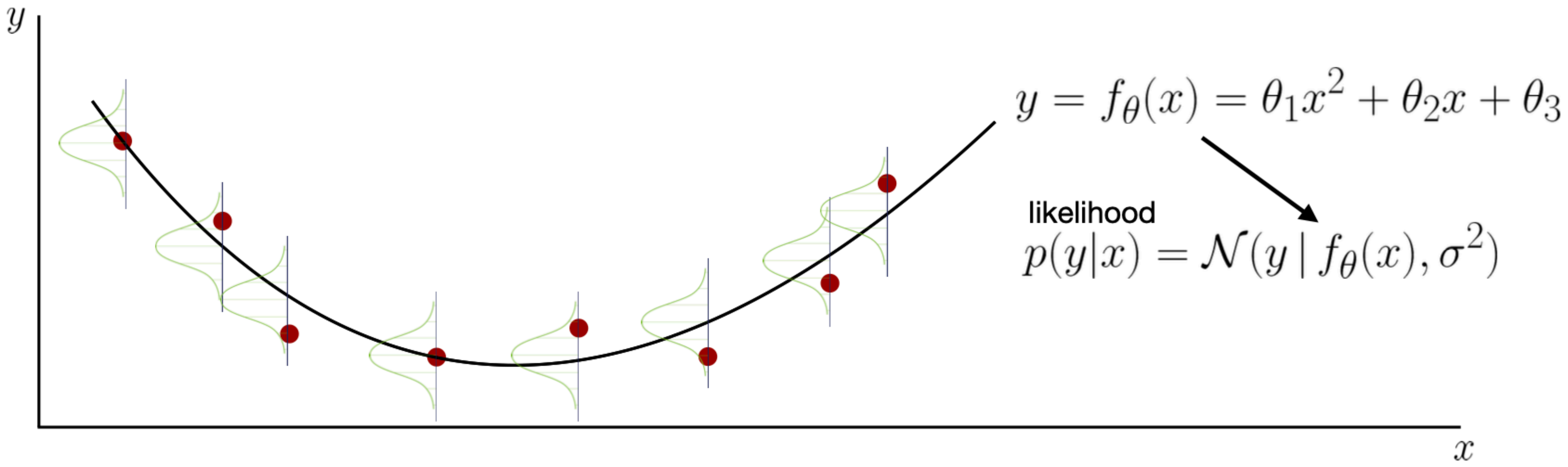


Learning through minimization of squared error

$$\theta^* = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$$

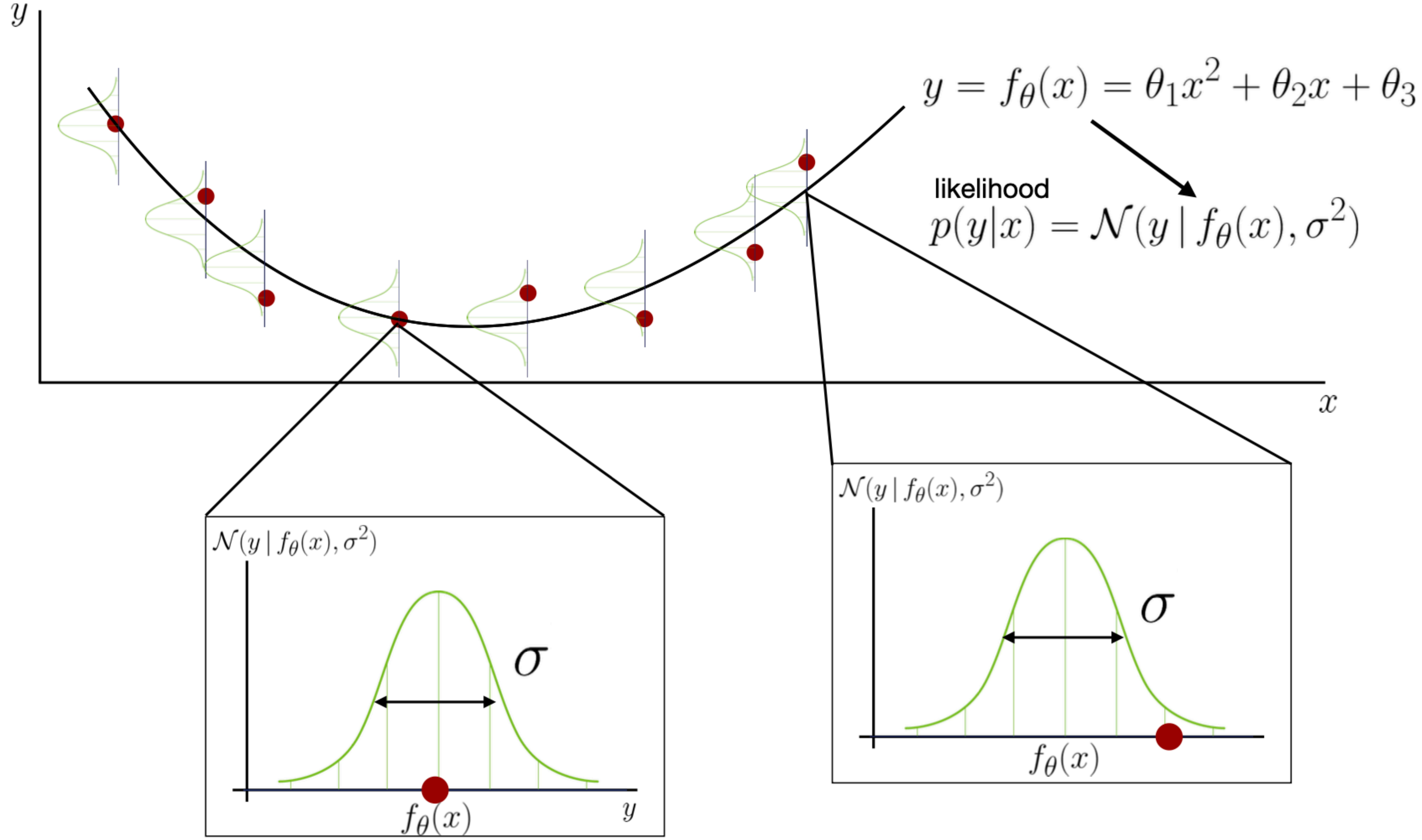
Learning from a probabilistic standpoint

- Let’s consider regression as an example:



Learning from a probabilistic standpoint

- Let’s consider regression as an example:

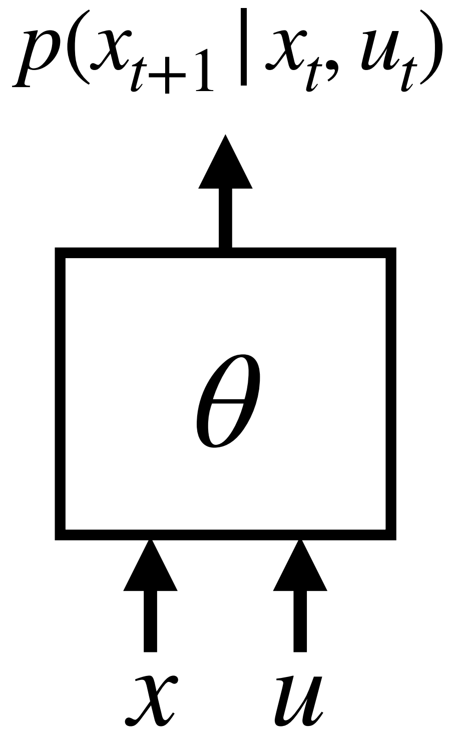


Learning through likelihood maximization

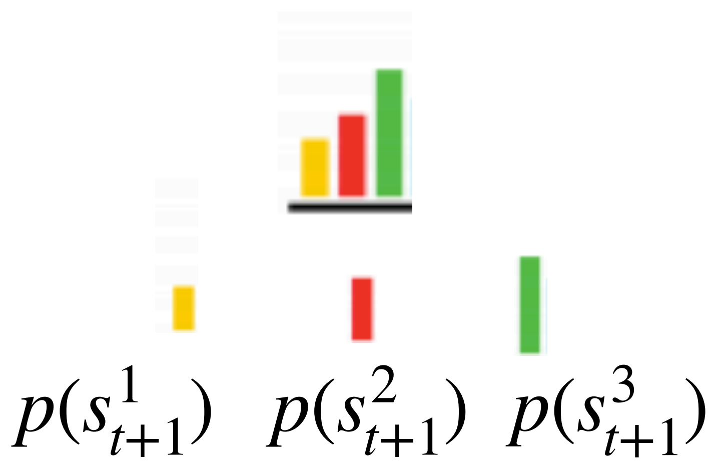
$$\theta^* = \arg\max_\theta \prod_{i=1}^n \mathcal{N}(y_i \mid f_\theta(x_i), \sigma^2)$$

How can we model uncertainty?

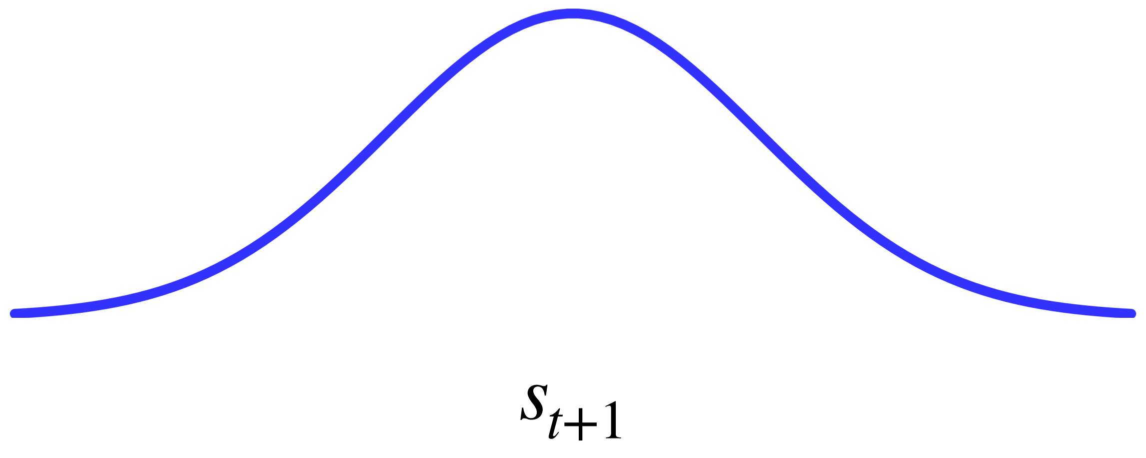
- **Idea 1:** use output entropy (spoiler: this does not work)
- Suppose we estimated a model, why not use its entropy?



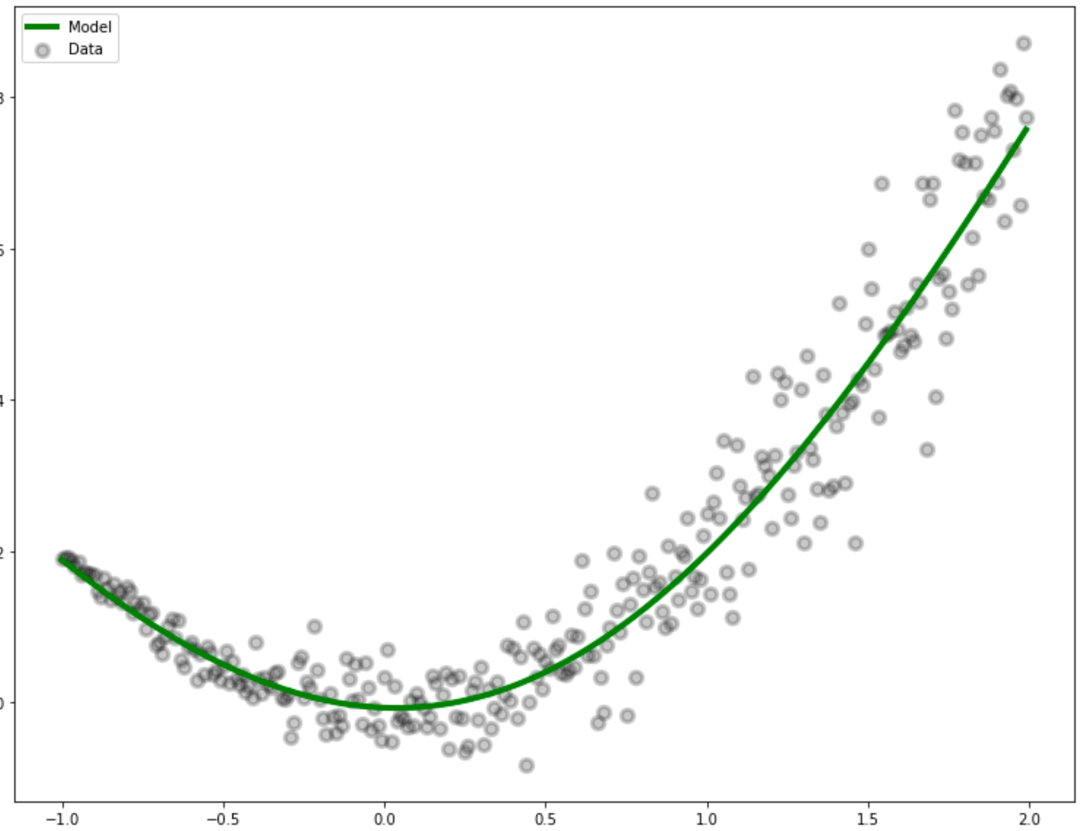
Discrete state-space



Continuous state-space

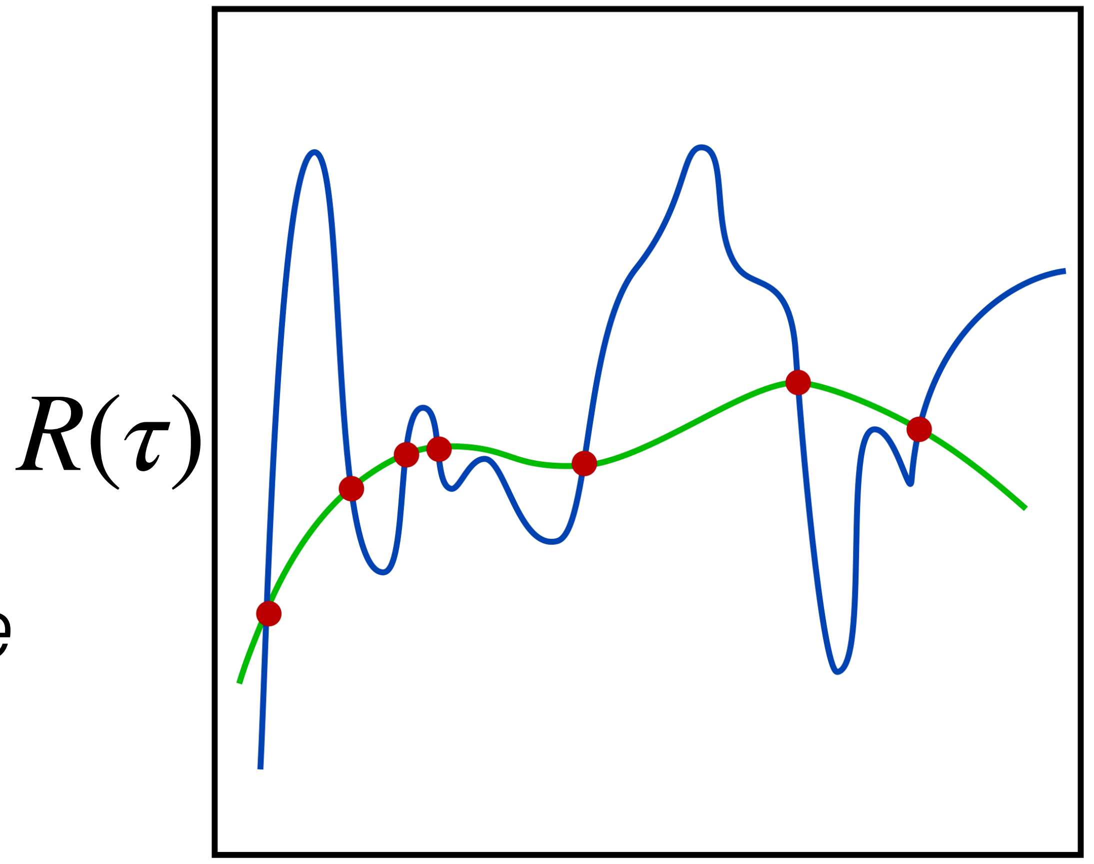


- Doing so will not take *epistemic* uncertainty into account



Aleatoric uncertainty: *"The process is intrinsically noisy"*

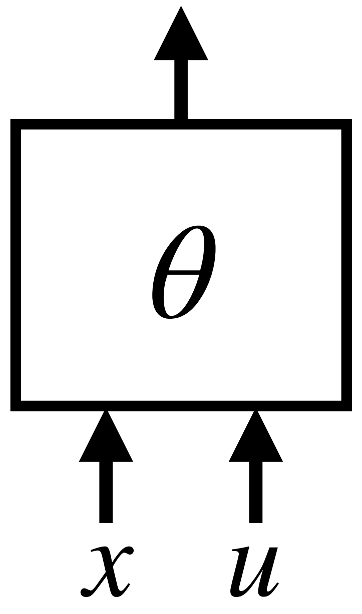
Epistemic uncertainty: *"Uncertainty about the model"*



How can we model uncertainty?

- **Idea 2:** estimate model uncertainty

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



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

$$\arg\max_{\theta} \log p(\mathscr{D} \mid \theta)$$

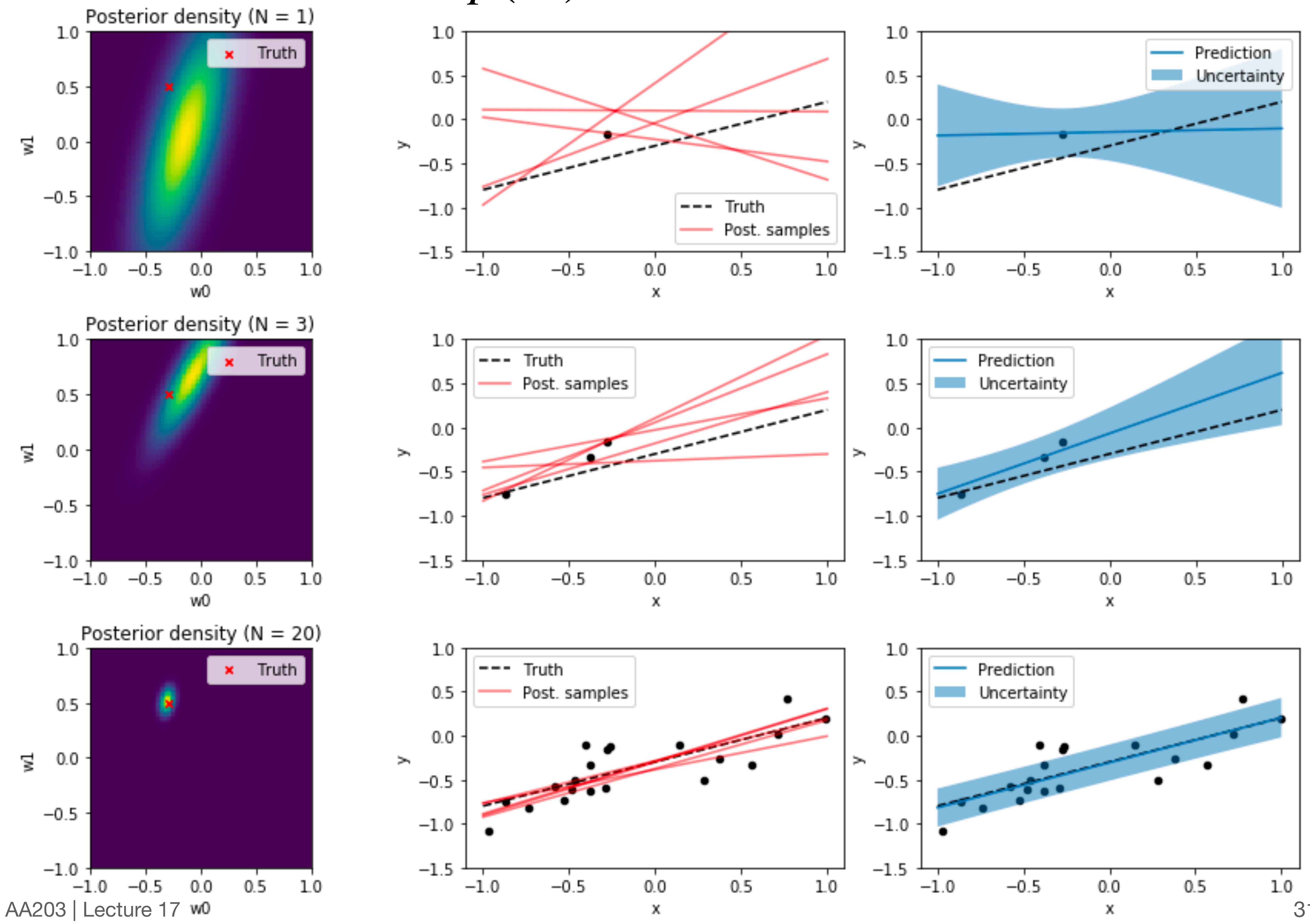
- To express model uncertainty means estimating:

$$p(\theta \mid \mathscr{D})$$

and predict according to the *predictive posterior distribution* $\int p\left(x_{t+1} \mid x_t, u_t, \theta\right) p(\theta \mid \mathscr{D}) d\theta$

Prior: $p(\theta)$, Likelihood: $p(\mathscr{D} \mid \theta)$, Posterior $p(\theta \mid \mathscr{D})$

Bayes' Theorem $p(\theta \mid \mathscr{D}) = \dfrac{p(\mathscr{D} \mid \theta)p(\theta)}{p(\mathscr{D})}$

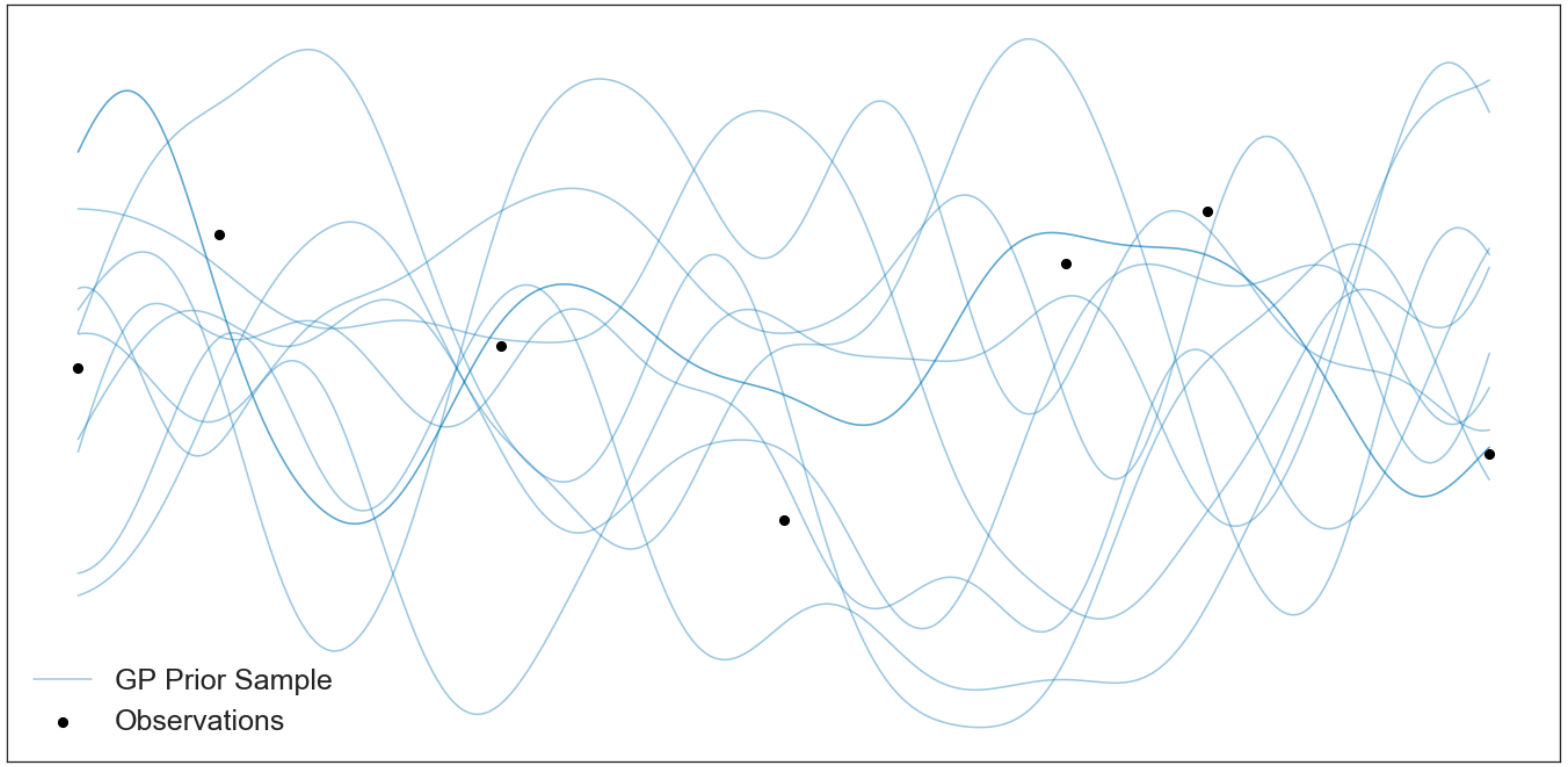


(1) Gaussian Processes

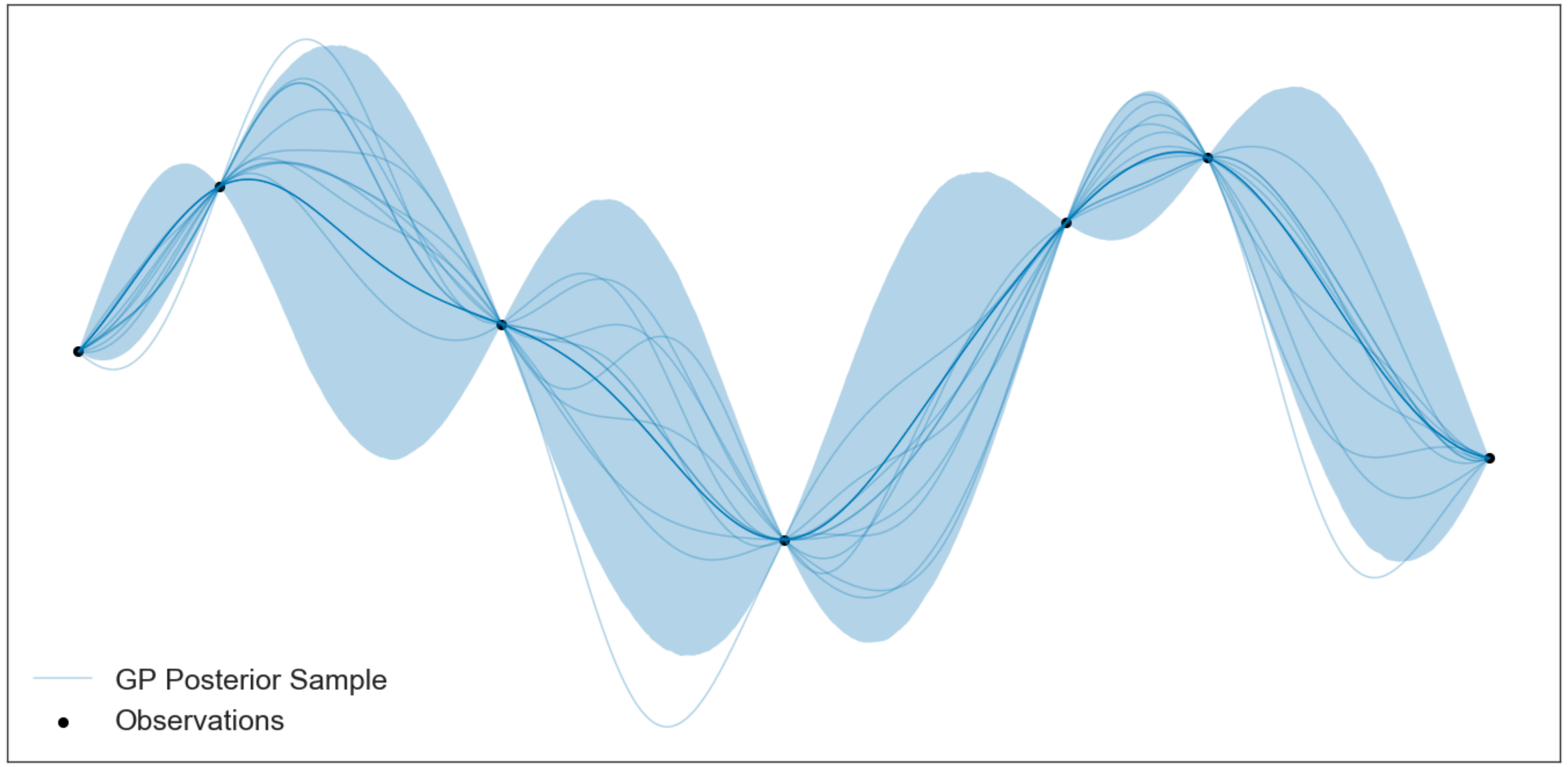
- Represent "distribution over functions"

Bayesian inference

Samples from **prior** distribution



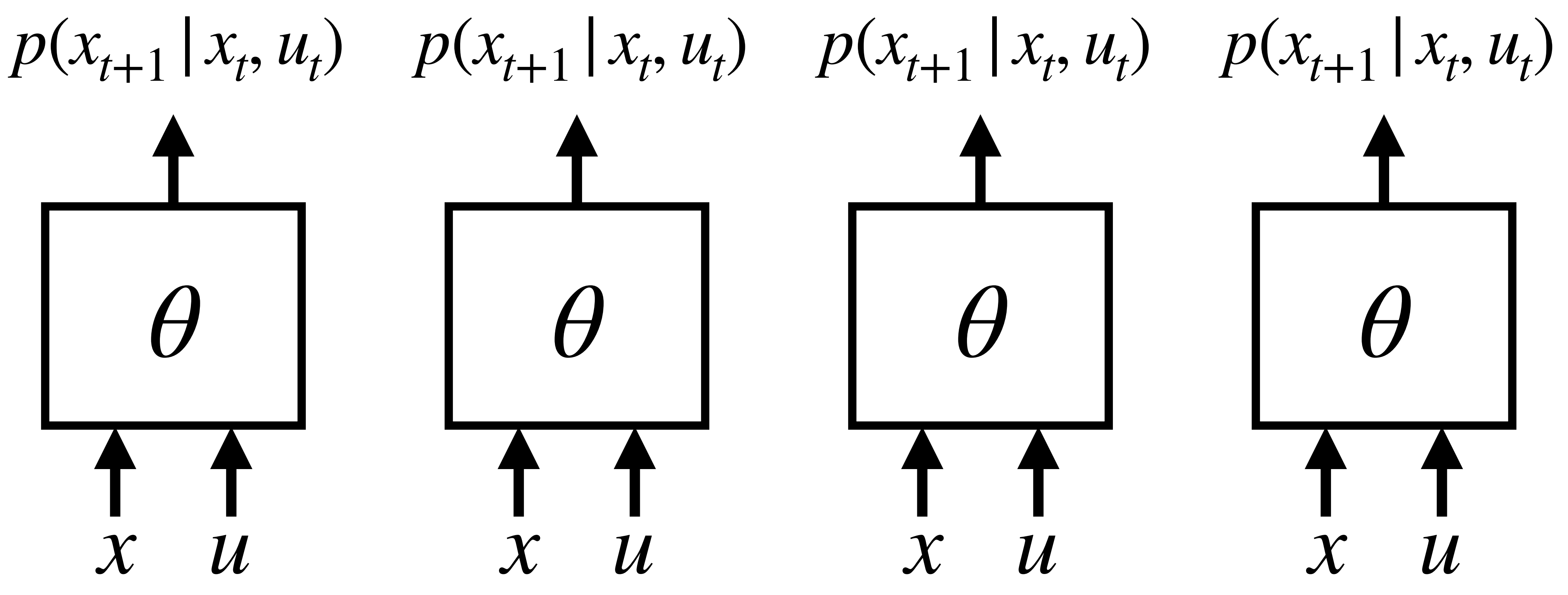
Samples from **posterior** distribution

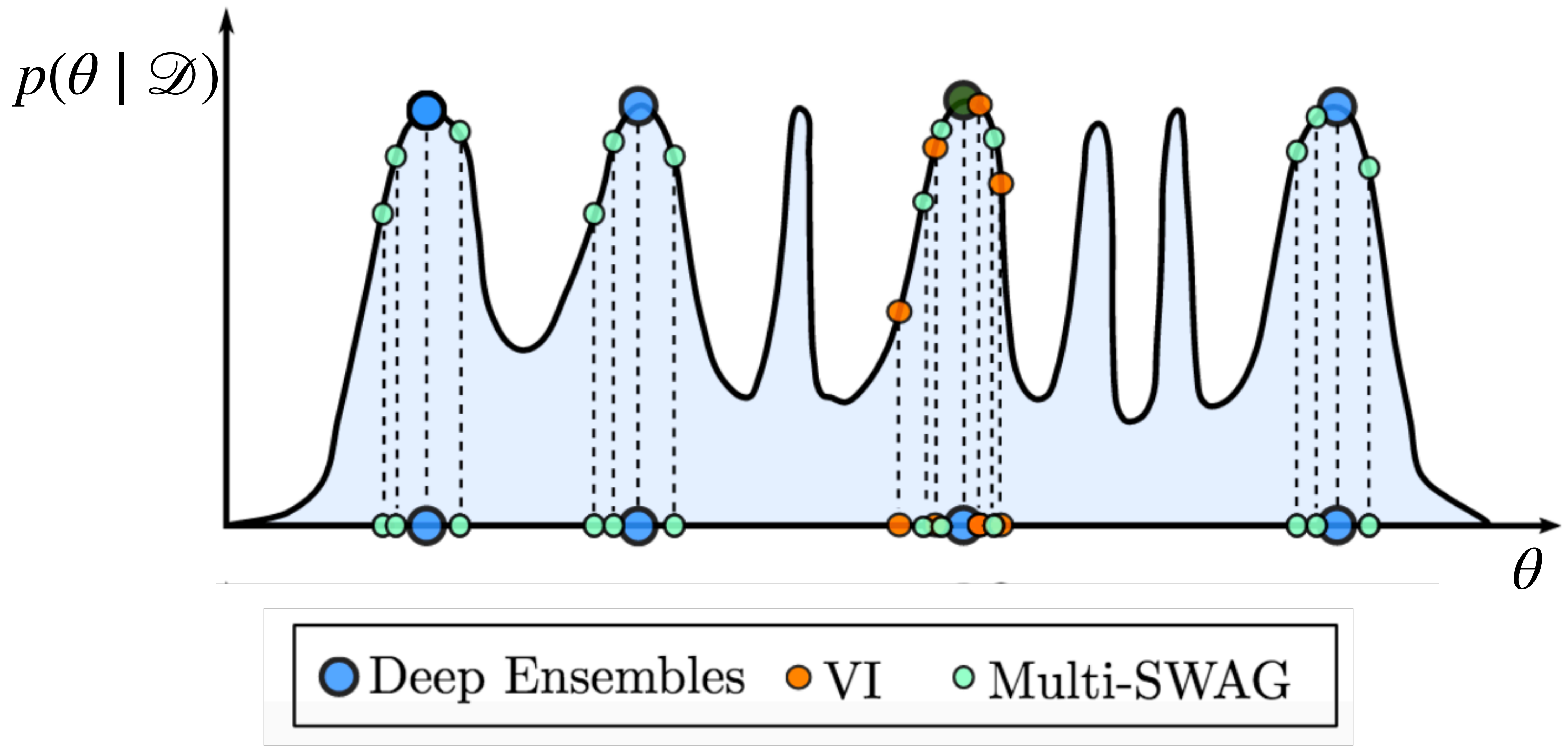


- Strengths
  - Data efficient
  - Exact posterior
  - Predictable behavior via the choice of kernel
- Weaknesses
  - High computational complexity
  - Cannot learn expressive features

(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:

$$p(\theta \mid \mathscr{D}) \approx \frac{1}{N} \sum_i \delta\left(\theta_i\right)$$

$$\int p\left(x_{t+1} \mid x_t, u_t, \theta\right) p(\theta \mid \mathscr{D}) d\theta \approx \frac{1}{N} \sum_i p\left(x_{t+1} \mid x_t, u_t, \theta_i\right)$$

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

Planning with uncertainty

- How can we use this additional knowledge in planning?

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

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

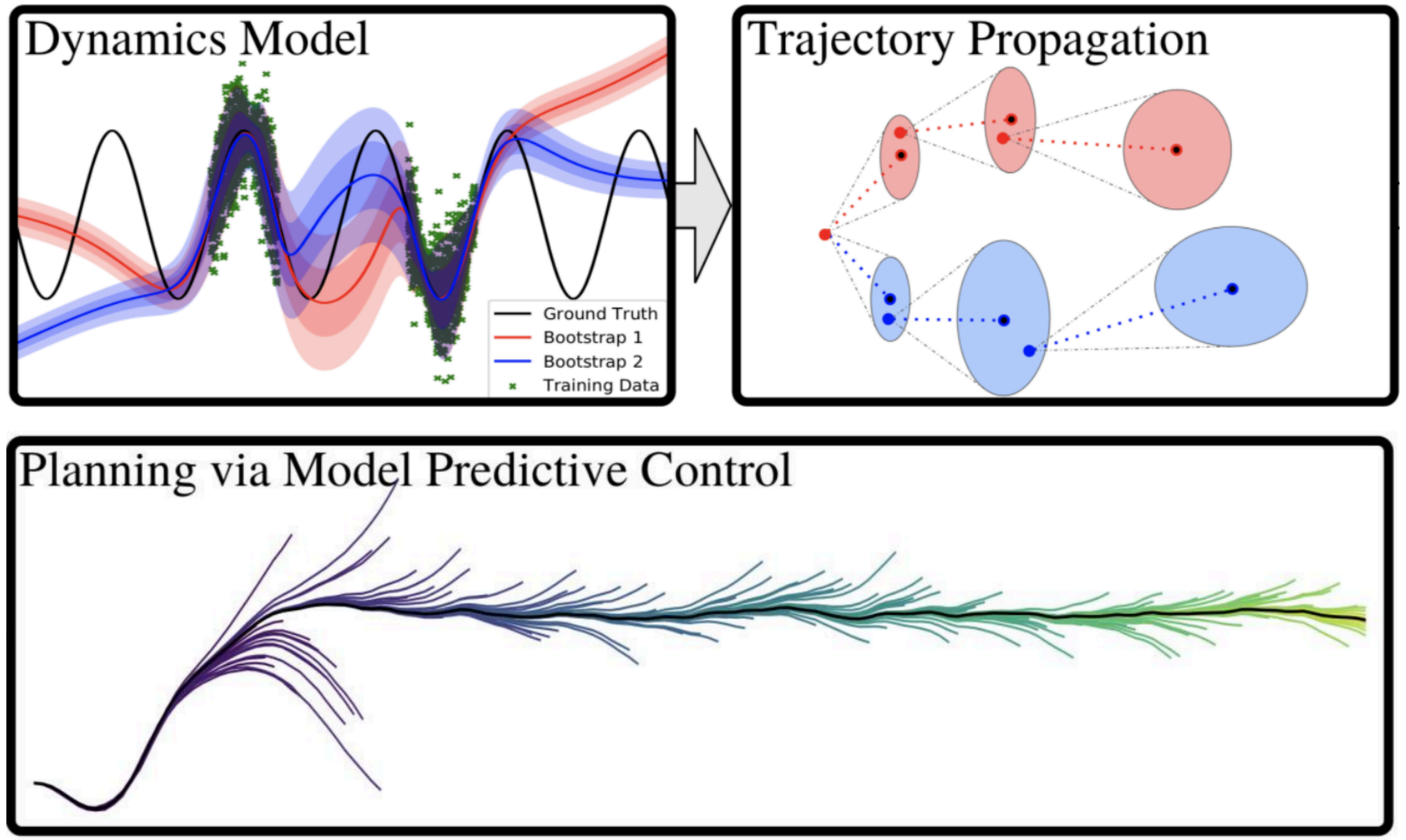
- Caveat: this is only a choice, one could think of other ways to approximate the posterior predictive distribution.
  - The general idea is that, when planning, we want to evaluate the expected reward under our model

Case study: PETS

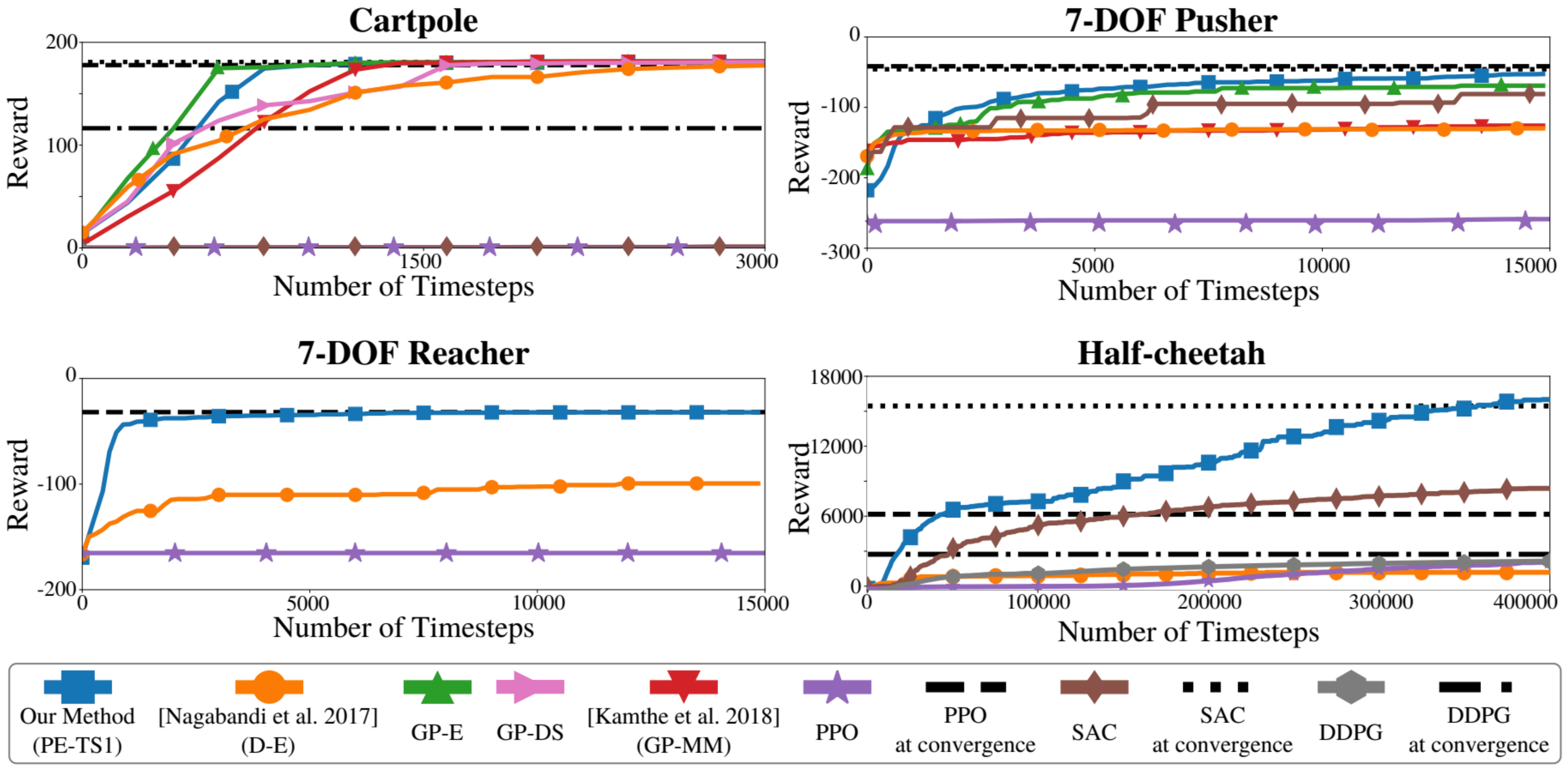
- Probabilistic Ensembles with Trajectory Sampling
- Key idea:
  - **Model:** Use ensemble of NNs to approximate posterior over model
  - **Propagation:** sample different models and use them to generate predictions of different “futures”
  - **Planning:** apply MPC (compute action sequence via sampling,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

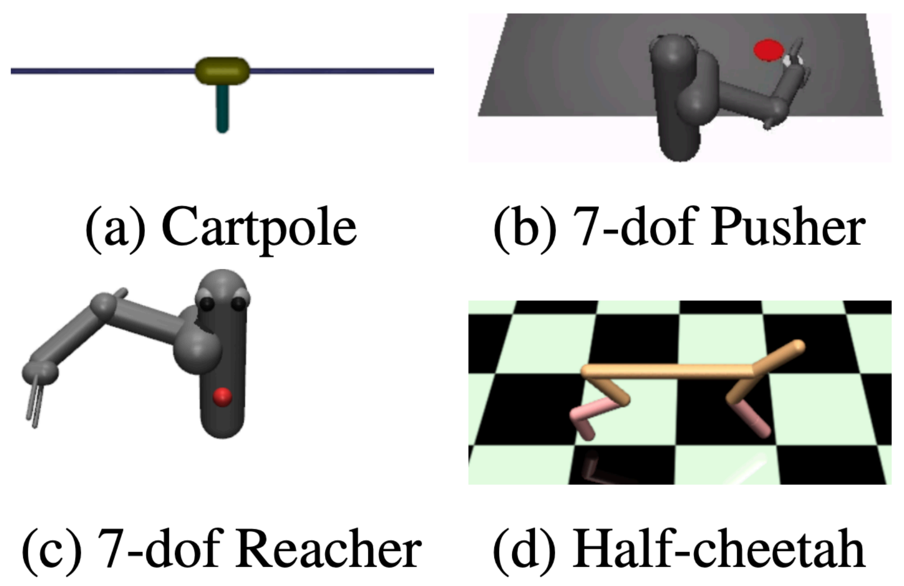
Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

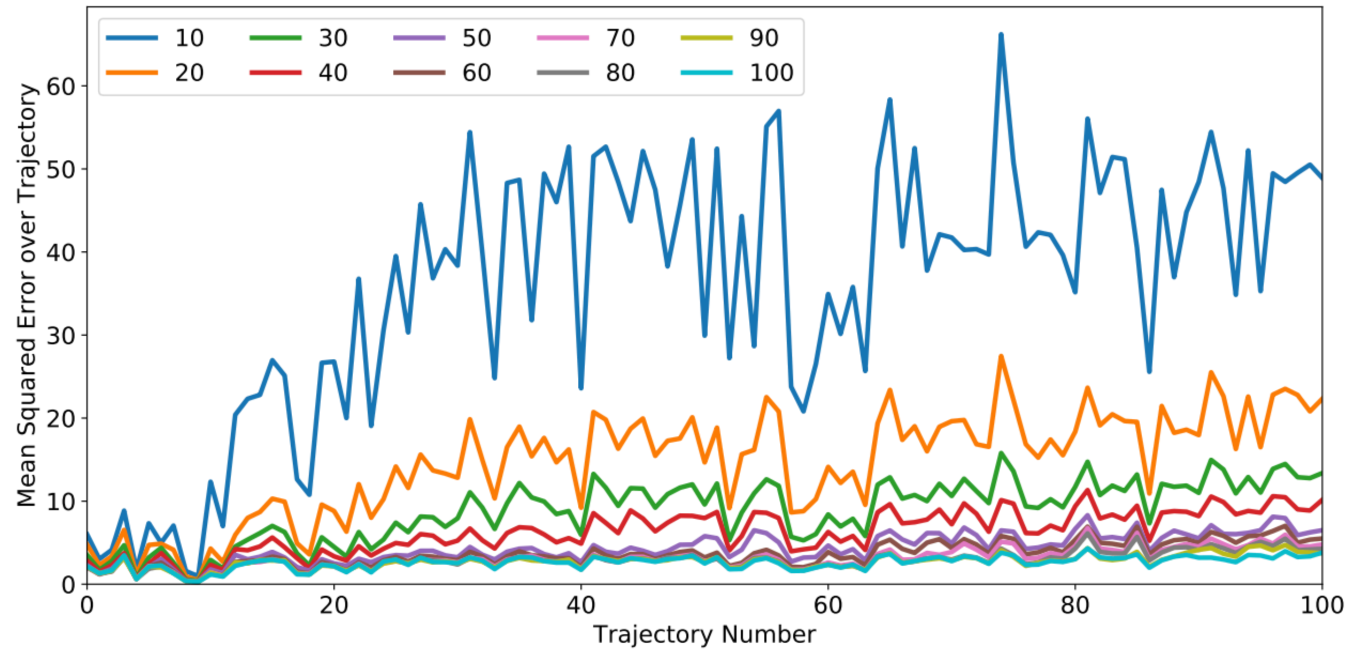
Kurtland Chua **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu



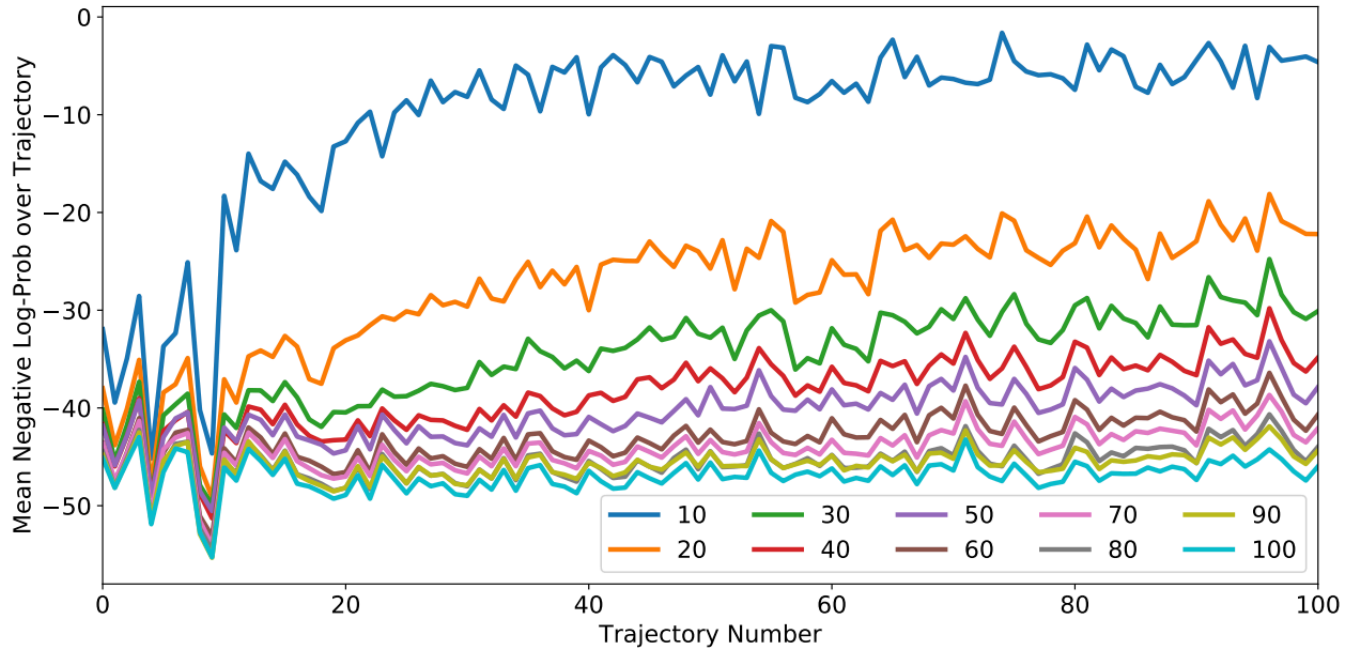
Case study: PETS







(a) Mean squared error.



(b) Negative log likelihood.

Next time

- Model-based RL: Policy learning