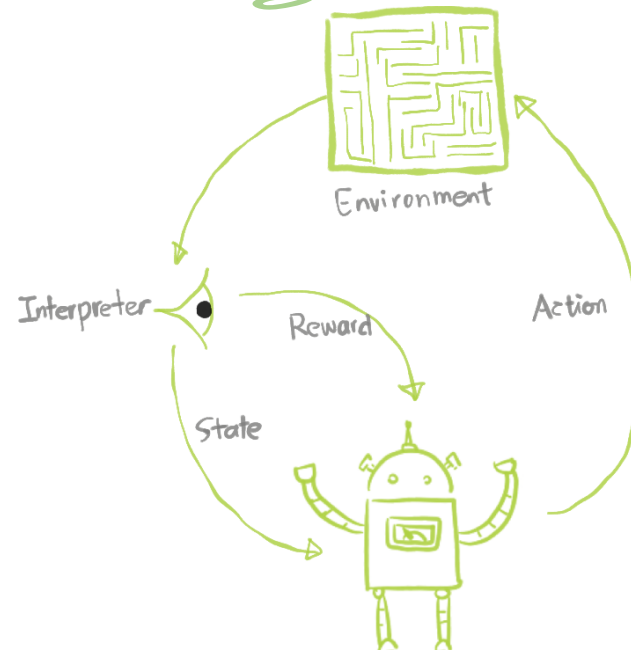


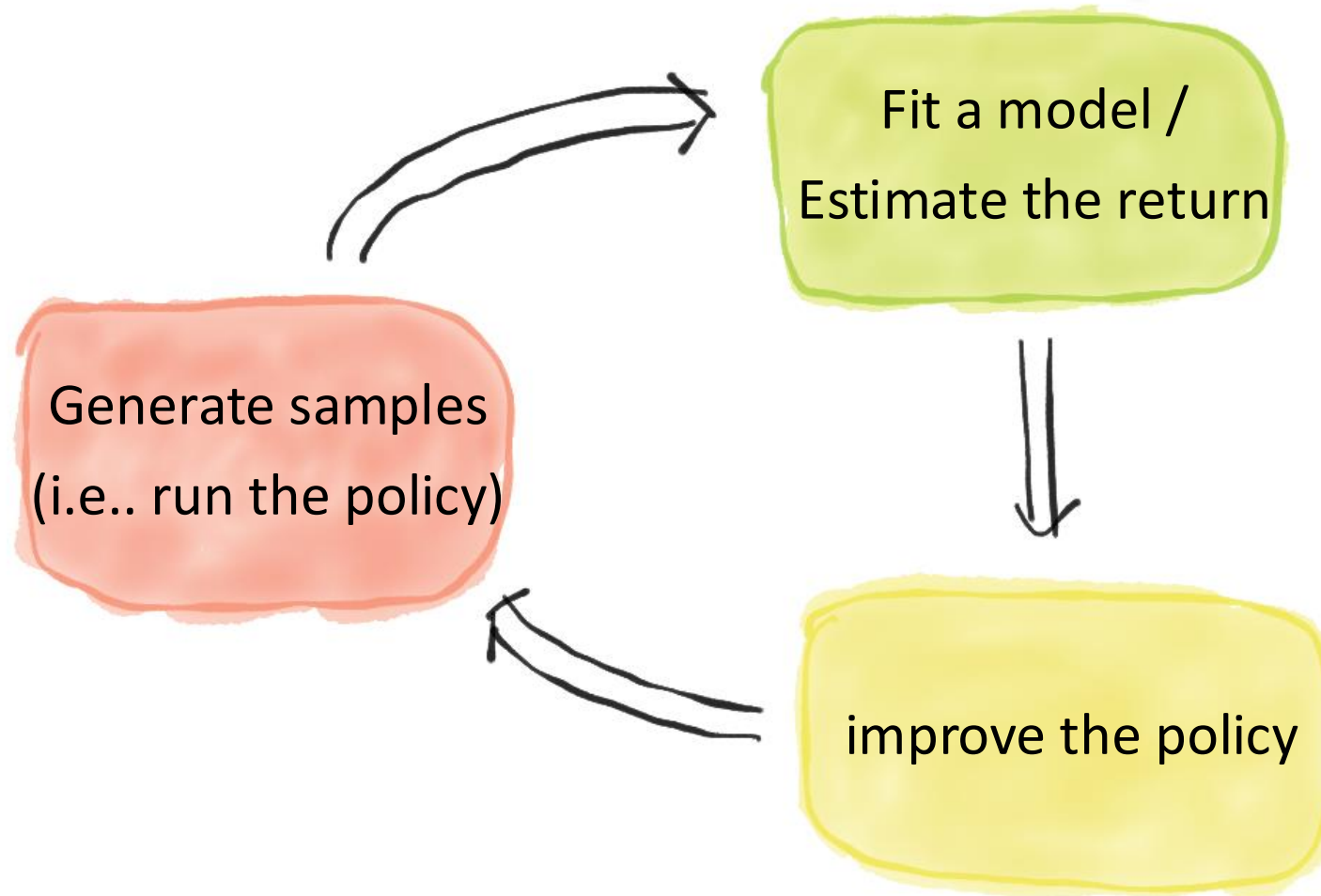
# CS5491: Artificial Intelligence

## Reinforcement Learning

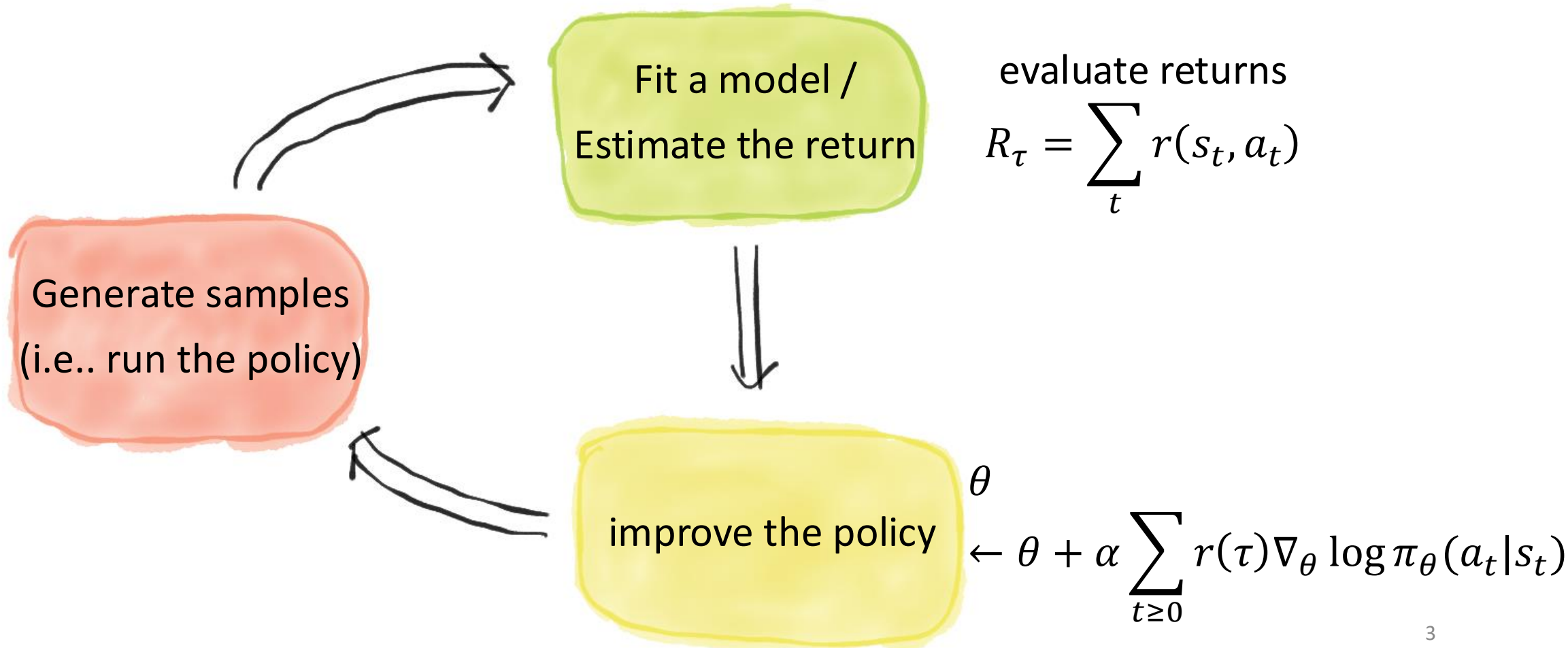


Instructor: Kai Wang

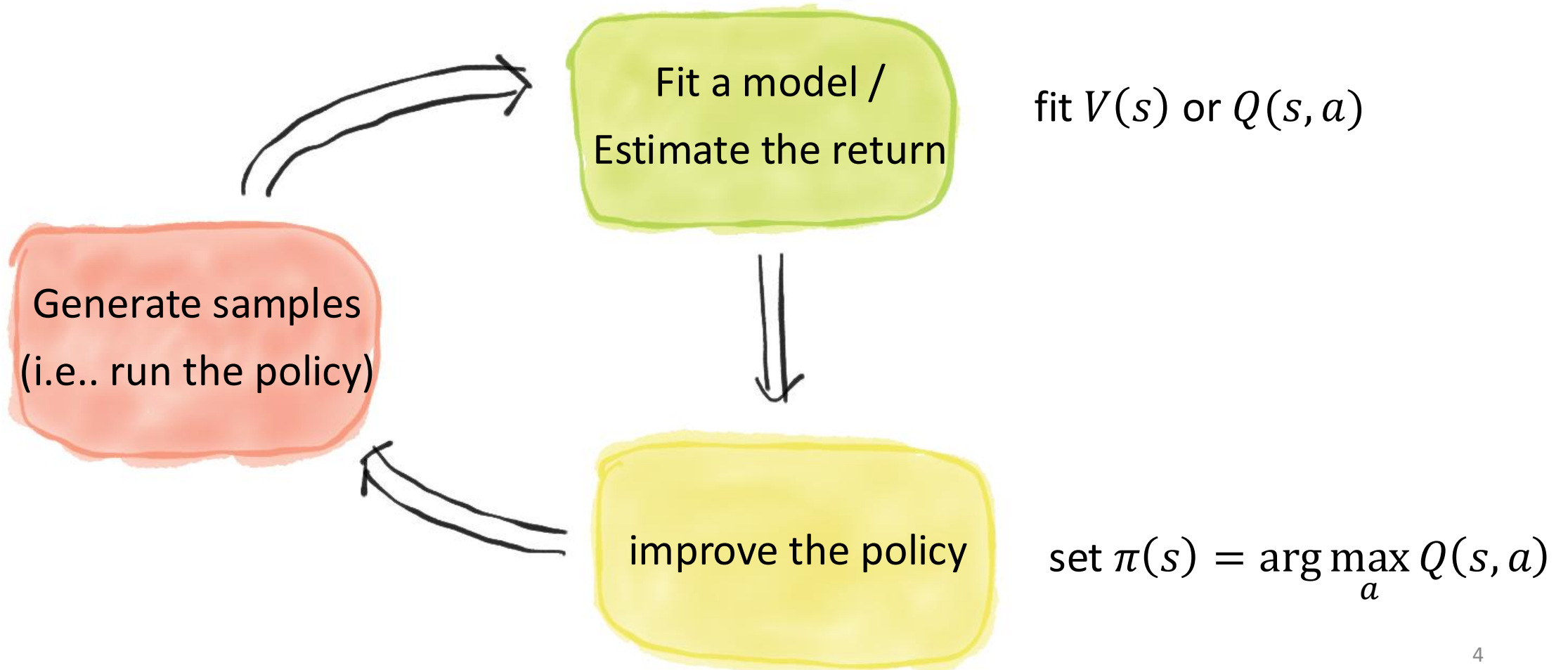
# Recap: Anatomy of Reinforcement Learning



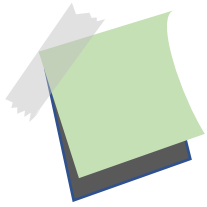
# Recap: Policy-gradient Algorithms



# Recap: Value function-based Algorithms



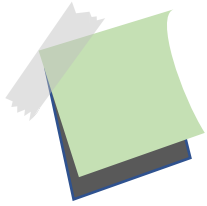
# Recap: Value Function



Following a policy produces sample trajectories  $(s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ . The value function evaluates how good a state  $s$  is by measuring the expected cumulative reward from following the policy from state  $s$ .

$$V^\pi(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

# Recap: Q-value Function



Following a policy produces sample trajectories  $(s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ . The **Q-value function** evaluates how good a state-action pair  $(s, a)$  is by measuring the expected cumulative reward from **taking action  $a$**  in state  $s$  and following the policy.

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

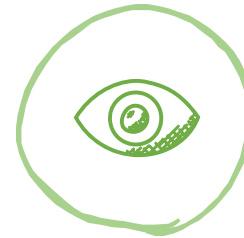
# Today



Model-based RL



Uncertainty

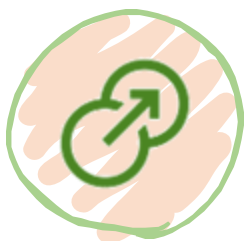


Complex  
Observations

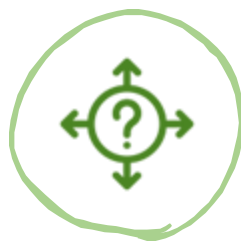


Policy  
learning

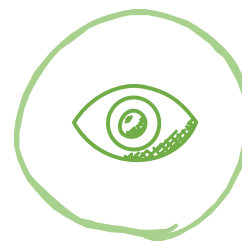
# Today



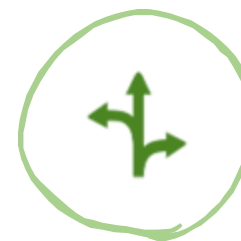
Model-based RL



Uncertainty



Complex  
Observations



Policy  
learning

# Why to Learn the Model

- ✦ If we knew  $f(s_t, a_t) = s_{t+1}$ , we could plan the actions with the highest rewards. **FIJO**
- ✦  $p(s_{t+1}|s_t, a_t)$  in the stochastic case
- ✦ Two questions:
  - How can we make decisions (or plan) based on the dynamics  $p(s_{t+1}|s_t, a_t)$ ?
  - How can we know or learn the model dynamics  $p(s_{t+1}|s_t, a_t)$ ?

# Random Shooting Method

- ✦ Abstract away optimal control/planning

$$a_1, \dots, a_T = \operatorname{argmax}_{a_1, \dots, a_T} J(a_1, \dots, a_T)$$

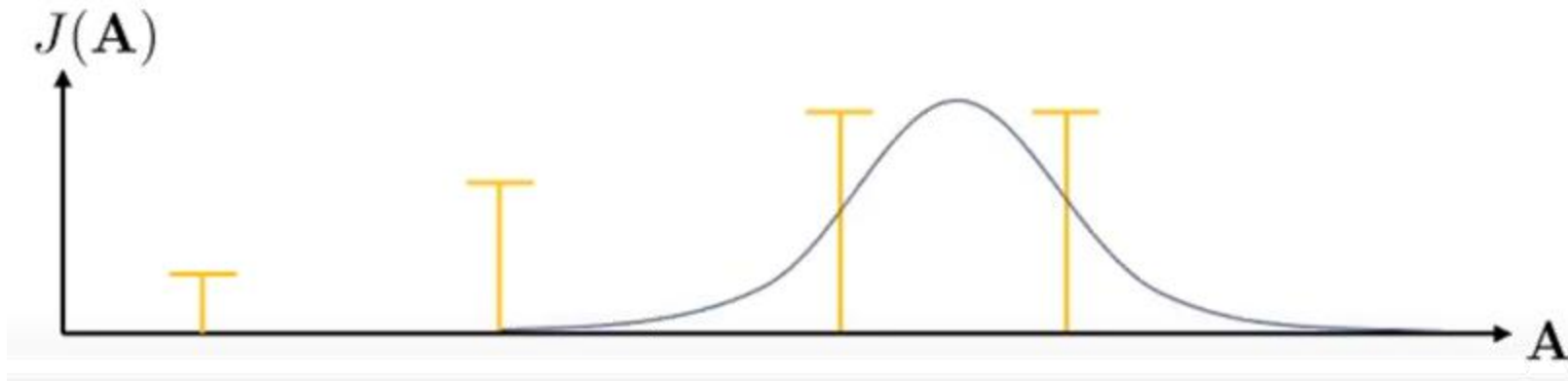
We do not care what this  $\mathbf{A}$  is  
It's just the action seq.


$$\mathbf{A} = \operatorname{argmax}_{\mathbf{A}} J(\mathbf{A})$$

- ✦
  1. Pick  $\mathbf{A}_1, \dots, \mathbf{A}_N$  from some distributions (e.g., uniform)
  2. Choose  $\mathbf{A}_i$  based on  $\operatorname{argmax}_i J(\mathbf{A}_i)$

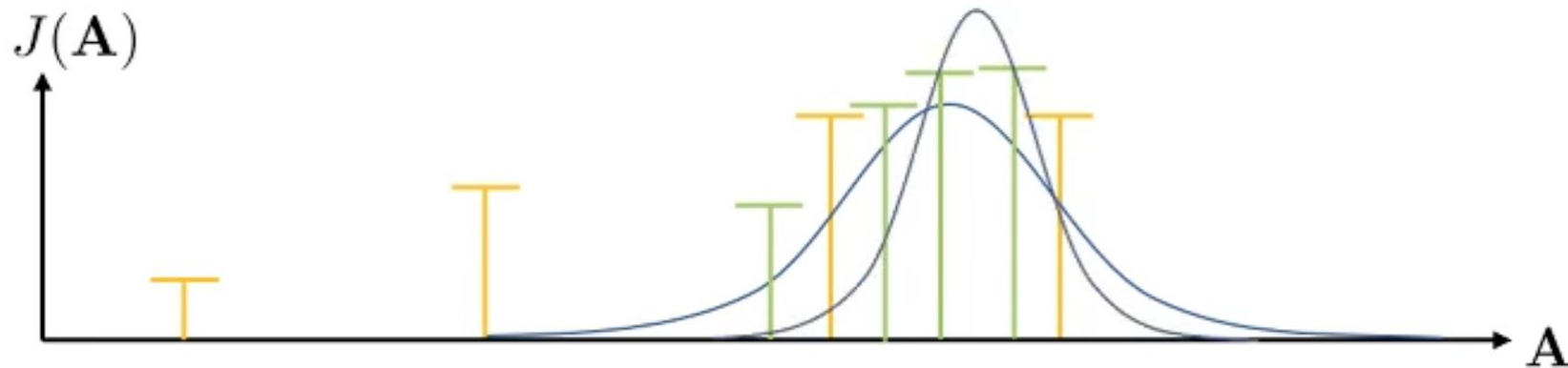
# Cross Entropy Method


Adaptive adjustment of  $A$



- 
1. Sample  $\mathbf{A}_1, \dots, \mathbf{A}_N$  from  $p(\mathbf{A})$
  2. Evaluate  $J(\mathbf{A}_1), \dots, J(\mathbf{A}_N)$
  3. Pick the *elites*  $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$  with the highest value, where  $M < N$
  4. **Refit**  $p(\mathbf{A})$  to the **elites**  $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$

# Cross Entropy Method



- 
1. Sample  $\mathbf{A}_1, \dots, \mathbf{A}_N$  from  $p(\mathbf{A})$
  2. Evaluate  $J(\mathbf{A}_1), \dots, J(\mathbf{A}_N)$
  3. Pick the *elites*  $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$  with the highest value, where  $M < N$
  4. Refit  $p(\mathbf{A})$  to the elites  $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$

# Monte Carlo Search

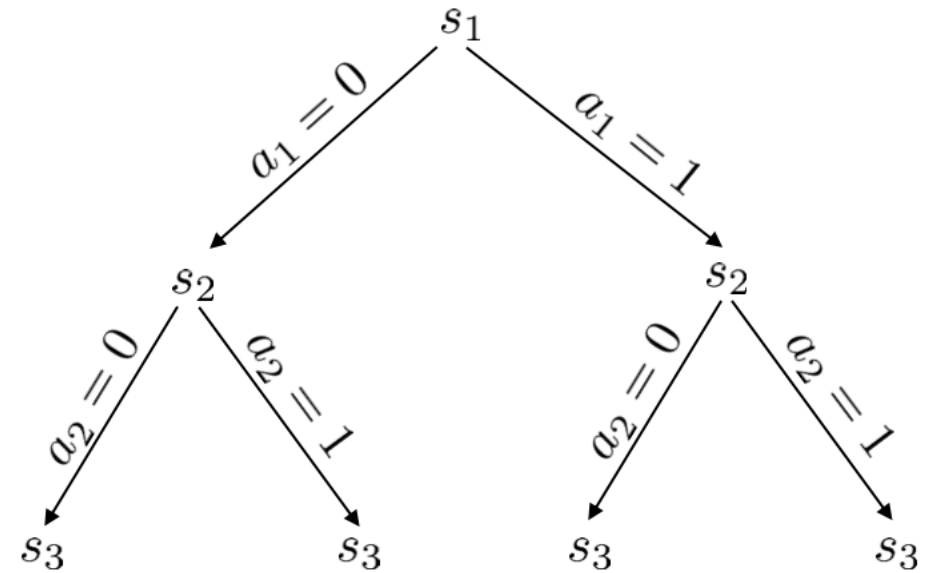
discrete planning as a search problem



$s_t$



$a_t$



# Model-based RL Version 0.5

1. Run base policy  $\pi_0(a_t|s_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn dynamics model  $f(s, a)$  to minimize  $\sum_i \|f(s_i, a_i) - s'_i\|^2$
3. Plan through  $f(s, a)$  to choose actions

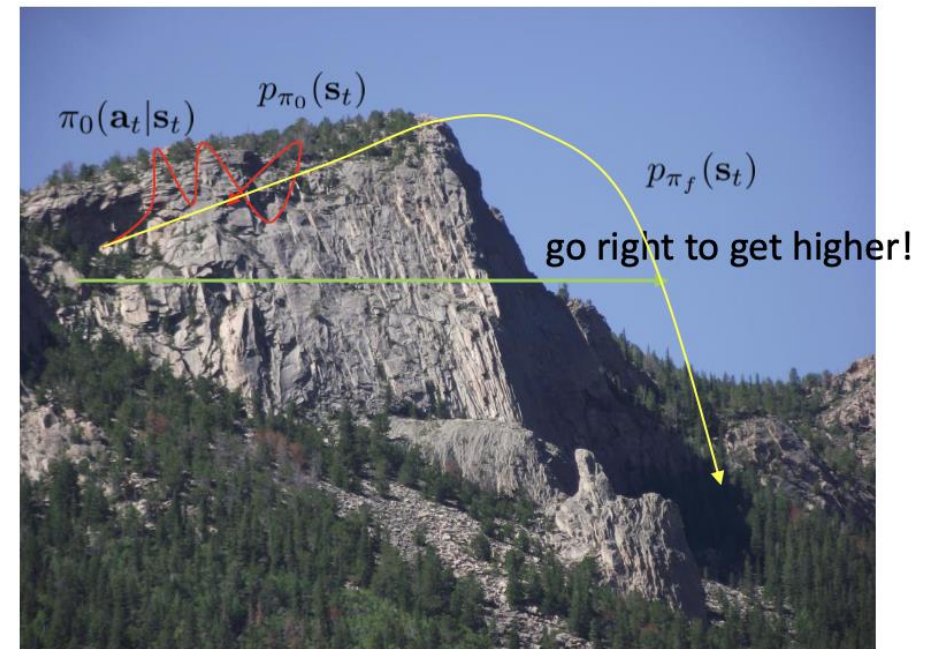
# Does it Work?

- ✦ Yes!
- ✦ Essentially how system identification works in classical robotics
- ✦ Some care should be taken to design a good base policy
- ✦ Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters

# Does it Work?

- ✧ No! (Local Optimal)
- ✧ Distribution mismatch problem becomes exacerbated as we use more expressive model classes

$$p_{\pi_f}(s_t) \neq p_{\pi_0}(s_t)$$

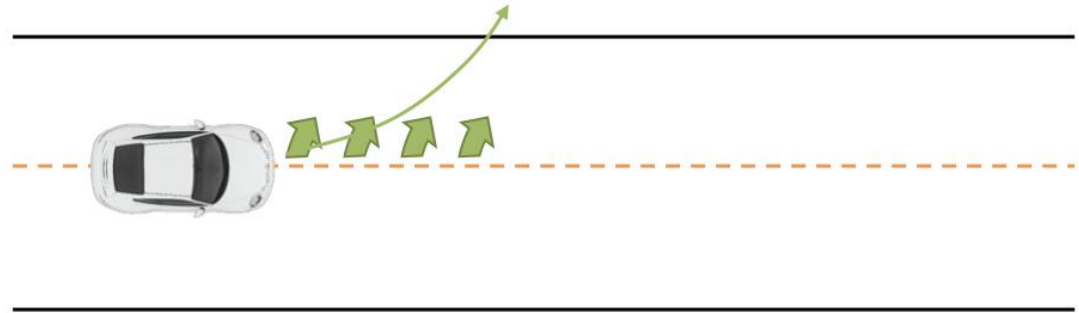


# Model-based RL Version 1.0

1. Run base policy  $\pi_0(a_t|s_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn dynamics model  $f(s, a)$  to minimize  $\sum_i \|f(s_i, a_i) - s'_i\|^2$
3. Plan through  $f(s, a)$  to choose actions
4. Execute those actions and add the resulting data  $\{(s, a, s')_j\}$  to  $\mathcal{D}$ .



# What If We Make a Mistake?

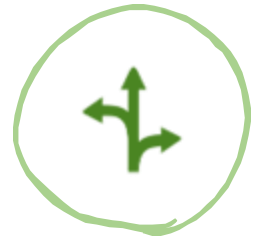
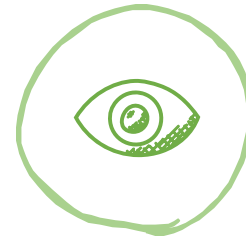
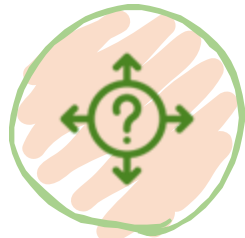


# Model-based RL Version 1.5

1. Run base policy  $\pi_0(a_t|s_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn dynamics model  $f(s, a)$  to minimize  $\sum_i \|f(s_i, a_i) - s'_i\|^2$
3. Plan through  $f(s, a)$  to choose actions
4. Execute the first planned action, observe resulting  $s'$
5. Append  $(s, a, s')$  to  $\mathcal{D}$ .



# Today



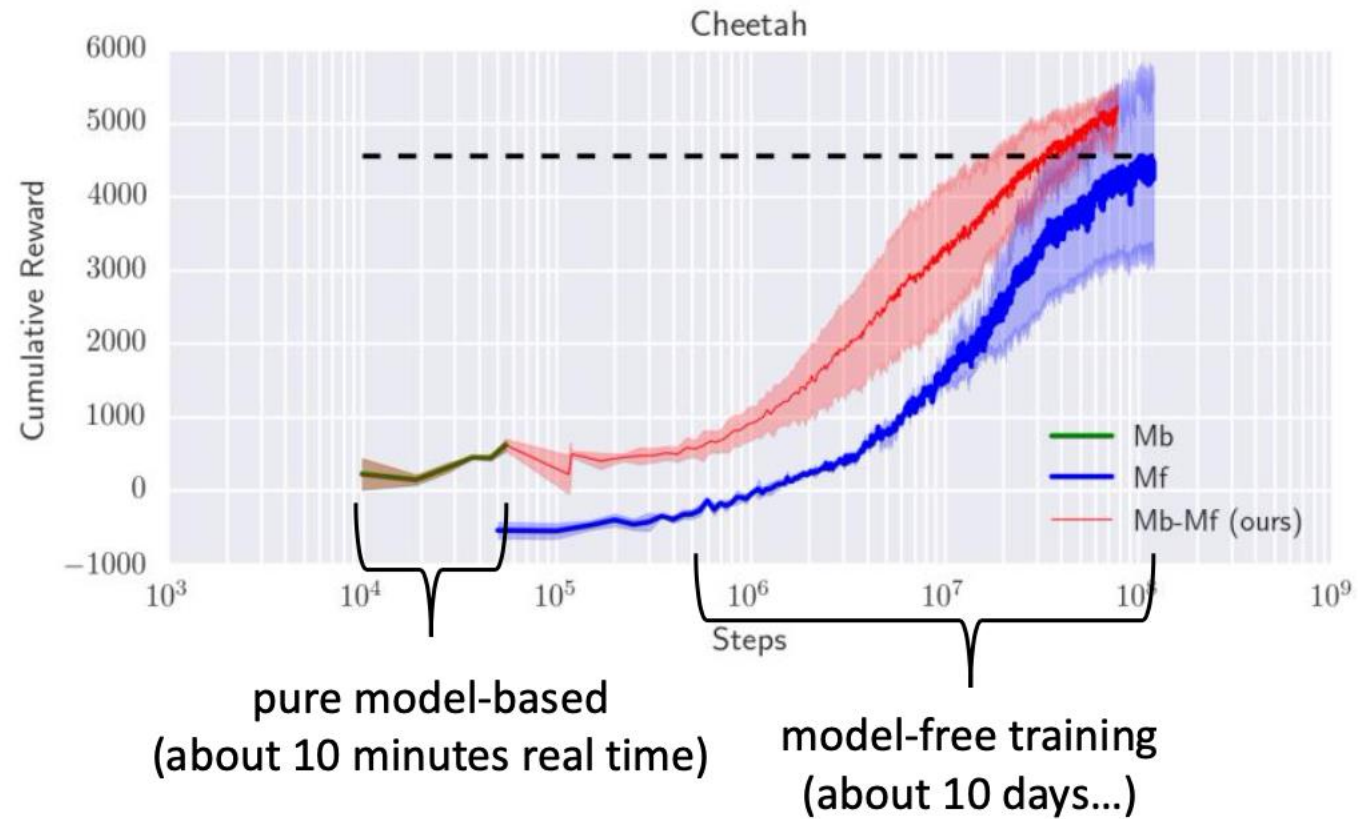
Model-based RL

Uncertainty

Complex  
Observations

Policy  
learning

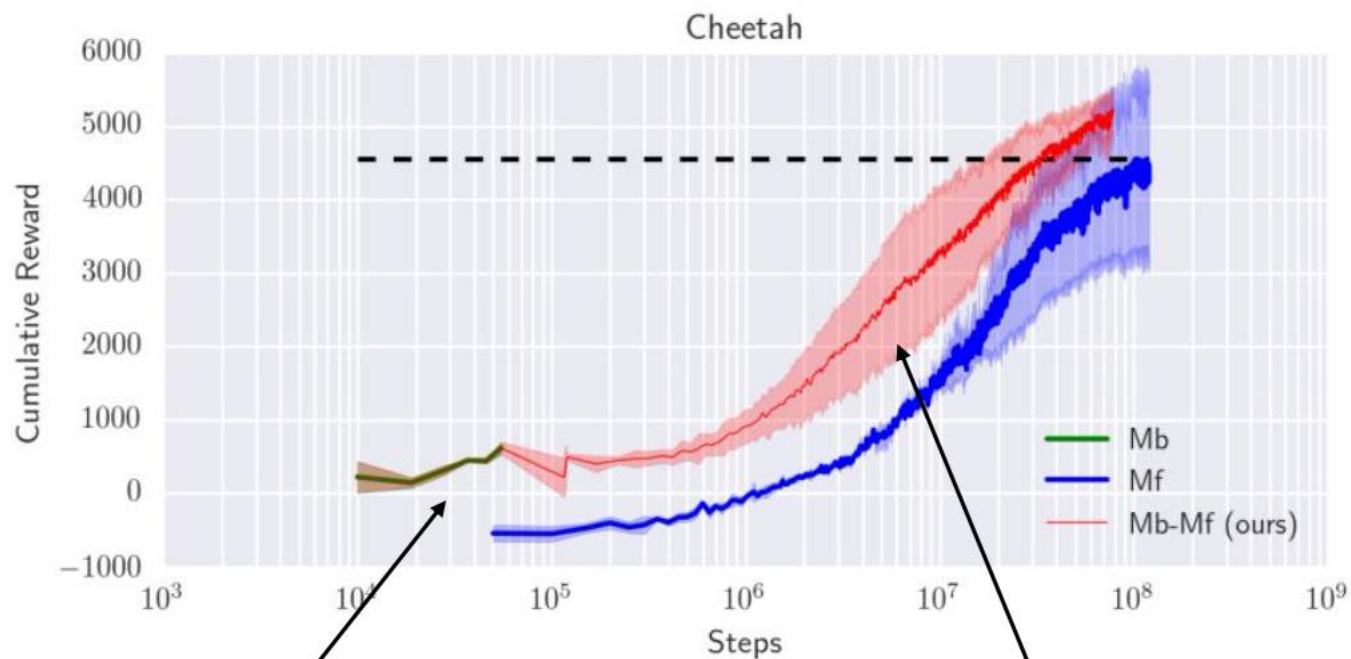
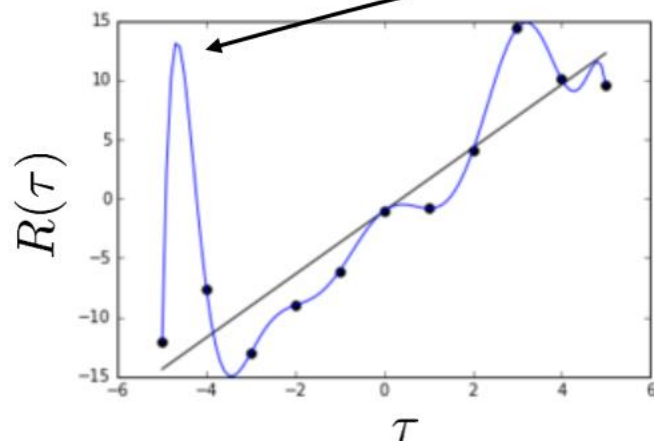
# A Performance Gap in Model-based RL



# Why the Performance Gap?



very attempting to go here



need to not overfit here...

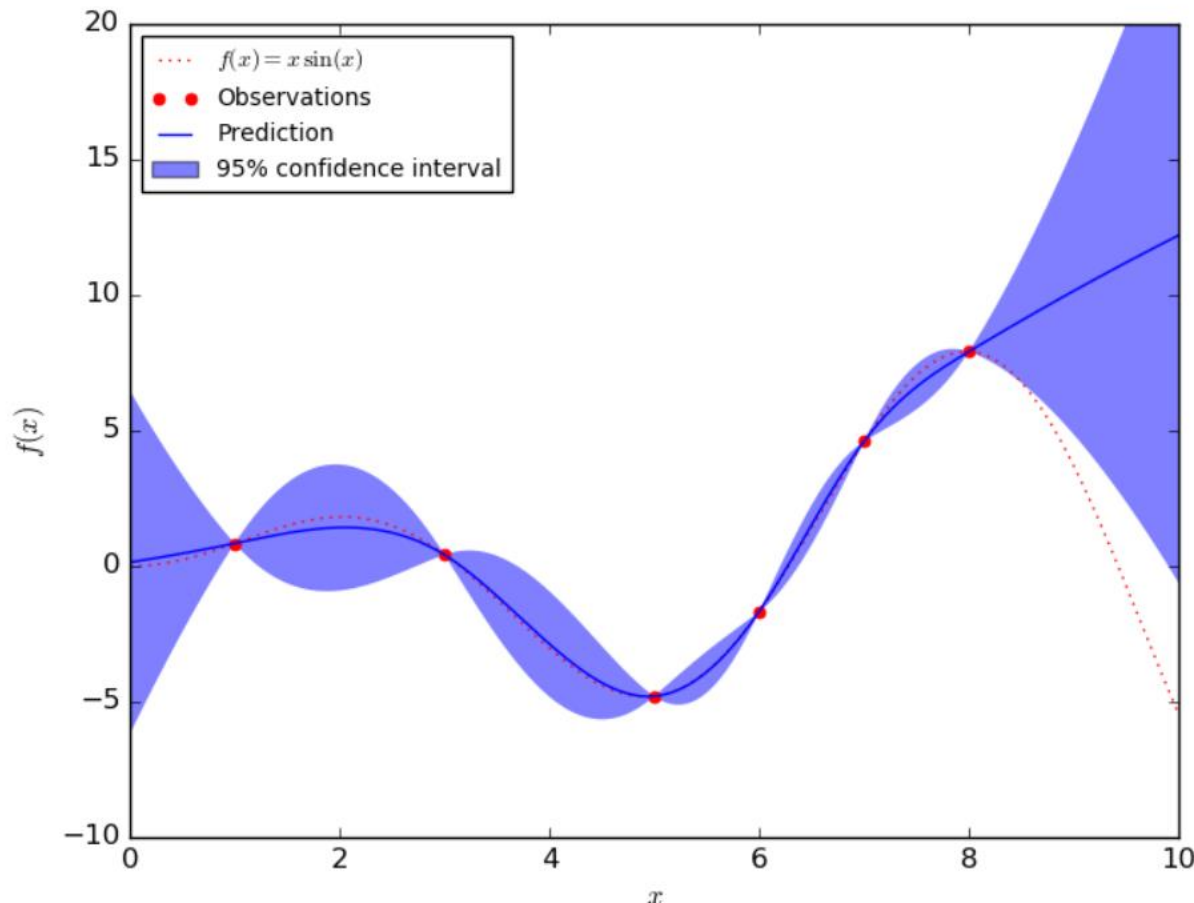
...but still have high capacity over here

Fitting vs. Generalization

# How Can Uncertainty Estimation Help?

Observe the circle with different radius/diameter

$$p_{\pi_f}(s_t) \neq p_{\pi_0}(s_t)$$



expected reward under high-variance prediction is **very** low, even though mean is the same!

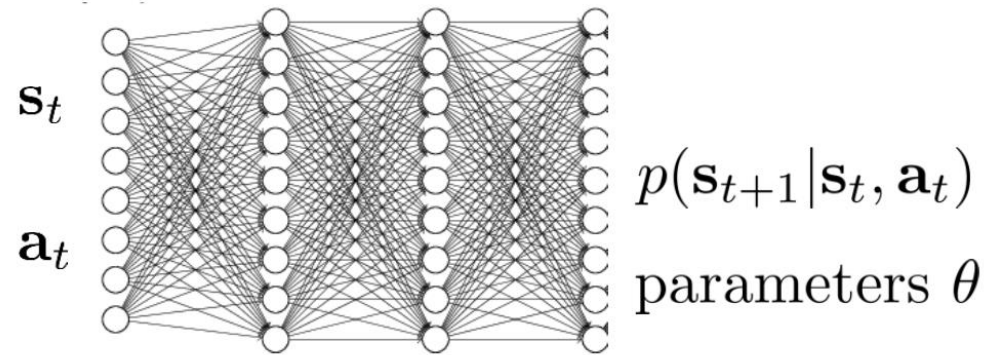
# Uncertainty-aware Models

1. Run base policy  $\pi_0(a_t|s_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn dynamics model  $f(s, a)$  to minimize  $\sum_i \|f(s_i, a_i) - s'_i\|^2$
3. Plan through  $f(s, a)$  to choose actions
4. Execute the first planned action, observe resulting  $s'$
5. Append  $(s, a, s')$  to  $\mathcal{D}$ .



only take actions for which we think we'll get high reward in expectation (w.r.t. uncertain dynamics)

# Uncertainty-aware Neural Network Models

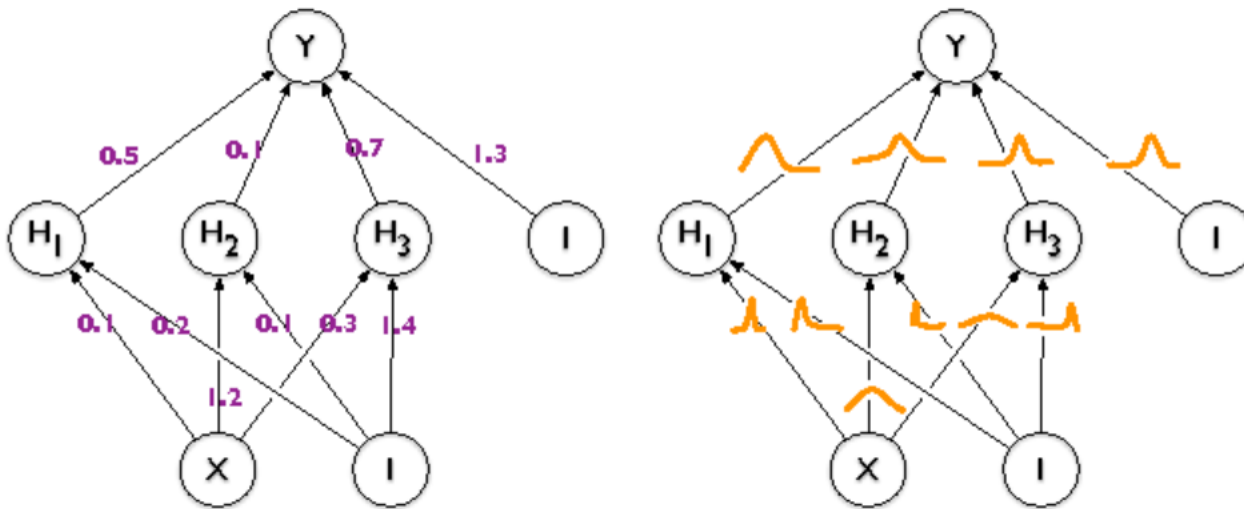


✦ Usually we estimate **MLE**  
$$\operatorname{argmax}_{\theta} \log p(\theta | \mathcal{D}) = \operatorname{argmax}_{\theta} \log p(\mathcal{D} | \theta)$$

✦ Can we instead estimate  $p(\theta | \mathcal{D})$ ? **BPE**

So that we can predict according to  $\int p(s_{t+1} | s_t, a_t, \theta) p(\theta | \mathcal{D}) d\theta$

# Bayesian Neural Networks



common approximation:

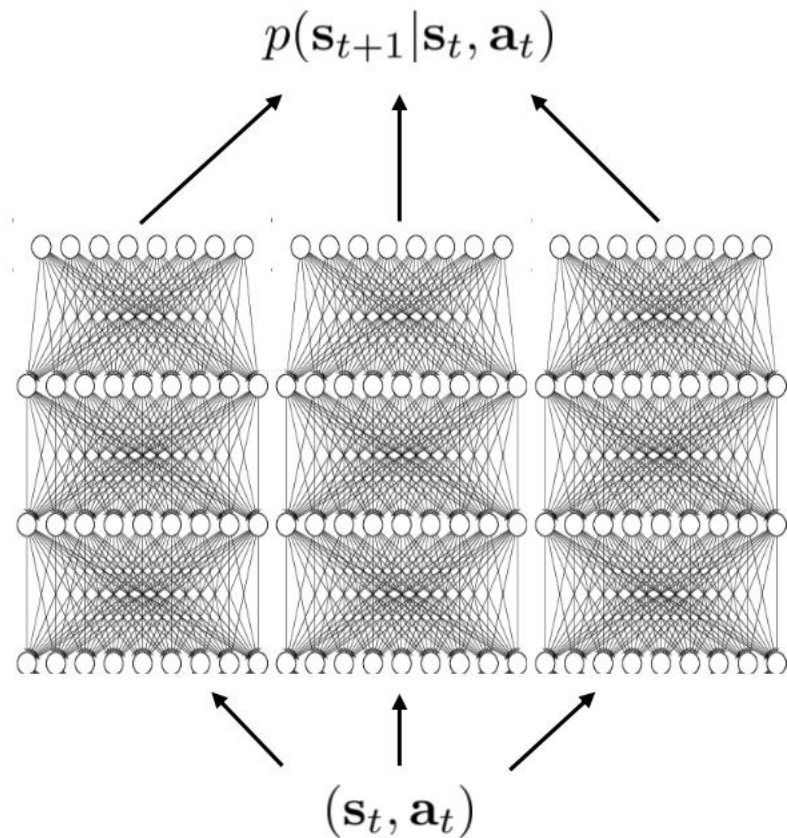
$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$$

expected weight

uncertainty  
about the weight

# Bootstrap Ensembles



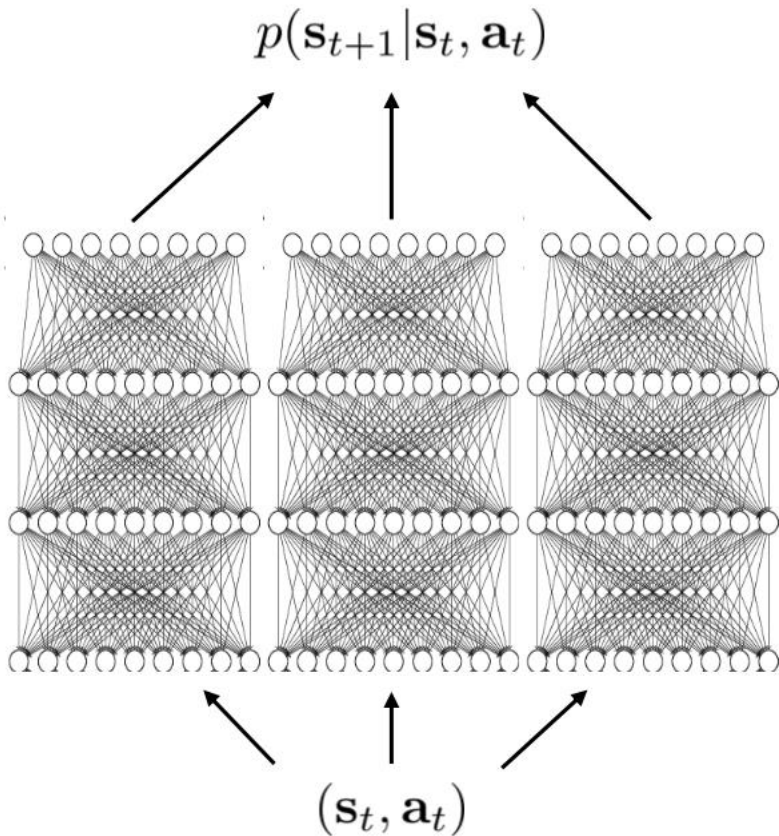
- ✦ Train multiple models and see if they agree:  

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

$$\int p(s_{t+1}|s_t, a_t, \theta) p(\theta|\mathcal{D}) d\theta$$

$$\approx \frac{1}{N} \sum_i p(s_{t+1}|s_t, a_t, \theta_i)$$
- ✦ How to train?
- ✦ Main idea: need to generate “independent” datasets to get “independent” models
- ✦  $\theta_i$  is trained on  $\mathcal{D}_i$ , sampled with replacement from  $\mathcal{D}$

# Bootstrap Ensembles



- ✧ This basically works
- ✧ Very crude approximation, because the number of models is usually small ( $< 10$ )
- ✧ Resampling with replacement is usually unnecessary, because SGD and random initialization usually makes the models sufficiently independent

# Planning with Uncertainty

- ✧ Before  $J(a_1, \dots, a_H) = \sum_{t=1}^H r(s_t, a_t)$ , where  $s_{t+1} = f(s_t, a_t)$
- ✧ Now  $J(a_1, \dots, a_H) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^H r(s_{t,i}, a_t)$ , where  $s_{t+1,i} = f(s_{t,i}, a_t)$ ; we got  $N$  sub-models

1. Sample  $\theta \sim p(\theta|\mathcal{D})$
2. At each time step  $t$ , sample  $s_{t+1} \sim p(s_{t+1}|s_t, a_t, \theta)$
3. Calculate  $R = \sum_t r(s_t, a_t)$
4. Repeat steps 1 to 3 and accumulate the average reward

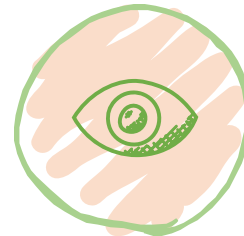
# Today



Model-based RL



Uncertainty

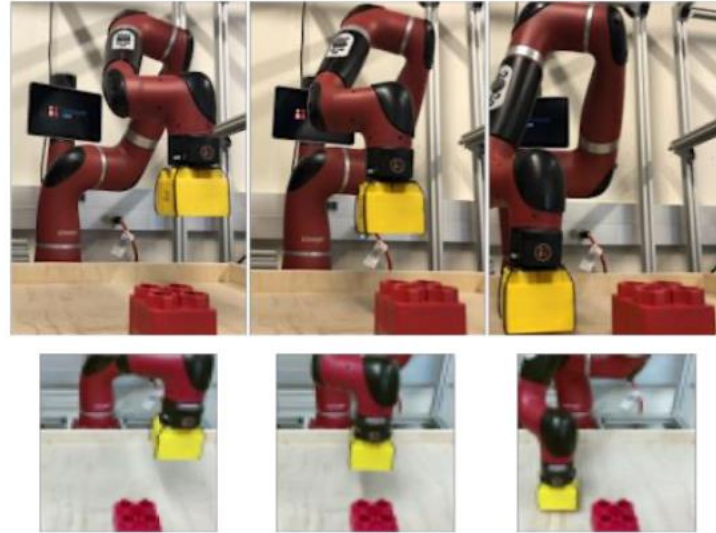


Complex  
Observations



Policy  
learning

# Difficulty about Complex Observations



$$f(s_t, a_t) = s_{t+1}$$

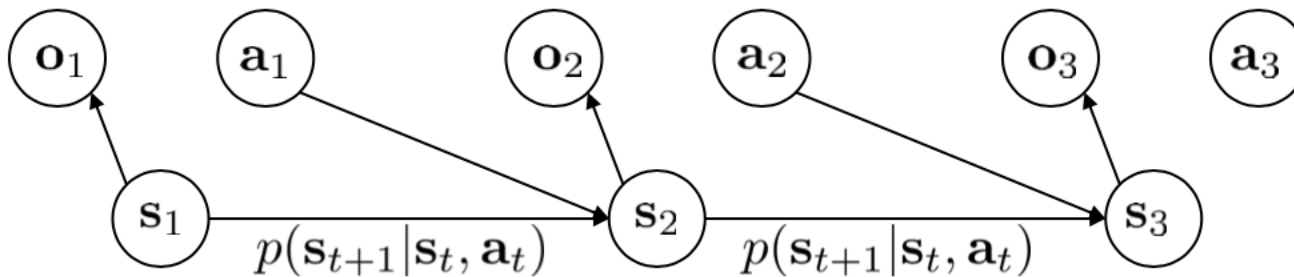
## Challenges ( $O_i$ )

- High dimensionality
- Redundancy
- Partial observability

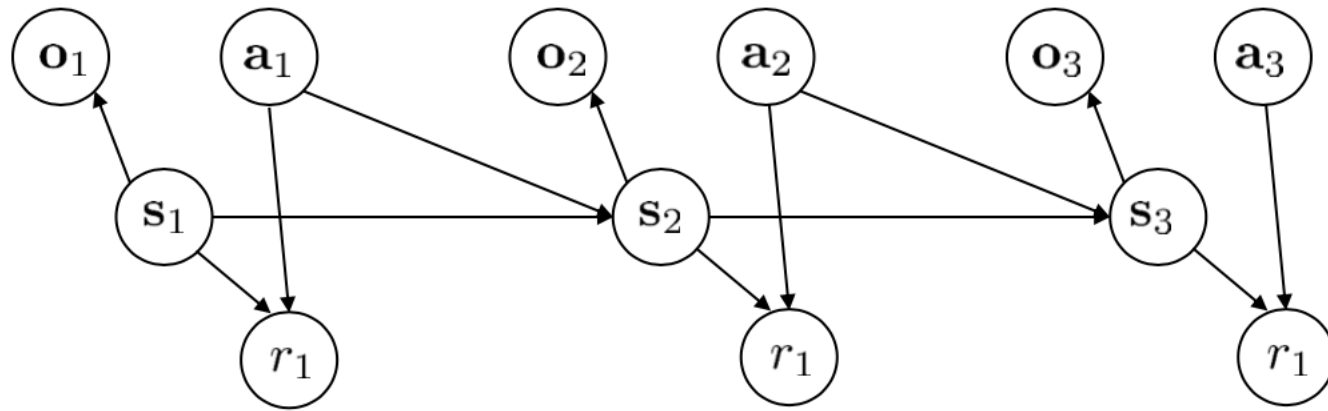
high-dimensional  
but not dynamic

low-dimension  
but dynamic

Separately learn  $p(o_t | s_t)$  and  $p(s_{t+1} | s_t, a_t)$



# State Space Models



$p(o_t|s_t)$  observation model  
 $p(s_{t+1}|s_t, a_t)$  dynamics model  
 $p(r_t|s_t, a_t)$  reward model

- Standard (full observed model)  $\max_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log p_{\phi}(s_{t+1,i} | s_{t,i}, a_{t,i})$
- Latent space based model  $\max_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T E[\log p_{\phi}(s_{t+1,i} | s_{t,i}, a_{t,i}) + \log p_{\phi}(o_{t,i} | s_{t,i})]$

expectation w.r.t.  $(s_t, s_{t+1}) \sim p(s_t, s_{t+1} | o_{1:T}, a_{1:T})$

# Model-based RL with Latent Space Models

$$\diamond \max_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T E[\log p_{\phi}(s_{t+1,i} | s_{t,i}, a_{t,i}) + \log p_{\phi}(o_{t,i} | s_{t,i})]$$

↑  
expectation w.r.t.  $(s_t, s_{t+1}) \sim p(s_t, s_{t+1} | o_{1:T}, a_{1:T})$

→ Learn approximate posterior  $q_{\phi}(s_t | o_{1:T}, a_{1:T})$  “encoder”

→ Full smoothing posterior  $q_{\phi}(s_t, s_{t+1} | o_{1:T}, a_{1:T})$

→ Single-step encoder  $q_{\phi}(s_t | o_t)$

# Model-based RL with Latent Space Models

1. Run base policy  $\pi_0(a_t|o_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(o, a, o')_i\}$
2. Learn  $p_\phi(s_{t+1}|s_t, a_t), p_\phi(r_t|s_t), p(o_t|s_t)$
3. Plan through the model to choose actions
4. Execute the first planned action, observe resulting  $o'$
5. Append  $(o, a, o')$  to  $\mathcal{D}$ .



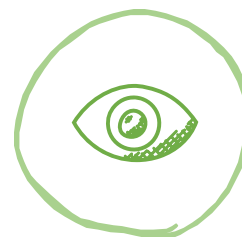
# Today



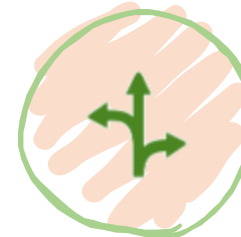
Model-based RL



Uncertainty

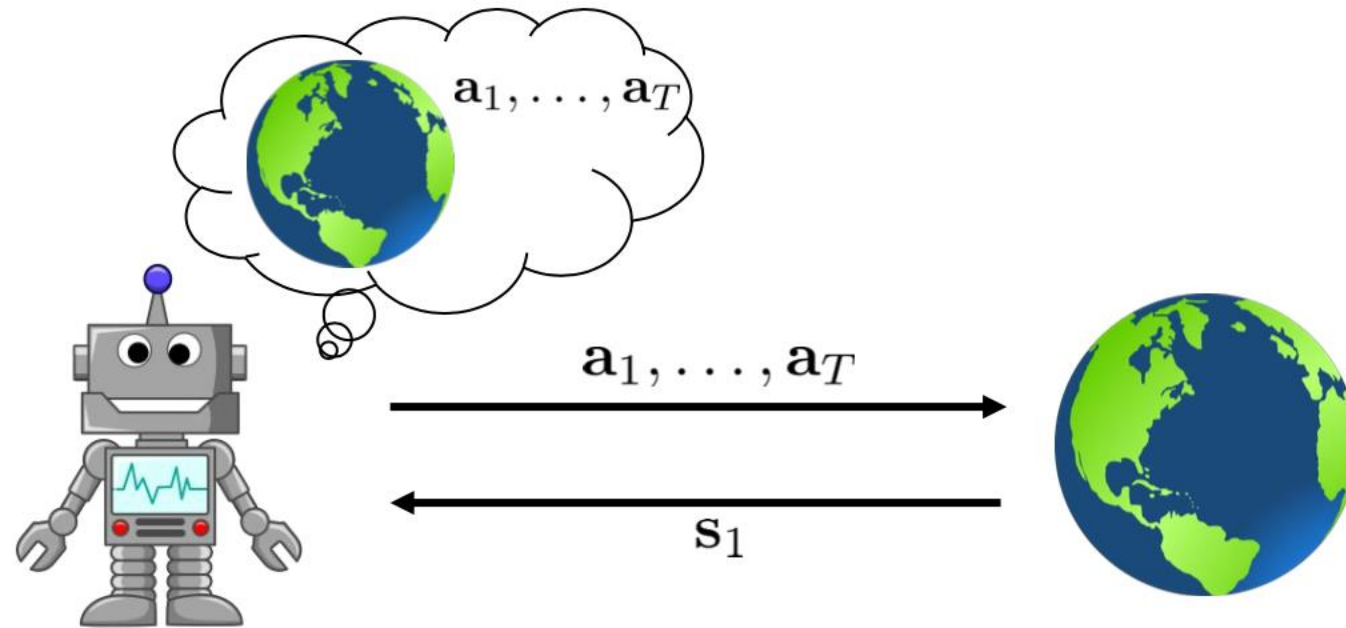


Complex  
Observations



Policy  
learning

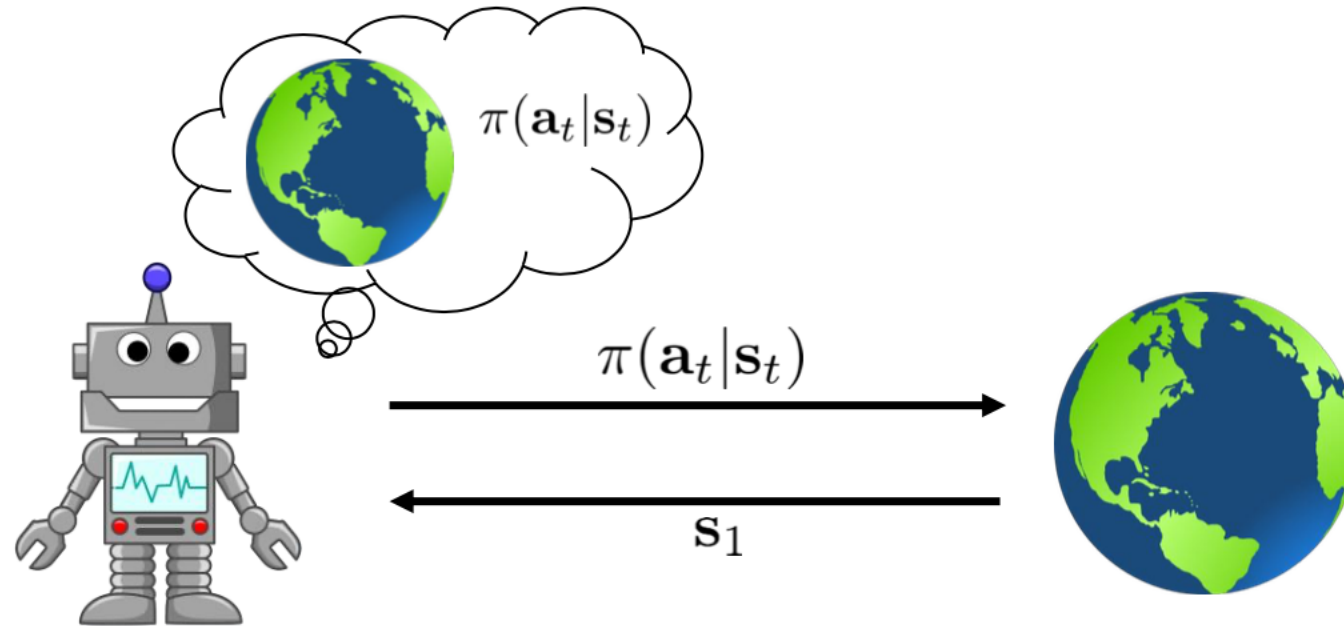
# The Stochastic Open-loop Case



$$p_{\theta}(\mathbf{s}_1, \dots, \mathbf{s}_T | \mathbf{a}_1, \dots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} E \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \dots, \mathbf{a}_T \right]$$

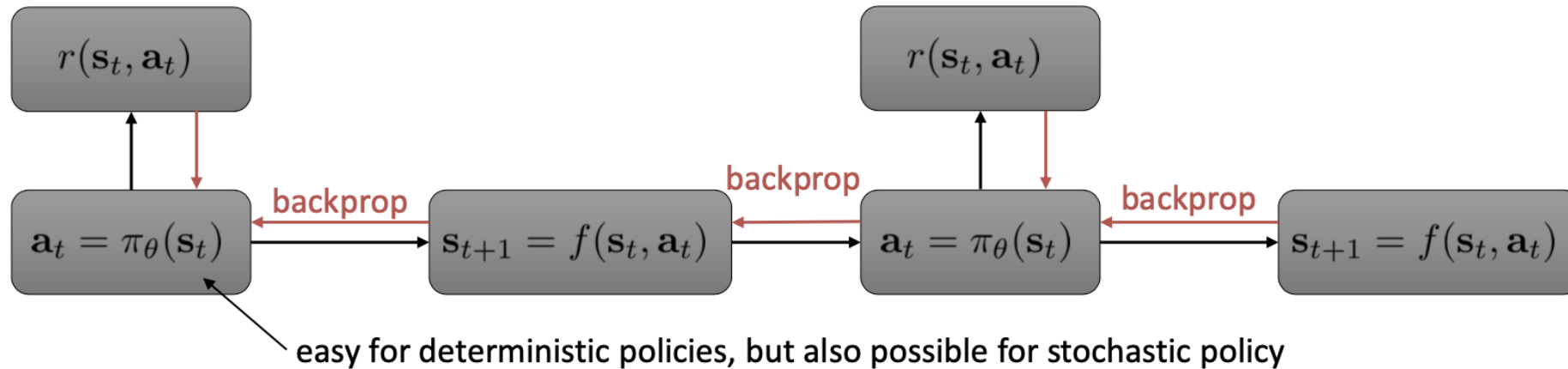
# The Stochastic Closed-loop Case



$$p(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T \pi(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

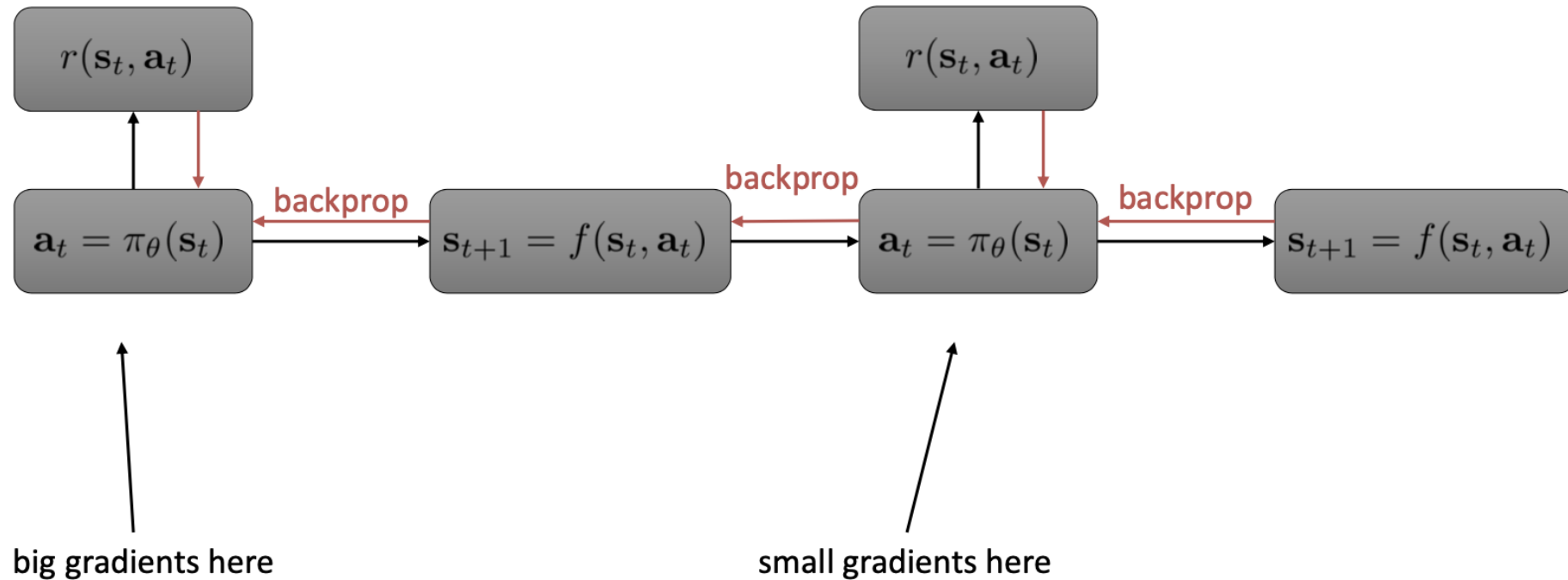
$$\pi = \arg \max_{\pi} E_{\tau \sim p(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

# Backpropagate into the Policy



1. Run base policy  $\pi_\theta(a_t|s_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn dynamics model  $f(s, a)$  to minimize  $\sum_i \|f(s_i, a_i) - s'_i\|^2$
3. Backpropagate through  $f(s, a)$  into the policy to optimize  $\pi_\theta(a_t|s_t)$
4. Run  $\pi_\theta(a_t|s_t)$ , appending the visited tuples  $(s, a, s')$  to  $\mathcal{D}$ .

# Issue: Vanishing and Exploding gradients



# Solution: Model-free Optimization with a Model

- ✧ Policy gradient might be more *stable* (if enough samples are used) because it does not require multiplying many Jacobians

→ Policy gradient:  $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \hat{Q}_{i,t}^{\pi}$

→ Backprop (pathwise) gradient:  $\nabla_{\theta} J(\theta) = \sum_{t=1}^T \frac{dr_t}{ds_t} \prod_{t'=2}^t \frac{ds_{t'}}{da_{t'-1}} \frac{da_{t'-1}}{ds_{t'-1}}$

# Dyna

1. Give state  $s$ , pick action  $a$  using exploration policy
2. Observe  $s'$  and  $r$ , and get transition  $(s, a, s', r)$
3. Update model  $p_\phi(s'|s, a)$  and  $r(s, a)$  using  $(s, a, s')$
4. Q-update  $Q_\phi(s, a) = Q_\phi(s, a) + \alpha E_{s', r} [r + \max_{a'} Q_\phi(s, a') - Q_\phi(s, a)]$
5. Repeat K times:
  6. Sample  $(s, a) \sim \mathcal{B}$  from buffer of past states and actions
  7. Q-update  $Q_\phi(s, a) = Q_\phi(s, a) + \alpha E_{s', r} [r + \max_{a'} Q_\phi(s, a') - Q_\phi(s, a)]$

# Goals

- ✓ Understand the basics for model-based reinforcement learning methods.
- ✓ Understand the algorithmic insight and workflow behind model-based algorithms.
- ✓ Understand how to leverage model-based algorithms to solve real-world problems.
- ✓ Learn how to implement the model-based reinforcement learning algorithms.

# Important This Week



Read [Reinforcement Learning: An Introduction](#).



Know more about Reinforcement Learning algorithms here.  
<http://rail.eecs.berkeley.edu/deeprlcourse/>



Know more about implementation issues of RL here.  
<https://spinningup.openai.com/en/latest/>