Model-Based Policy Learning

CS 285

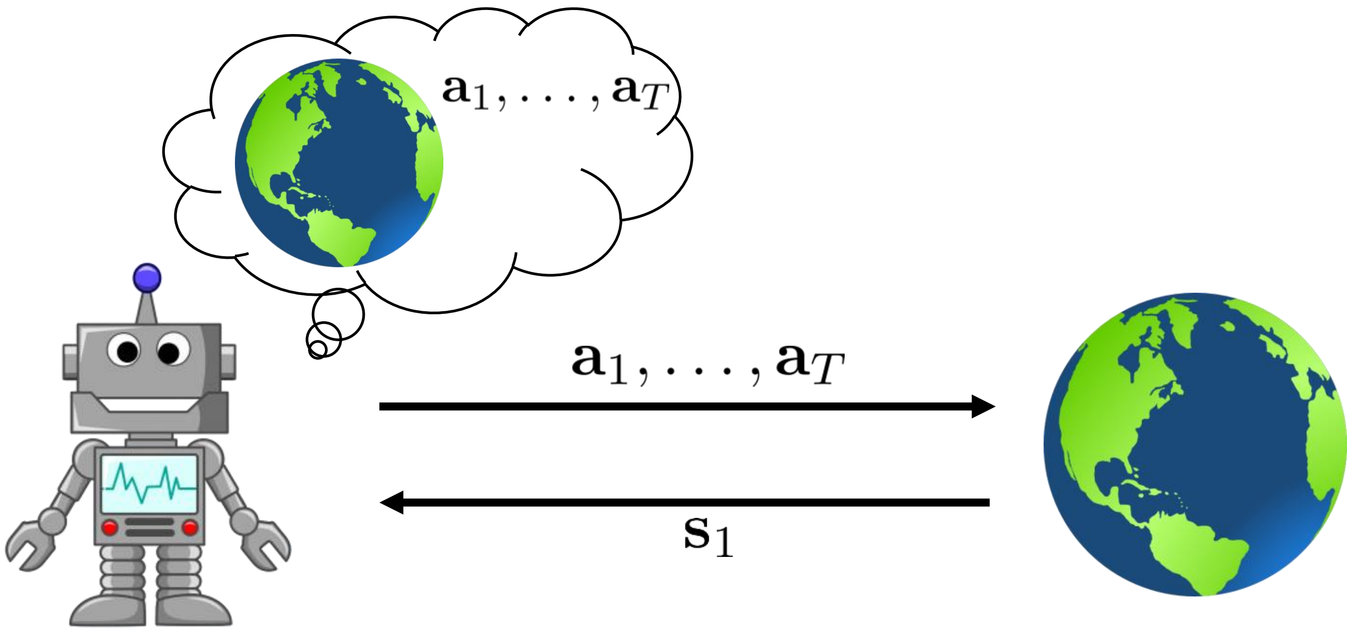
Instructor: Sergey Levine
UC Berkeley

Last time: model-based RL with MPC

model-based reinforcement learning version 1.5:

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. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

The stochastic open-loop case

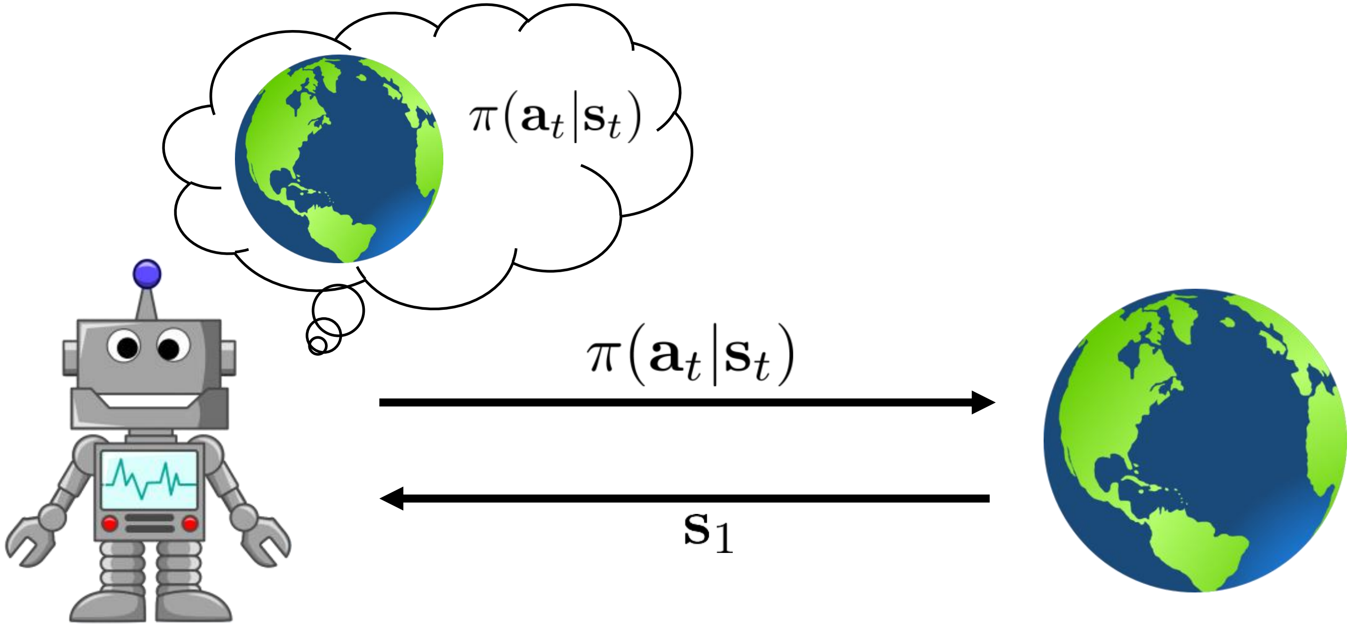


$$p_\theta(\mathbf{s}_1, \ldots, \mathbf{s}_T | \mathbf{a}_1, \ldots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

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

why is this suboptimal?

The stochastic **closed**-loop case



form of π?

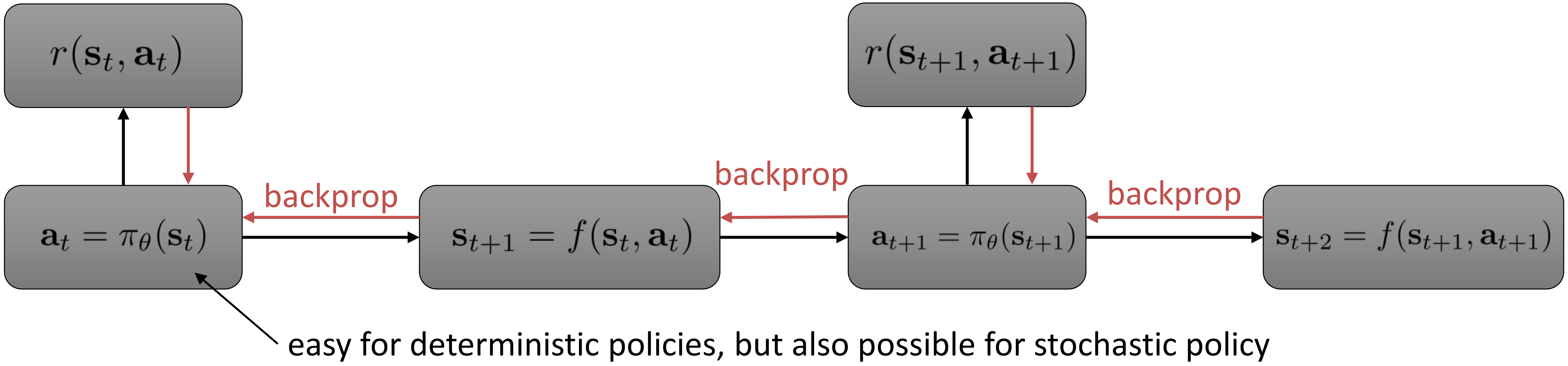
neural net — global

time-varying linear
$\mathbf{K}_t\mathbf{s}_t + \mathbf{k}_t$ — local

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

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

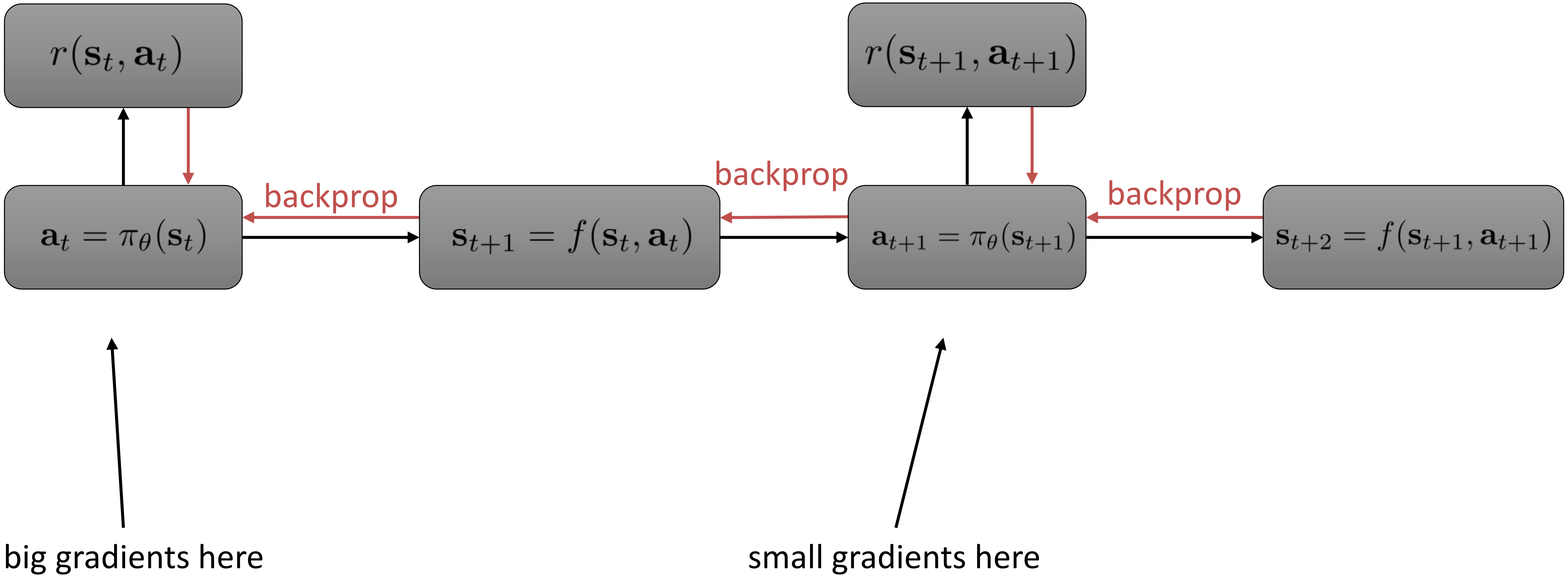
Backpropagate directly into the policy?



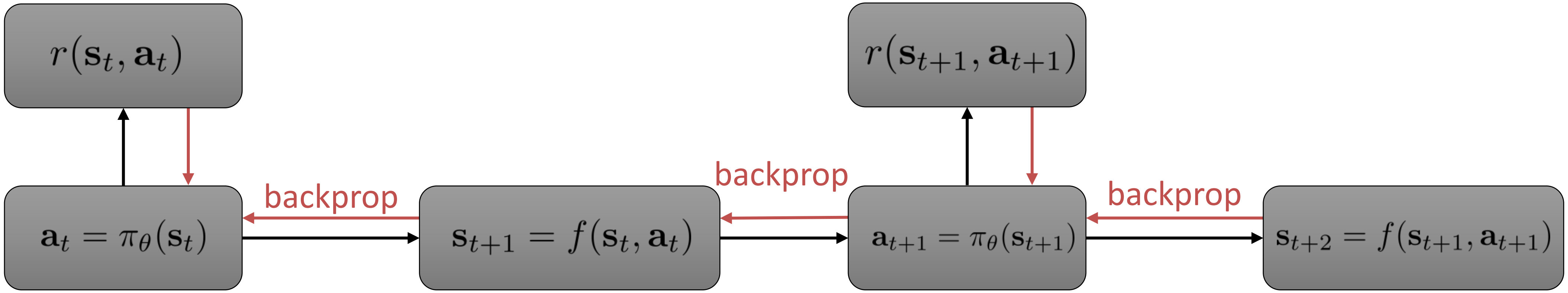
model-based reinforcement learning version 2.0:

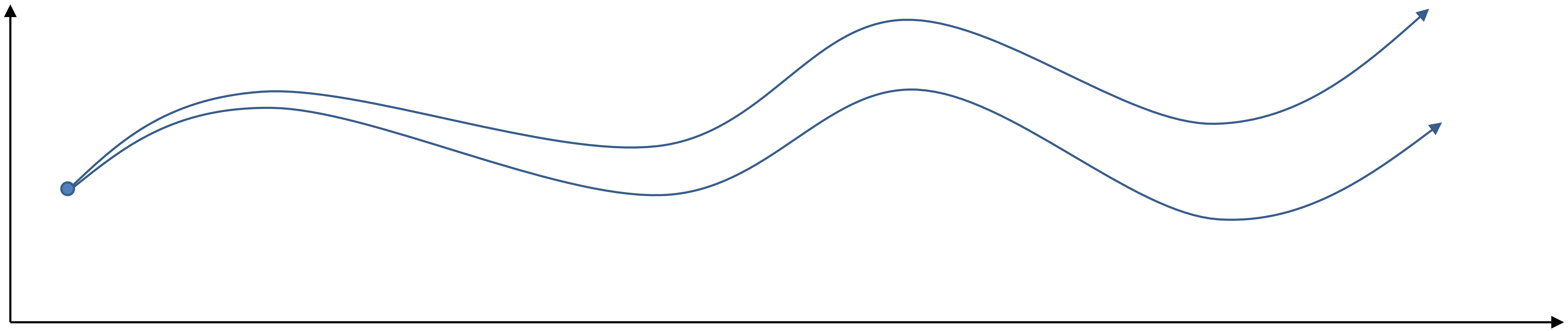
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. backpropagate through $f(\mathbf{s},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

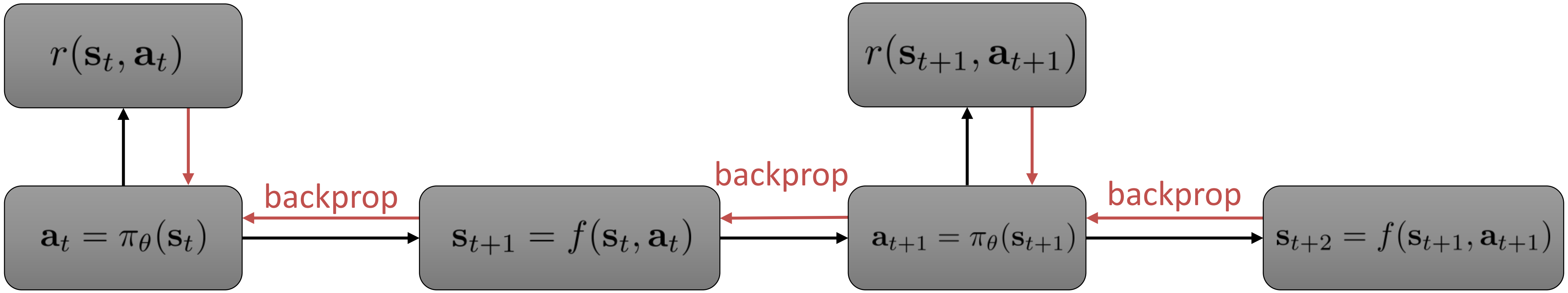


What's the problem with backprop into policy?





What's the problem with backprop into policy?



- 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 solution?

- Use derivative-free (“model-free”) RL algorithms, with the model used to generate synthetic samples
  - Seems weirdly backwards
  - Actually works very well
  - Essentially “model-based acceleration” for model-free RL

Model-Free Learning With a Model

Model-free optimization with a model

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

Backprop (pathwise) gradient: $$\nabla_\theta J(\theta) = \sum_{t=1}^{T}\frac{d\mathbf{a}_t}{d\theta}\frac{d\mathbf{s}_{t+1}}{d\mathbf{a}_t}\left(\sum_{t'=t+1}^{T}\frac{dr_{t'}}{d\mathbf{s}_{t'}}\left(\prod_{t''=t+2}^{t'}\frac{d\mathbf{s}_{t''}}{d\mathbf{a}_{t''-1}}\frac{d\mathbf{a}_{t''-1}}{d\mathbf{s}_{t''-1}}+\frac{d\mathbf{s}_{t''}}{d\mathbf{s}_{t''-1}}\right)\right)$$

- Policy gradient might be more *stable* (if enough samples are used) because it does not require multiplying many Jacobians
- See a recent analysis here:
  - Parmas et al. ‘18: PIPP: Flexible Model-Based Policy Search Robust to the Curse of Chaos

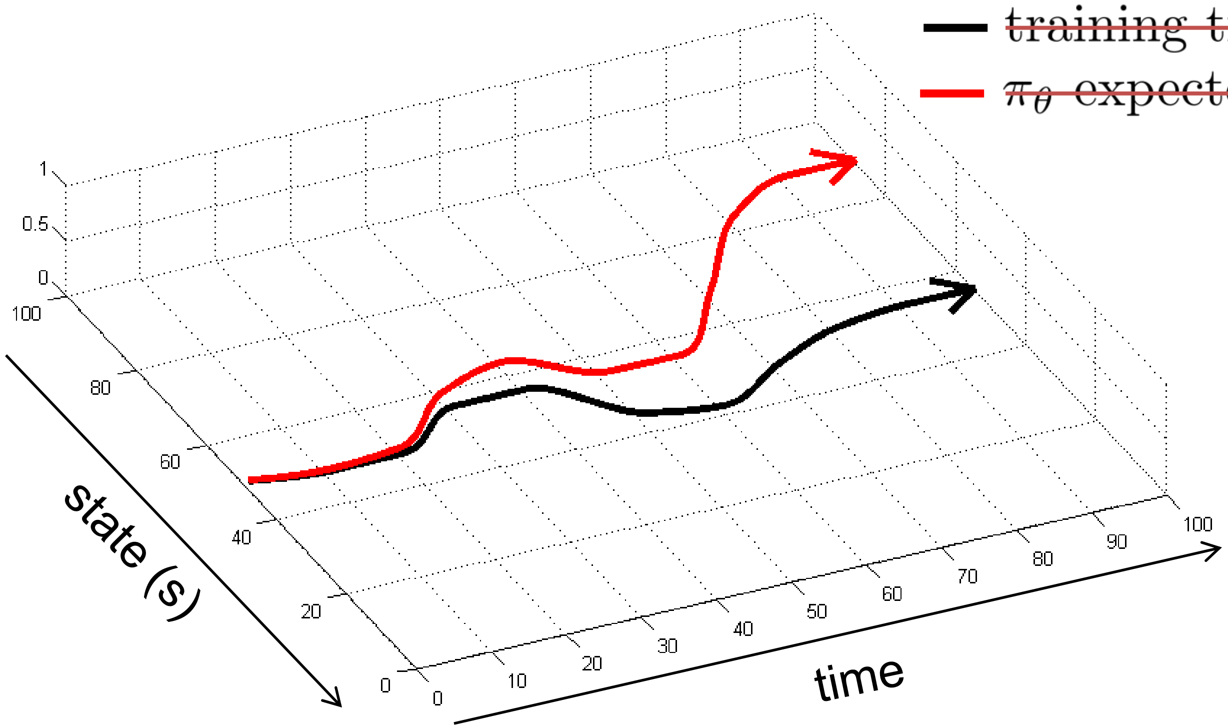
Model-based RL via policy gradient

model-based reinforcement learning version 2.5:

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. use $f(\mathbf{s}, \mathbf{a})$ to generate trajectories $\{\tau_i\}$ with policy $\pi_\theta(\mathbf{a}|\mathbf{s})$
4. use $\{\tau_i\}$ to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ via policy gradient
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

What's a potential **problem** with this approach?

The curse of long model-based rollouts



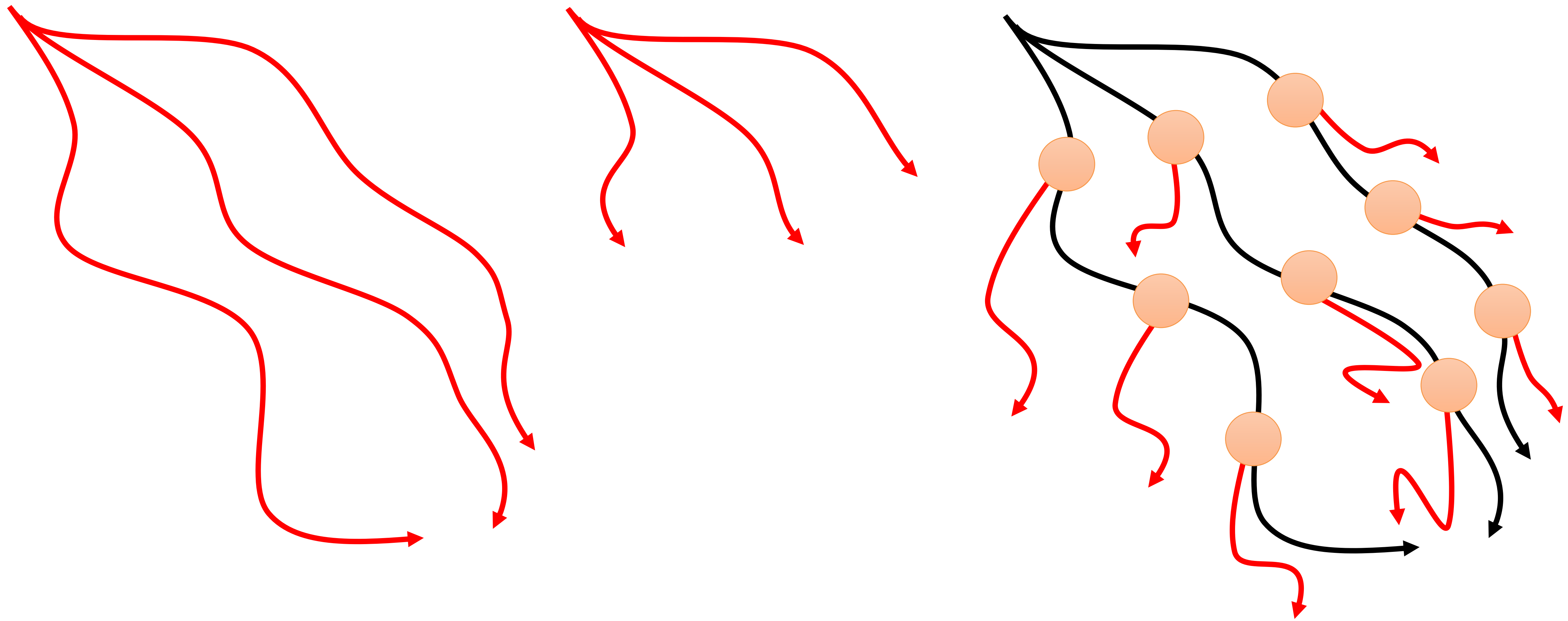
~~training trajectory~~ run π_θ with true dynamics

~~π_θ expected trajectory~~ run π_θ with learned model

How quickly does error accumulate?

$$\mathcal{O}(\epsilon T^2)$$

How to get away with **short** rollouts?



- huge accumulating error

+ much lower error
- never see later time steps

+ much lower error
+ see all time steps
- wrong state distribution

Model-based RL with **short** rollouts

model-based reinforcement learning version 3.0:

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. pick states $\mathbf{s}_i$ from $\mathcal{D}$, use $f(\mathbf{s}, \mathbf{a})$ to make *short* rollouts from them
4. use *both* real and model data to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ with *off-policy RL*
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

Dyna-Style Algorithms

Model-based RL with **short** rollouts

model-based reinforcement learning version 3.0:

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. pick states $\mathbf{s}_i$ from $\mathcal{D}$, use $f(\mathbf{s}, \mathbf{a})$ to make *short* rollouts from them
4. use *both* real and model data to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ with *off-policy RL*
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

Model-free optimization with a model

Dyna

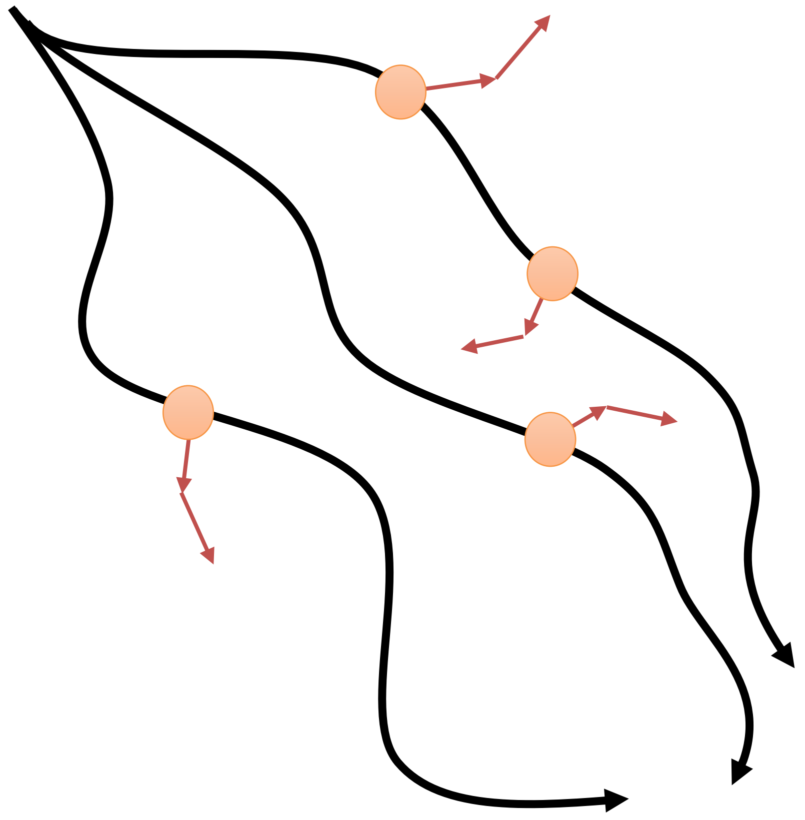
online Q-learning algorithm that performs model-free RL with a model

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

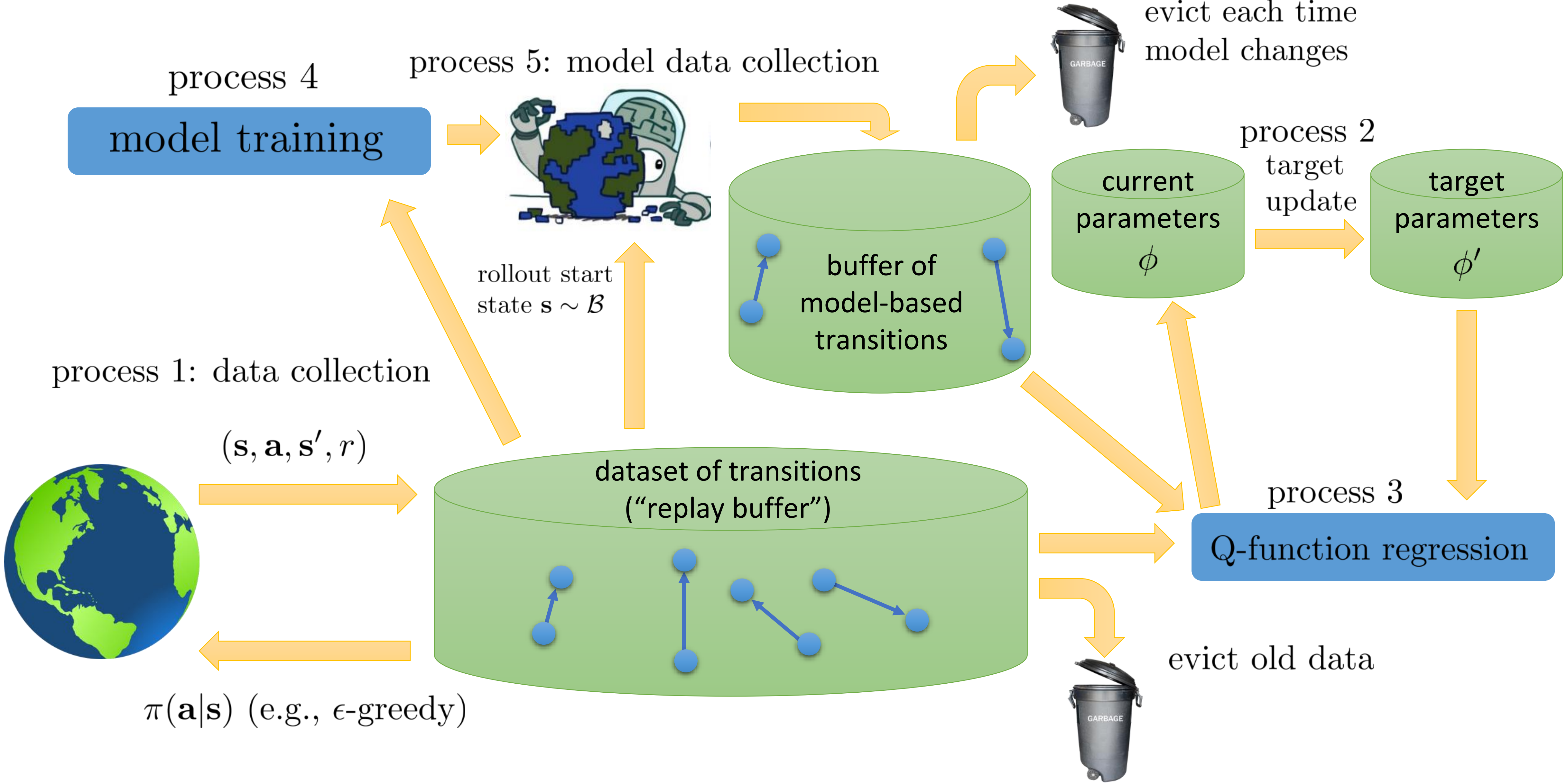
1. collect some data, consisting of transitions (s, a, s', r)
2. learn model $\hat{p}(s'|s, a)$ (and optionally, $\hat{r}(s, a)$)
3. repeat K times:
    4. sample $s \sim \mathcal{B}$ from buffer
    5. choose action a (from $\mathcal{B}$, from π, or random)
    6. simulate $s' \sim \hat{p}(s'|s, a)$ (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

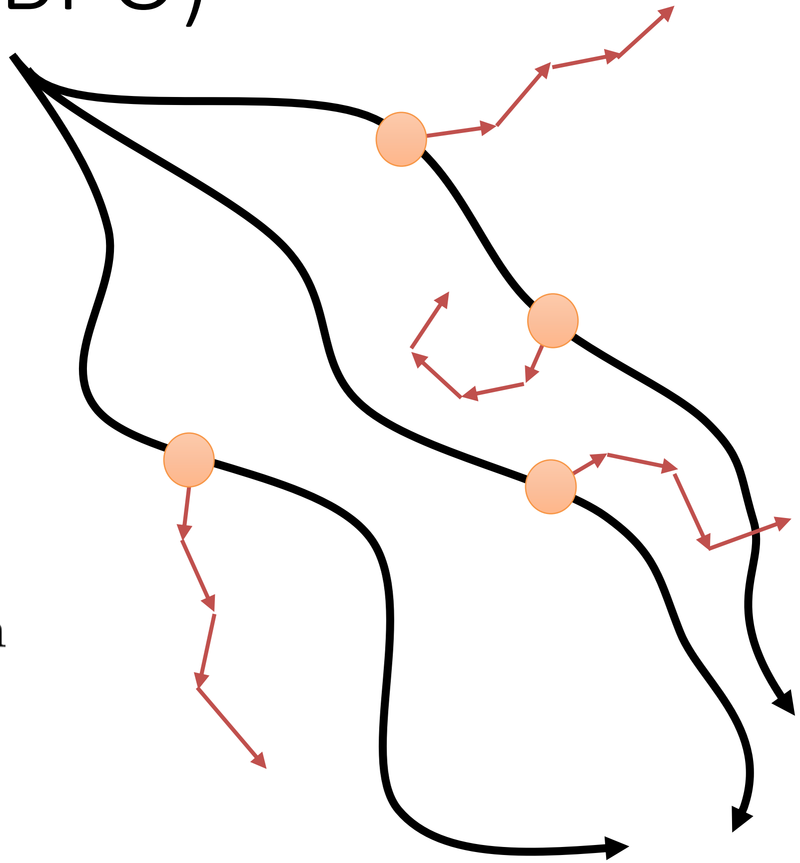
+ still sees diverse states

Model-accelerated off-policy RL



Model-Based Acceleration (MBA)
Model-Based Value Expansion (MVE)
Model-Based Policy Optimization (MBPO)

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



+ why is this a *good* idea?

- why is this a *bad* idea?

Gu et al. Continuous deep Q-learning with model-based acceleration. '16
Feinberg et al. Model-based value expansion. '18
Janner et al. When to trust your model: model-based policy optimization. '19

Multi-Step Models & Successor Representations

What kind of model do we need to **evaluate** a policy?

The job of the model is to **evaluate** the policy

(if you can evaluate it, you can make it better)

$$J(\pi) = E_{s \sim p(s_1)}[V^\pi(s_1)]$$

$$V^\pi(s_t) = \sum_{t=t'}^{\infty} \gamma^{t'-t} E_{p(s_{t'}|s_t)} E_{\mathbf{a}_{t'} \sim \pi(\mathbf{a}_{t'}|\mathbf{s}_{t'})}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})]$$

let's keep it simple

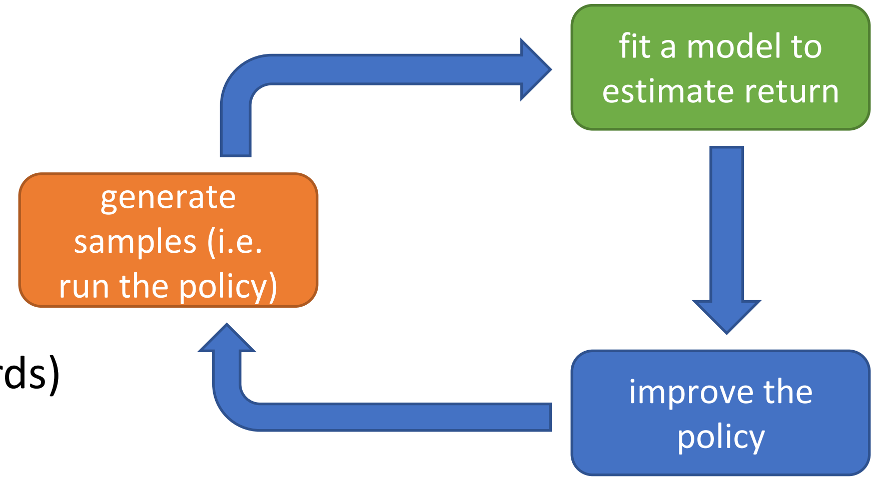
$$= \sum_{t=t'}^{\infty} \gamma^{t'-t} E_{p(s_{t'}|s_t)}[r(\mathbf{s}_{t'})]$$

(easy to re-derive for action-dependent rewards)

$$= \sum_{t=t'}^{\infty} \gamma^{t'-t} \sum_{\mathbf{s}} p(\mathbf{s}_{t'} = \mathbf{s}|\mathbf{s}_t) r(\mathbf{s})$$

$$= \sum_{\mathbf{s}} \left(\sum_{t=t'}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = \mathbf{s}|\mathbf{s}_t) \right) r(\mathbf{s})$$

fit model $f(\mathbf{s}, \mathbf{a})$



What kind of model do we need to **evaluate** a policy?

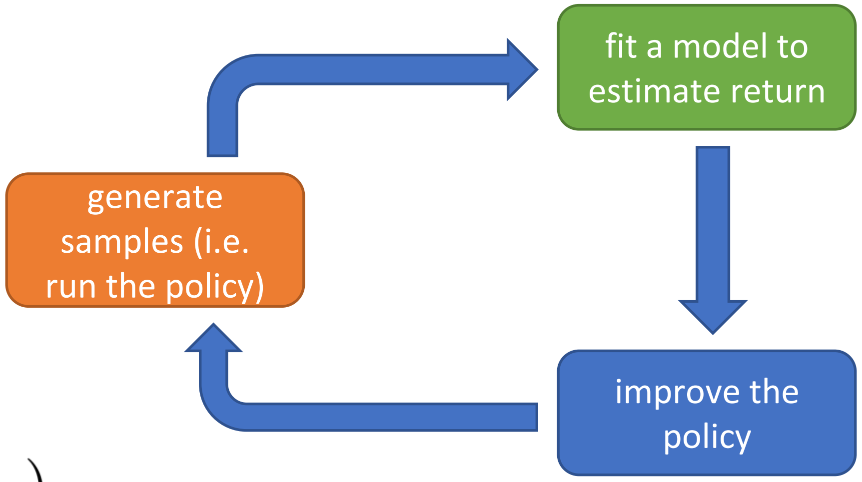
$$V^\pi(\mathbf{s}_t) = \sum_{t=t'}^{\infty} \gamma^{t'-t} E_{p(s_{t'}|s_t)}[r(\mathbf{s}_{t'})]$$

$$= \sum_{\mathbf{s}} \underbrace{\left(\sum_{t=t'}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = \mathbf{s} | s_t) \right)}_{p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s}|\mathbf{s}_t)} r(\mathbf{s})$$

$$p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s}|\mathbf{s}_t) = \underbrace{(1-\gamma)}_{\text{just to ensure it sums to 1}} \sum_{t'=t}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = \mathbf{s}|\mathbf{s}_t)$$

(if you can evaluate it, you can make it better)

fit model $f(\mathbf{s}, \mathbf{a})$



What kind of model do we need to **evaluate** a policy?

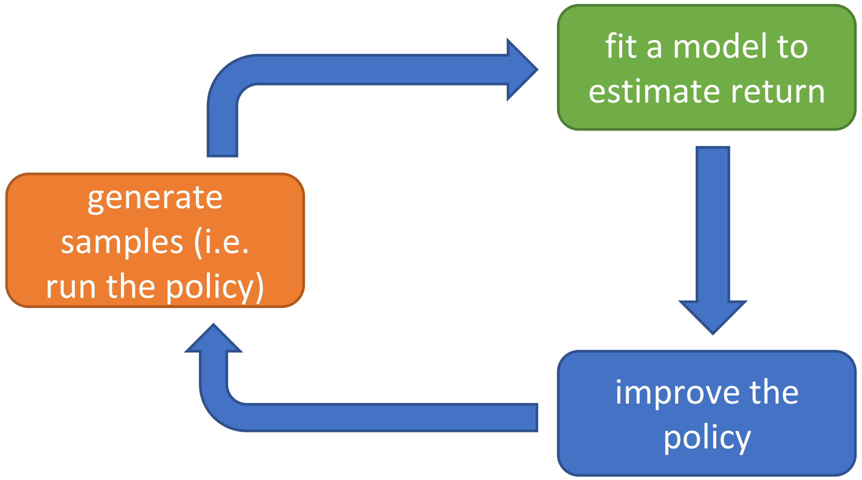
$$p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s}|\mathbf{s}_t) = (1-\gamma)\sum_{t'=t}^{\infty}\gamma^{t'-t}p(\mathbf{s}_{t'} = \mathbf{s}|\mathbf{s}_t)$$

$$V^\pi(\mathbf{s}_t) = \frac{1}{1-\gamma}\underbrace{\sum_{\mathbf{s}} p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s}|\mathbf{s}_t)r(\mathbf{s})}_{\mu^\pi(\mathbf{s}_t)^T\vec{r}}$$

$$\mu_i^\pi(\mathbf{s}_t) = p_\pi(s_{\text{future}} = i|\mathbf{s}_t)$$

(if you can evaluate it, you can make it better)

$\text{fit model } f(\mathbf{s}, \mathbf{a})$



This is called a **successor representation**

Dayan. **Improving Generalisation for Temporal Difference Learning: The Successor Representation**. 1993.

Successor representations

$$\mu_i^\pi(\mathbf{s}_t) = (1-\gamma)\sum_{t'=t}^{\infty}\gamma^{t'-t}p(\mathbf{s}_{t'} = i|\mathbf{s}_t)$$

$$= \underbrace{(1-\gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t\sim\pi(\mathbf{a}_t|\mathbf{s}_t),\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}[\mu_i^\pi(\mathbf{s}_{t+1})]}$$

like a Bellman backup with "reward" $r(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t = i)$

in practice, we can use vectorized backups for all i at once

A few issues…

- Not clear if learning successor representation is easier than model-free RL
- How to scale to large state spaces?
- How to extend to continuous state spaces?

Successor features

$$\mu_i^\pi(\mathbf{s}_t) = (1-\gamma)\sum_{t'=t}^{\infty}\gamma^{t'-t}p(\mathbf{s}_{t'} = i|\mathbf{s}_t) \qquad \psi_j^\pi(\mathbf{s}_t) = \sum_{\mathbf{s}}\mu_{\mathbf{s}}^\pi(\mathbf{s}_t)\phi_j(\mathbf{s}) \qquad \psi_j^\pi(\mathbf{s}_t) = \mu^\pi(\mathbf{s}_t)^T\vec{\phi}_j$$

$$V^\pi(\mathbf{s}_t) = \mu^\pi(\mathbf{s}_t)^T\vec{r}$$

$$\text{if } r(\mathbf{s}) = \textstyle\sum_j \phi_j(\mathbf{s})w_j = \phi(\mathbf{s})^T\mathbf{w}$$

$$\begin{aligned}\text{then } V^\pi(\mathbf{s}_t) &= \psi^\pi(\mathbf{s}_t)^T\mathbf{w} \\ &= \sum_j \psi_j^\pi(\mathbf{s}_t)w_j \\ &= \sum_j \mu^\pi(\mathbf{s}_T)^T\vec{\phi}_j\mathbf{w} \\ &= \mu^\pi(\mathbf{s}_T)^T\sum_j\vec{\phi}_j\mathbf{w} = \mu^\pi(\mathbf{s}_t)^T\vec{r}\end{aligned}$$

so what?

If the number of features is much less than the number of states, learning them is much easier!

Successor features

$$\mu_i^\pi(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[\mu_i^\pi(\mathbf{s}_{t+1})]$$

$$\psi_j^\pi(\mathbf{s}_t) = \phi_j(\mathbf{s}_t) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[\psi_j^\pi(\mathbf{s}_{t+1})]$$

special case with
$\phi_i(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t = i)$

can also construct a "Q-function-like" version:

$$\psi_j^\pi(\mathbf{s}_t, \mathbf{a}_t) = \phi_j(\mathbf{s}_t) + \gamma E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_{t+1} \sim \pi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})}[\psi_j^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})]$$

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^\pi(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w}$ when $r(\mathbf{s}_t) \approx \phi(\mathbf{s}_t)^T \mathbf{w}$

Using successor features

Idea 1: recover a Q-function very quickly

1. Train $\psi^\pi(\mathbf{s}_t, \mathbf{a}_t)$ (via Bellman backups)
2. Get some reward samples $\{\mathbf{s}_i, r_i\}$
3. Get $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} \sum_i ||\phi(\mathbf{s}_i)^T \mathbf{w} - r_i||^2$
4. Recover $Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^\pi(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w}$

Is this the **optimal** Q-function?

$$\pi'(\mathbf{s}) = \arg\max_{\mathbf{a}} \psi^\pi(\mathbf{s}, \mathbf{a})^T \mathbf{w}$$

Equivalent to **one step** of policy iteration

Better than nothing, but **not** optimal

Using successor features

Idea 2: recover many Q-functions

1. Train $\psi^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t)$ for many policies π_k (via Bellman backups)
2. Get some reward samples $\{\mathbf{s}_i, r_i\}$
3. Get $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} \sum_i ||\phi(\mathbf{s}_i)^T \mathbf{w} - r_i||^2$
4. Recover $Q^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w}$ for every π_k

$$\pi'(\mathbf{s}) = \arg\max_{\mathbf{a}} \max_{k} \psi^{\pi_k}(\mathbf{s}, \mathbf{a})^T \mathbf{w}$$

Finds the highest reward policy in **each state**

Barreto et al. **Successor Features for Transfer in Reinforcement Learning**. 2016.

Continuous successor representations

$$\mu_i^\pi(\mathbf{s}_t) = (1-\gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[\mu_i^\pi(\mathbf{s}_{t+1})]$$

always zero for any sampled state if states are continuous

Framing successor representation as *classification*:

$$p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

binary classifier

$F = 1$ means $\mathbf{s}_{\text{future}}$ is a future state from $\mathbf{s}_t, \mathbf{a}_t$ under π

$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) \qquad \mathcal{D}_- \sim p^\pi(\mathbf{s})$

Continuous successor representations

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) \qquad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

$$p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

$$p^\pi(F=0|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}})}{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

$$\frac{p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F=0|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})} = \frac{p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}})}$$

$$\frac{p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F=0|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})} p^\pi(\mathbf{s}_{\text{future}}) = p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)$$

constant independent of $\mathbf{a}_t, \mathbf{s}_t$

The C-Learning algorithm

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t) \qquad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

$$p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) + p^\pi(\mathbf{s}_{\text{future}})}$$

To train:

1. Sample $\mathbf{s} \sim p^\pi(\mathbf{s})$ (e.g., run policy, sample from trajectories)
2. Sample $\mathbf{s} \sim p^\pi(\mathbf{s}_{\text{future}}|\mathbf{s}_t, \mathbf{a}_t)$ (e.g., pick $\mathbf{s}_{t'}$ where $t' = t+\Delta$, $\Delta \sim \text{Geom}(\gamma)$)
3. Update $p^\pi(F=1|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s})$ using SGD with cross entropy loss

This is an **on policy** algorithm

Could also derive an **off policy** algorithm

Eysenbach, Salakhutdinov, Levine. **C-Learning: Learning to Achieve Goals via Recursive Classification**. 2020.