CS 175: Project in Artificial Intelligence

Winter 2020

Lecture 4: Deep Reinforcement Learning

Roy Fox
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

Today's lecture

- Deep Learning basics
- Reinforcement learning with function approximation
- Some basic Deep RL algorithms

Basics: Gradient Descent

- Hard to optimize $\min_\theta \mathcal{L}_\theta(\mathcal{D})$ over high-dimensional parameter space $\theta \in \mathbb{R}^d$
- But try to improve gradually by following direction of maximal decrease

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_\theta(\mathcal{D})$$

- All we need is a differentiable loss function, and hope it's "well-behaved"

Basics: Stochastic Gradient Descent (SGD)

- If $\mathcal{L}_\theta(\mathcal{D}) = \sum_{x \in \mathcal{D}} \mathcal{L}_\theta(x)$, we can do Gradient Descent with **big data**:
- Sample a **batch** $\mathcal{B} \subseteq \mathcal{D}$ and take a gradient step with

$$\theta \leftarrow \theta - \alpha \sum_{x \in \mathcal{B}} \nabla_\theta \mathcal{L}_\theta(x)$$

- Fast growing body of theory + heuristics for how to make this work

Basics: Deep Learning

- If we represent $\mathcal{L}_\theta(x) = f_{\theta_L}(\cdots f_{\theta_2}(f_{\theta_1}(x))\cdots)$
  - Denote $x_0 = x \qquad x_\ell = f_\ell(x_{\ell-1}) \qquad \mathcal{L}_\theta(x) = x_L$
- We get **back-propagation**

$$\nabla_{\theta_\ell}\mathcal{L}_\theta(x) = \nabla_{x_{L-1}} f_{\theta_L}(x_{L-1}) \cdots \nabla_{x_\ell} f_{\theta_{\ell+1}}(x_\ell) \nabla_{\theta_\ell} f_{\theta_\ell}(x_{\ell-1})$$

- Enables very expressive model classes from very simple layers
- Shifts algorithmic challenge from optimizer to loss, architecture, and data

Deep MC policy evaluation

- Monte Carlo (MC) evaluation:

$$\xi_i | s \sim p_\pi \qquad V(s) = \frac{1}{N} \sum_i R_i$$

- What if the state space is large?

$$\mathcal{L}_\theta(\xi) = (V_\theta(s_0) - R)^2$$

- With proper parametrization, this can yield generalization over state space

- But still very data inefficient

Deep TD policy evaluation

- On-policy Temporal-Difference (TD) evaluation:

$$\text{for each } (\mathrm{s_i}, \mathrm{a_i}, \mathrm{r_i}, \mathrm{s_i'}) : \quad \Delta V(s_i) \leftarrow \alpha(r_i + \gamma V(s_i') - V(s_i))$$

- Lends itself nicely to SGD:

$$\mathcal{L}_\theta(s, a, r, s') = (r + \gamma V_\theta(s') - V_\theta(s))^2$$

- Using both current-state $V_\theta(s)$ and next-state $V_\theta(s')$ may be unstable
  - Heuristic: use **target network** $V_{\bar{\theta}}(s')$, update it periodically with $\bar{\theta} \leftarrow \theta$

Deep MC reinforcement learning

- A variant of Monte Carlo Tree Search (MCTS):

$$\xi \sim p_{\pi_{\bar{\theta}}} \qquad \mathcal{L}_\theta(\xi) = (Q_\theta(s_0, a_0) - R)^2$$

  - With $\pi_{\bar{\theta}}$ greedy for a snapshot of Q_θ

- We need a representation of Q_θ that allows computing

$$\pi_\theta(s) = \operatorname*{argmax}_a Q_\theta(s, a)$$

- For a small action space: Deep Q Network

$$(q_\theta(s))_a = Q_\theta(s, a)$$

$$s \longrightarrow \boxed{q_\theta} \longrightarrow Q_\theta(s, \cdot)$$

- π_θ is not differentiable, but we don't need it to be

Deep TD reinforcement learning

- Deep Q Learning (historically called DQN):

$$\mathcal{L}_\theta(s, a, r, s') = (r + \gamma \max_{a'} Q_{\boxed{\bar{\theta}}}(s', a') - Q_\theta(s, a))^2$$

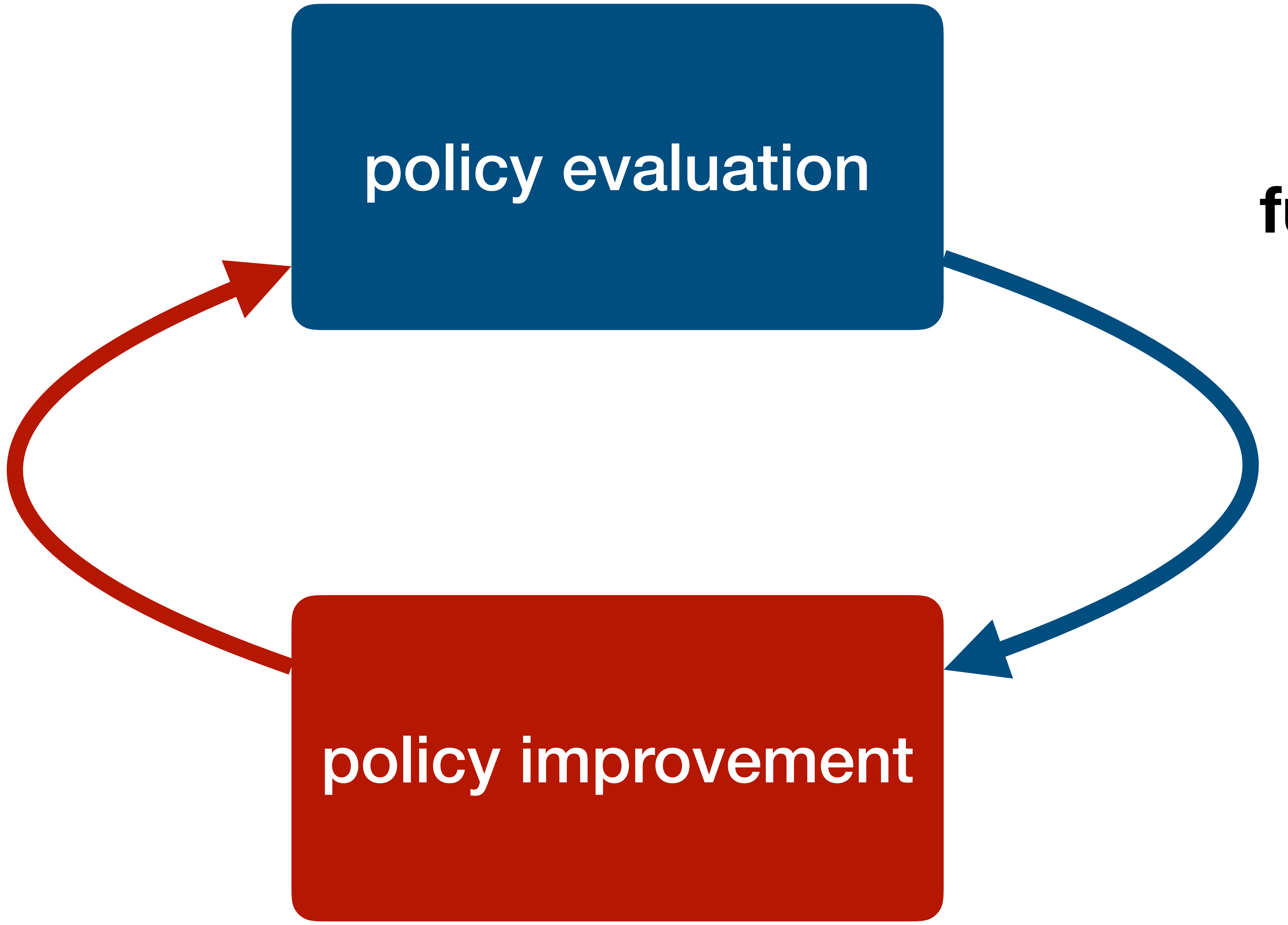
- This algorithm should work off-policy, so we can keep **replay buffer**

- Variants differ on

    - How to add experience to the buffer

    - How to sample from the buffer

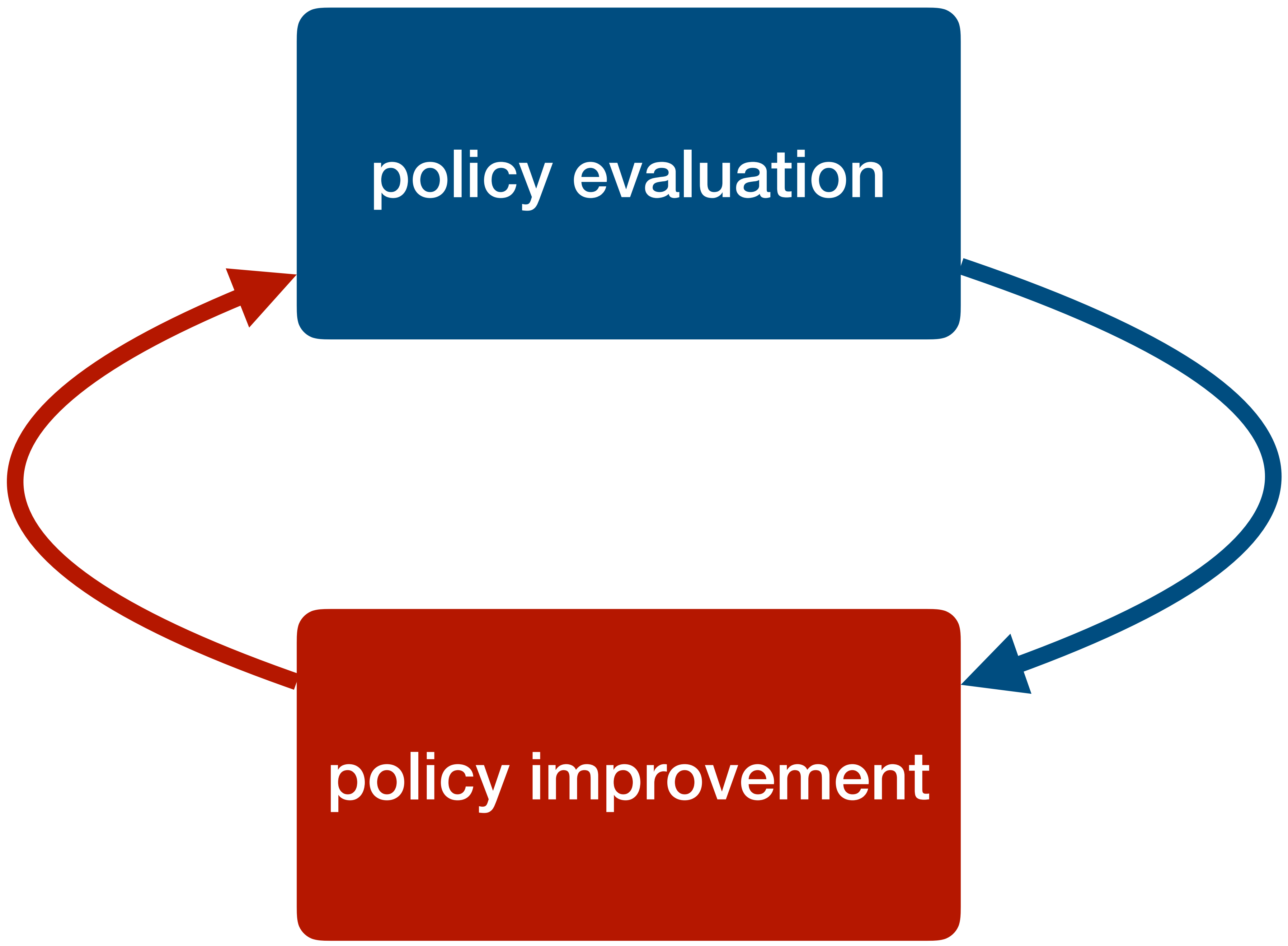
DQN



differentiable value function approximation

greedy policy

Policy gradient



e.g. MC

differentiable policy

Policy gradient

- Unlike minimizing $\mathcal{L}_\theta(\mathcal{D})$ in general ML, in RL we maximize $\mathcal{J}_\theta = \mathbb{E}_{\xi\sim p_{\pi_\theta}}[R]$
- This is harder since the "data" distribution depends on θ
- But there's a trick:

$$\begin{aligned}\nabla_\theta \mathcal{J}_\theta &= \nabla_\theta \int p_\theta(\xi) R(\xi) d\xi \\ &= \int p_\theta(\xi) \nabla_\theta \log p_\theta(\xi) R(\xi) d\xi \\ &= \mathbb{E}_{\xi\sim p_\theta}[\nabla_\theta \log p_\theta(\xi) R]\end{aligned}$$

REINFORCE (1992 !)

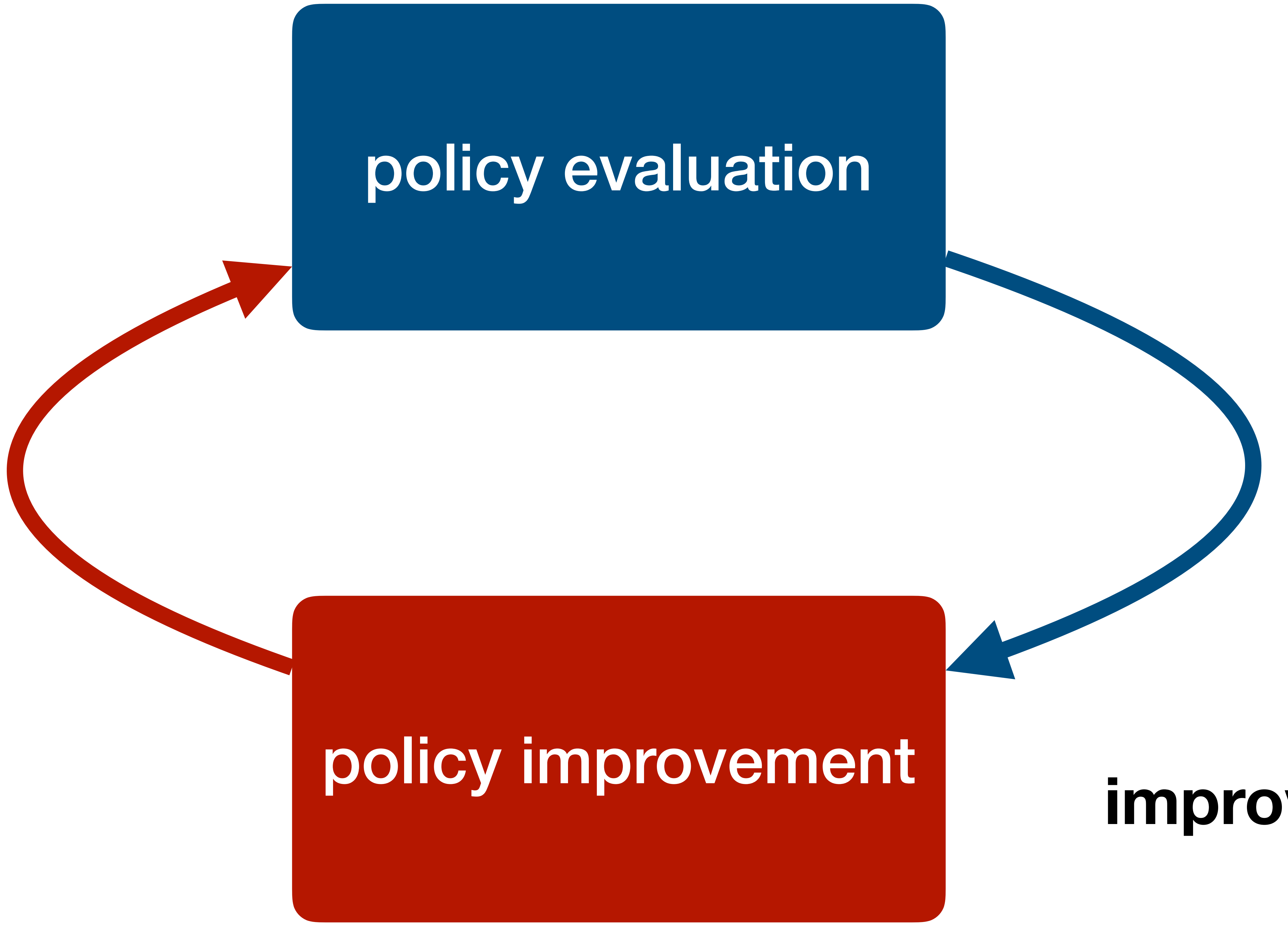
- Roll out π_θ to sample $\xi \sim p_\theta$
- Compute R and

$$\nabla_\theta \log p_\theta(\xi) = \nabla_\theta\left(\log p(s_0) + \sum_t (\log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t))\right)$$

- Take a gradient step with $\nabla_\theta \log p_\theta(\xi) R$
- Repeat

- This is on-policy + has very high variance of the gradient estimator

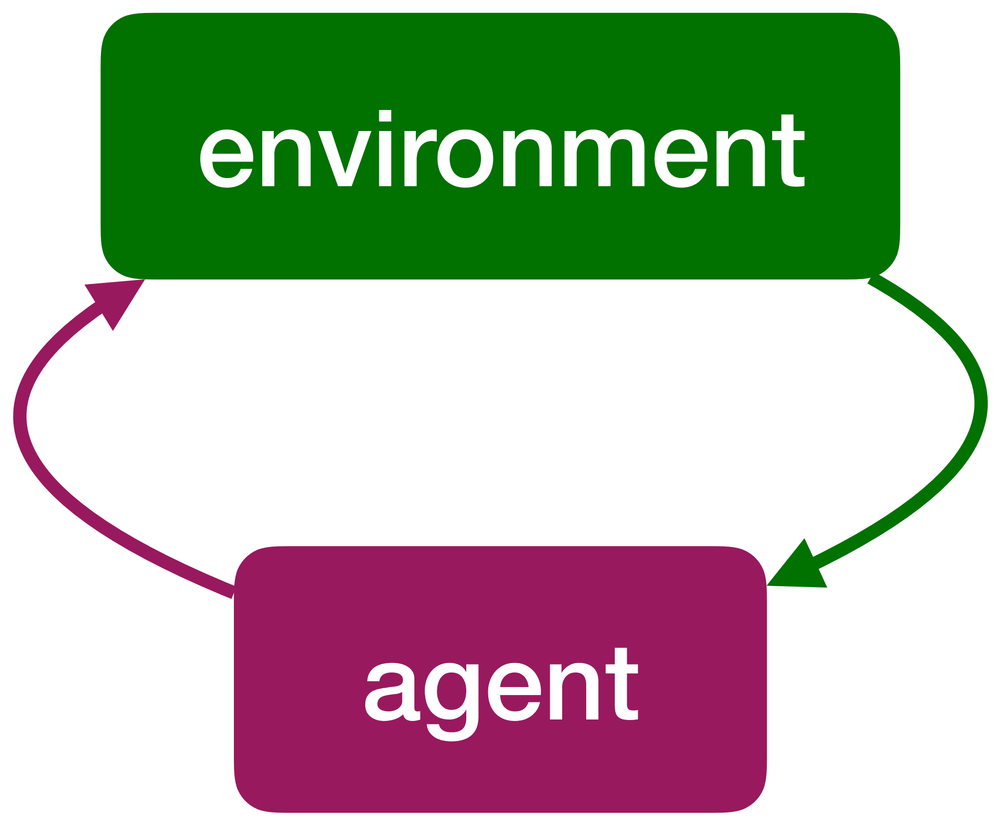
Actor–Critic

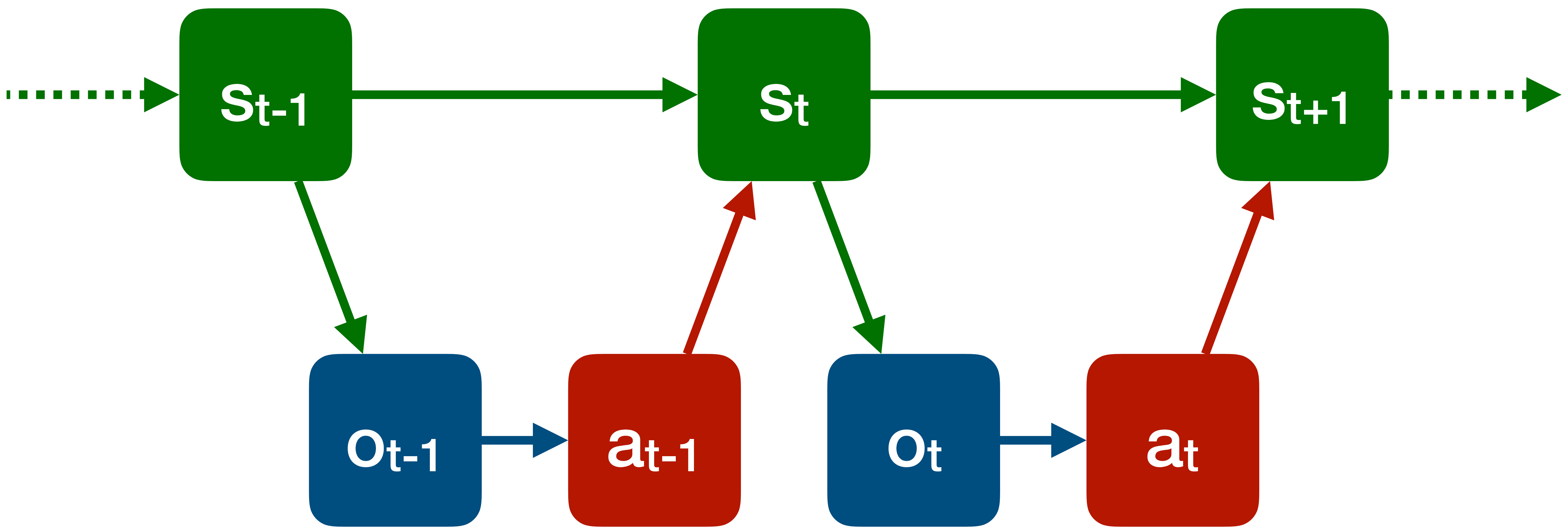


critic:
evaluates the actor

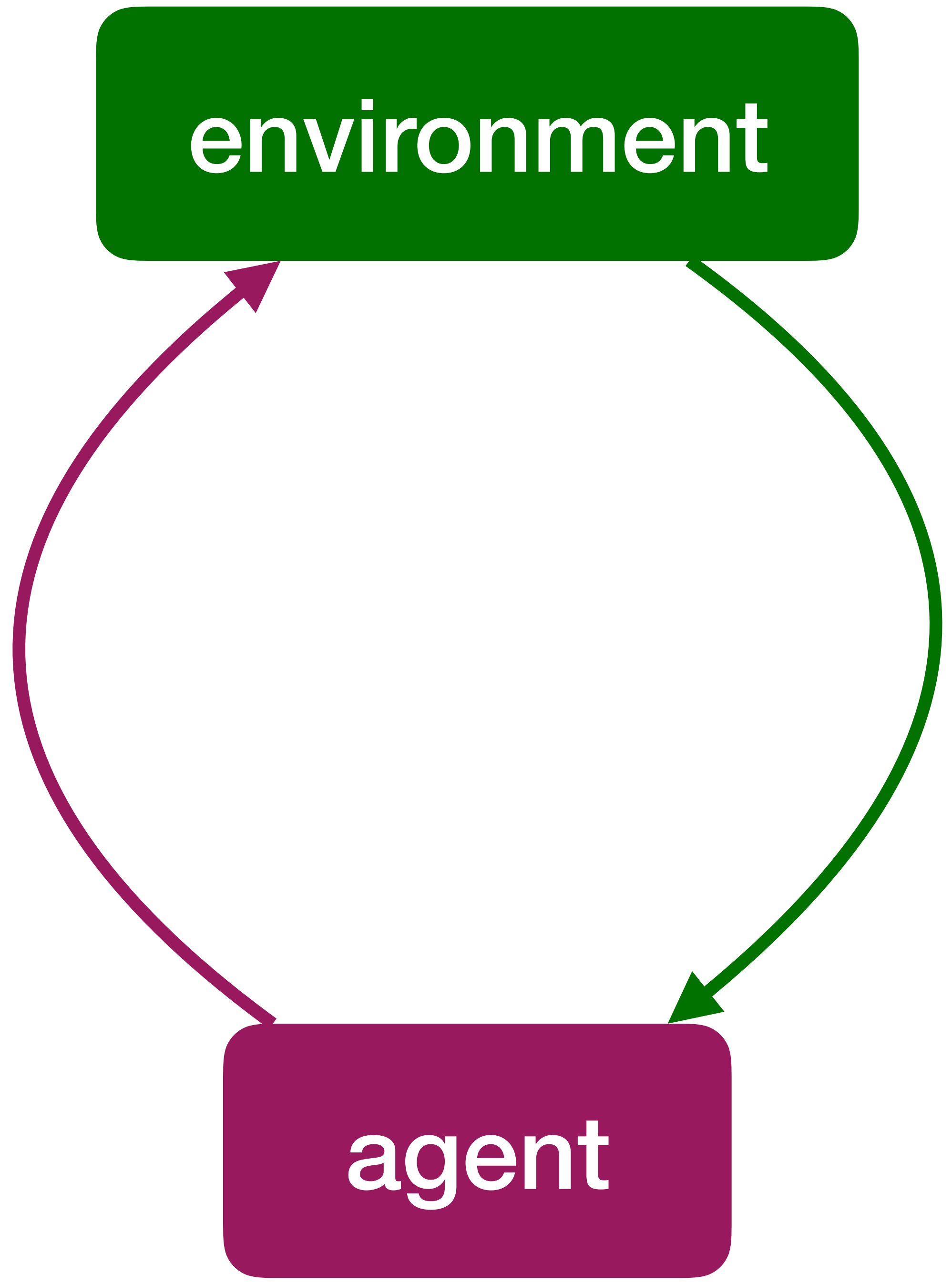
actor:
improves using the critic's advice

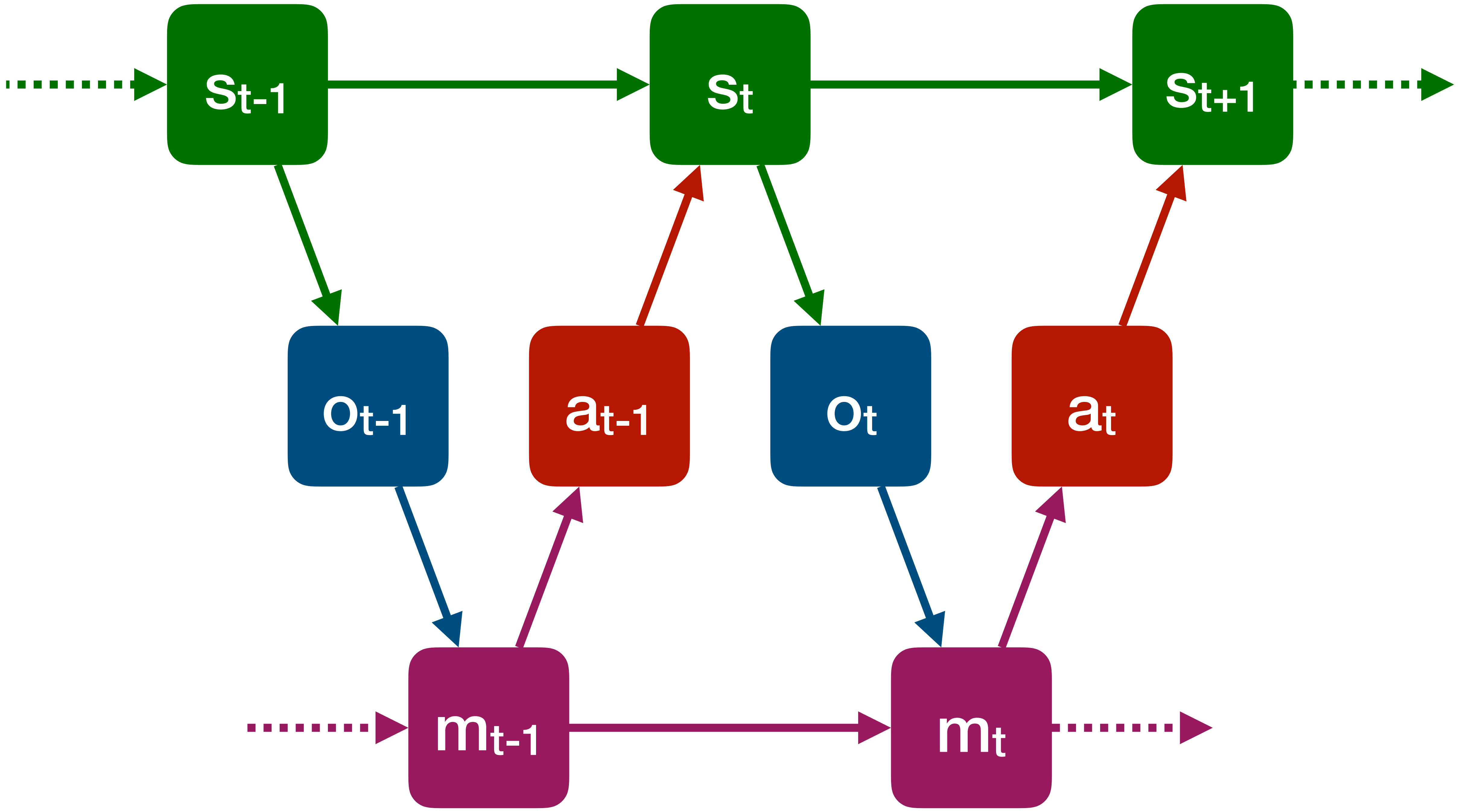
Partial observability





Partial observability





$$\pi_\theta(m_t, a_t | m_{t-1}, o_t)$$

- Can be represented by a Recurrent Neural Network (RNN)
  - For example, Long Short-Term Memory (LSTM)

How to choose a Deep RL algorithm?

- Continuous or discrete action space?
- Stochastic or deterministic policy?
- Sample efficiency — generally speaking:
    - Off-policy > on-policy
    - Model-based > TD > PG
- Robustness
- Well-studied, well-supported

Recap

- Policy evaluation and reinforcement learning with function approximation
- Can represent the value function: DQN, SQL, etc.
- Or the policy: PG, DPG, DDPG, TRPO, PPO, etc.
- Or both: A2C, SAC, etc.
- Did not mention model-based Deep RL, derivative-free methods, etc.