SAFE ROBOTICS LABORATORY

Reinforcement Learning Tutorial

Kai-Chieh Hsu

Aug. 11, 2022

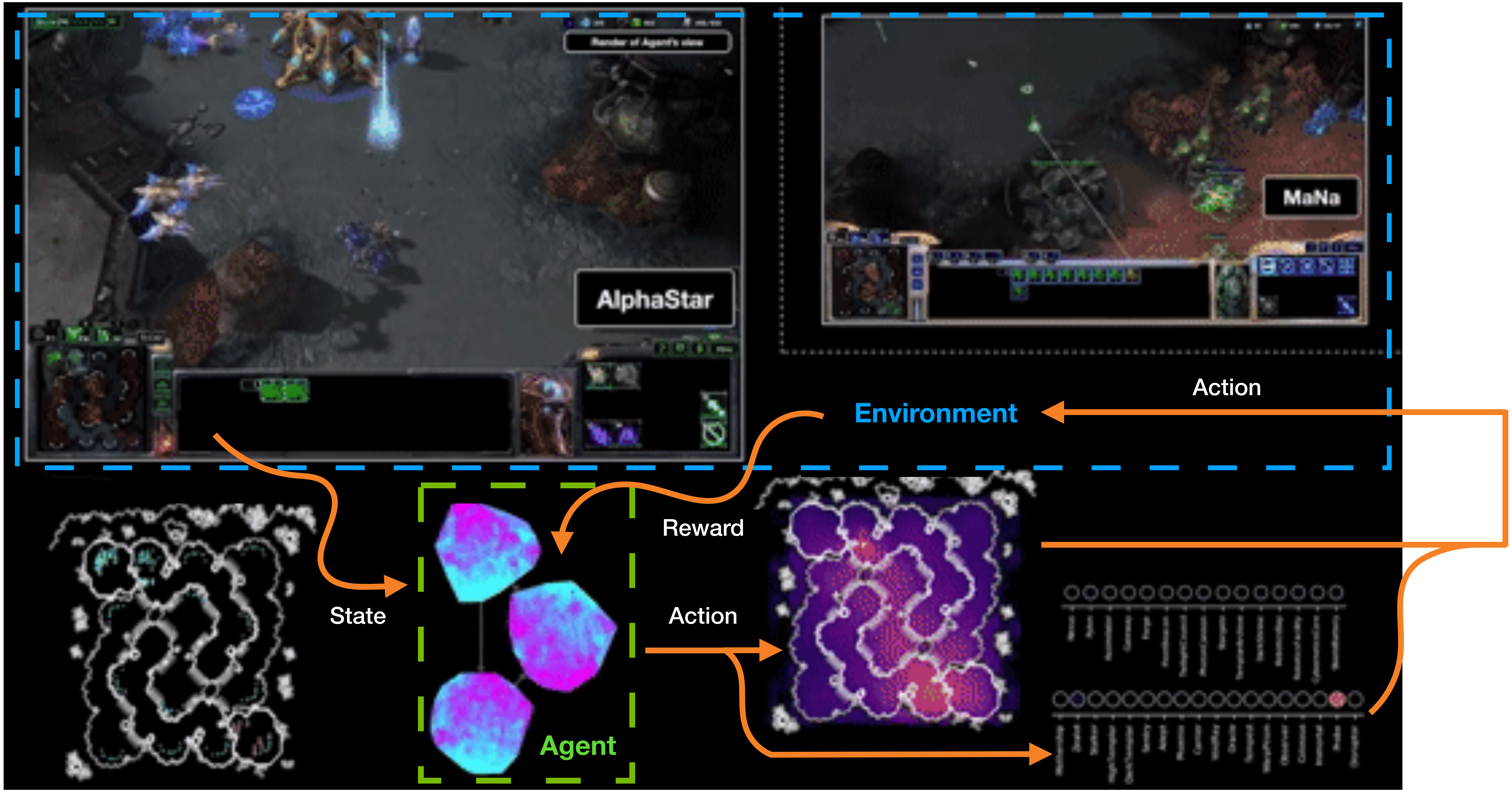
Outline

- Introduction: Reinforcement Learning and Markov Decision Process
- Dynamic Programming
  - Value Iteration and Policy Iteration
  - Model-Based Reinforcement Learning
- Model-Free Reinforcement Learning
  - Temporal-Difference Learning
  - Policy Gradient and Actor-Critic
- Discussion: Research Directions
  - Safe Reinforcement Learning
  - Multiagent Reinforcement Learning

David Silver, Lectures on Reinforcement Learning, https://www.davidsilver.uk/teaching/

Sequential decision making
- **Long-term effect**
- **Delayed reward**

Reinforcement Learning (RL)



Vinyals et al., AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning

Markov Decision Process (MDP)

- MDP is a mathematical framework to describe an environment in RL.
- Markov Property (the main assumption in MDP)

The future is independent of the past given the present

A MDP is a tuple of $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{P}_0, \mathcal{R}, \gamma)$

- $\mathcal{S} = \{s_1, s_2, \cdots\}$ is the state space
- $\mathcal{A} = \{a_1, a_2, \cdots\}$ is the action space
- $\mathcal{P}: \mathcal{S} \times \mathcal{A} \rightarrow \Delta\mathcal{S}$ is the transition function: $\mathcal{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$
- $\mathcal{P}_0 \in \Delta\mathcal{S}$ is the distribution of the initial state: $\mathcal{P}_0(s_0 = s)$
- $\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function: $\mathcal{R}(s, a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a]$
- $\gamma \in (0, 1]$ is the discount factor (how much you care about the future)

$$\pi(a \mid s) = \Pr(a_t = a \mid s_t = s)$$

- RL wants to find a (stochastic) policy $\pi : \mathcal{S} \to \Delta\mathcal{A}$, that maximizes the return at time t=0 in this environment

$$\mathbb{E}_{s_0 \sim \mathcal{P}_0, s_{t+1} \sim \mathcal{P}(\cdot|s_t,a_t), a_t \sim \pi(\cdot|s_t)} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \right] =: \mathbb{E}_{\mathcal{P}_0} \left[\mathbb{E}_\pi \left[G_0 \mid s_0 = s \right] \right]$$

where the return at time t is the total discounted reward from time step t

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- Value function is the reward-to-go (or cost-to-go) from each state:

$$V^\pi(s) := \mathbb{E}_\pi[G_t \mid s_t = s] = \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s]$$

- Quality function (or Q-function) is the reward-to-go (or cost-to-go) given an initial state-action pair:

$$Q^\pi(s, a) := \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a]$$

- The connection between the value function and the Q-function

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) Q^\pi(s, a) \qquad Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) V^\pi(s')$$

Planning by Dynamic Programming (DP)

- Principle of Optimality: the subsolutions of an optimal solution of the problem are themselves optimal solutions for their subproblems.
- Overlapping subproblems: cached and reused
- MDP satisfies these properties as
  - Bellman equation provides the optimal structure
  - Value function serves as the cache
- Full knowledge about the underlying MDP

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) V^{\pi}(s') \right)$$

Bellman Expectation Eq.

$$V^{*}(s) = \max_{a \in \mathcal{A}} \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) V^{*}(s')$$

Bellman Optimality Eq.

Policy Iteration (PI) and Value Iteration (VI)

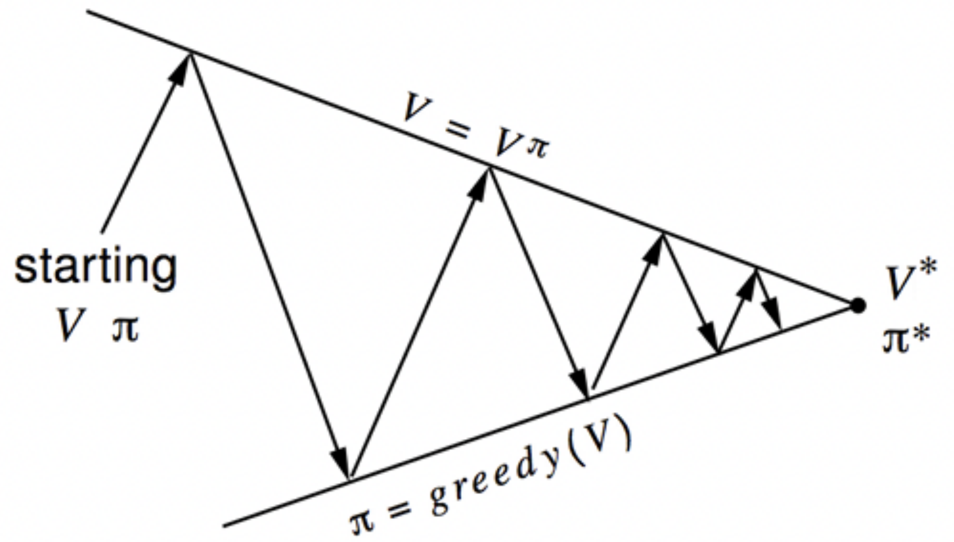
Policy Evaluation

$$V^{\pi^{k+1}}(s) = \sum_{a \in \mathcal{A}} \pi^k(a \mid s) Q^{\pi^{k+1}}(s,a)$$

$$Q^{\pi^{k+1}}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s,a) V^{\pi^k}(s')$$

Policy Improvement

$$\pi^{k+1}(a \mid s) = \begin{cases} 1, & a = \text{argmax}_{a' \in \mathcal{A}} Q^{\pi^{k+1}}(s,a') \\ 0, & \text{otherwise} \end{cases}$$



Bellman Expectation Eq.

Value Evaluation

$$V^{k+1}(s) = \max_{a \in \mathcal{A}} \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s,a) V^k(s')$$

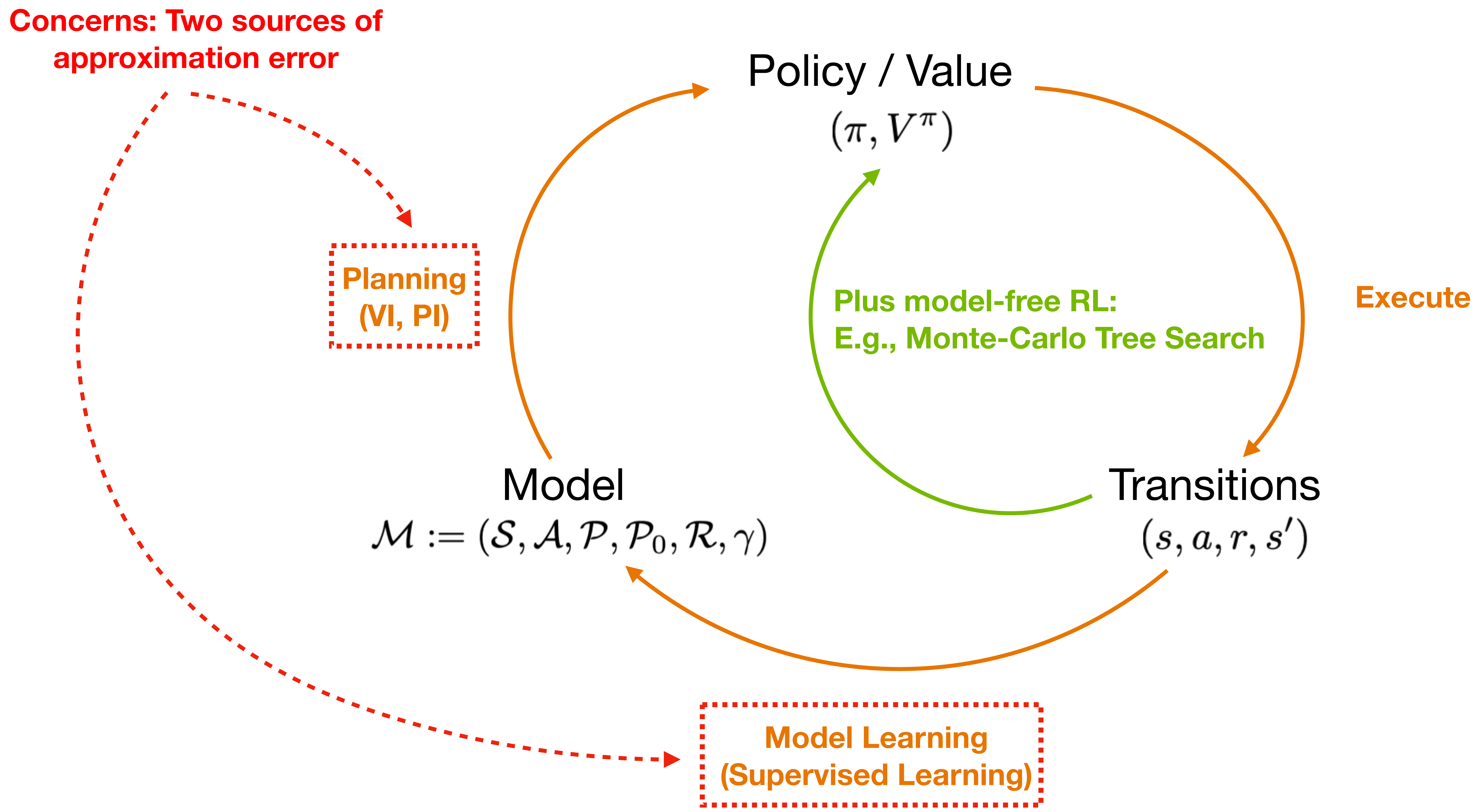
Optimal Policy

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s,a) V^*(s')$$

$$\pi^*(a \mid s) = \begin{cases} 1, & a = \text{argmax}_{a' \in \mathcal{A}} Q^*(s,a') \\ 0, & \text{otherwise} \end{cases}$$

Bellman Optimality Eq.

Model-Based RL



Model-Free RL

- Instead of full backup, can we learn the Q-function/policy from episodes of transitions?
- Q-learning: learns the Q-function and then obtains the optimal policy by arg max
- Policy Gradient methods: directly learns the optimal policy by experiences
- Actor-Critic methods: combines both TD learning and policy gradient methods

Monte-Carlo (MC) and Temporal-Difference (TD)

$\approx r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$

Sampling: updates with multiple transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ instead of a full backup

Bootstrapping: updates toward an estimated return (TD target)

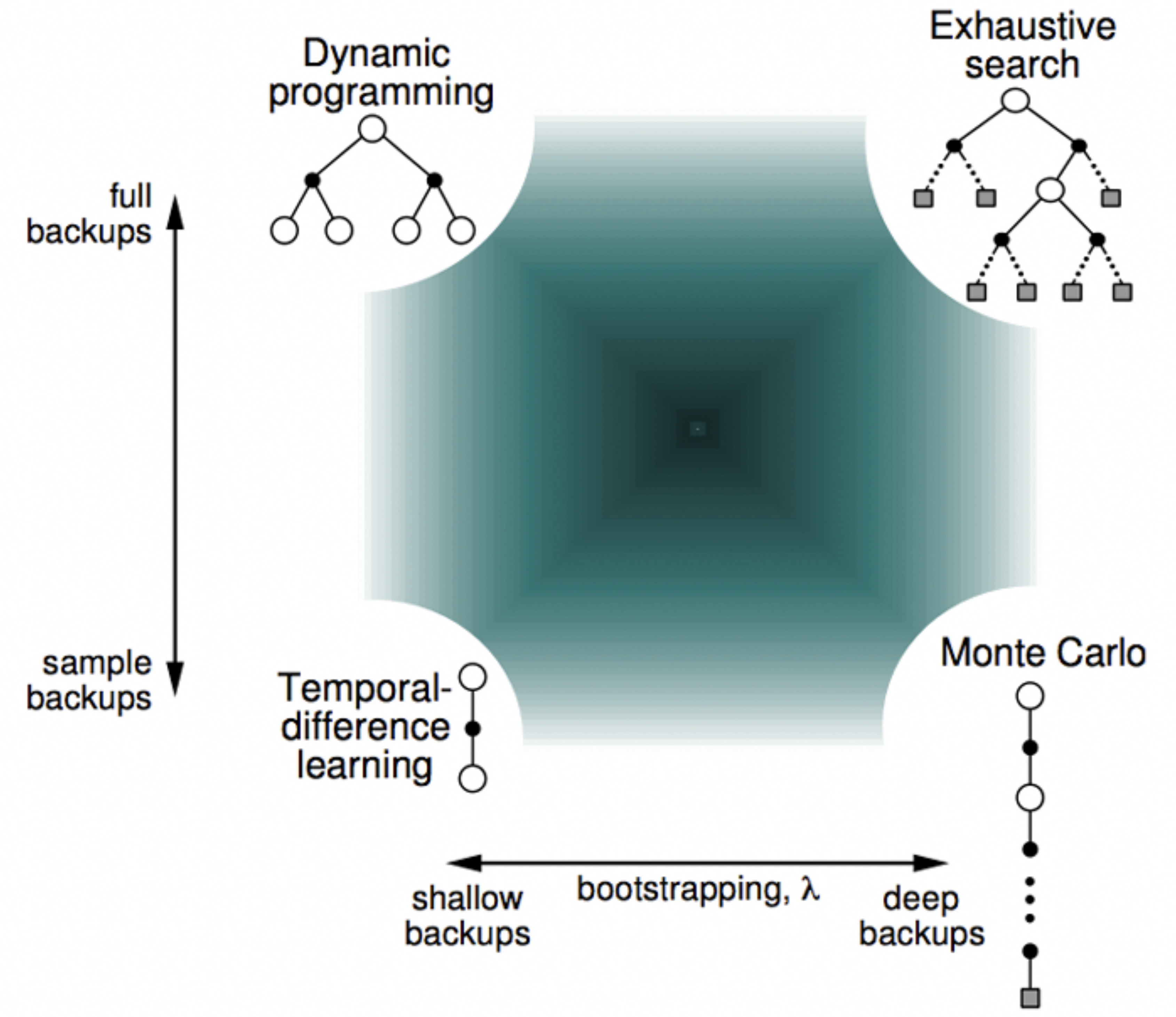
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)$$

TD error

- **This Q-function can be parameterized by a neural network (DQN)!**
- **Discrete action space and ϵ-greedy exploration**

$$a_t = \begin{cases} a \sim \pi(\cdot \mid s_t), & x > \epsilon \\ a \sim \mathrm{Unif}(\mathcal{A}), & x \leq \epsilon \end{cases}$$

- **Off-Policy algorithm: the policy to sample actions is different from the policy we optimize.**



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(G_t - Q(s_t, a_t)\right)$$

MC prediction:
- **episodic environments**
- **zero bias, high variance**

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Mnih et al., Playing Atari with Deep Reinforcement Learning, NuerIPS workshop, 2013

Temporal-Difference (TD) Learning

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \approx r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$$

Sampling: updates with multiple transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ instead of a full backup

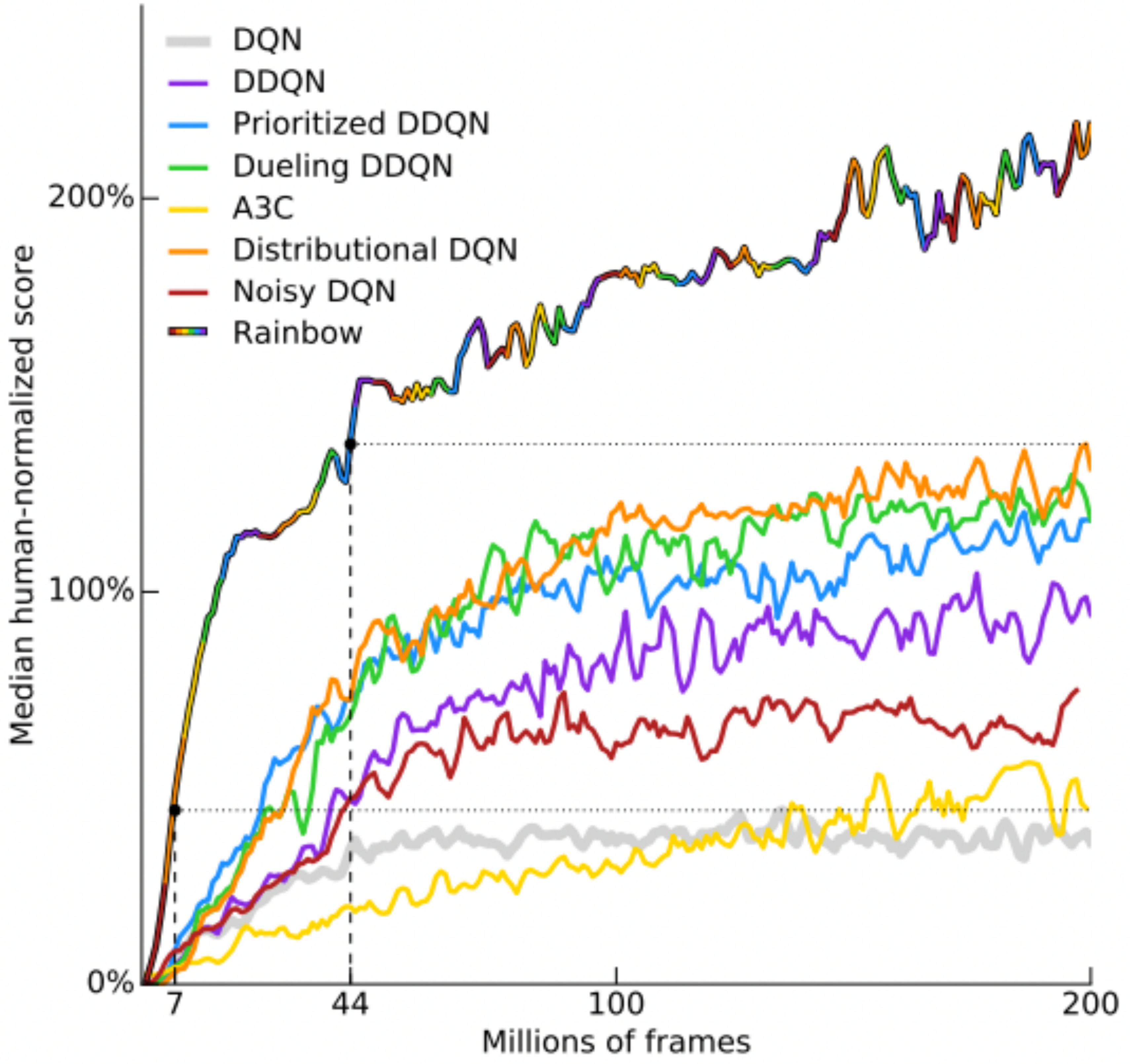
Bootstrapping: updates toward a estimated return (TD target)

$$Q_\omega(s_t, a_t) \leftarrow Q_\omega(s_t, a_t) + \alpha\Big(\underbrace{r_{t+1} + \gamma Q_\omega(s_{t+1}, a_{t+1}) - Q_\omega(s_t, a_t)}_{\text{TD error}}\Big)$$

- **There are a couple of tricks to make this moving target update more stable.**
- **One well-known trick is called double Q-network (DDQN).**

$$Q_\omega(s_t, a_t) \leftarrow Q_\omega(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma Q_{\omega'}(s_{t+1}, a_{t+1}) - Q_\omega(s_t, a_t)\Big)$$

$$a_{t+1} = \operatorname*{argmax}_a Q_\omega(s_{t+1}, a)$$



Hessel et al., Rainbow: Combining Improvements in Deep Reinforcement Learning, AAAI, 2018
Van Hasselt et al., Deep Reinforcement Learning with Double Q-learning, AAAI, 2016

Policy Gradient

- **What if the action space is continuous?**
- **Can we have a stochastic policy?**

→ Directly optimizes the policy

visitation frequency

$$\rho^{\pi_\theta}(s) = \mathcal{P}_0(s) + \sum_{t=1}^{\infty} \gamma^t P(s_t = s)$$

$$J(\theta) := \mathbb{E}_{\mathcal{P}_0}\left[\mathbb{E}_{\pi_\theta}\left[G_0 \mid s_0 = s\right]\right]$$

$$= \sum_s \rho^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) Q^{\pi_\theta}(s,a) = \mathbb{E}_{\rho^{\pi_\theta},\pi_\theta}\left[Q^{\pi_\theta}(s_t, a_t)\right]$$

$$\nabla_\theta J(\theta) = \sum_s \rho^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log(\pi_\theta) Q^{\pi_\theta}(s,a)$$

$$= \mathbb{E}_{\rho^{\pi_\theta},\pi_\theta}\left[\nabla_\theta \log(\pi_\theta) Q^{\pi_\theta}(s_t, a_t)\right]$$

$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$	**REINFORCE**
$A_t^{\pi_\theta} := Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$	**Proximal Policy Optimization**
$Q_\omega(s_t, a_t)$	**Soft Actor-Critic**

Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine Learning, 1992
Schulman et al., Proximal Policy Optimization Algorithms, arXiv, 2017
Haarnoja et al., Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, ICML, 2018

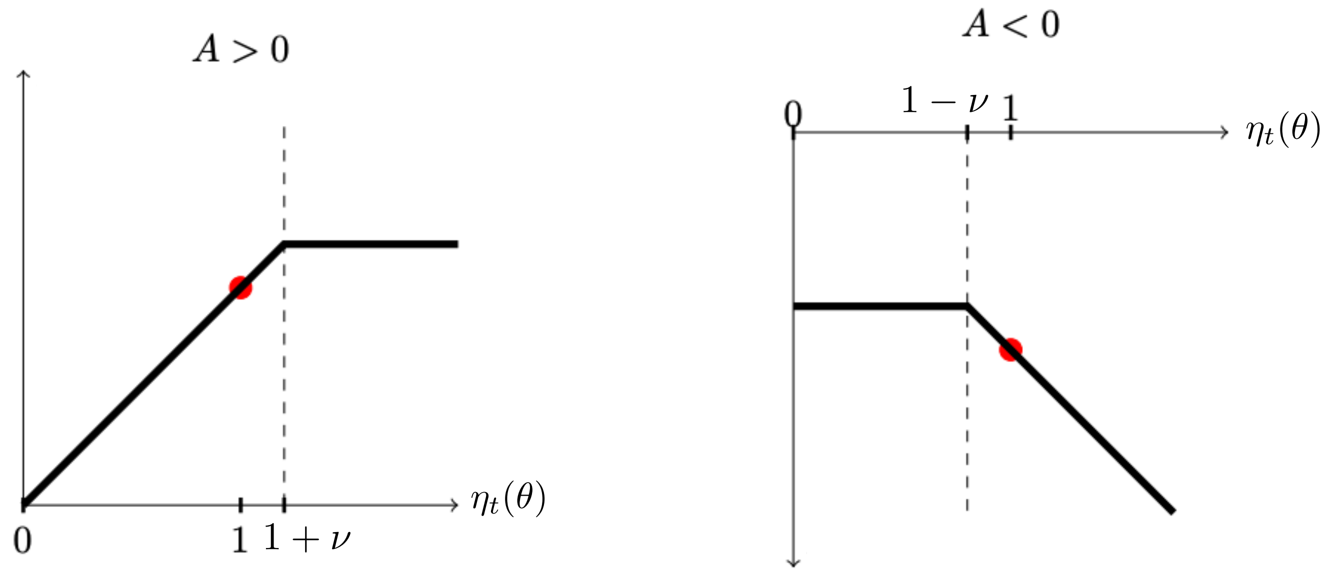
$$J(\theta) = \mathbb{E}_{\rho^{\pi_\theta}, \pi_\theta}\left[Q^{\pi_\theta}(s_t, a_t)\right]$$

Proximal Policy Optimization

$$\begin{aligned}\mathbb{E}_{\rho^{\pi_\theta}, \pi_\theta}\left[A_t^{\pi_\theta}\right] &\approx \mathbb{E}_{\rho^{\pi_{\theta_k}}, a_t \sim \pi_\theta(\cdot|s_t)}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)} A_t^{\pi_{\theta_k}}\right] \\ &=: \mathbb{E}_{\rho^{\pi_{\theta_k}}, a_t \sim \pi_\theta(\cdot|s_t)}\left[\eta_t(\theta) A_t^{\pi_{\theta_k}}\right] \\ &\approx \frac{1}{\sum_{n=1}^{N} T_n} \sum_{n=1}^{N} \sum_{t=0}^{T_n - 1} \eta_{n,t}(\theta) A_{n,t}^{\pi_{\theta_k}}\end{aligned}$$

- **On-Policy Algorithm: trajectories are sampled by π_{θ_k}.**
- **The updated policy should not be too far from the old one.**

$$J(\theta) = \frac{1}{\sum_{n=1}^{N} T_n} \sum_{n=1}^{N} \sum_{t=0}^{T_n - 1} \min\left\{\eta_{n,t}(\theta) A_{n,t}^{\pi_{\theta_k}},\ \text{clip}\big(\eta_{n,t}(\theta), 1-\nu, 1+\nu\big) A_{n,t}^{\pi_{\theta_k}}\right\}$$

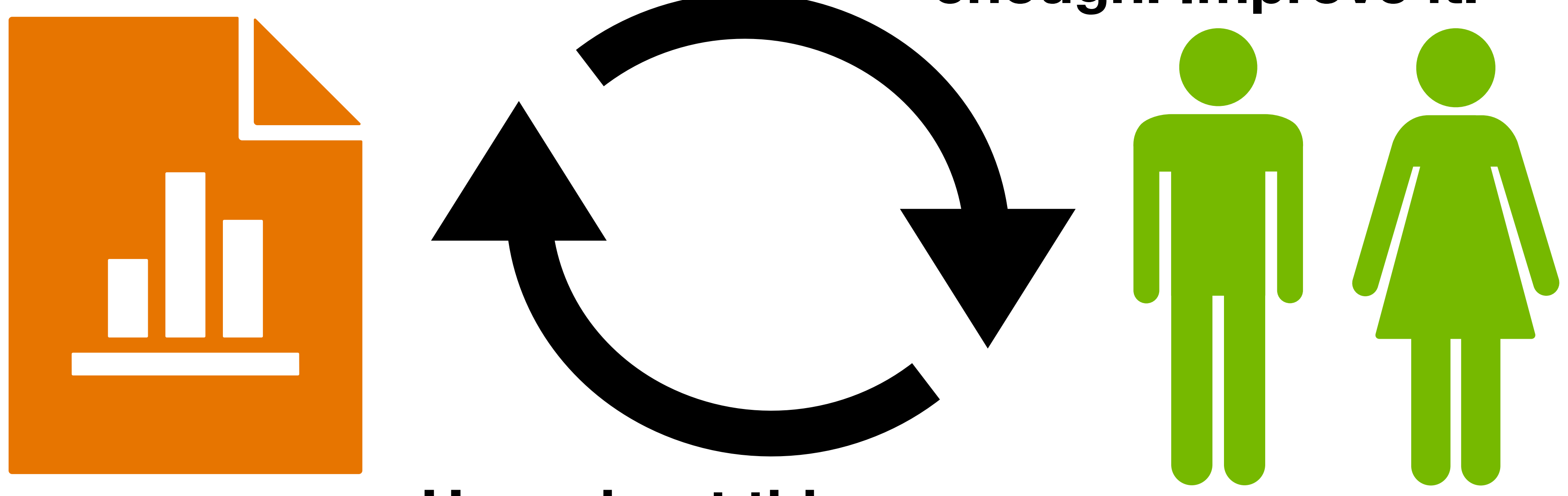


Soft Actor-Critic

- **Two neural networks to parameterize Q-function and policy**
- Q_ω **is called critic because it estimates the quality of the parameterized policy.**
- π_θ **is called actor since it determines how agent reacts in the environment.**
- **Off-Policy algorithm**

$$J(\theta) = \mathbb{E}_{s \sim \mathcal{B}}\left[\mathbb{E}_{a \sim \pi_\theta(\cdot|s)}\left[Q_\omega(s,a) - \alpha \log \pi_\theta(a \mid s)\right]\right] \qquad a' \sim \pi_\theta(\cdot \mid s')$$

$$L(\omega) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{B}}\left[\frac{1}{2}\Big(Q_\omega(s,a) - \big(r + \gamma Q_{\omega'}(s', a')\big)\Big)^2\right]$$

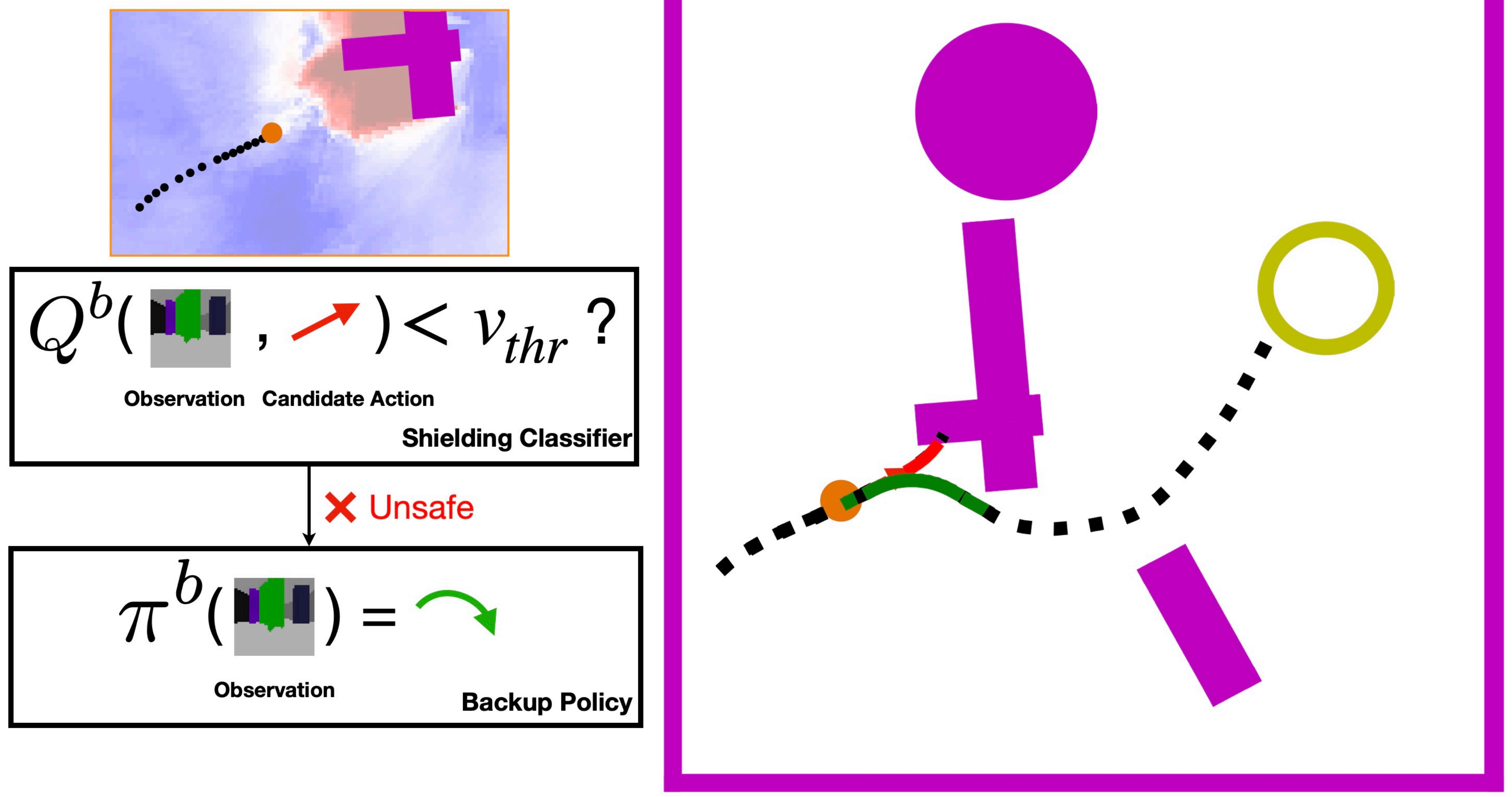


Schulman et al., Proximal Policy Optimization Algorithms, arXiv, 2017
Haarnoja et al., Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, ICML, 2018

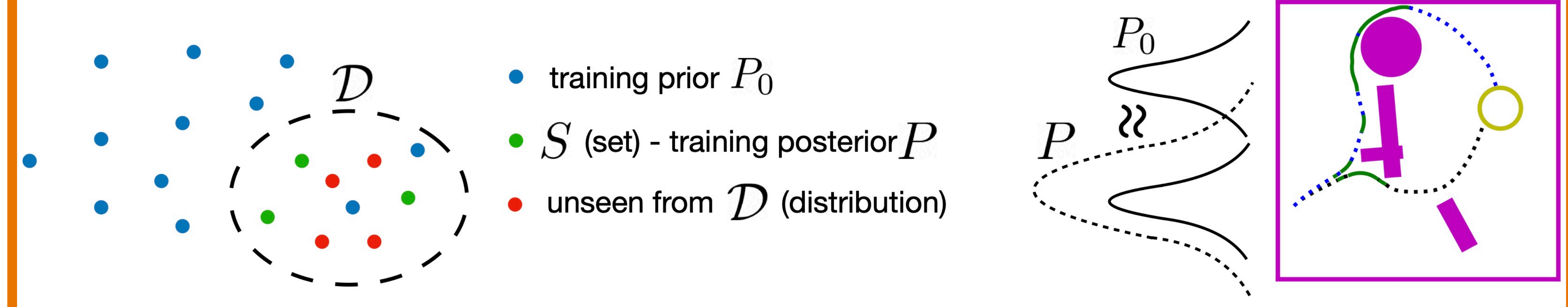
Safe Reinforcement Learning

Safe Exploration

The safety Q-function Q^b and policy π^b



Sim-to-Real Transfer



PAC-Bayes Control: With high probability:

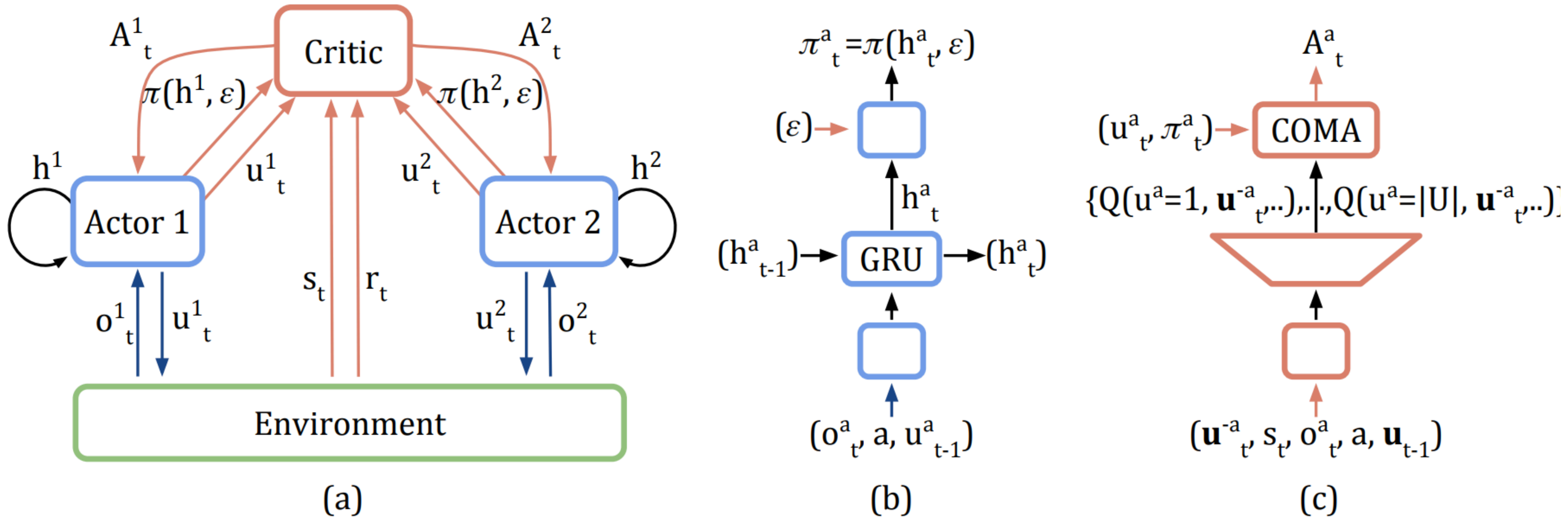
$$R_{\mathcal{D}}(P) \geq R_{\text{PAC}}(P, P_0) := R_S(P) - \sqrt{C(P, P_0)}$$

Test reward	Generalization Bound	Training reward	Regularizer

Hsu, Ren* et al., Sim-to-Lab-to-Real: Safe Reinforcement Learning with Shielding and Generalization Guarantees, arXiv, 2022*

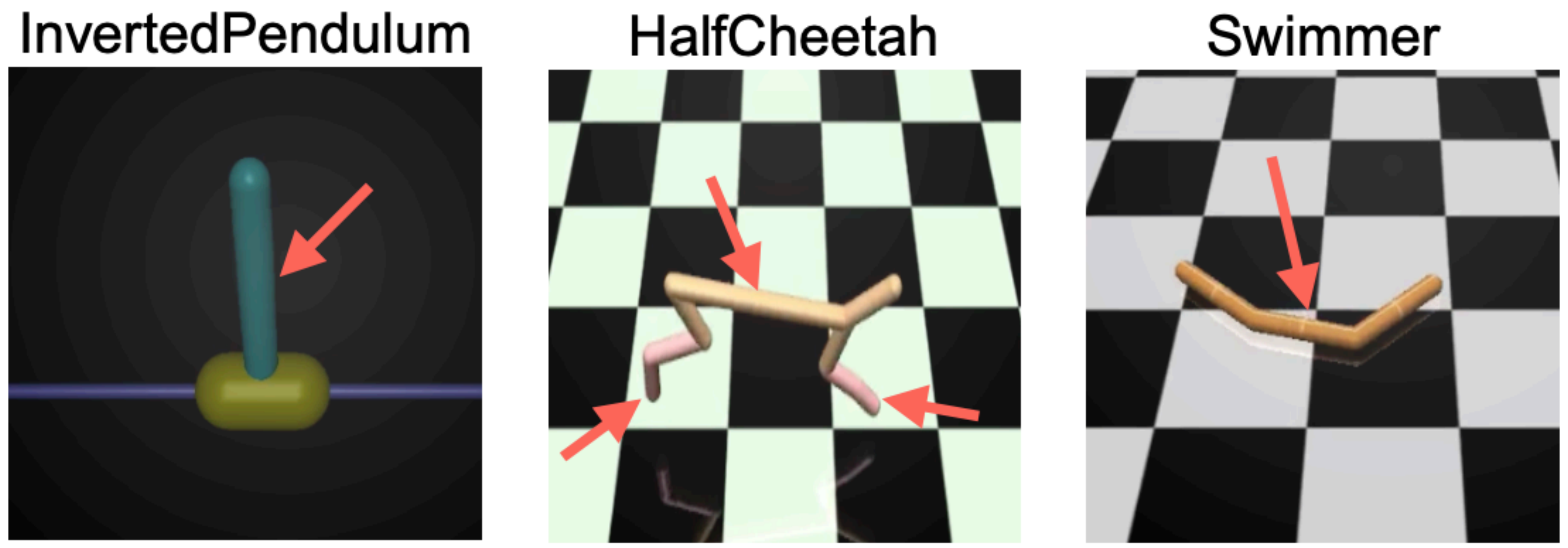
Multiagent Reinforcement Learning

Centralized Training Decentralized Execution



$$A^i(s, \mathbf{a}) := Q(s, \mathbf{a}) - \sum_{a'^i \in \mathcal{A}^i} \pi^i(a'^i \mid h^i) Q\Big(s, (a'^i, \mathbf{a}^{-i})\Big)$$

Robust RL (Zero-Sum Game)



$$\max_{\theta} \min_{\psi} J(\pi_\theta, \pi_\psi) := \mathbb{E}_{\mathcal{P}_0}\Big[\mathbb{E}_{\pi_\theta, \pi_\psi}\big[G_0 \mid s_0 = s\big]\Big]$$

Foerster et al., Counterfactual Multi-Agent Policy Gradients, AAAI, 2018
Pinto et al., Robust Adversarial Reinforcement Learning, ICML, 2017

Other Directions

- How to formulate the reward function? — Inverse Reinforcement Learning
- Can we learn from static data set? — Offline Reinforcement Learning
- Sample complexity and convergence? — Reinforcement Learning Theory
- Is Markov property necessary? — Representation Learning, Transformer
- OTHER THOUGHTS…