

Reinforcement learning

Policy gradient, state-value function, advantage function, actor-critic methods, inverse reinforcement learning

Karel Zimmermann

Problems often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$

$\mathbf{x}$ 

Problems often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$



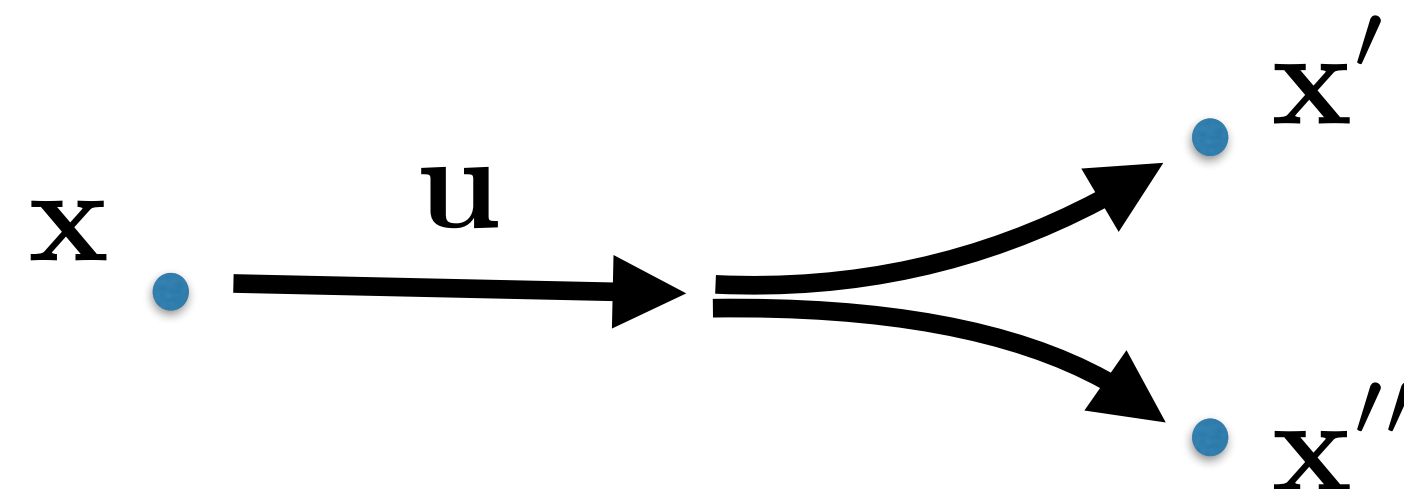
Actions: $\mathbf{u} \in \mathcal{R}^m$

Problems often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$

Actions: $\mathbf{u} \in \mathcal{R}^m$

Model: $p(\mathbf{x}' | \mathbf{x}, \mathbf{u})$



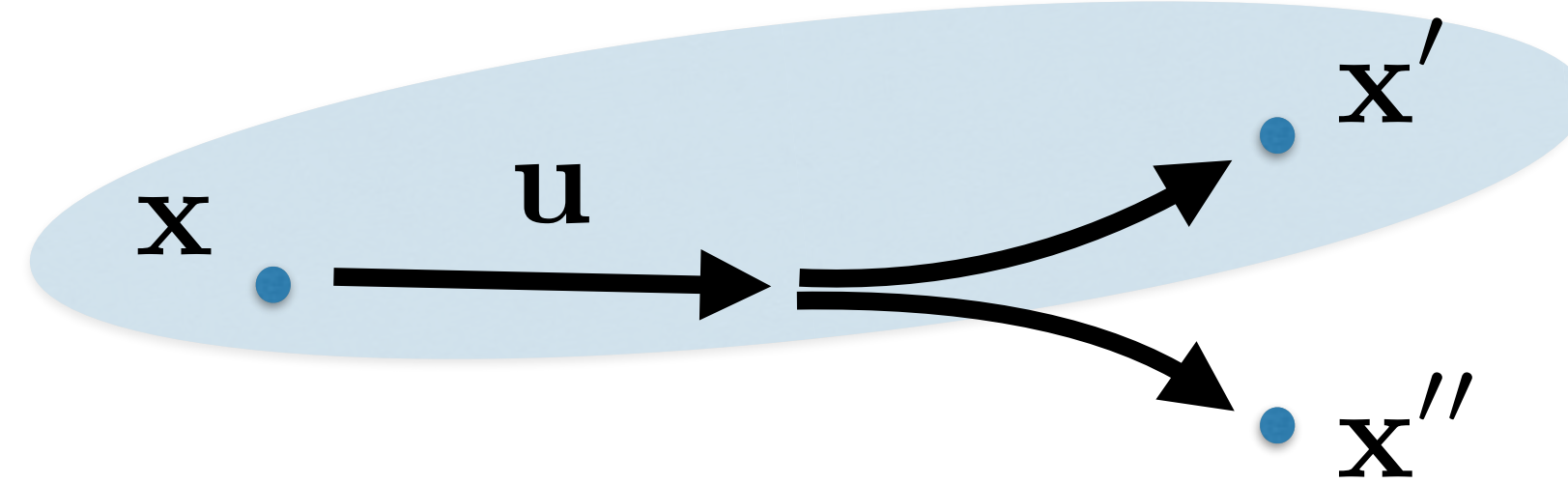
Problems often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$

Actions: $\mathbf{u} \in \mathcal{R}^m$

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

Rewards: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$



Problems often formalised as MDP

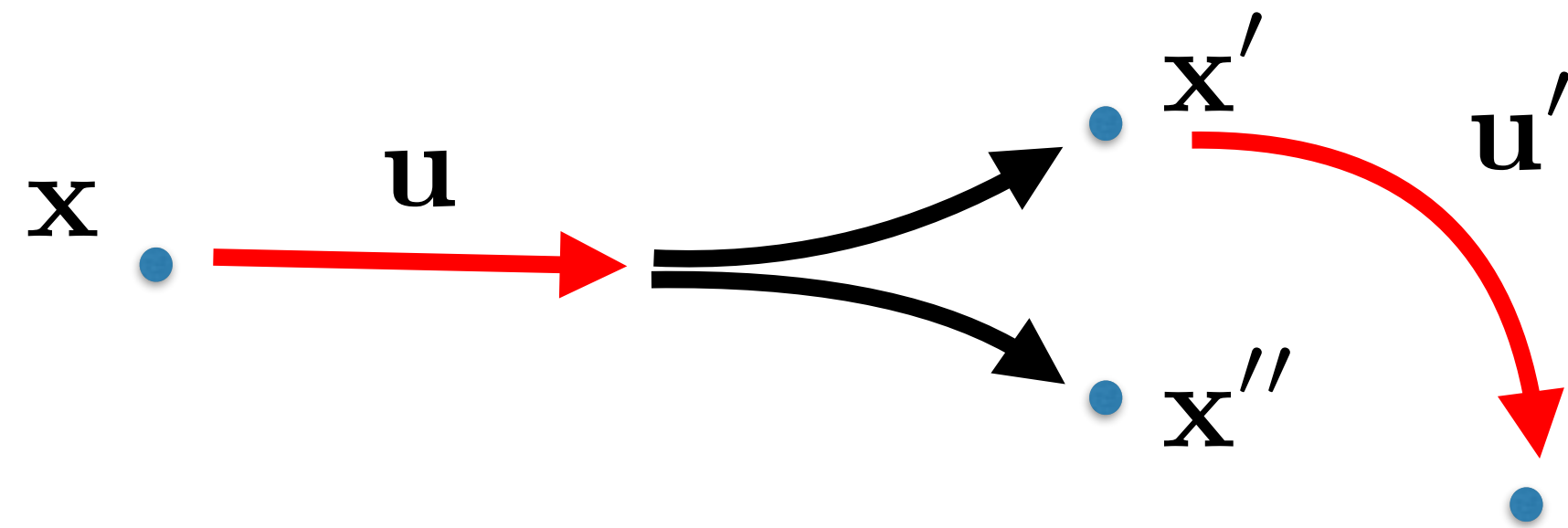
States: $\mathbf{x} \in \mathcal{R}^n$

Actions: $\mathbf{u} \in \mathcal{R}^m$

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

Rewards: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$

Policy: $\pi(\mathbf{u}|\mathbf{x})$



Problems often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$

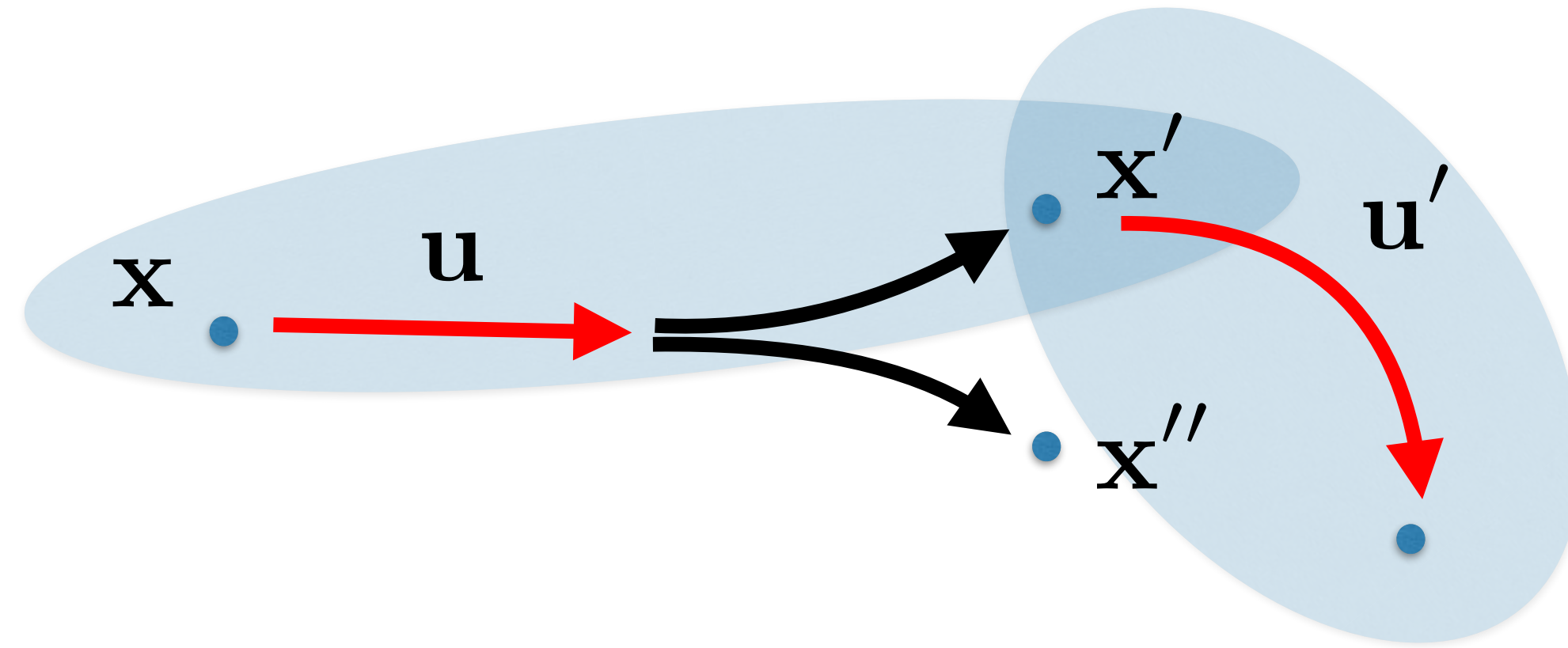
Actions: $\mathbf{u} \in \mathcal{R}^m$

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

Rewards: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$

Policy: $\pi(\mathbf{u}|\mathbf{x})$

Goal: $\pi^* = \arg \max_{\pi} J_{\pi}$ (e.g. $J_{\pi} = \mathbb{E}_{\tau \sim \pi} \{ \sum_{r_t \sim \tau} \gamma^t r_t \}$)



Problems often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$ incomplete, noisy

Actions: $\mathbf{u} \in \mathcal{R}^m$ continuous high-dimensional

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$ inaccurate model

Rewards: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$ hard to engineer

Policy: $\pi(\mathbf{u}|\mathbf{x})$ execution endanger the robot

Goal: $\pi^* = \arg \max_{\pi} J_{\pi}$ (e.g. $J_{\pi} = \mathbb{E}_{\tau \sim \pi} \{ \sum_{r_t \sim \tau} \gamma^t r_t \}$)

Typical problems

τ
→

Model identification:

- given some trajectories
estimate model

$p(\mathbf{x}' | \mathbf{x}, \mathbf{u})$
→

$p(\mathbf{x}' | \mathbf{x}, \mathbf{u})$
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Model predictive control / Planning

- given the model and reward
estimate optimal policy/plan

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

??control vs planning??

τ
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Reinforcement learning:

- given rewards and trajectories,
estimate optimal policy

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

**??model-free vs
model-based??**

τ^*
→

Inverse reinforcement learning:

- given optimal trajectories
estimate reward function

$r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$
→

Typical problems

τ
→

Model identification:

- given some trajectories
estimate model

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Model predictive control / Planning

- given the model and reward
estimate optimal policy/plan

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Reinforcement learning:

- given rewards and trajectories,
estimate optimal policy

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ^*
→

Inverse reinforcement learning:

- given optimal trajectories
estimate reward function

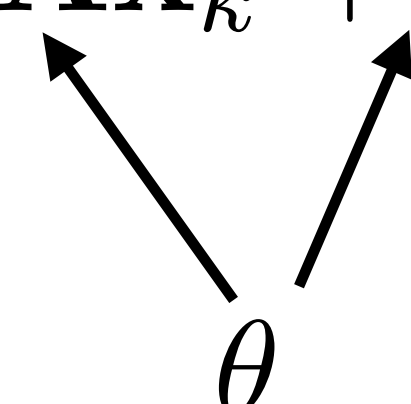
$r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$
→

Model identification:

- **Employ physics engine** and experimentally identify its physical quantities
- **Learn network** (deep convolutional) to predict following state $\mathbf{x}_{k+1} = p_{\theta}(\mathbf{x}_k, \mathbf{u}_k)$.

$$\arg \min_{\theta} \sum_k \|p_{\theta}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1}\|_2^2$$

- For example fully observable, time-discrete, linear system:

$$\mathbf{x}_{k+1} = p_{\theta}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$


A diagram with the symbol θ at the bottom. Two arrows originate from θ and point upwards to the matrices $\mathbf{A}$ and $\mathbf{B}$ in the equation $\mathbf{x}_{k+1} = p_{\theta}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ shown above.

Typical problems

τ
→

Model identification:

- given some trajectories
estimate model

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Model predictive control / Planning

- given the model and reward
estimate optimal policy/plan

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Reinforcement learning:

- given rewards and trajectories,
estimate optimal policy

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ^*
→

Inverse reinforcement learning:

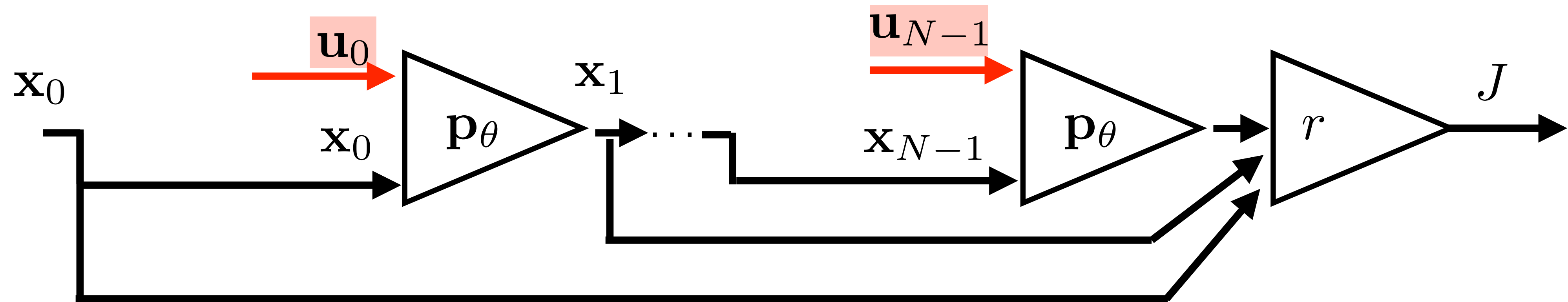
- given optimal trajectories
estimate reward function

$r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$
→

Optimizing actions

Optimize **sequence of actions** (planning) maximizing the rewards on model-based trajectories

$$\arg \max_{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{N-1}} \left\{ \sum_k r(\mathbf{x}_k, \mathbf{u}_k) \mid \mathbf{x}_{k+1} = p_\theta(\mathbf{x}_k, \mathbf{u}_k) \right\}$$

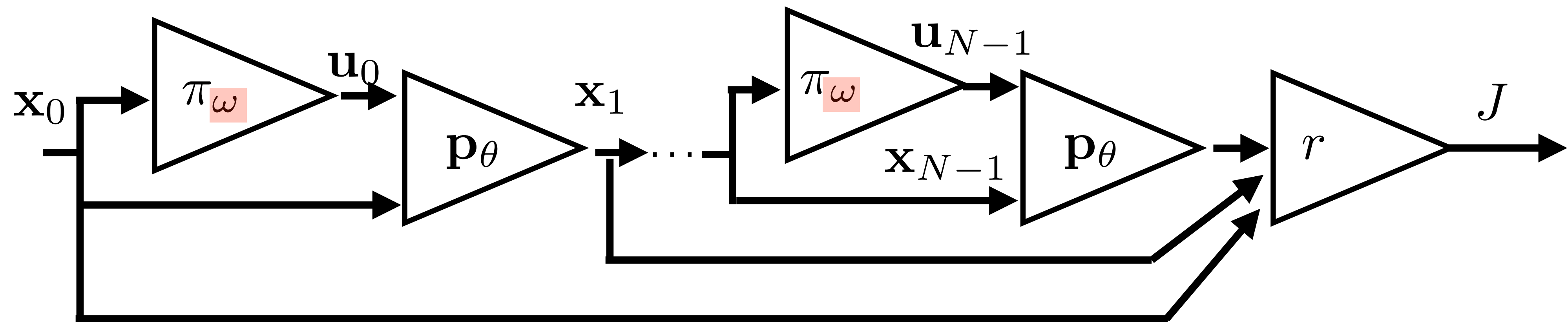


- typically strongly non-convex => gradient optimization inefficient
- exhaustive search (BFS, Dijkstra, A*, RRT), or (iterative) QP formulation (MPC)
- **repeated open loop control**

Learning policy

Optimize **policy parameters** (learn linear function or deep net) maximizing the rewards on model-based trajectories

$$\arg \max_{\omega} \left\{ \sum_k r(\mathbf{x}_k, \mathbf{u}_k) \mid \mathbf{x}_{k+1} = p_{\theta}(\mathbf{x}_k, \mathbf{u}_k), \mathbf{u}_k = \pi_{\omega}(\mathbf{x}_k) \right\}$$



- Especially: linear system and policy + quadratic reward function
- LQR has closed form solution
- **closed loop control**

Typical problems

τ
→

Model identification:

- given some trajectories
estimate model

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Model predictive control / Planning

- given the model and reward
estimate optimal policy/plan

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Reinforcement learning:

- given rewards and trajectories,
estimate optimal policy

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ^*
→

Inverse reinforcement learning:

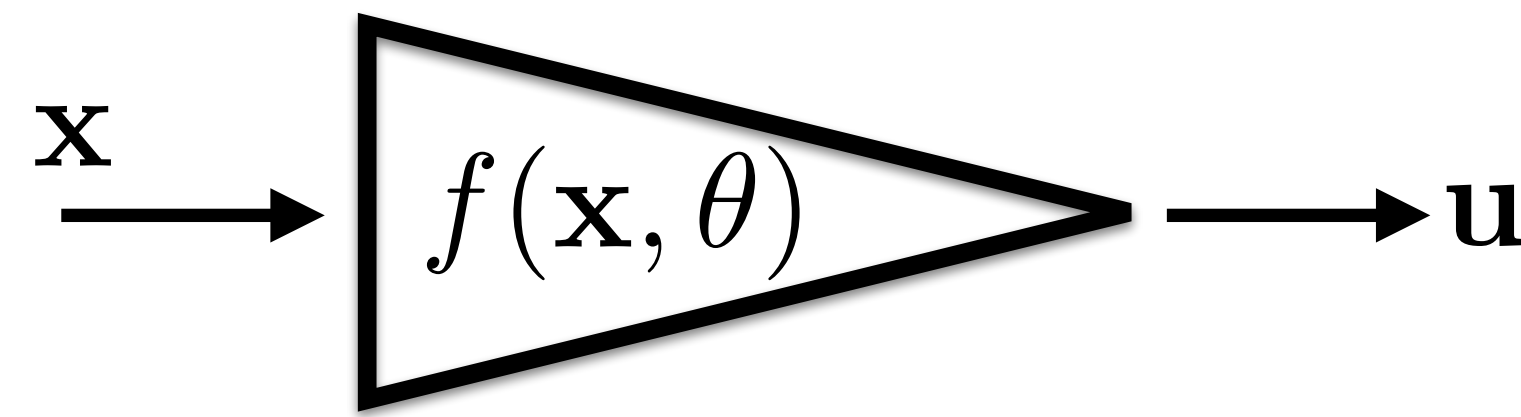
- given optimal trajectories
estimate reward function

$r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$
→

Deterministic vs stochastic policy

Deterministic policy for continuous control:

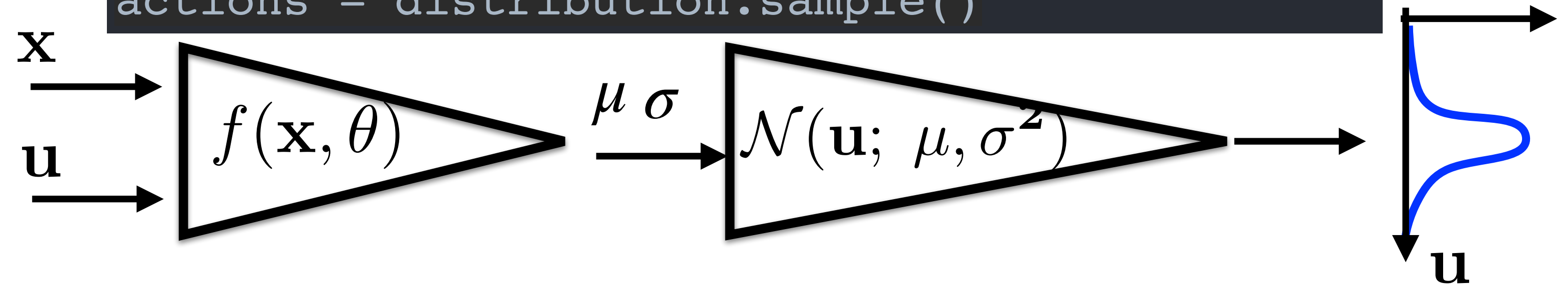
$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$



```
from torch.distributions.normal import Normal
distribution = Normal(mu, sigma)
actions = distribution.sample()
```

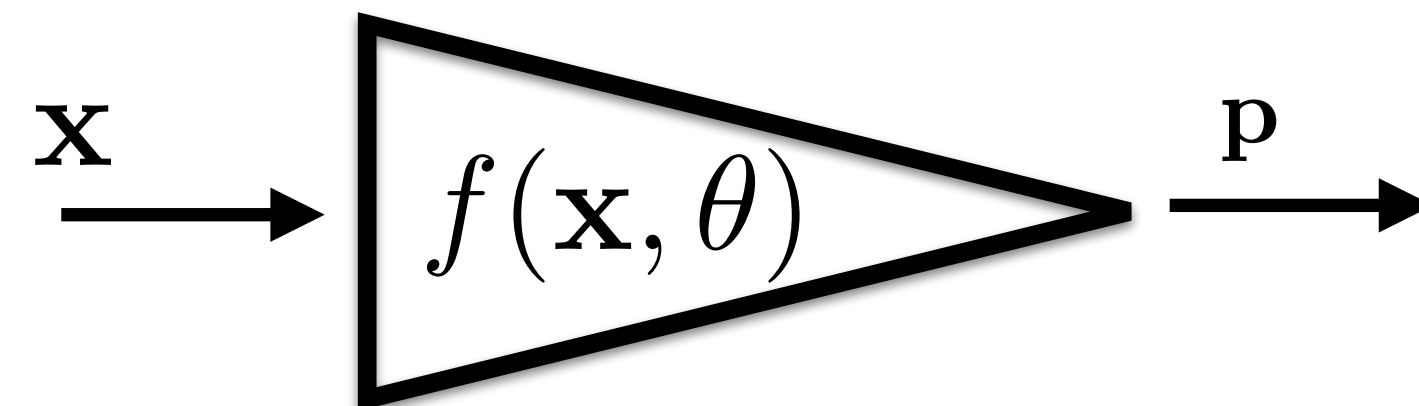
Stochastic policy for continuous control:

$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) \sim \mathcal{N}(\mathbf{u}; \mu, \sigma^2)$$



Stochastic policy for discrete control:

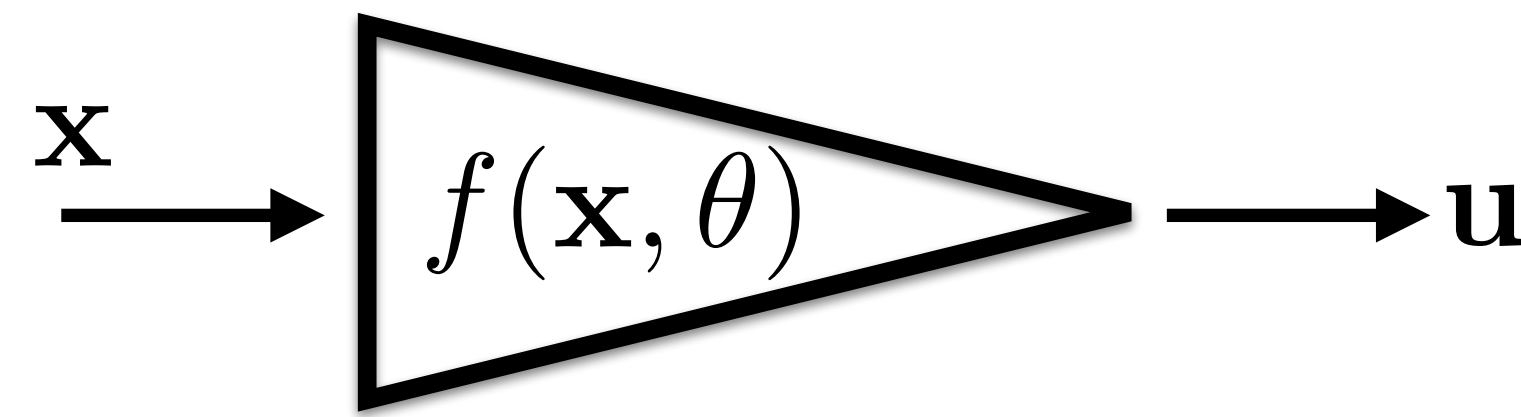
$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) \sim p(\mathbf{u}; \mathbf{p})$$



Deterministic vs stochastic policy

Deterministic policy for continuous control:

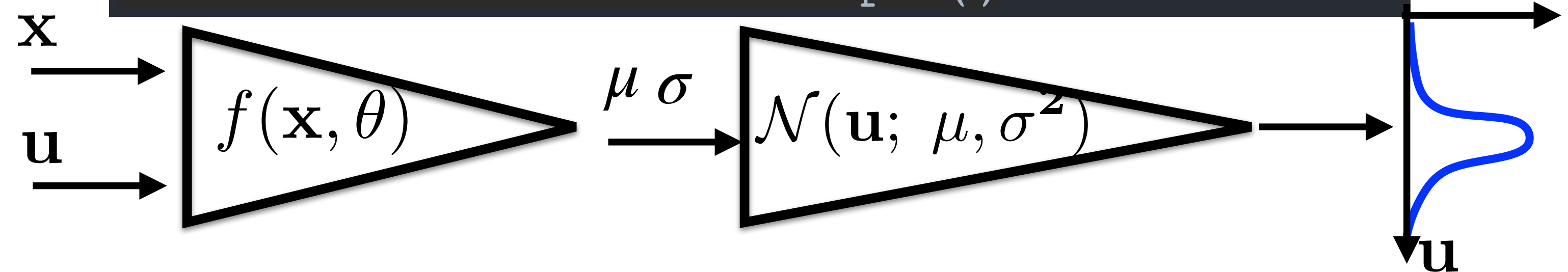
$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$



```
from torch.distributions.normal import Normal
distribution = Normal(mu, sigma)
actions = distribution.sample()
```

Stochastic policy for continuous control:

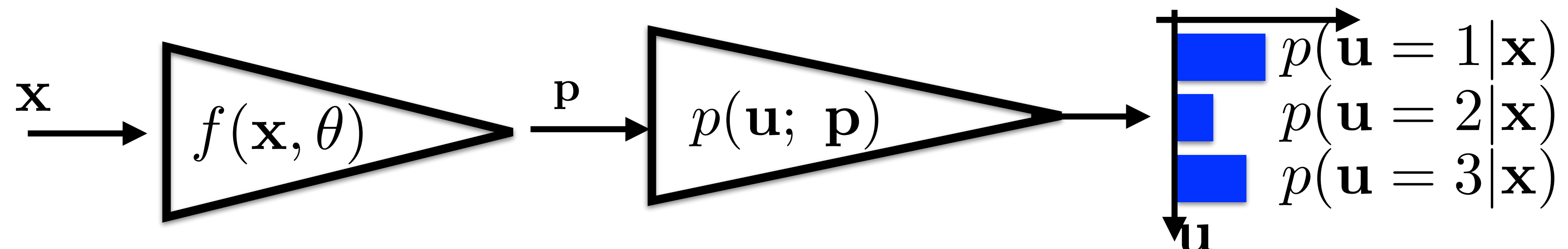
$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) \sim \mathcal{N}(\mathbf{u}; \mu, \sigma^2)$$

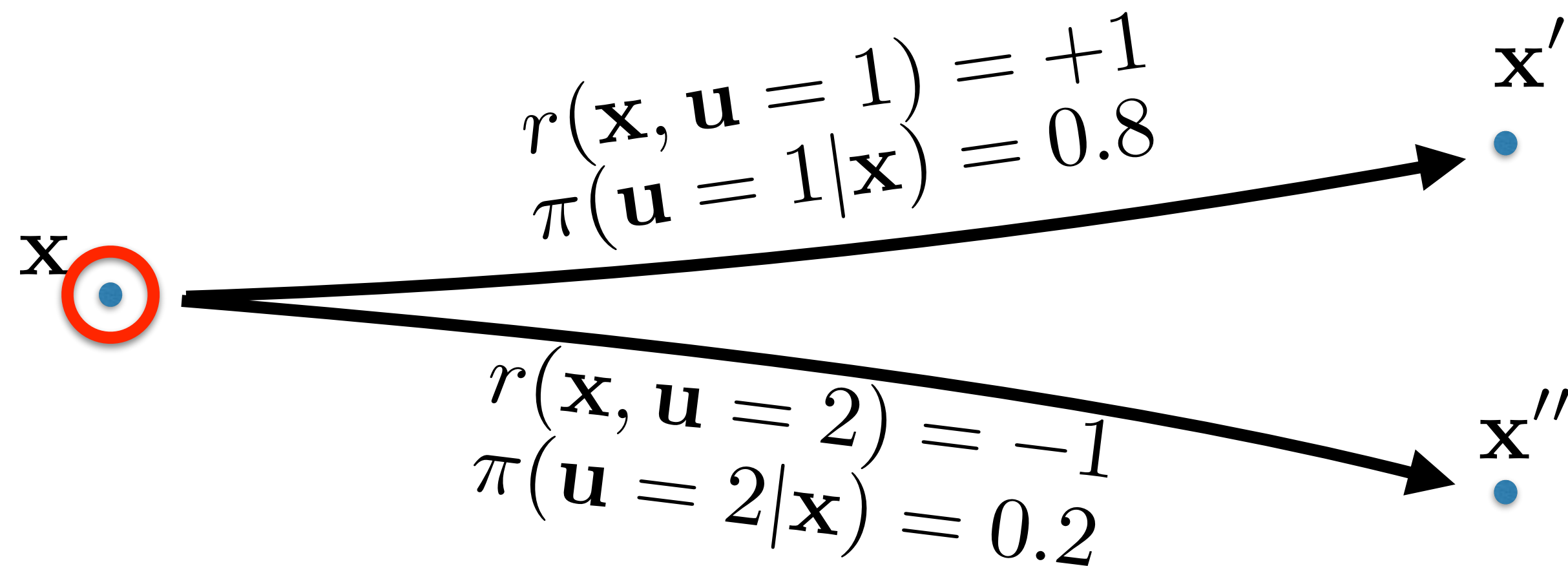


```
from torch.distributions.normal import Categorical
distribution = Categorical(mu, sigma)
actions = distribution.sample()
```

Stochastic policy for discrete control:

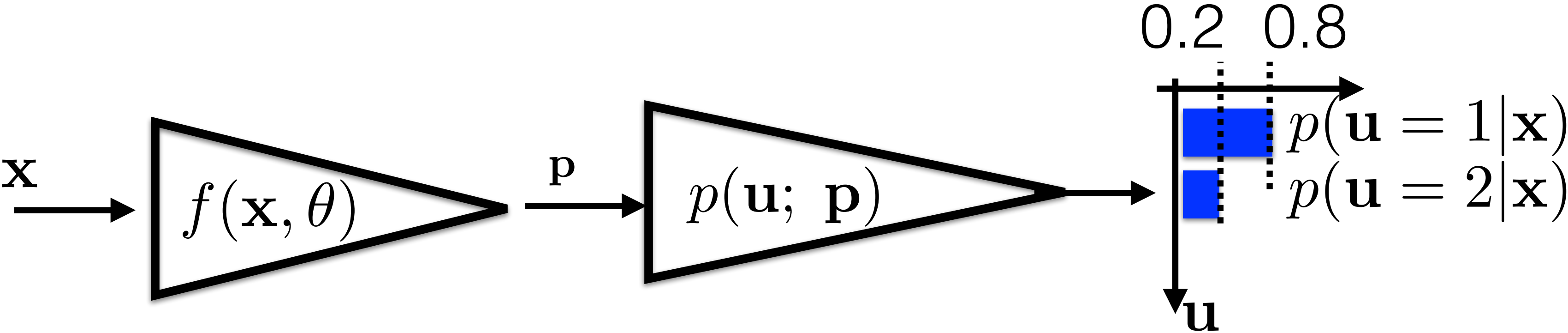
$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) \sim p(\mathbf{u}; \mathbf{p})$$

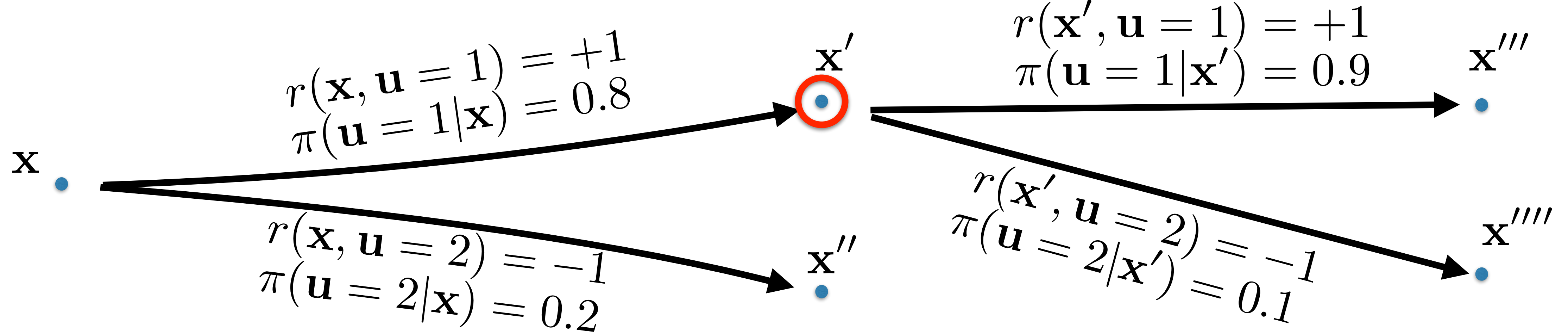




Stochastic policy for discrete control:

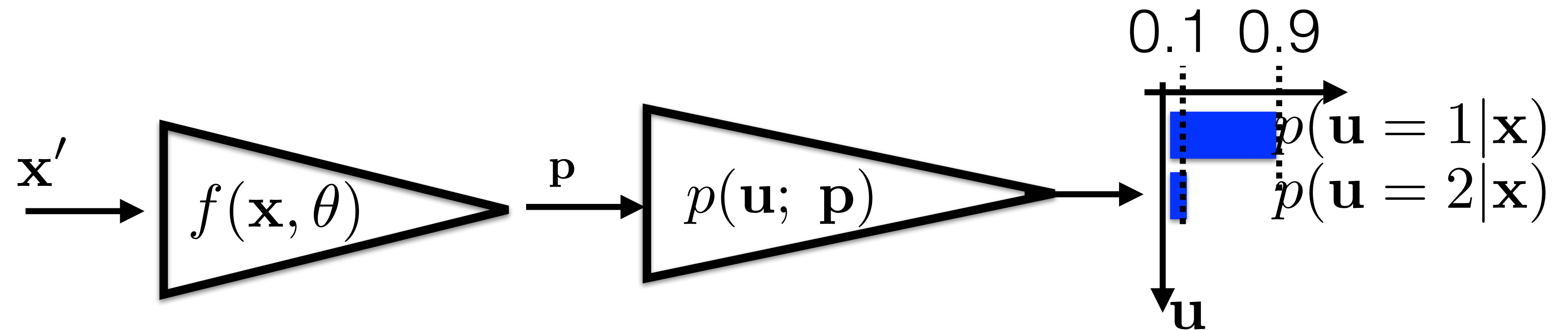
$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) \sim p(\mathbf{u}; \mathbf{p})$$

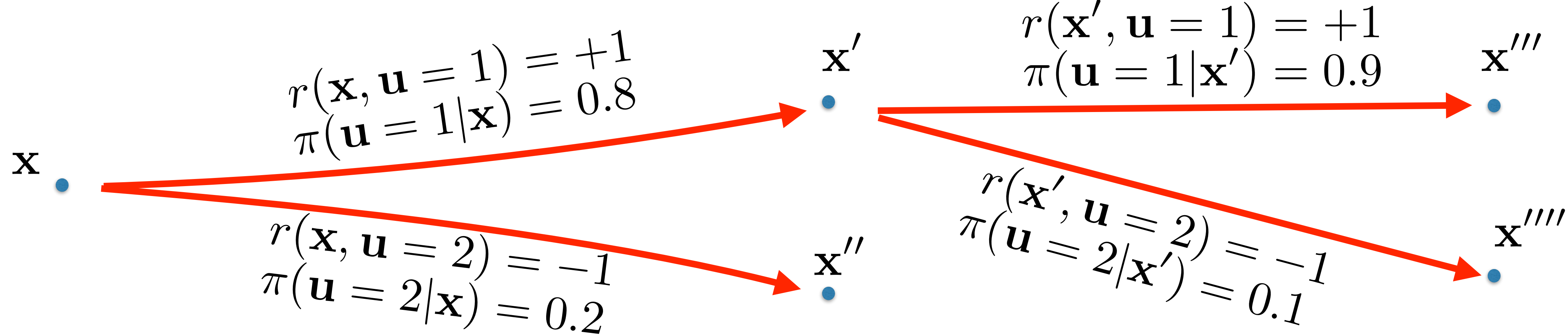




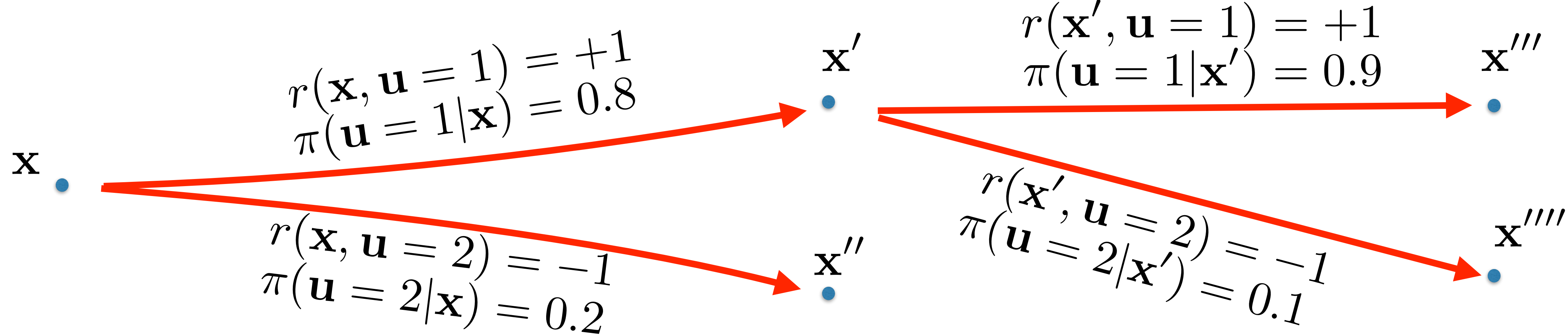
Stochastic policy for discrete control:

$$\pi_{\theta}(\mathbf{u}|\mathbf{x}) \sim p(\mathbf{u}; \mathbf{p})$$

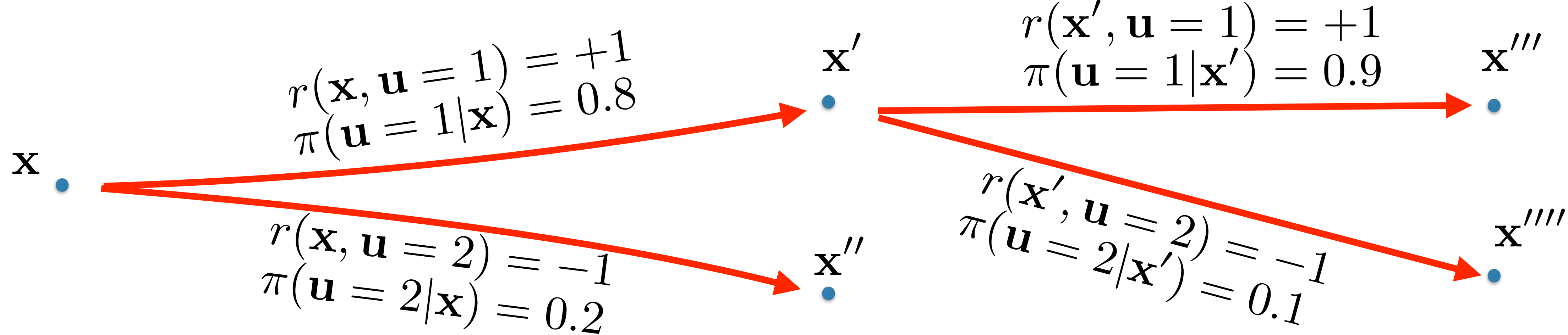




$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]$$

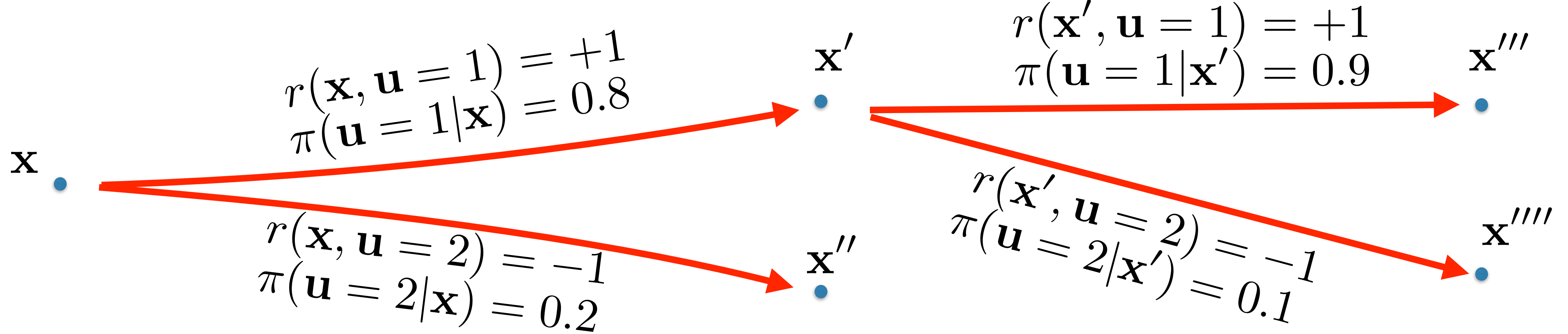


$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)]$$



$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau | \pi) r(\tau)$$

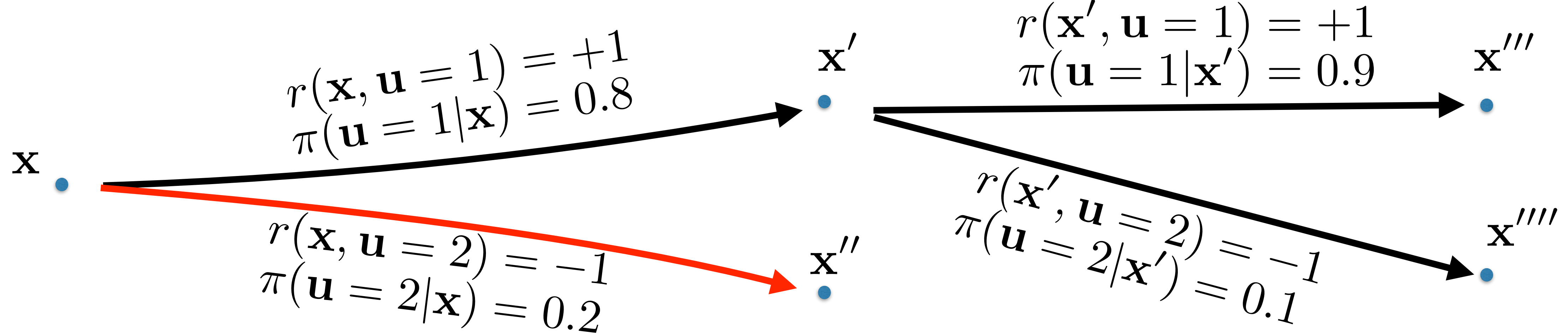
$$V^\pi(\mathbf{x}) = \text{???}$$



$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau|\pi) r(\tau)$$

$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$

$$Q^\pi(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} [r(\tau)] = \int_{\substack{\tau: \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} p(\tau|\pi) r(\tau)$$

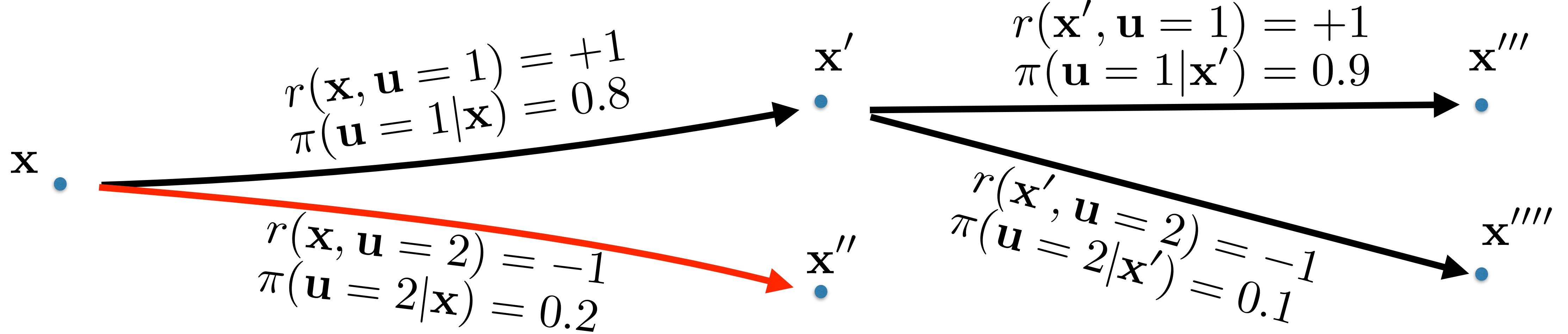


$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau | \pi) r(\tau)$$

$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$

$$Q^\pi(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} [r(\tau)] = \int_{\substack{\tau: \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} p(\tau | \pi) r(\tau)$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 2) = \text{???}$$

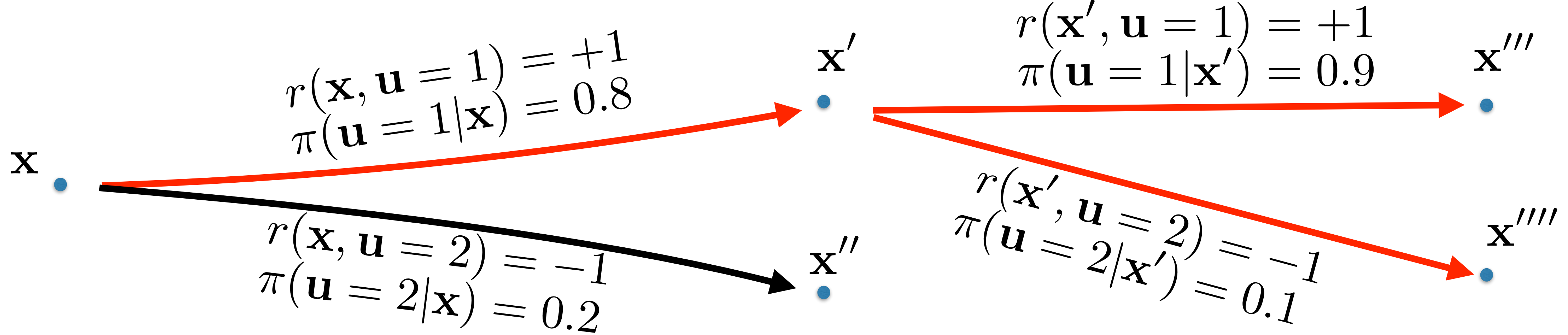


$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau|\pi) r(\tau)$$

$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$

$$Q^\pi(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} [r(\tau)] = \int_{\substack{\tau: \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} p(\tau|\pi) r(\tau)$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 2) = -1$$



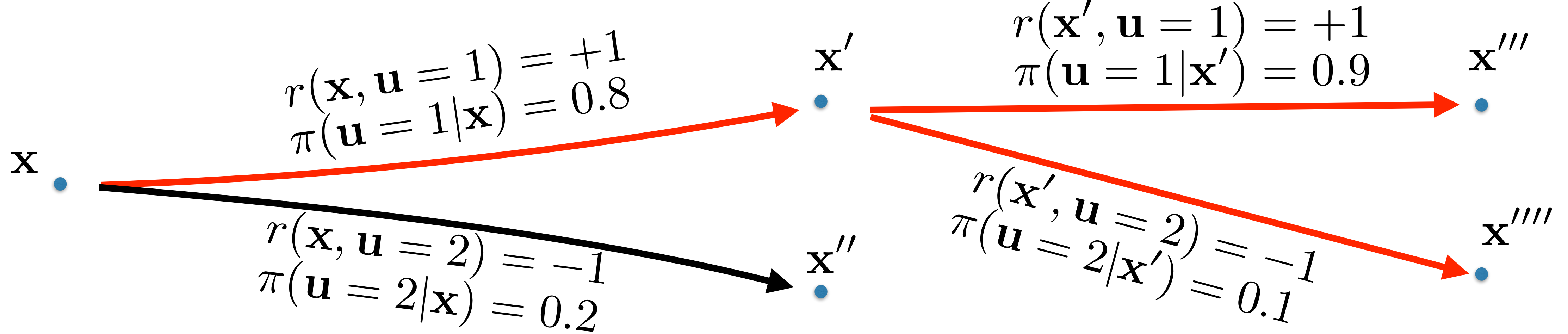
$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau | \pi) r(\tau)$$

$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$

$$Q^\pi(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} [r(\tau)] = \int_{\substack{\tau: \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} p(\tau | \pi) r(\tau)$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 2) = -1$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 1) = 1 + 0.9 * 1 + 0.1 * (-1) = \text{???}$$



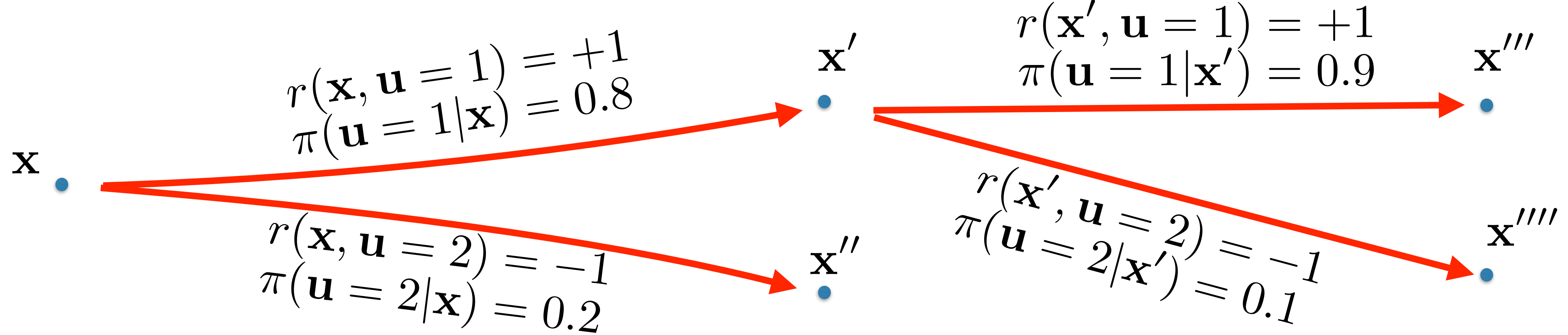
$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau|\pi) r(\tau)$$

$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$

$$Q^\pi(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} [r(\tau)] = \int_{\substack{\tau: \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} p(\tau|\pi) r(\tau)$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 2) = -1$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 1) = 1 + 0.9 * 1 + 0.1 * (-1) = 1.8$$



$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau | \pi) r(\tau)$$

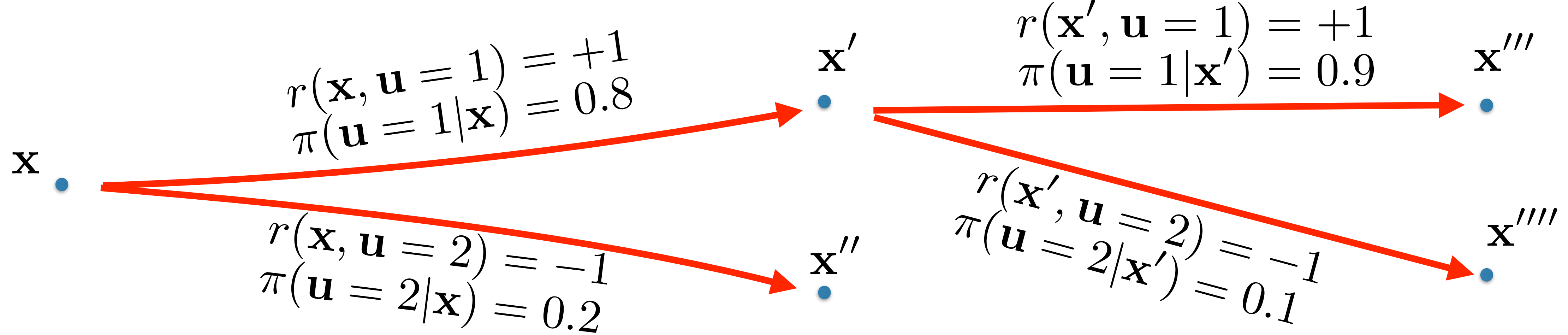
$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$

$$Q^\pi(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} [r(\tau)] = \int_{\substack{\tau: \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} p(\tau | \pi) r(\tau)$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 2) = -1$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 1) = 1 + 0.9 * 1 + 0.1 * (-1) = 1.8$$

$$A^\pi(\mathbf{x}, \mathbf{u} = 1) = Q^\pi(\mathbf{x}, \mathbf{u} = 1) - V^\pi(\mathbf{x}) = \text{???}$$



$$V^\pi(\mathbf{x}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x}}} [r(\tau)] = \int_{\tau: \mathbf{x}_0 = \mathbf{x}} p(\tau|\pi) r(\tau)$$

$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$

$$Q^\pi(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right] = \mathbb{E}_{\substack{\tau \sim \pi \\ \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} [r(\tau)] = \int_{\substack{\tau: \mathbf{x}_0 = \mathbf{x} \\ \mathbf{u}_0 = \mathbf{u}}} p(\tau|\pi) r(\tau)$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 2) = -1$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 1) = 1 + 0.9 * 1 + 0.1 * (-1) = 1.8$$

**Can you
interpret it?**

$$A^\pi(\mathbf{x}, \mathbf{u} = 1) = Q^\pi(\mathbf{x}, \mathbf{u} = 1) - V^\pi(\mathbf{x}) = 1.8 - 1.24 = 0.56$$

$$A^\pi(\mathbf{x}, \mathbf{u} = 2) = Q^\pi(\mathbf{x}, \mathbf{u} = 2) - V^\pi(\mathbf{x}) = -1 - 1.24 = -2.24$$

Attention in RL

Value function $V(x)$



Advantage function $A(x,u)$



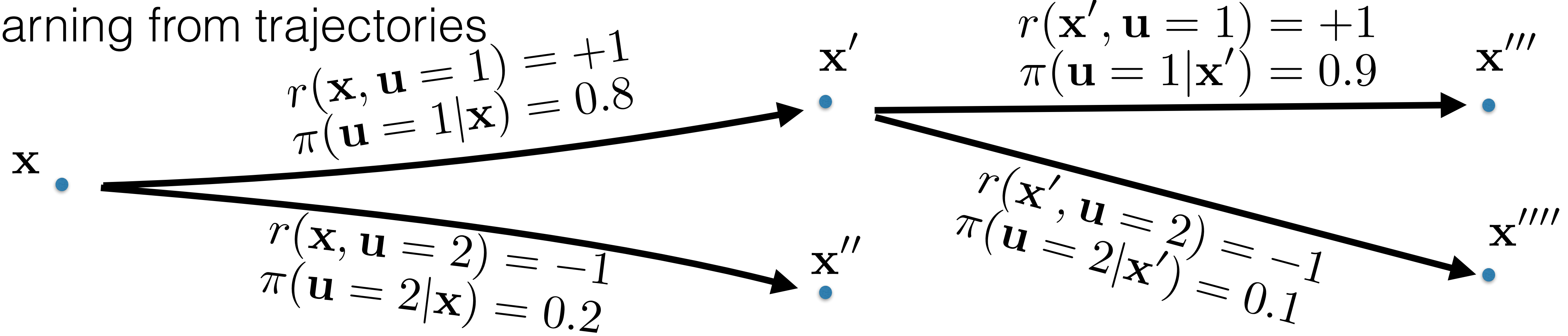
00034
1 £198

ACTIVISION

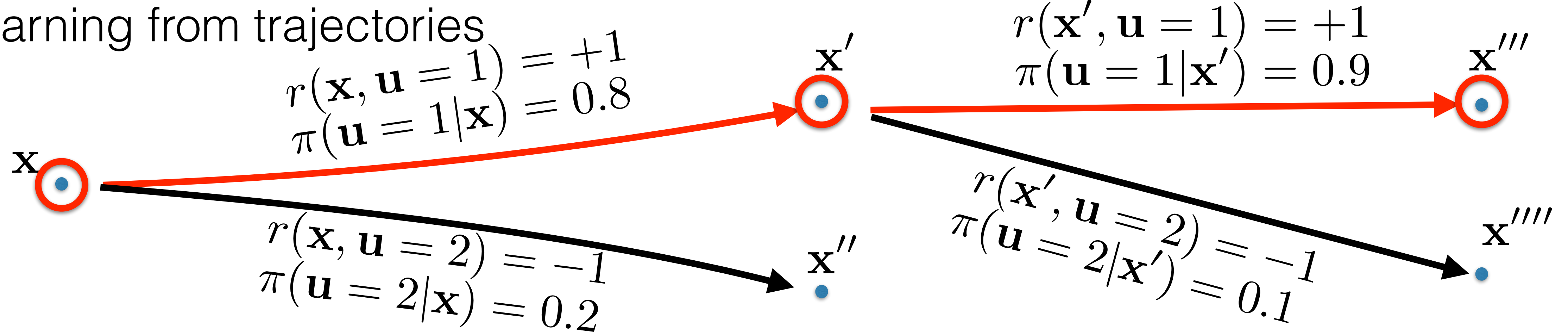
00034
1 £198

ACTIVISION

Learning from trajectories



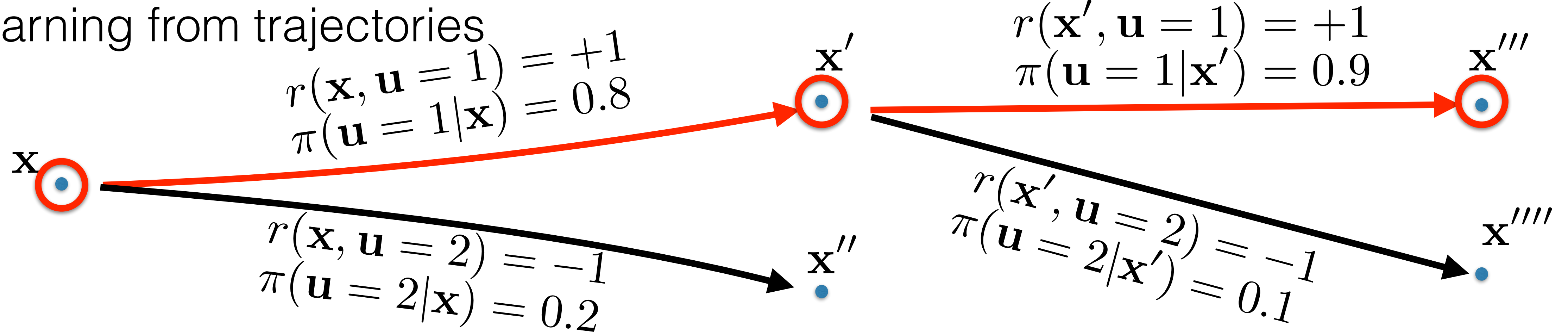
Learning from trajectories



$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_1 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}''$,

$Q^\pi(\mathbf{x}, \mathbf{u} = 1) \approx ???$

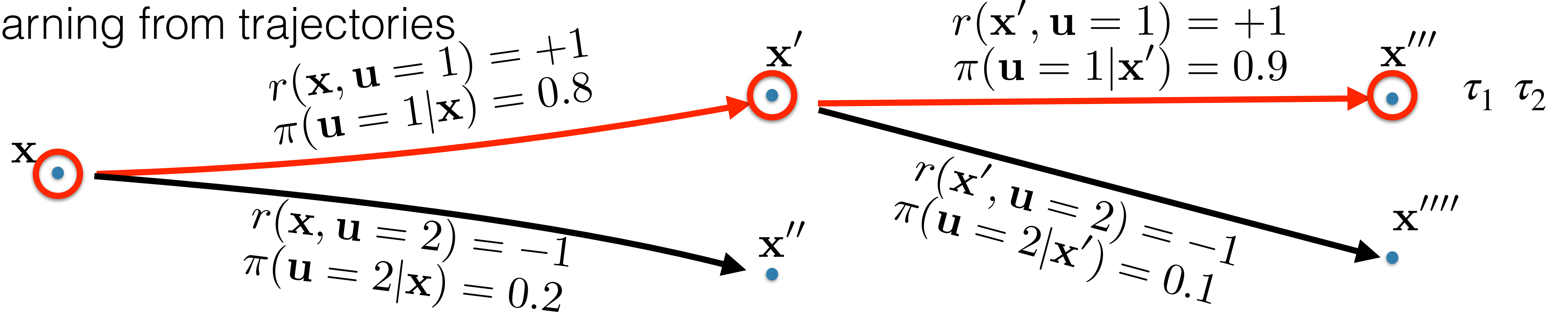
Learning from trajectories



$$\tau_1 : \quad \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_1 = 1, r_1 = +1, \quad \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_2 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}''',$$

$$Q^\pi(\mathbf{x}, \mathbf{u}=1) \approx r_1 + \gamma r_2$$

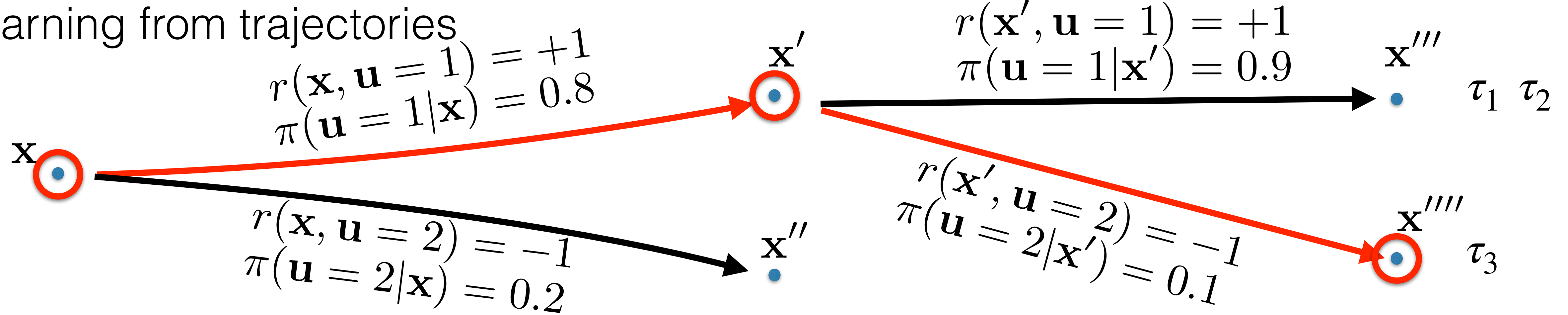
Learning from trajectories



$$\tau_1 : \quad \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_1 = 1, r_1 = +1, \quad \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_2 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}''$$

$$\tau_2 : \quad \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_1 = 1, r_1 = +1, \quad \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_2 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}'''$$

Learning from trajectories



$$\tau_1 : \quad \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_1 = 1, r_1 = +1, \quad \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_2 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}''',$$

$$\tau_2 : \quad \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_1 = 1, r_1 = +1, \quad \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_2 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}''',$$

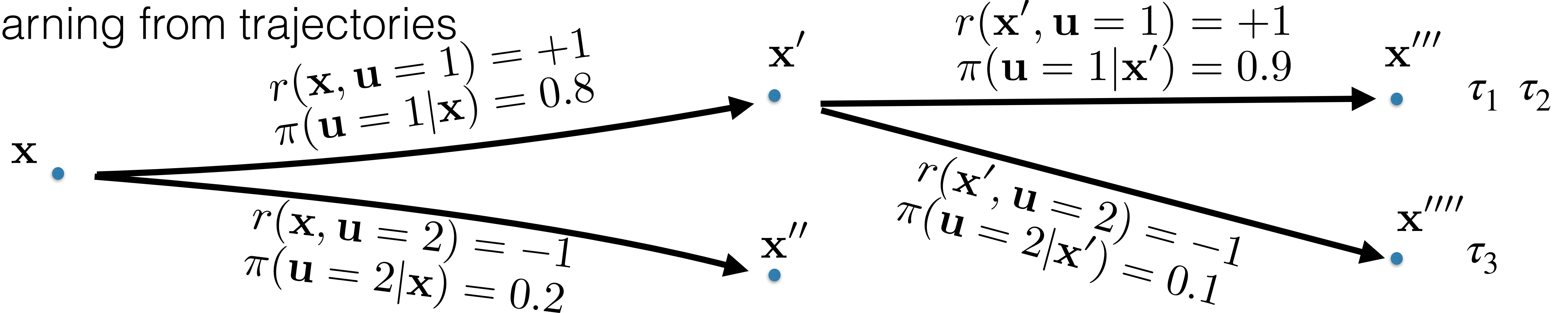
$$\tau_3 : \quad \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_1 = 1, r_1 = +1, \quad \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_2 = 2, r_2 = -1, \mathbf{x}_2 = \mathbf{x}''',$$

$$Q^\pi(\mathbf{x}, \mathbf{u} = 1) \approx \frac{1}{N} \sum_{\tau} r_1 + \gamma r_2 = \frac{1}{3} (1+1 + 1+1 + 1+(-1)) = 1.33 \quad \text{vs} \quad Q^\pi(\mathbf{x}, \mathbf{u} = 1) = 1.8$$

If you are given true value function $V^\pi(\mathbf{x}') = 0.8$ then

$$Q^\pi(\mathbf{x}, \mathbf{u} = 1) \approx \frac{1}{N} \sum_{\tau} r_1 + \gamma V^\pi(\mathbf{x}') = \frac{1}{3} (1+0.8 + 1+0.8 + 1+0.8) = 1.8$$

Learning from trajectories



There are many possible approximations of $Q^\pi(\mathbf{x}, \mathbf{u})$ when $V^\pi(\mathbf{x})$ is available:

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \dots$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V^\pi(\mathbf{x}_4)$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 V^\pi(\mathbf{x}_3)$$

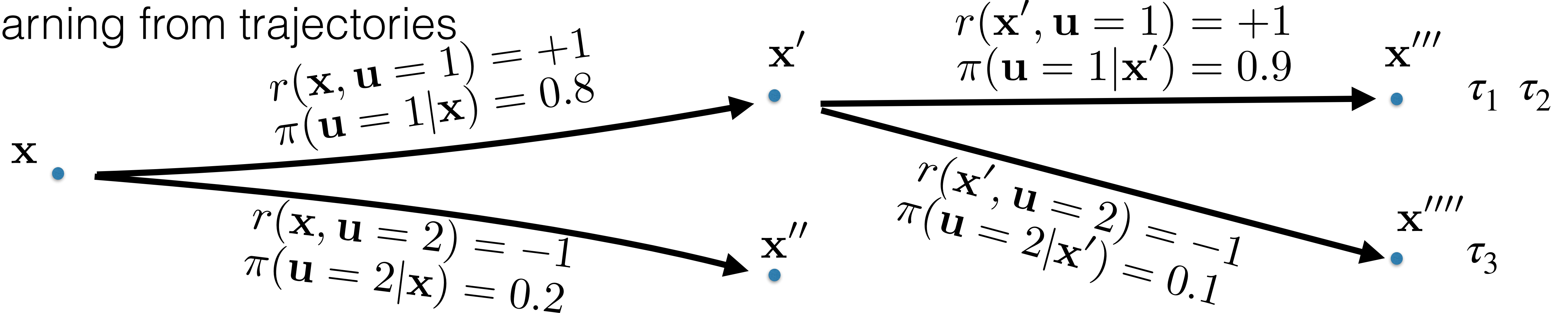
$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma V^\pi(\mathbf{x}_2)$$

Which I should use?

TD(0) estimate (if reliable $V^\pi(\mathbf{x})$ available):

- small variance,
- strong bias

Learning from trajectories



There are many possible approximations of $Q^\pi(\mathbf{x}, \mathbf{u})$ when $V^\pi(\mathbf{x})$ is available:

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \dots$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V^\pi(\mathbf{x}_4)$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 V^\pi(\mathbf{x}_3)$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma V^\pi(\mathbf{x}_2)$$

Which I should use?

TD(1) estimate (if no reliable $V^\pi(\mathbf{x})$ available):

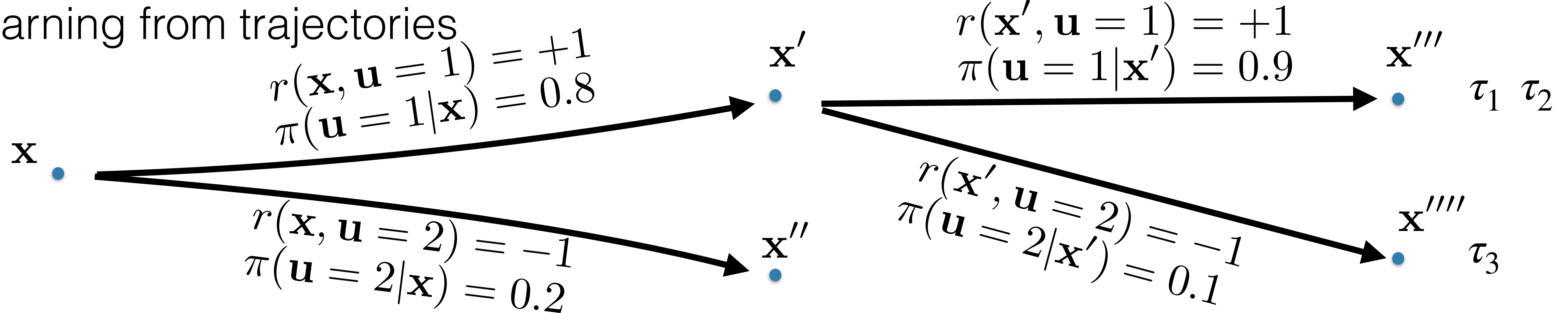
- high variance,
- no bias



TD(0) estimate (if reliable $V^\pi(\mathbf{x})$ available):

- small variance,
- strong bias

Learning from trajectories



There are many possible approximations of $Q^\pi(\mathbf{x}, \mathbf{u})$ when $V^\pi(\mathbf{x})$ is available:

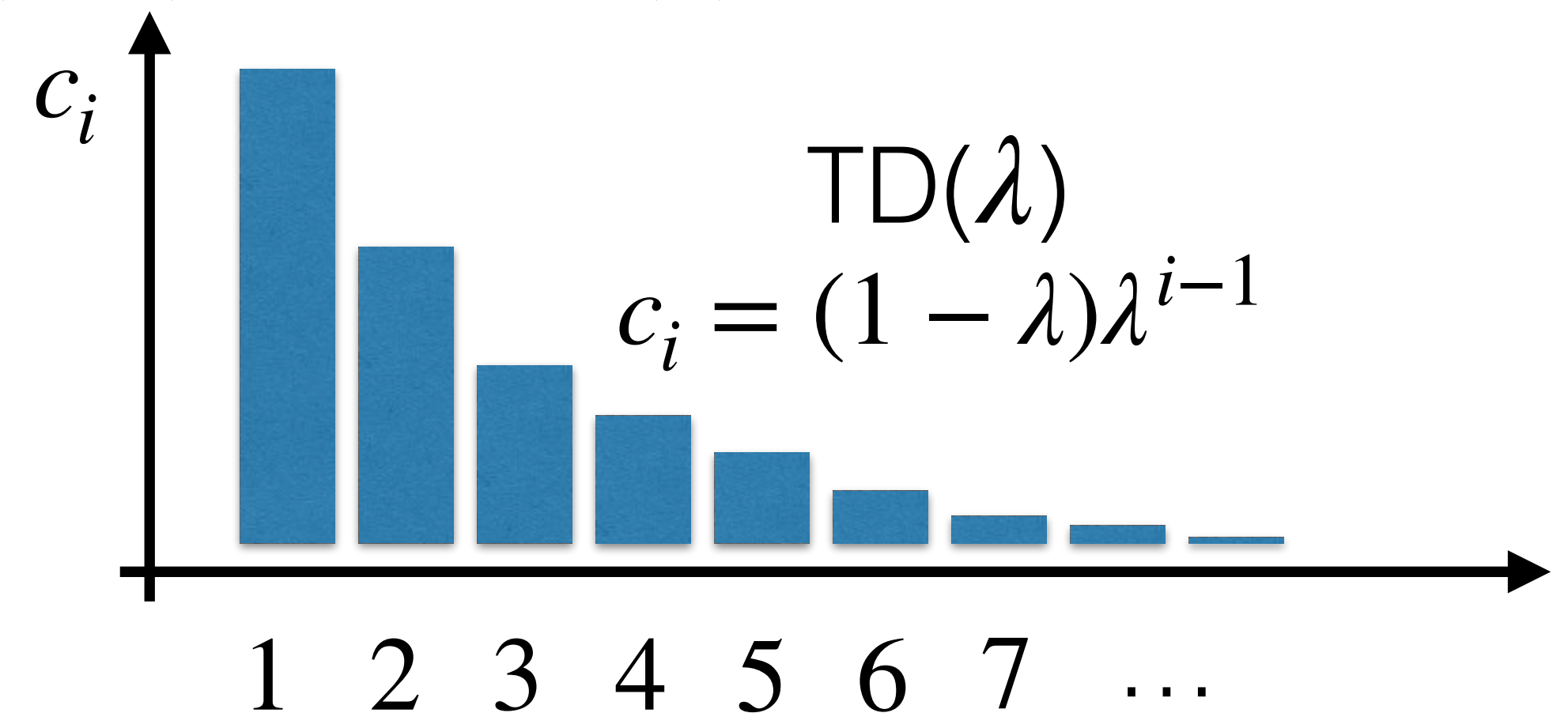
$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \dots = G^{(4)}$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V^\pi(\mathbf{x}_4) = G^{(3)}$$

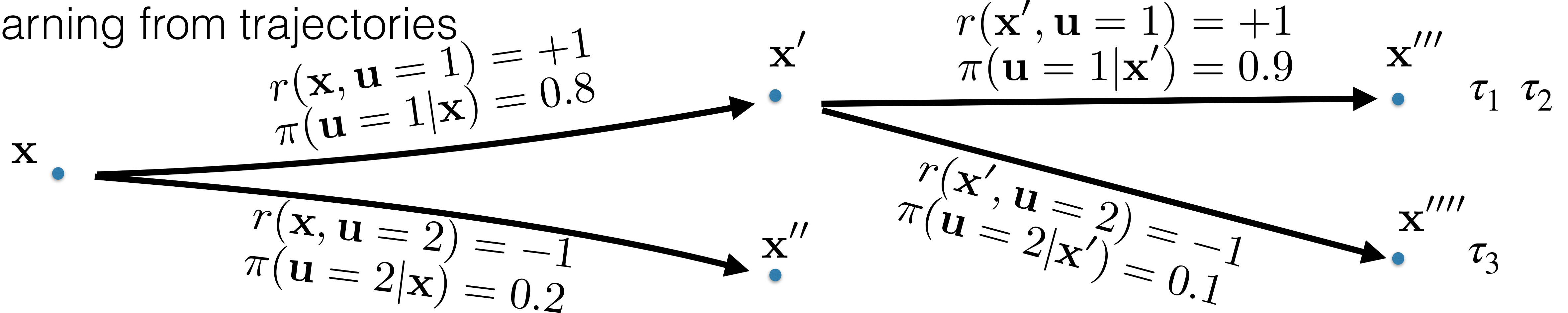
$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 V^\pi(\mathbf{x}_3) = G^{(2)}$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma V^\pi(\mathbf{x}_2) = G^{(1)}$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx c_1 G^{(1)} + c_2 G^{(2)} + c_3 G^{(3)} + c_4 G^{(4)} + \dots$$

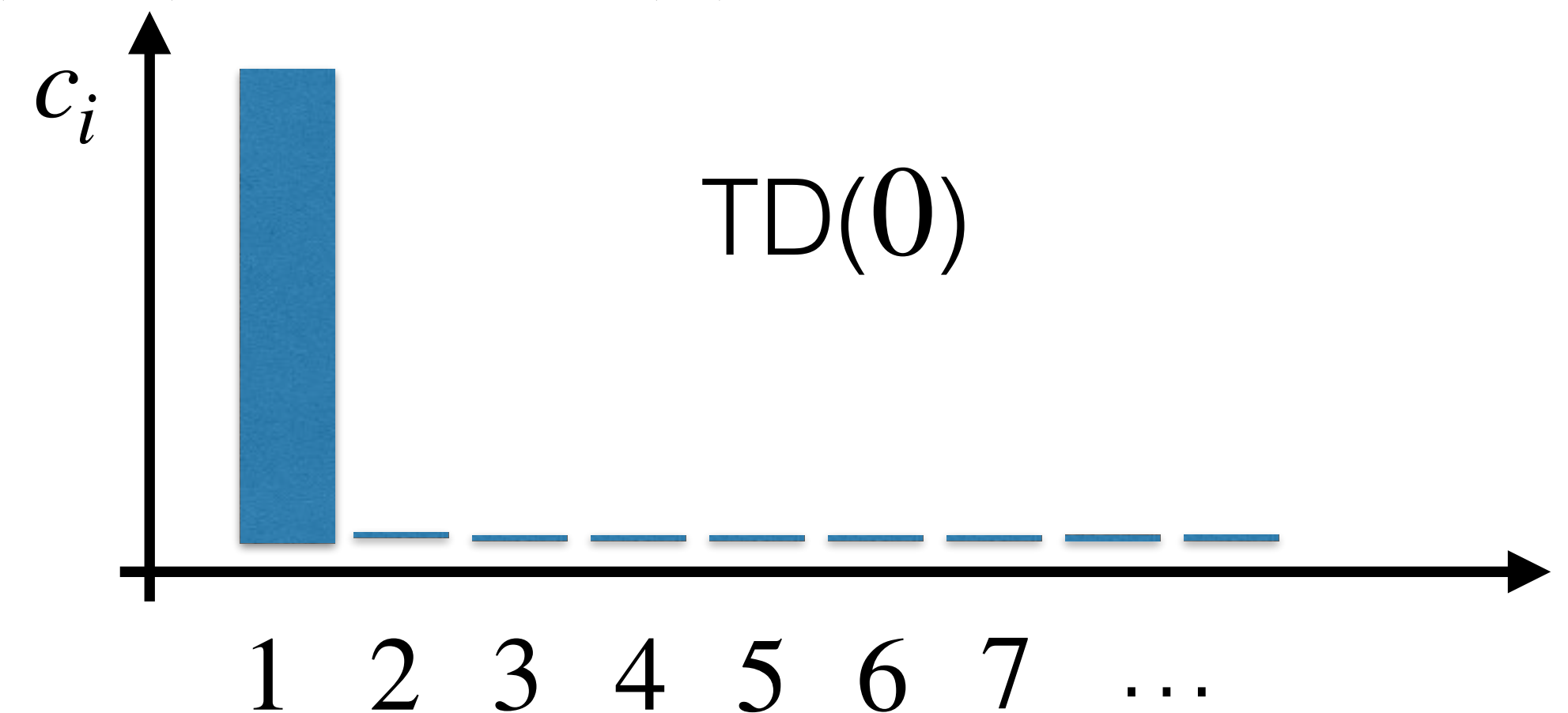


Learning from trajectories

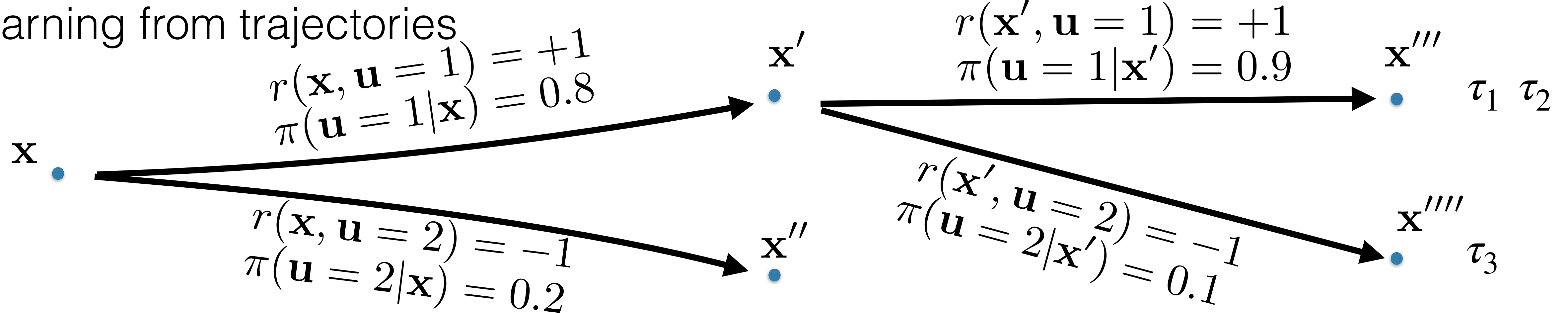


There are many possible approximations of $Q^\pi(\mathbf{x}, \mathbf{u})$ when $V^\pi(\mathbf{x})$ is available:

$$\begin{aligned}
 Q^\pi(\mathbf{x}_1, \mathbf{u}_1) &\approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \dots &= G^{(4)} \\
 Q^\pi(\mathbf{x}_1, \mathbf{u}_1) &\approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V^\pi(\mathbf{x}_4) &= G^{(3)} \\
 Q^\pi(\mathbf{x}_1, \mathbf{u}_1) &\approx r_1 + \gamma r_2 + \gamma^2 V^\pi(\mathbf{x}_3) &= G^{(2)} \\
 Q^\pi(\mathbf{x}_1, \mathbf{u}_1) &\approx r_1 + \gamma V^\pi(\mathbf{x}_2) &= G^{(1)} \\
 Q^\pi(\mathbf{x}_1, \mathbf{u}_1) &\approx c_1 G^{(1)} + c_2 G^{(2)} + c_3 G^{(3)} + c_4 G^{(4)} + \dots
 \end{aligned}$$



Learning from trajectories



There are many possible approximations of $Q^\pi(\mathbf{x}, \mathbf{u})$ when $V^\pi(\mathbf{x})$ is available:

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \dots = G^{(4)}$$

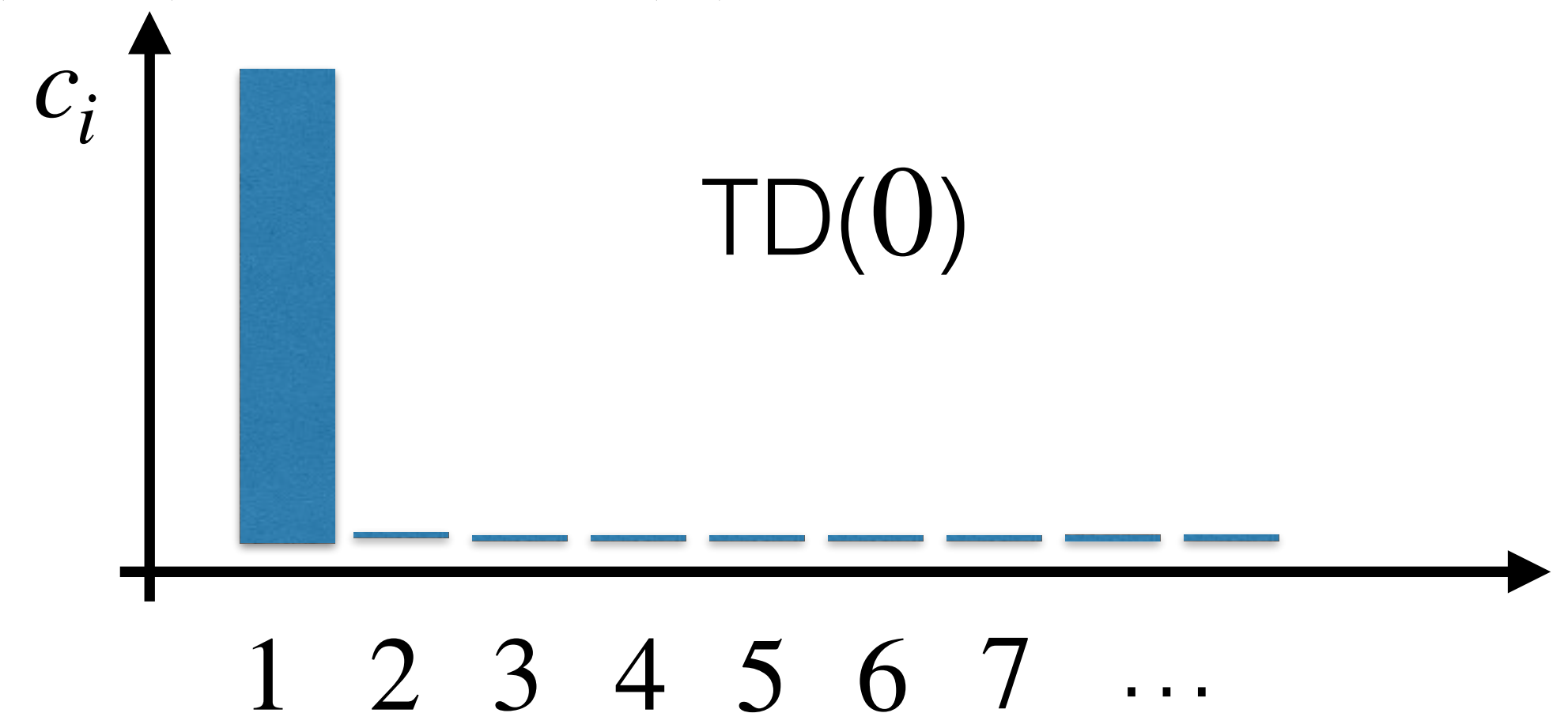
$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V^\pi(\mathbf{x}_4) = G^{(3)}$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma r_2 + \gamma^2 V^\pi(\mathbf{x}_3) = G^{(2)}$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx r_1 + \gamma V^\pi(\mathbf{x}_2) = G^{(1)}$$

$$Q^\pi(\mathbf{x}_1, \mathbf{u}_1) \approx c_1 G^{(1)} + c_2 G^{(2)} + c_3 G^{(3)} + c_4 G^{(4)} + \dots$$

$$A^\pi(\mathbf{x}_1, \mathbf{u}_1) = Q^\pi(\mathbf{x}_1, \mathbf{u}_1) - V^\pi(\mathbf{x}_1) \approx G^{(1)} - V^\pi(\mathbf{x}_1) = r_1 + \gamma V^\pi(\mathbf{x}_2) - V^\pi(\mathbf{x}_1)$$



This determines $Q^\pi(\mathbf{x}, \mathbf{u})$, $V^\pi(\mathbf{x})$, $A^\pi(\mathbf{x}, \mathbf{u})$ for a given policy

How do I find $\pi^* = \arg \max_{\pi} J_{\pi}$?

How do I find $\pi^* = \arg \max_{\pi} J_{\pi}$?

There are two ways:

Primal task:

- Gradient optimization in parameter space of the criterion

$$\theta^* = \arg \max_{\theta} J_{\pi_{\theta}}$$

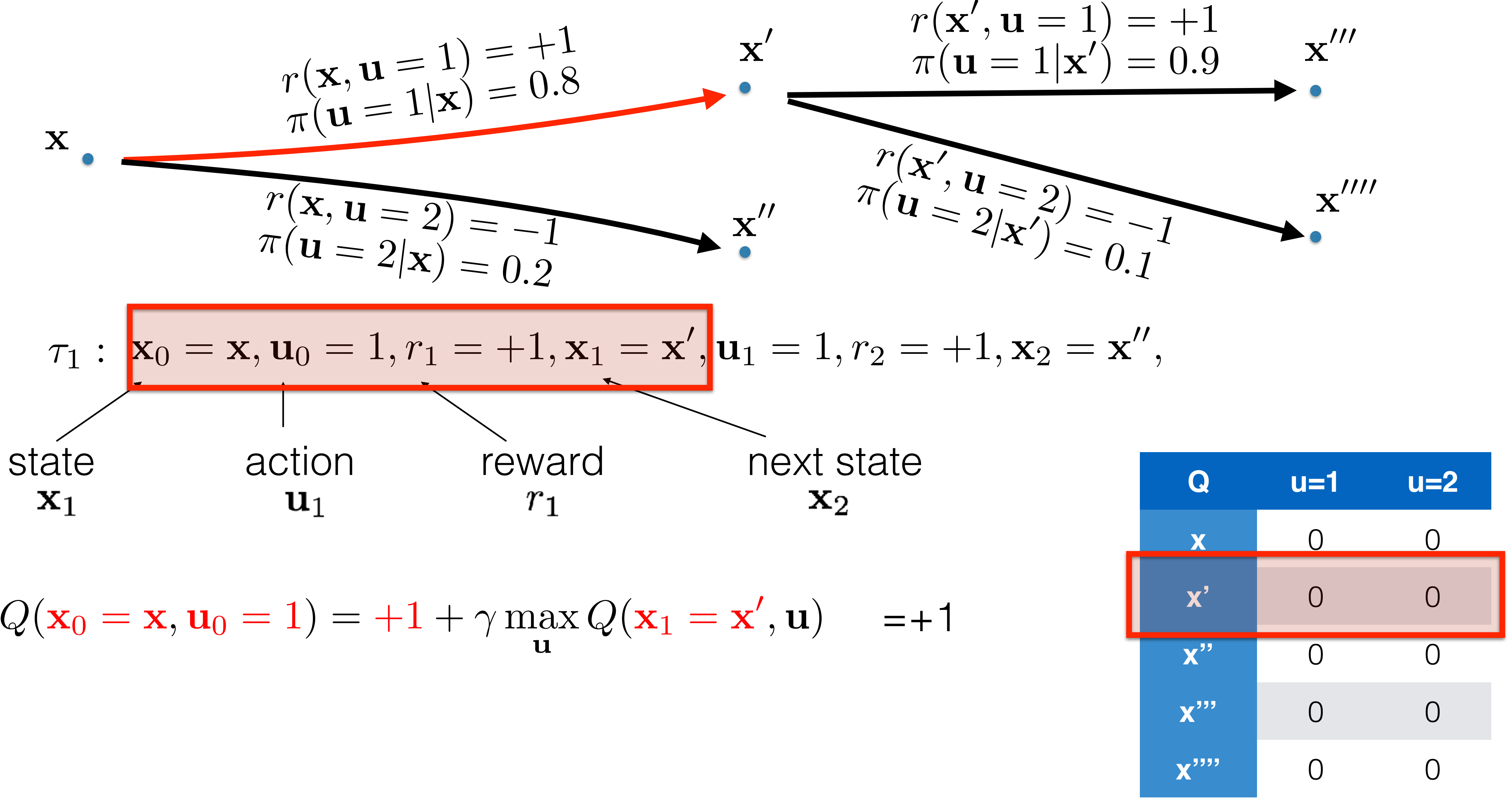
Dual task:

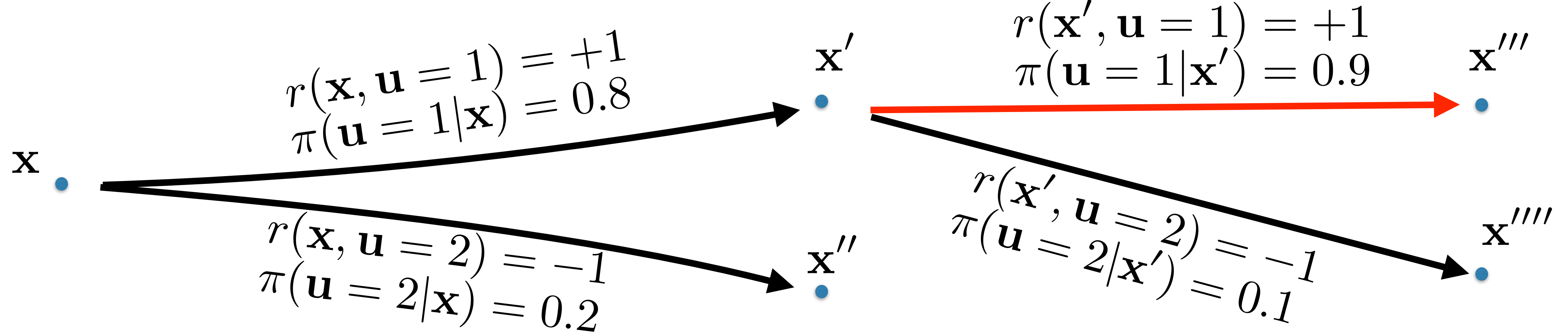
- Search for Q satisfying Bellman equation (for every transition):

$$Q^{\pi}(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}) + \gamma \max_{\mathbf{u}'} Q^{\pi}(\mathbf{x}', \mathbf{u}')$$

- Once we find it, the optimal policy is:

$$\pi^*(\mathbf{x}) = \arg \max_{\mathbf{u}} Q^{\pi}(\mathbf{x}, \mathbf{u}) = \arg \max_{\pi} J_{\pi}$$





$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}'',$

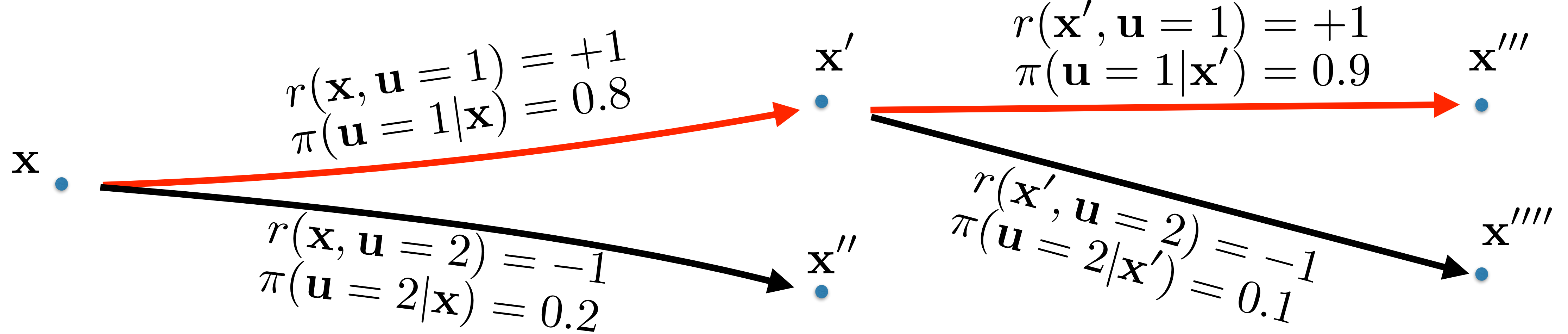
Labels and arrows for the sequence:

- state $\mathbf{x}_1$ points to $\mathbf{x}_0$
- action $\mathbf{u}_1$ points to $\mathbf{u}_0$
- reward r_1 points to r_1
- next state $\mathbf{x}_2$ points to $\mathbf{x}_1$

Q	u=1	u=2
$\mathbf{x}$	0	0
$\mathbf{x}'$	0	0
$\mathbf{x}''$	0	0
$\mathbf{x}'''$	0	0
$\mathbf{x}''''$	0	0

$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) = +1$$

$$Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2 = \mathbf{x}'', \mathbf{u}) = +1$$



$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}''$,

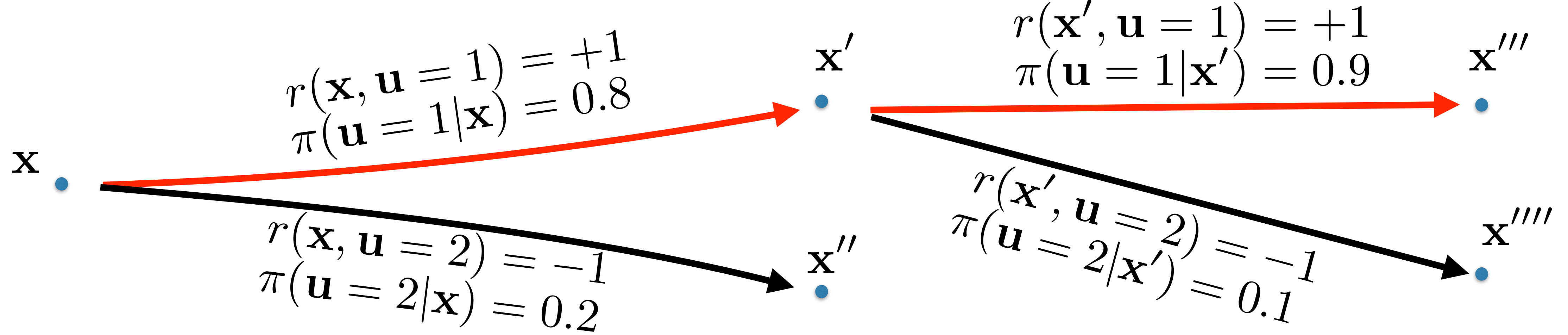
state $\mathbf{x}_1$ action $\mathbf{u}_1$ reward r_1 next state $\mathbf{x}_2$

Q	u=1	u=2
x	0	0
x'	0	0
x''	0	0
x'''	0	0
x''''	0	0

$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) = +1$$

$$Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2 = \mathbf{x}'', \mathbf{u}) = +1$$

Bellman equation is not satisfied



$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}'',$

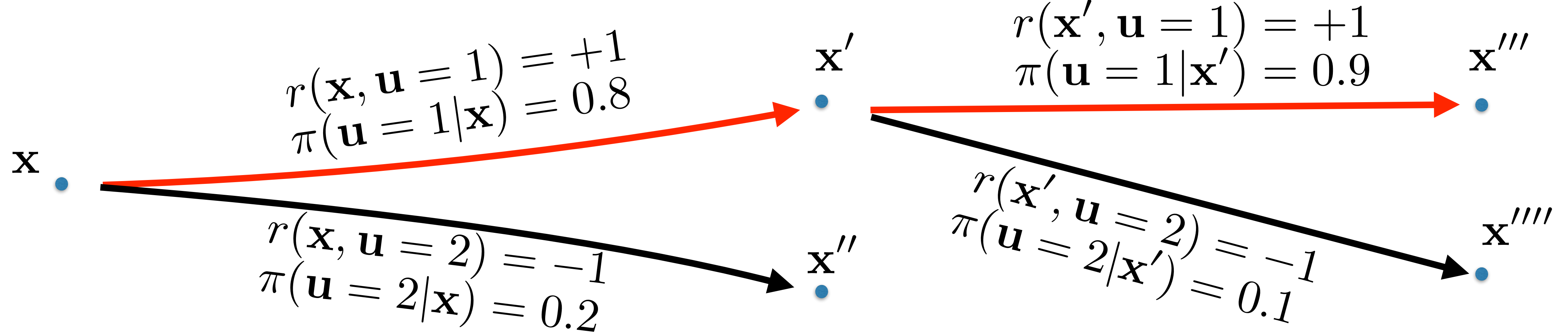
state $\mathbf{x}_1$ action $\mathbf{u}_1$ reward r_1 next state $\mathbf{x}_2$

Q	u=1	u=2
x	1	0
x'	0	0
x''	0	0
x'''	0	0
x''''	0	0

$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) = +1$$

$$Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2 = \mathbf{x}'', \mathbf{u}) = +1$$

Search for solution by successive subst. of RHS to LHS.



$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}'',$

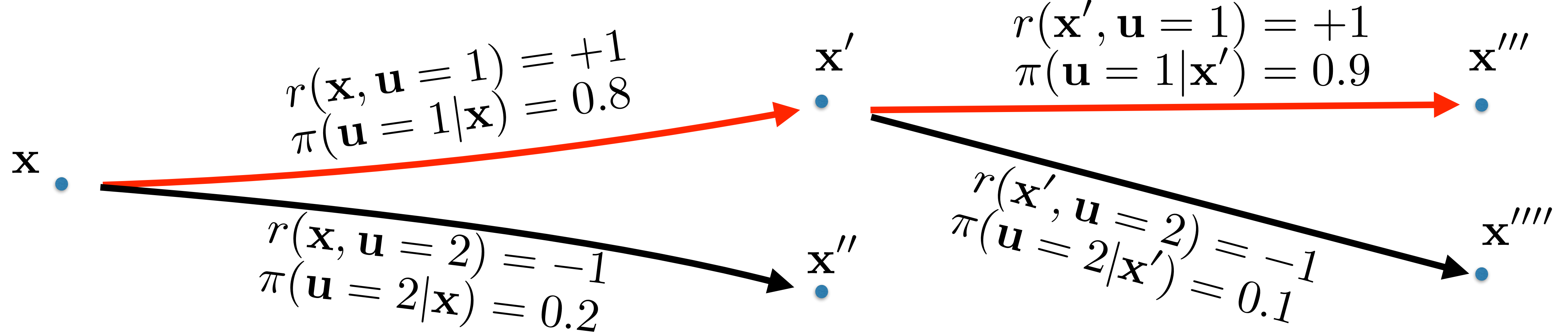
state $\mathbf{x}_1$ action $\mathbf{u}_1$ reward r_1 next state $\mathbf{x}_2$

$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) = +1$$

$$Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2 = \mathbf{x}'', \mathbf{u}) = +1$$

Search for solution by successive subst. of RHS to LHS.

Q	u=1	u=2
x	1	0
x'	1	0
x''	0	0
x'''	0	0
x''''	0	0



$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}''$,

Labels for the sequence:

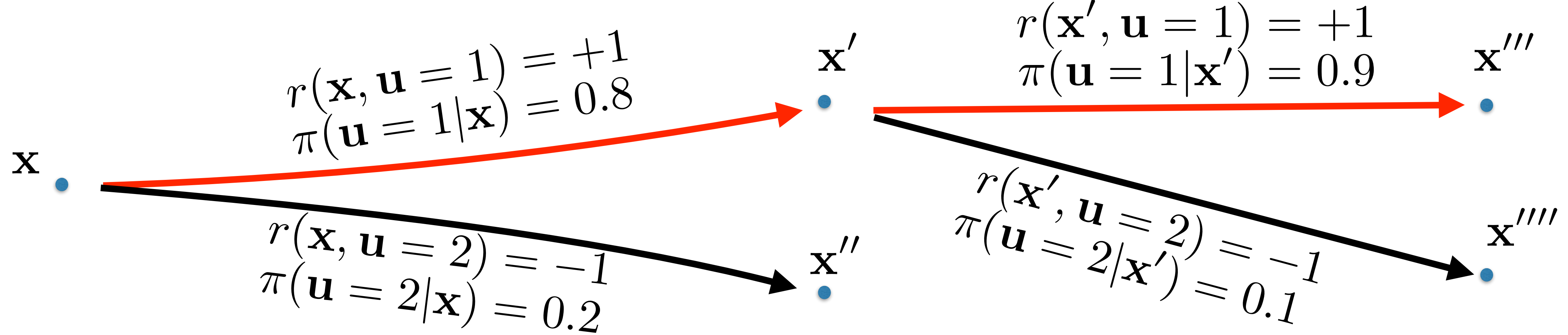
- state $\mathbf{x}_1$ points to $\mathbf{x}_0$
- action $\mathbf{u}_1$ points to $\mathbf{u}_0$
- reward r_1 points to r_1
- next state $\mathbf{x}_2$ points to $\mathbf{x}_1$

Q	u=1	u=2
$\mathbf{x}$	1.9	0
$\mathbf{x}'$	1	0
$\mathbf{x}''$	0	0
$\mathbf{x}'''$	0	0
$\mathbf{x}''''$	0	0

$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) = +1.9$$

$$Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2 = \mathbf{x}'', \mathbf{u}) = +1$$

Substitute of RHS to LHS.



$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}'',$

Labels for the sequence:

- state $\mathbf{x}_1$ points to $\mathbf{x}_0$
- action $\mathbf{u}_1$ points to $\mathbf{u}_0$
- reward r_1 points to r_1
- next state $\mathbf{x}_2$ points to $\mathbf{x}_1$

Q	u=1	u=2
x	1.9	0
x'	1	0
x''	0	0
x'''	0	0
x''''	0	0

$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) = +1.9$$

$$Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2 = \mathbf{x}'', \mathbf{u}) = +1$$

If Q is table, the mapping is contraction and iterations always converge to a fixed point of Bellman operator.

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
3. Repeat from 1

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

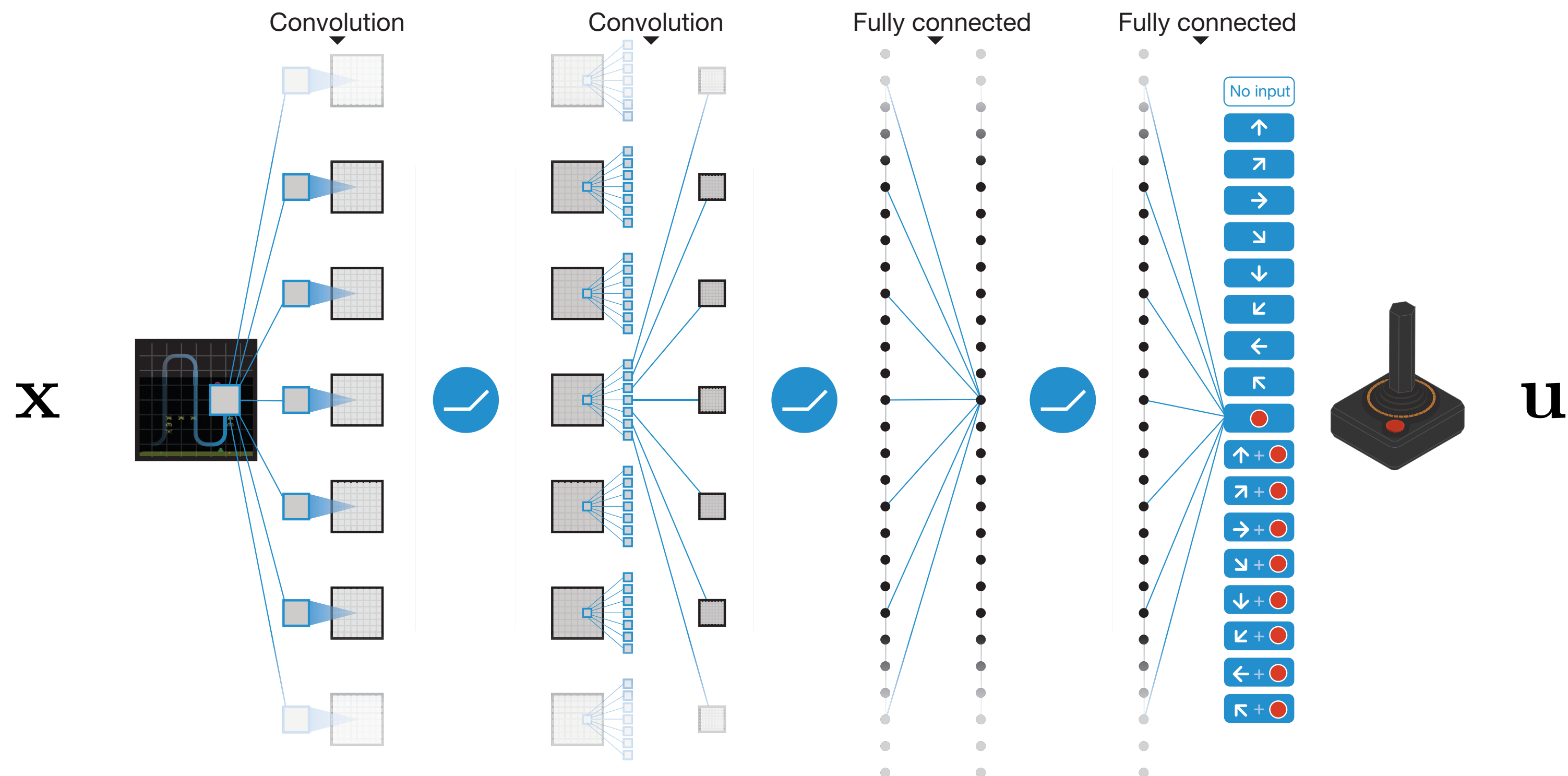
Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
2. Estimate $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\theta}(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning
$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$
4. Repeat from 1

Mnih et al. Nature 2015

- 2600 atari games
- **state space $\mathbf{x}$** : last four frames to capture dynamics (e.g. RGB images in VGA resolution)
- **action space $\mathbf{u}$** : 18 discrete joystic actions (8 direction + 8 direction with button + neutral action + neutral with button)

$$Q_{\theta}(\mathbf{x}, \mathbf{u})$$



Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

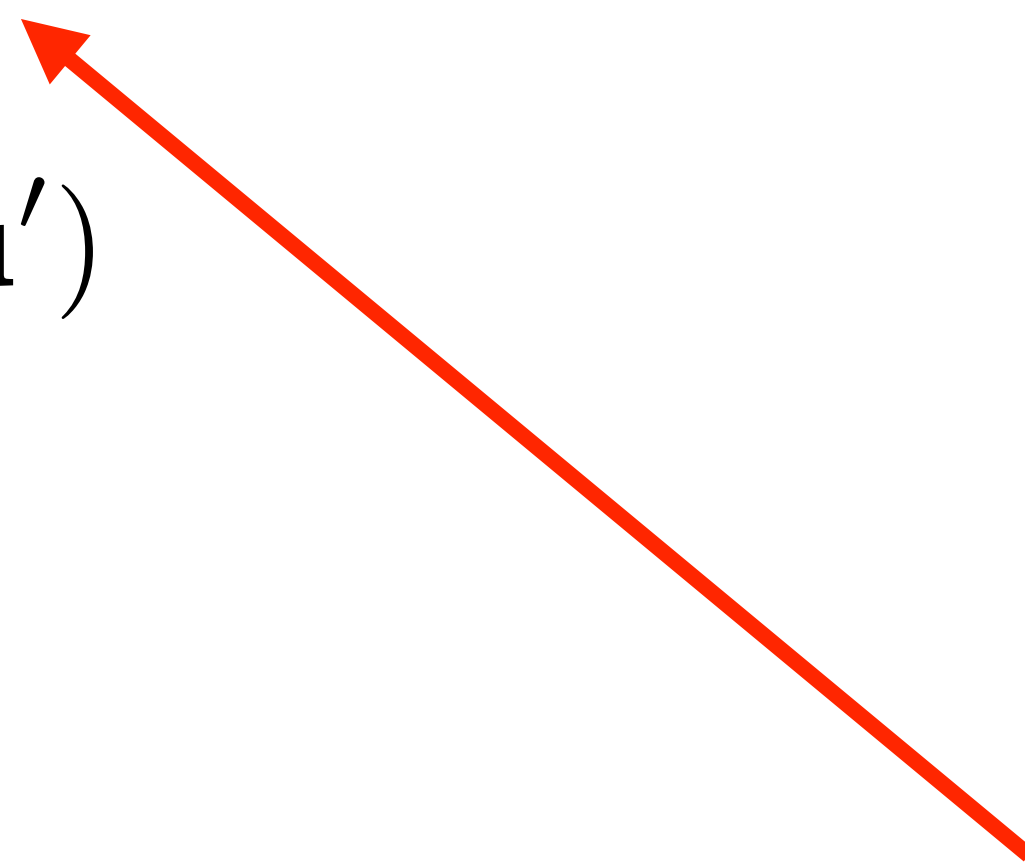
Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
2. Estimate target $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\theta}(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning

$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

4. Repeat from 1

**There are 2 wtf issues in this algorithm !
Do you see them?**



Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
2. Estimate target $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\theta}(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning (assumes i.i.d+n.n.)

$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

4. Repeat from 1
- ★ Transitions are strongly correlated !

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\theta}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

★ Transitions are strongly correlated !

5. Repeat from 1

★ Transitions are strongly correlated !

Solution: ReplayMemory => minibatch sampled at random
(decorrelates samples to be “more i.i.d”)

```
Transition = namedtuple( 'Transition',
                        ('state', 'action', 'next_state', 'reward'))

class ReplayMemory(object):
    def push(self, *args):
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)
```

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\theta}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

★ Transitions are strongly correlated !

5. Repeat from 1

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\theta}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

5. Repeat from 1

★ Transitions are strongly correlated !

★ Training/Testing distribution mismatch

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\bar{\theta}}(\mathbf{x}', \mathbf{u}')$ Target net (slowly upd.) encourage stability
4. Update parameters by learning (assumes i.i.d+n.n.)

$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

Policy net (regularly upd.)
encourage exploration

5. Repeat from 1

Q-learning

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$
 2. Solve $Q(\mathbf{x}, \mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
 3. Repeat from 1
- Curse of dimensionality
 - Replace table $Q(\mathbf{x}, \mathbf{u})$ by function $Q_{\theta}(\mathbf{x}, \mathbf{u})$

Approximate Q-learning (DQN)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\bar{\theta}}(\mathbf{x}', \mathbf{u}')$ Target net (slowly upd.) encourage stability
4. Update parameters by learning (assumes i.i.d+n.n.)

$$\arg \min_{\theta} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

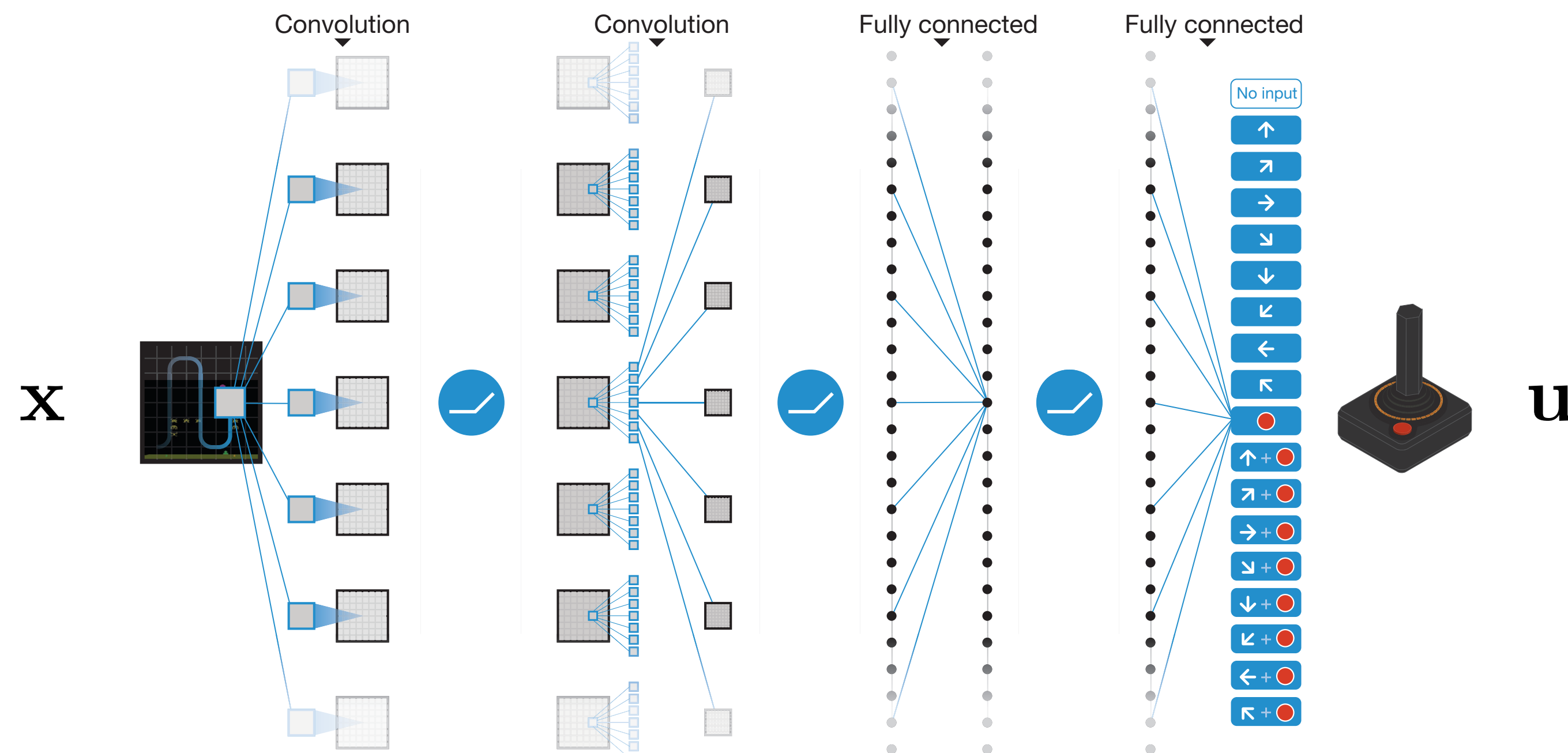
Policy net (regularly upd.) encourage exploration

5. Update target network $\bar{\theta} := \alpha \theta + (1 - \alpha) \bar{\theta}$
6. Repeat from 1

Mnih et al. Nature 2015

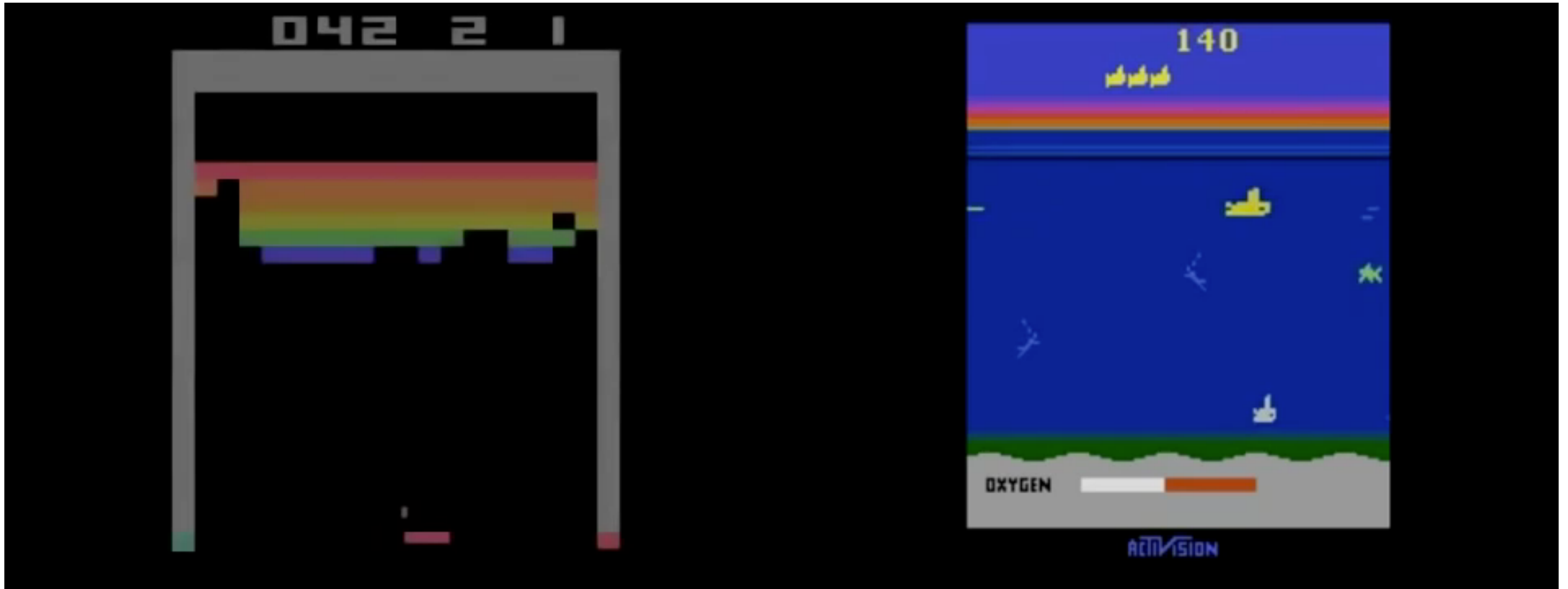
- 2600 atari games
- **state space $\mathbf{x}$** : last four frames to capture dynamics (e.g. RGB images in VGA resolution)
- **action space $\mathbf{u}$** : 18 discrete joystic actions (8 direction + 8 direction with button + neutral action + neutral with button)

$$Q_{\theta}(\mathbf{x}, \mathbf{u})$$



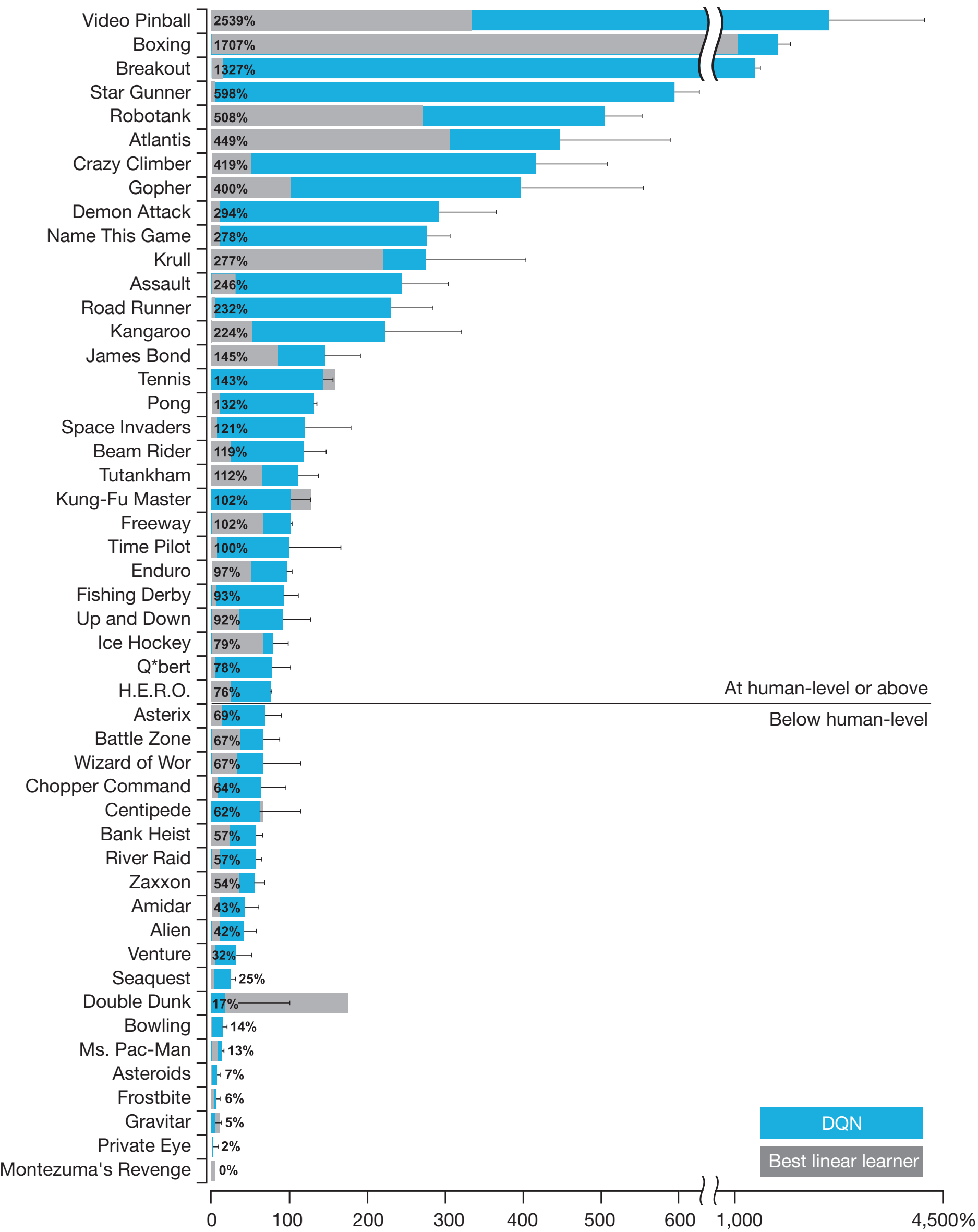
Mnih et al. Nature 2015

- **replay buffer:** decorrelates samples to be “more i.i.d”
- **two Q-networks:** suppress trn/tst mismatch (oscillations)



- collection of control tasks: <https://gym.openai.com>

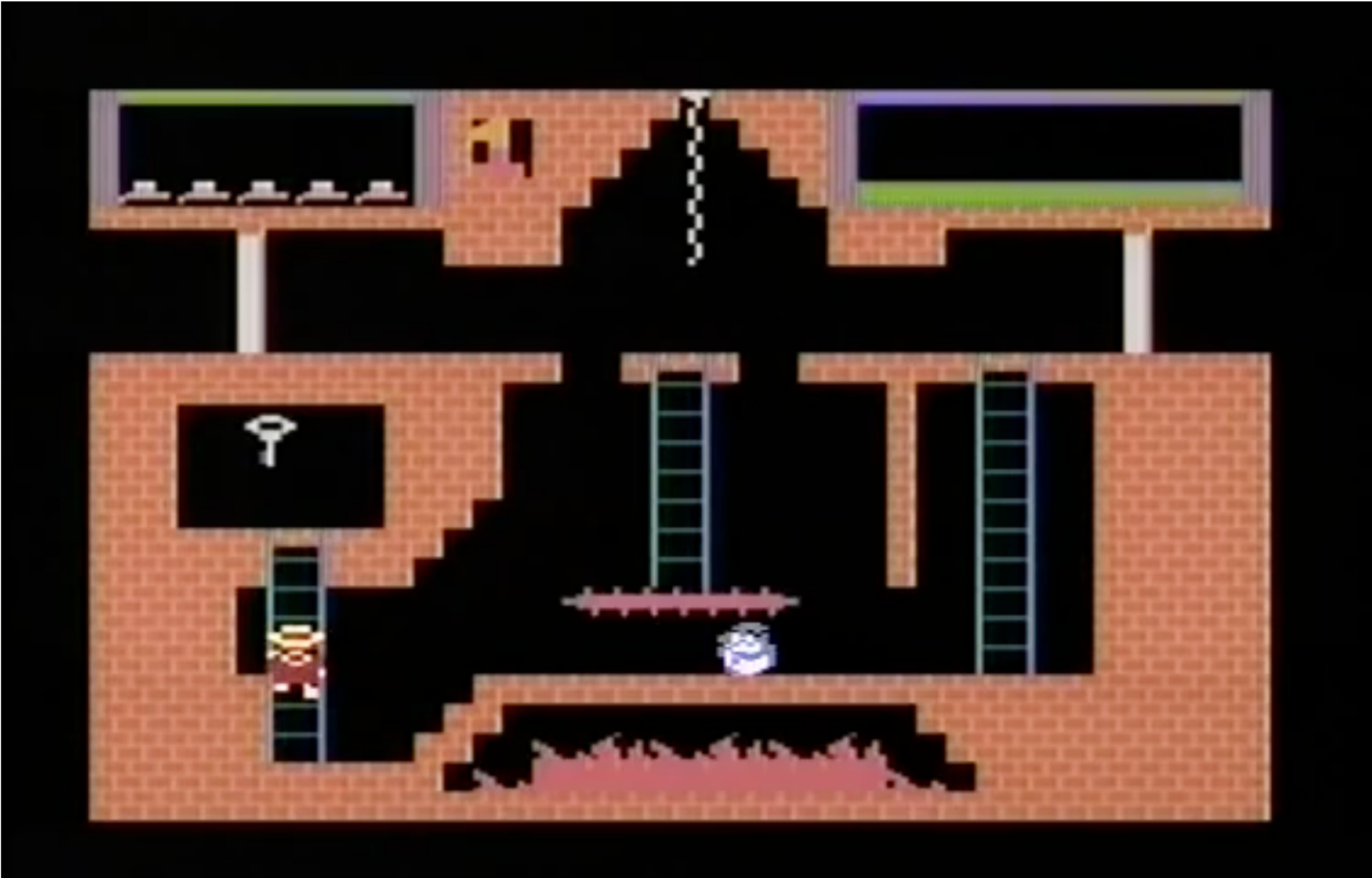
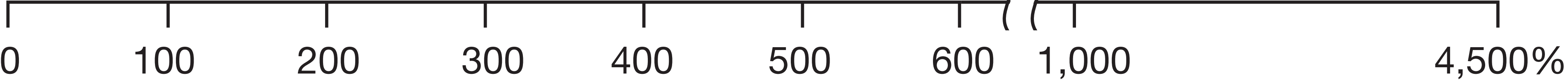
Mnih et al. Nature 2015



Montezuma's Revenge

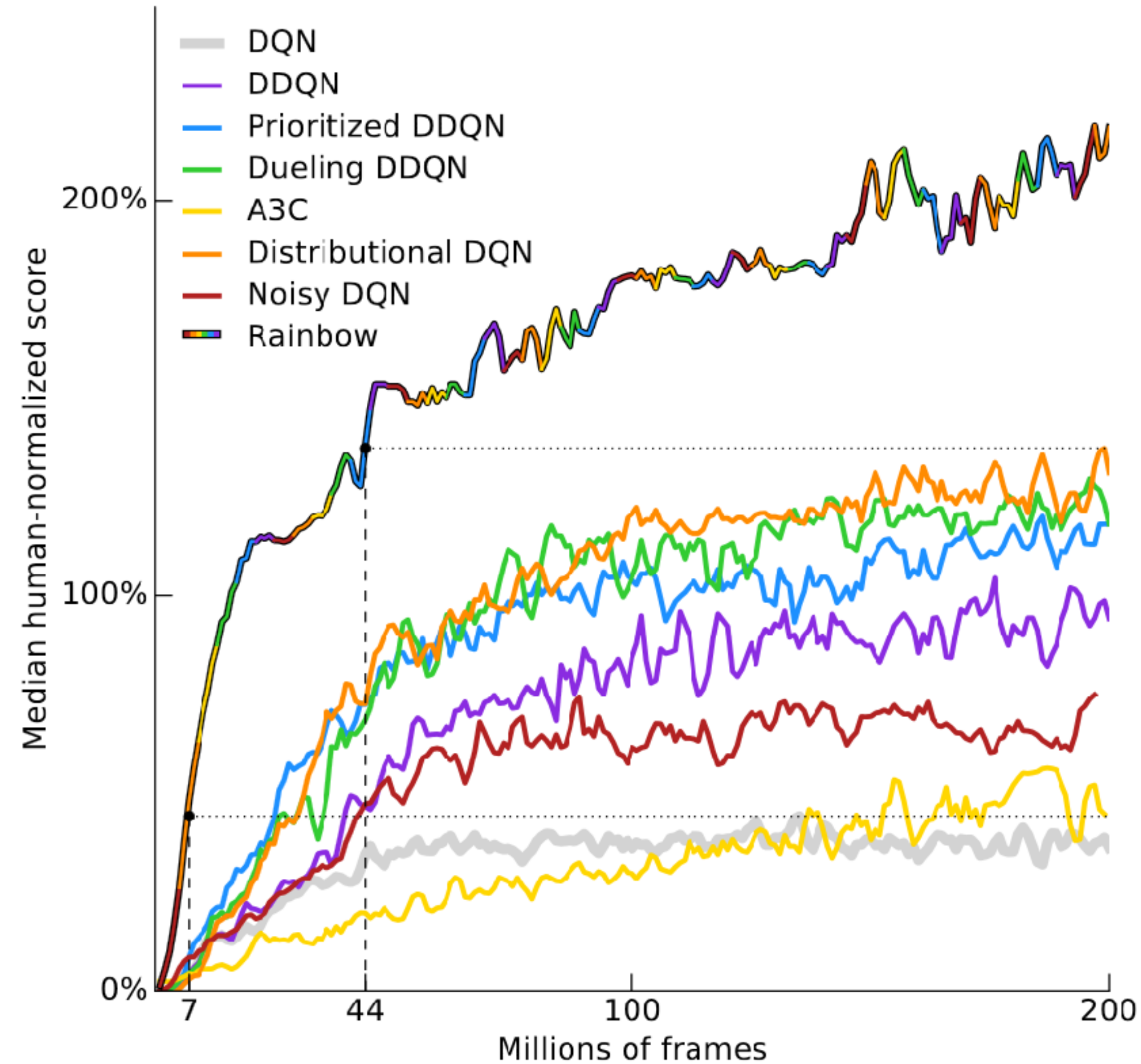
0%

Best linear learner

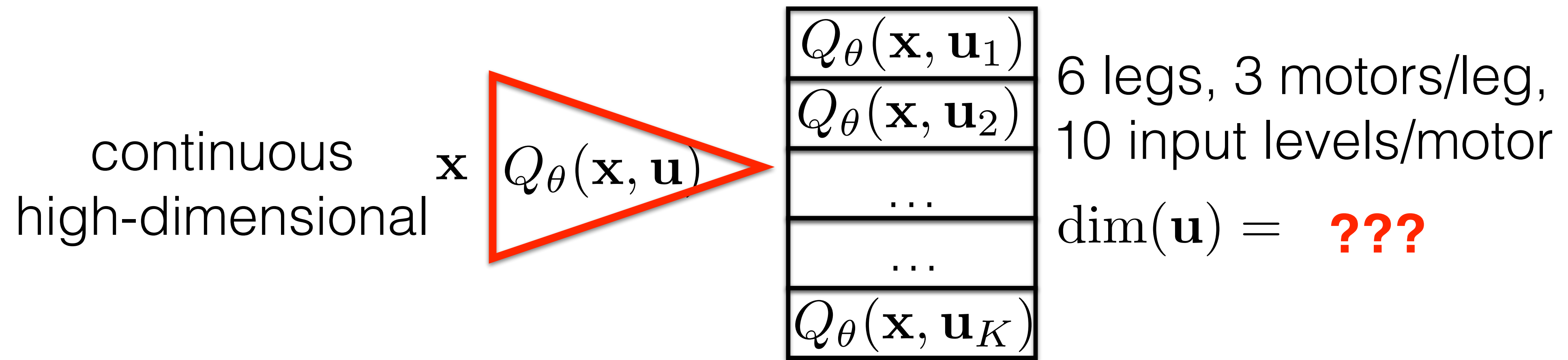


Hessel et. al Rainbow DQN, 2017

Ensemble of different RL methods

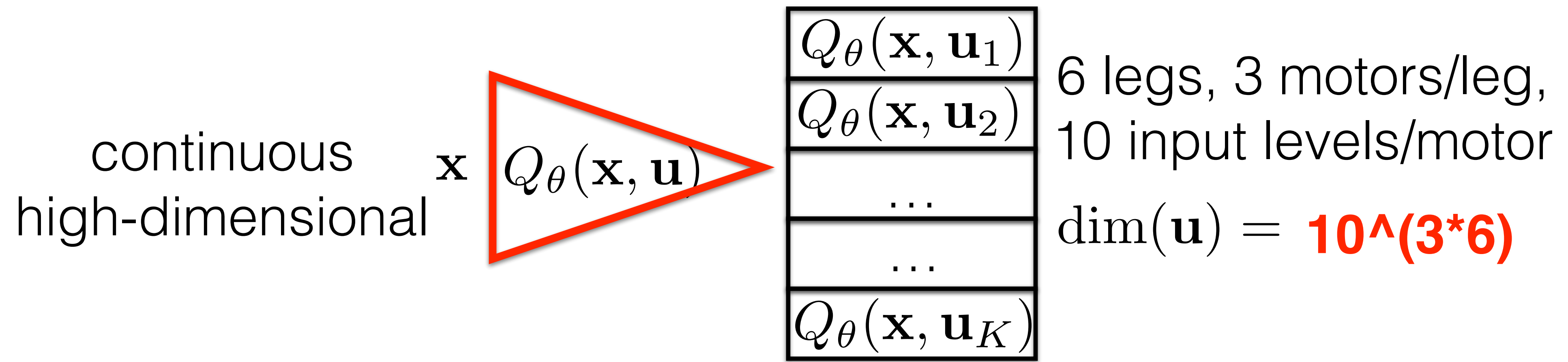


Main bottleneck of approximate Q-learning (DQN)



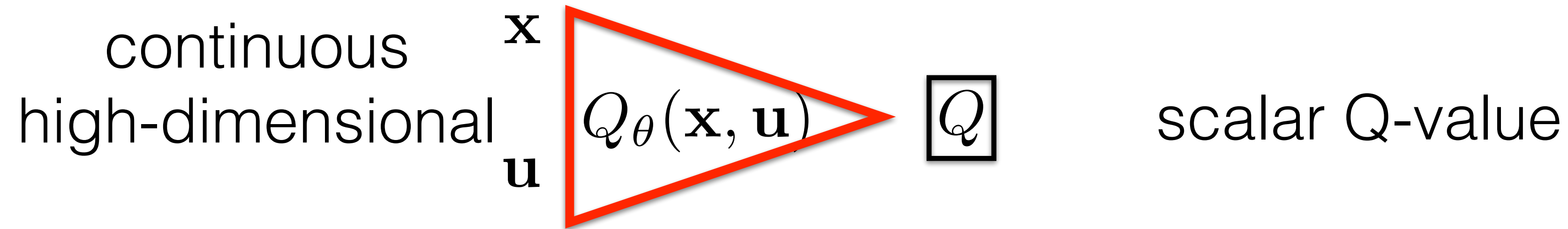
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\bar{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$
6. Update target network $\bar{\theta} := \alpha \theta + (1 - \alpha) \bar{\theta}$
7. Repeat from 1

Main bottleneck of approximate Q-learning (DQN)



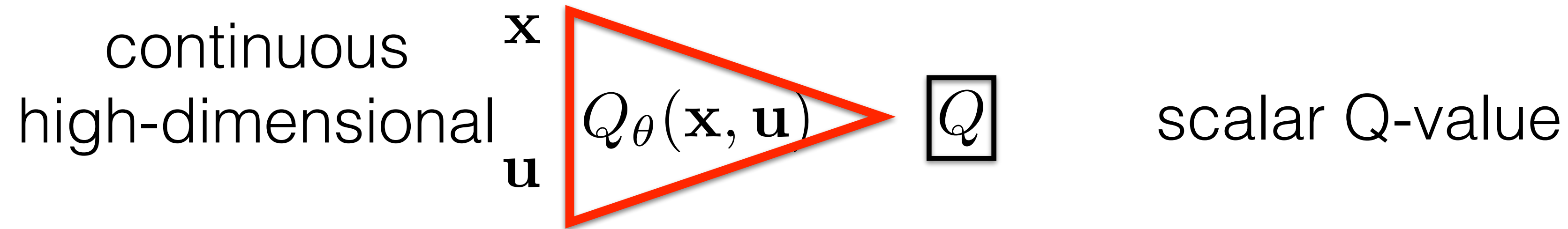
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\bar{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$
6. Update target network $\bar{\theta} := \alpha \theta + (1 - \alpha) \bar{\theta}$
7. Repeat from 1

Main bottleneck of approximate Q-learning (DQN)



1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\bar{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$
6. Update target network $\bar{\theta} := \alpha \theta + (1 - \alpha) \bar{\theta}$
7. Repeat from 1

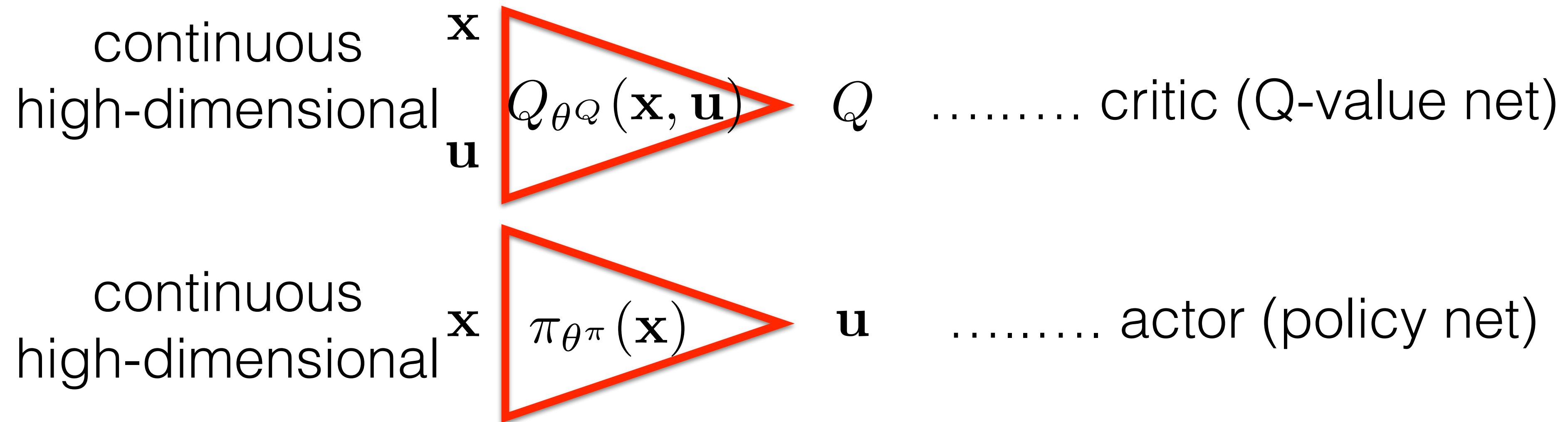
Main bottleneck of approximate Q-learning (DQN)



You cannot exhaustively maximize

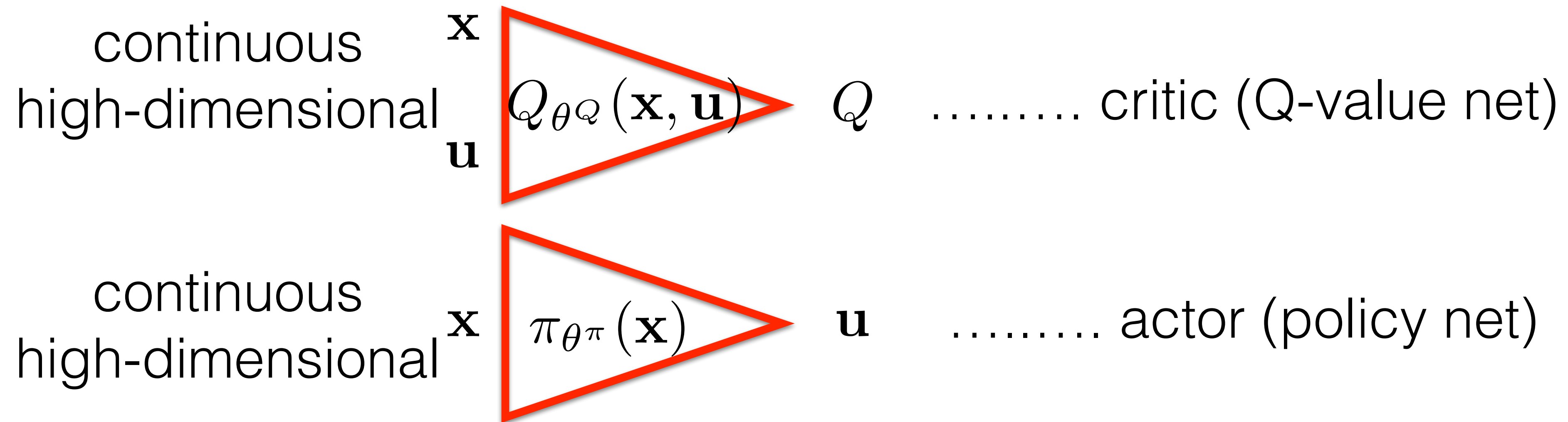
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\bar{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$
6. Update target network $\bar{\theta} := \alpha \theta + (1 - \alpha) \bar{\theta}$
7. Repeat from 1

Deep Deterministic Policy Gradient (DDPG)



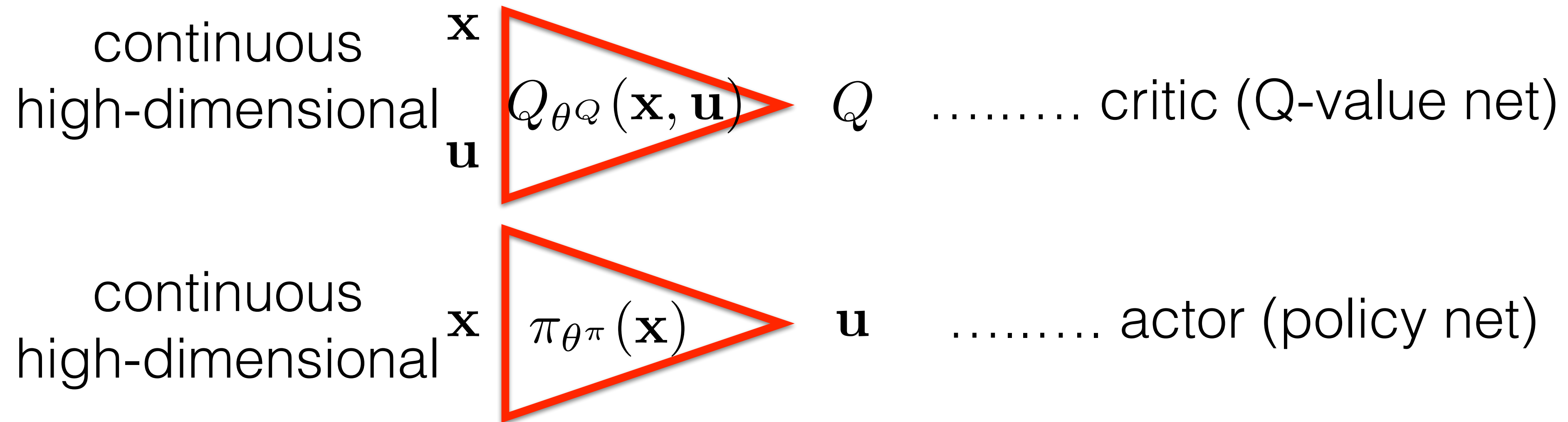
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta^Q}}(\mathbf{x}', \mathbf{u}')$
4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$
5. Update target network $\overline{\theta^Q} := \alpha \theta^Q + (1 - \alpha) \overline{\theta^Q}$
6. Repeat from 1

Deep Deterministic Policy Gradient (DDPG)



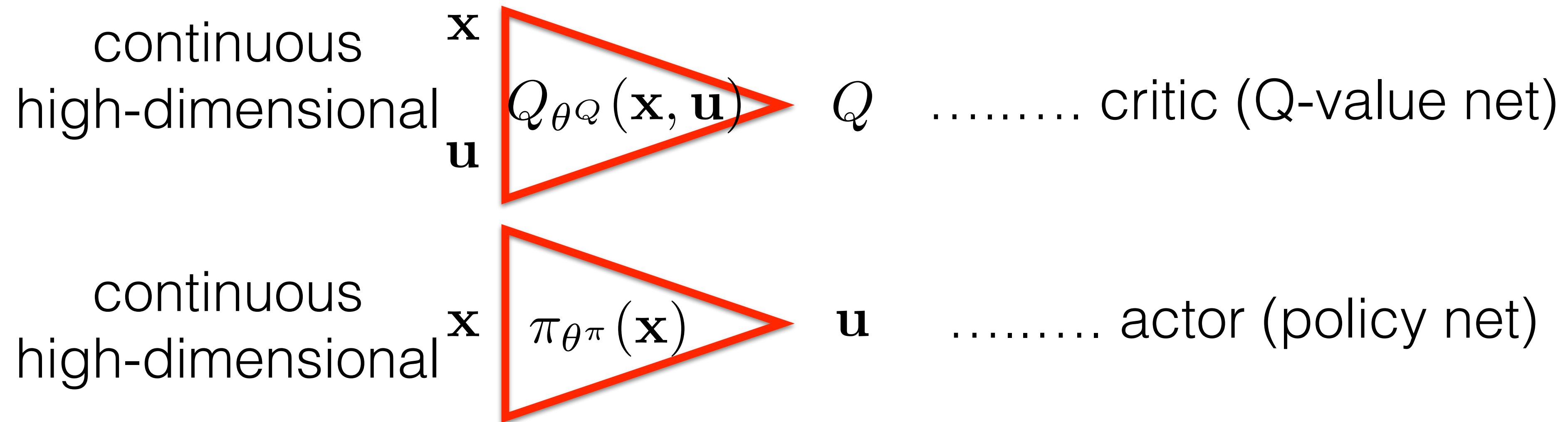
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta^Q}}(\mathbf{x}', \mathbf{u}')$
4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$
5. Update target network $\overline{\theta^Q} := \alpha \theta^Q + (1 - \alpha) \overline{\theta^Q}$
6. Repeat from 1

Deep Deterministic Policy Gradient (DDPG)



1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma Q_{\overline{\theta^Q}}(\mathbf{x}', \pi_{\overline{\theta^\pi}}(\mathbf{x}'))$
4. Update critic
$$\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$
5. Update target network $\overline{\theta^Q} := \alpha \theta^Q + (1 - \alpha) \overline{\theta^Q}$
6. Repeat from 1

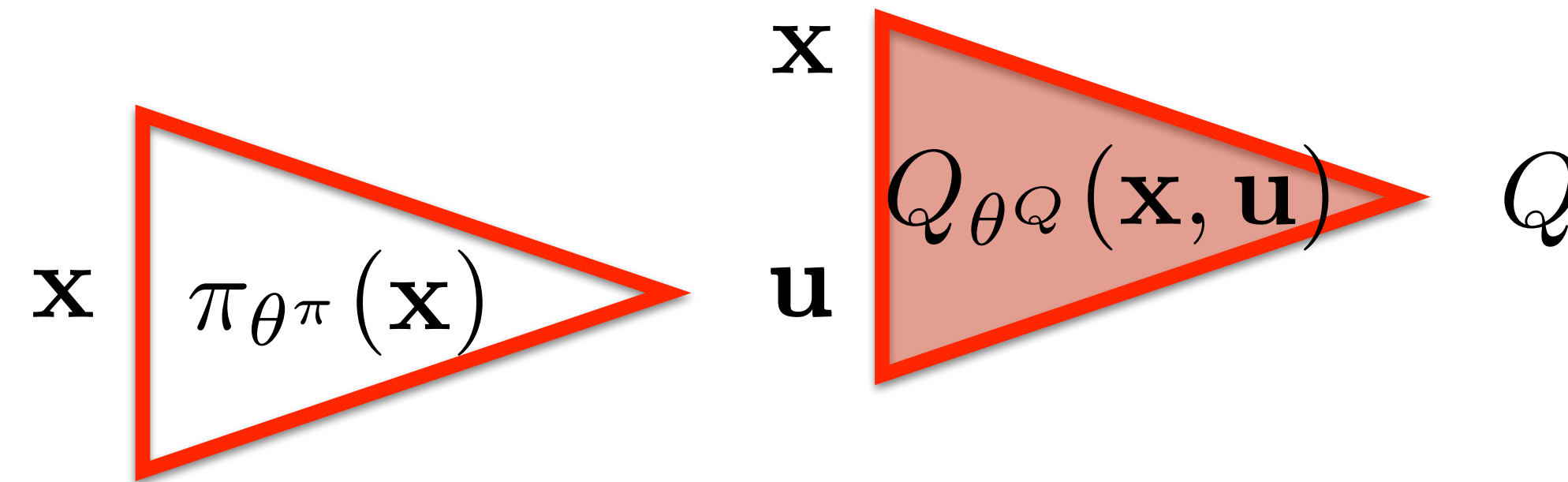
Deep Deterministic Policy Gradient (DDPG)



1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma Q_{\overline{\theta^Q}}(\mathbf{x}', \pi_{\overline{\theta^\pi}}(\mathbf{x}'))$
4. Update critic
$$\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$
5. Update actor
$$\arg \max_{\theta^\pi} \sum_{\mathbf{x}} Q_{\theta^Q}(\mathbf{x}, \pi_{\theta^\pi}(\mathbf{x}))$$
6. Update target network
$$\begin{aligned} \theta^Q &:= \alpha \theta^Q + (1 - \alpha) \overline{\theta^Q} \\ \theta^\pi &:= \alpha \theta^\pi + (1 - \alpha) \overline{\theta^\pi} \end{aligned}$$
7. Repeat from 1

Deep Deterministic Policy Gradient (DDPG)

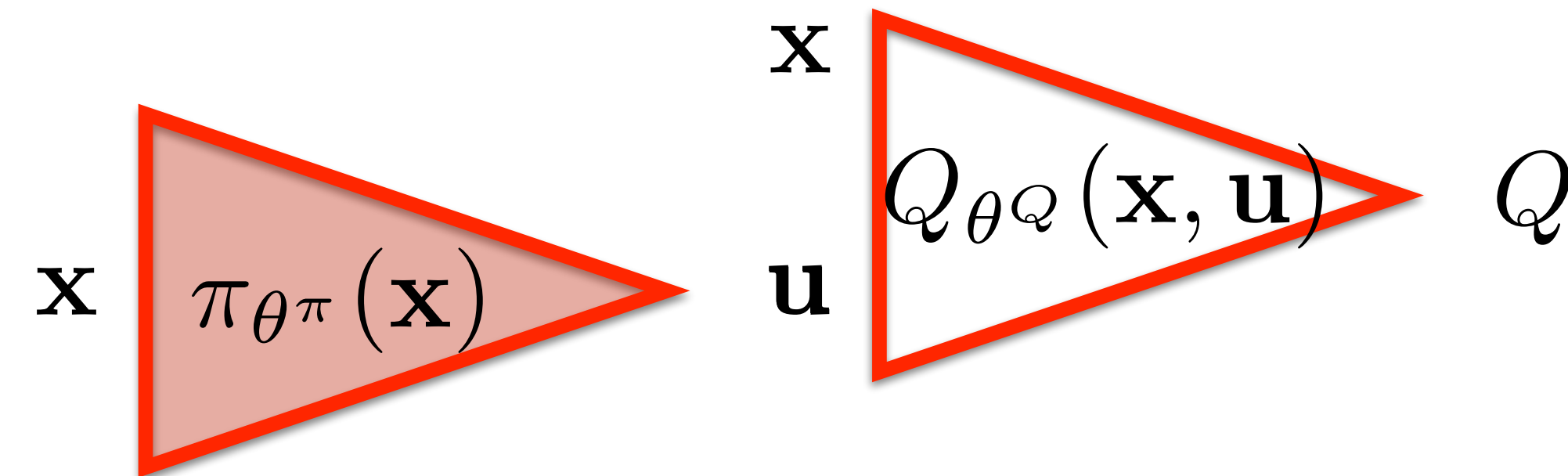
Update critic



1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma Q_{\overline{\theta^Q}}(\mathbf{x}', \pi_{\overline{\theta^\pi}}(\mathbf{x}'))$
4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$
5. Update actor $\arg \max_{\theta^\pi} \sum_{\mathbf{x}} Q_{\theta^Q}(\mathbf{x}, \pi_{\theta^\pi}(\mathbf{x}))$
6. Update target network $\begin{aligned} \overline{\theta^Q} &:= \alpha \theta^Q + (1 - \alpha) \overline{\theta^Q} \\ \overline{\theta^\pi} &:= \alpha \theta^\pi + (1 - \alpha) \overline{\theta^\pi} \end{aligned}$
7. Repeat from 1

Deep Deterministic Policy Gradient (DDPG)

Update actor



1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}'] \Rightarrow \text{ReplayMemory}$
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma Q_{\overline{\theta^Q}}(\mathbf{x}', \pi_{\overline{\theta^{\pi}}}(\mathbf{x}'))$

4. Update critic $\arg \min_{\theta^Q} \sum_{\mathbf{x}, \mathbf{u}, \mathbf{y}} \|Q_{\theta^Q}(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$

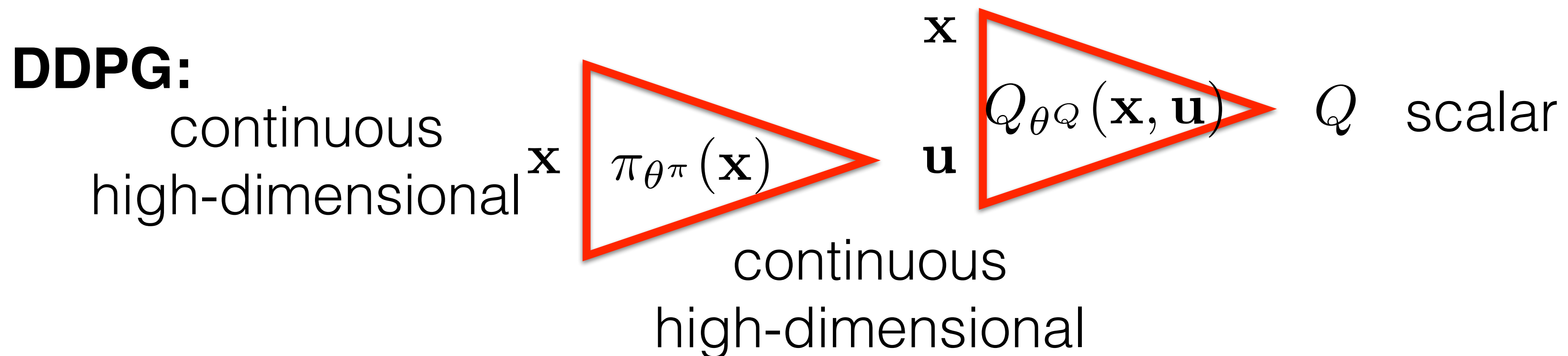
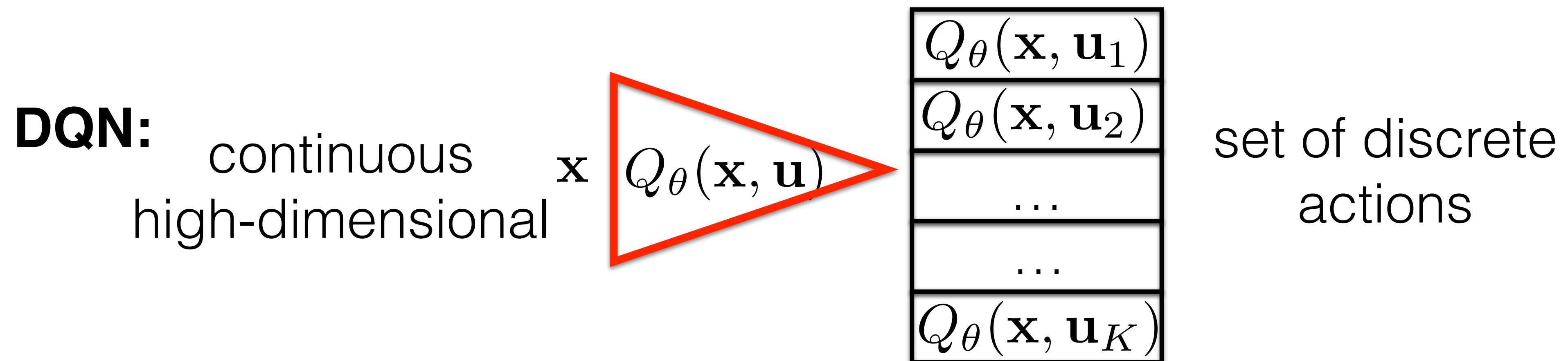
5. Update actor $\arg \max_{\theta^{\pi}} \sum_{\mathbf{x}} Q_{\theta^Q}(\mathbf{x}, \pi_{\theta^{\pi}}(\mathbf{x}))$

6. Update target network $\overline{\theta^Q} := \alpha \theta^Q + (1 - \alpha) \overline{\theta^Q}$
 $\overline{\theta^{\pi}} := \alpha \theta^{\pi} + (1 - \alpha) \overline{\theta^{\pi}}$

7. Repeat from 1

Summary

- DQN and DDPG are off-policy algorithms
(can learn from transitions collected by a different policy)

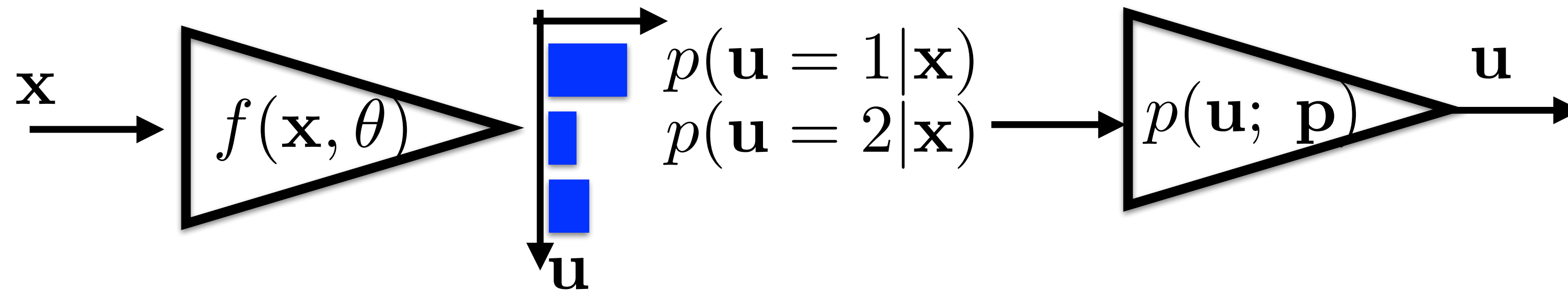


Summary

- **off-policy** algorithms can learn from transitions collected by a different policy
- **on-policy** algorithms can learn only from transitions collected by the current policy
- DQN and DDPG are **off-policy** algorithms
 - => Can use ReplayMemory (which includes outdated transitions)
 - => Can learn deterministic policy (while using synth.noise for exploration)
- **Replay memory** helps to decorrelate samples.
- **Slowly updated network** suppresses trn/tst distribution mismatch (and consequent learning oscillations).
- **Ensemble** of different algorithms helps a lot.
- Next: **On-policy** methods with stochastic gradient

REINFORCE

Stochastic policy
discrete control:
 $\pi_{\theta}(\mathbf{u}|\mathbf{x}) \sim p(\mathbf{u}; \mathbf{p})$

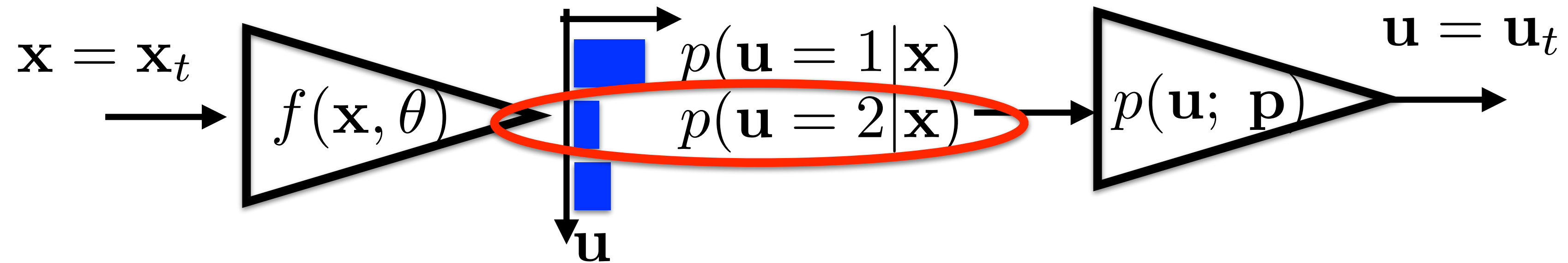


1. Initialize policy $\pi_{\theta}(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$
2. Collect trajectories τ with policy π_{θ}
3. Optimize criterion: $\theta := \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$ **What is the gradient???**
4. Repeat from 2

You control robot with $\mathbf{u} = 2$ and sum of rewards $r(\tau)$ is **high**, how to change policy?

You control robot with $\mathbf{u} = 1$ and sum of rewards $r(\tau)$ is **small**, how to change policy?

What is the gradient???



Assume that you performed action $\mathbf{u}_t$ in state $\mathbf{x}_t$

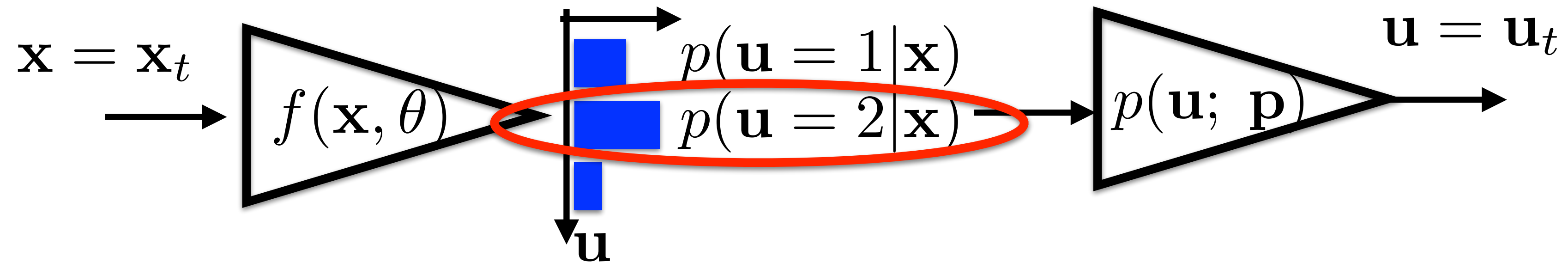
$$\frac{\partial J(\theta)}{\partial \theta} \approx \sum_{t=0}^T \frac{\partial \log(\pi_{\theta}(\mathbf{u}_t | \mathbf{x}_t))}{\partial \theta} \cdot r(\tau)$$

The term $\frac{\partial \log(\pi_{\theta}(\mathbf{u}_t | \mathbf{x}_t))}{\partial \theta}$ is highlighted in a light red box. A red arrow points from this box to the text "Direction (in θ -space) that increases probability of performed action." below. The term $r(\tau)$ is highlighted in a light blue box. A blue arrow points from this box to the text "Sum of rewards along the resulting trajectory." below. To the right, a green box contains the text $r(\tau) = +100$.

Sum of rewards along the resulting trajectory.

Direction (in θ -space) that increases probability of performed action.

What is the gradient???



Assume that you performed action $\mathbf{u}_t$ in state $\mathbf{x}_t$

$$\frac{\partial J(\theta)}{\partial \theta} \approx \sum_{t=0}^T \frac{\partial \log(\pi_{\theta}(\mathbf{u}_t | \mathbf{x}_t))}{\partial \theta} \cdot r(\tau)$$

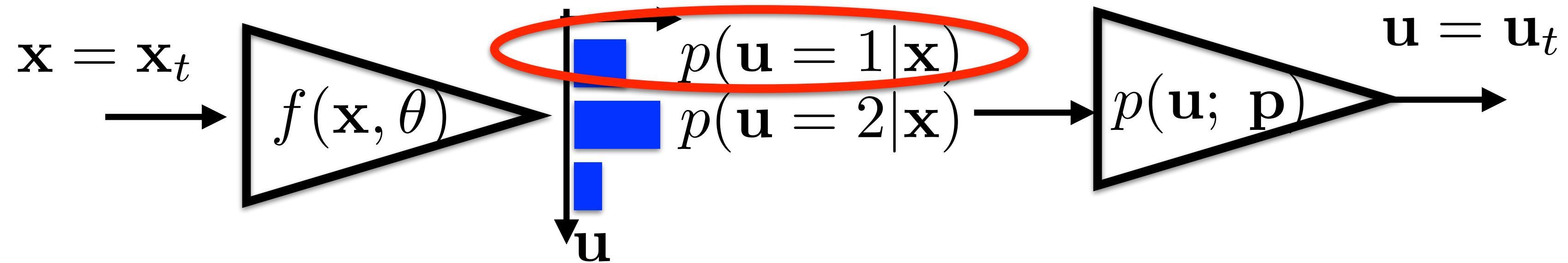
The equation shows the gradient of the objective function $J(\theta)$ with respect to the parameters θ . The sum is over time steps t from 0 to T . The term $\frac{\partial \log(\pi_{\theta}(\mathbf{u}_t | \mathbf{x}_t))}{\partial \theta}$ is highlighted in a red box, and $r(\tau)$ is highlighted in a blue box. A red arrow points from the red box to the text below, and a blue arrow points from the blue box to the text below.

$$r(\tau) = +100$$

Sum of rewards along the resulting trajectory.

Direction (in θ -space) that increases probability of performed action.

What is the gradient???



Assume that you performed action $\mathbf{u}_t$ in state $\mathbf{x}_t$

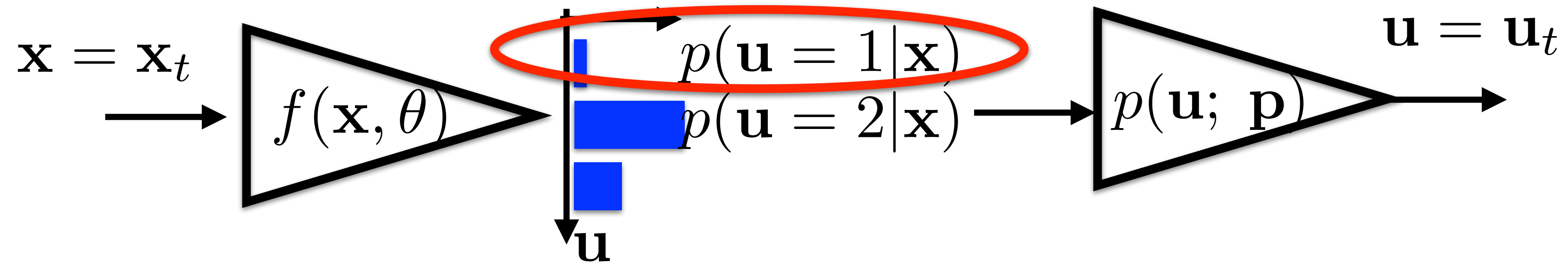
$$\frac{\partial J(\theta)}{\partial \theta} \approx \sum_{t=0}^T \frac{\partial \log(\pi_{\theta}(\mathbf{u}_t | \mathbf{x}_t))}{\partial \theta} \cdot r(\tau)$$

$$r(\tau) = -100$$

Sum of rewards along the resulting trajectory.

Direction (in θ -space) that increases probability of performed action.

What is the gradient???



Assume that you performed action $\mathbf{u}_t$ in state $\mathbf{x}_t$

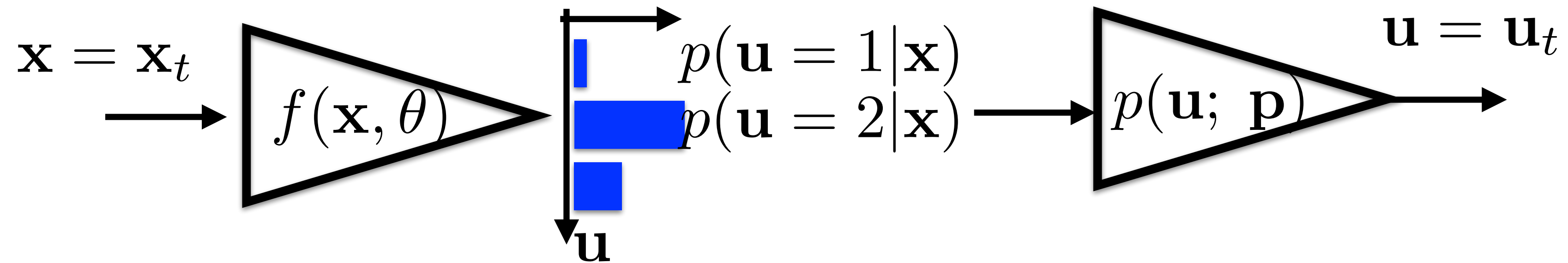
$$\frac{\partial J(\theta)}{\partial \theta} \approx \sum_{t=0}^T \frac{\partial \log(\pi_{\theta}(\mathbf{u}_t | \mathbf{x}_t))}{\partial \theta} \cdot r(\tau)$$

$$r(\tau) = -100$$

Sum of rewards along the resulting trajectory.

Direction (in θ -space) that increases probability of performed action.

REINFORCE



1. Initialize policy $\pi_{\theta}(\mathbf{u} | \mathbf{x}) = f(\mathbf{x}, \theta)$
2. Collect trajectories τ with policy $\pi_{\theta}(\mathbf{u} | \mathbf{x})$
3. Update policy (actor):

$$\frac{\partial J(\theta)}{\partial \theta} \approx \sum_{t=0}^T \frac{\partial \log(\pi_{\theta}(\mathbf{u}_t | \mathbf{x}_t))}{\partial \theta} \cdot r(\tau)$$

$$\theta := \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$

4. Repeat from 2

Policy gradient derivation

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\underbrace{\sum_{t=0}^T \gamma^t r_{t+1}}_{r(\tau)} \right] = \int_T p(\tau | \pi_\theta) r(\tau) d\tau$$

$$\frac{\partial J(\theta)}{\partial \theta} = \int_T \frac{\partial p(\tau | \pi_\theta)}{\partial \theta} r(\tau) d\tau = \int_T p(\tau | \pi_\theta) \frac{\partial \log p(\tau | \pi_\theta)}{\partial \theta} r(\tau) d\tau =$$

$$\frac{\partial p(\tau | \pi_\theta)}{\partial \theta} = p(\tau | \pi_\theta) \frac{\partial \log p(\tau | \pi_\theta)}{\partial \theta}$$

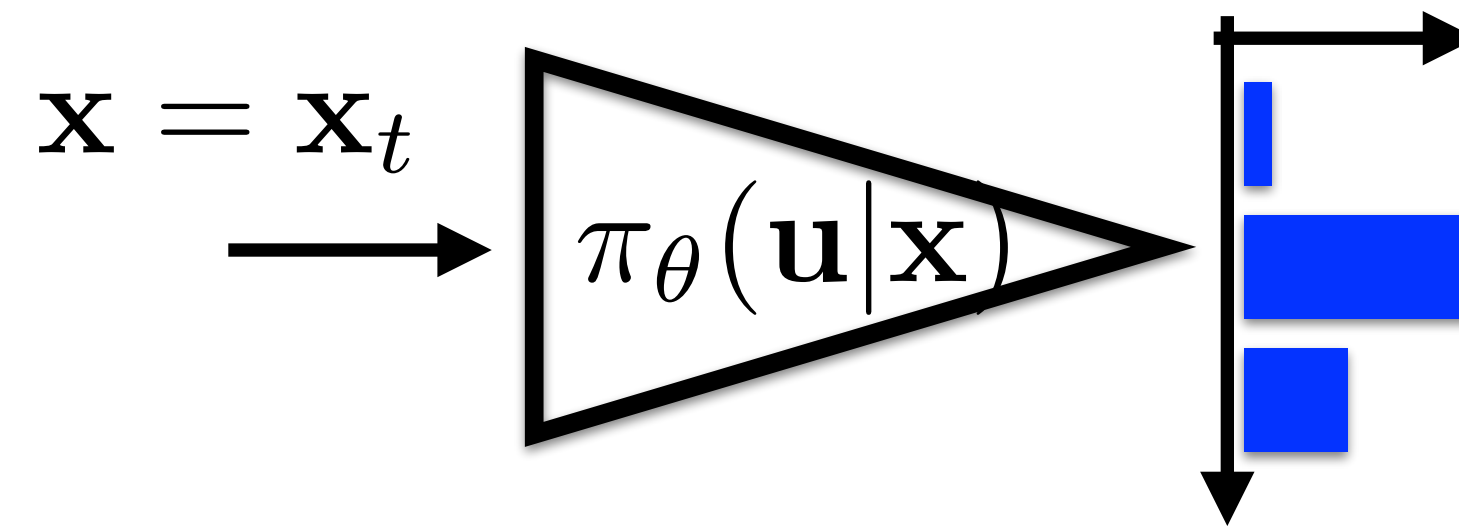
Policy gradient derivation

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\underbrace{\sum_{t=0}^T \gamma^t r_{t+1}}_{r(\tau)} \right] = \int_T p(\tau | \pi_\theta) r(\tau) d\tau$$

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= \int_T \frac{\partial p(\tau | \pi_\theta)}{\partial \theta} r(\tau) d\tau = \int_T p(\tau | \pi_\theta) \frac{\partial \log p(\tau | \pi_\theta)}{\partial \theta} r(\tau) d\tau = \mathbb{E}_{\tau \sim \pi_\theta} \left[\frac{\partial \log p(\tau | \pi_\theta)}{\partial \theta} r(\tau) \right] \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} r(\tau) \right] \approx \frac{1}{N} \sum_{\tau \in \mathcal{T}} \sum_{t=0}^T \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} r(\tau) \end{aligned}$$

$$p(\tau | \pi_\theta) = p(\mathbf{x}_0) \prod_{t=0}^T p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) \pi_\theta(\mathbf{u}_t | \mathbf{x}_t) \quad \dots \text{assuming MDP}$$

$$\frac{\partial \log p(\tau | \pi_\theta)}{\partial \theta} = \frac{\partial}{\partial \theta} [\cancel{\log p(\mathbf{x}_0)} + \sum_{t=0}^T \cancel{\log(p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t))} + \sum_{t=0}^T \log(\pi_\theta(\mathbf{u}_t | \mathbf{x}_t))]$$



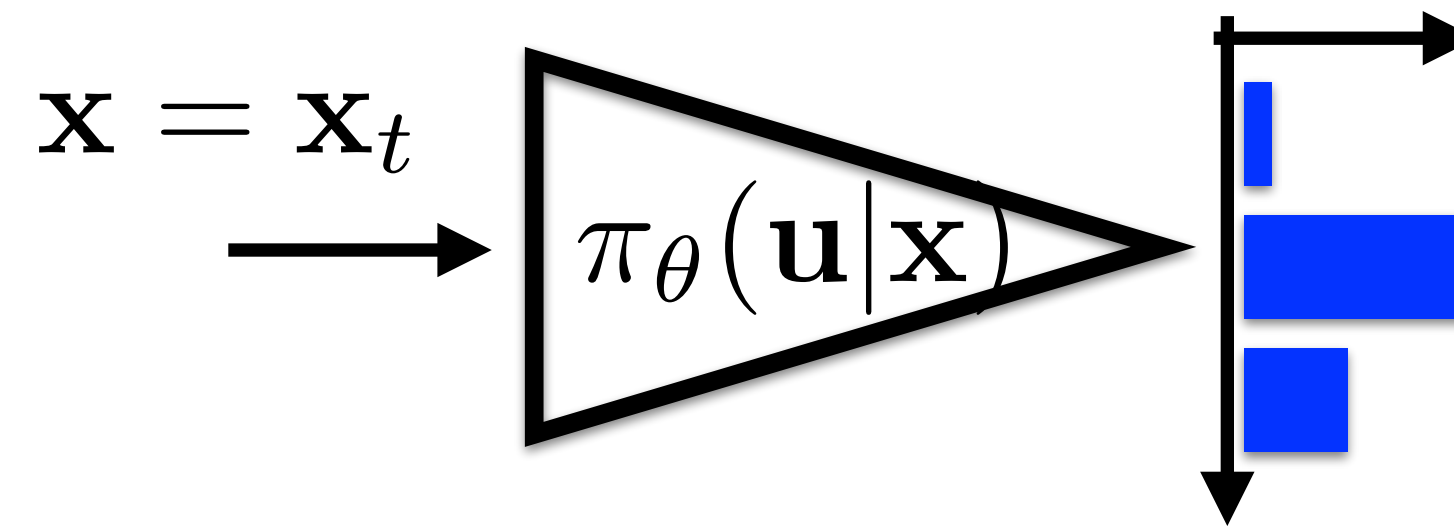
1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
2. Collect trajectories τ with policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
4. **Actor:** Update policy by policy gradient:

$$\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot r(\tau)$$

$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$$

replace by advantage function

5. Repeat from 2



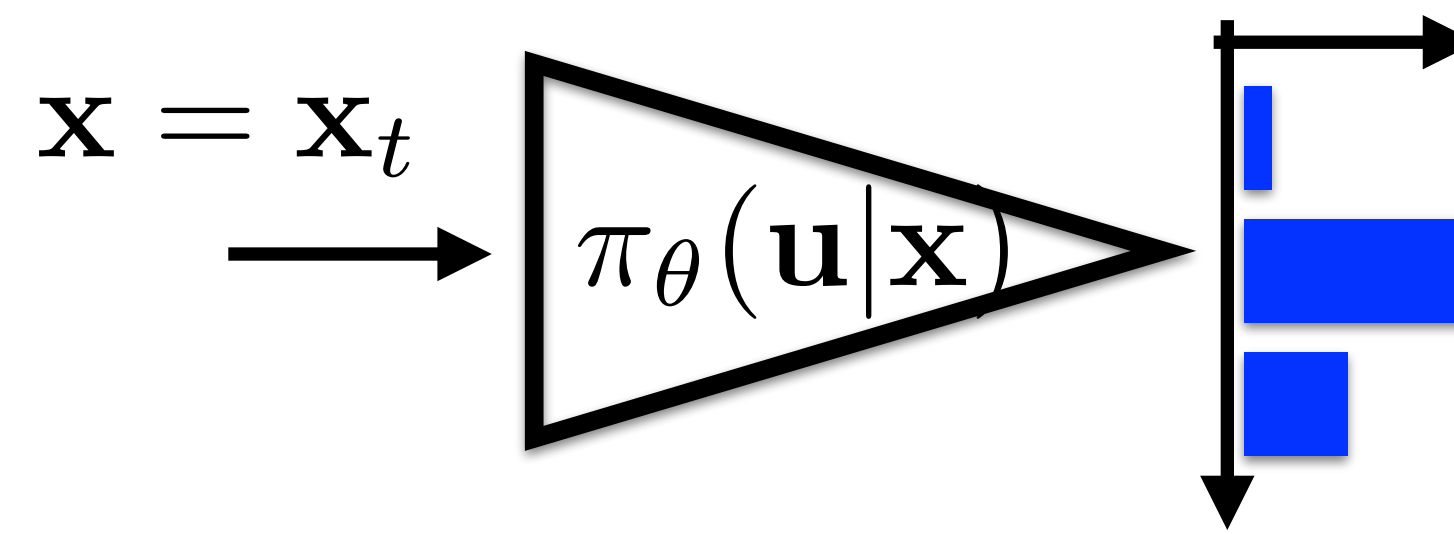
1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
2. Collect trajectories τ with policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
4. **Actor:** Update policy by policy gradient:

$$\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot A(\mathbf{u}_t, \mathbf{x}_t)$$

$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$$

replace by approximation

5. Repeat from 2



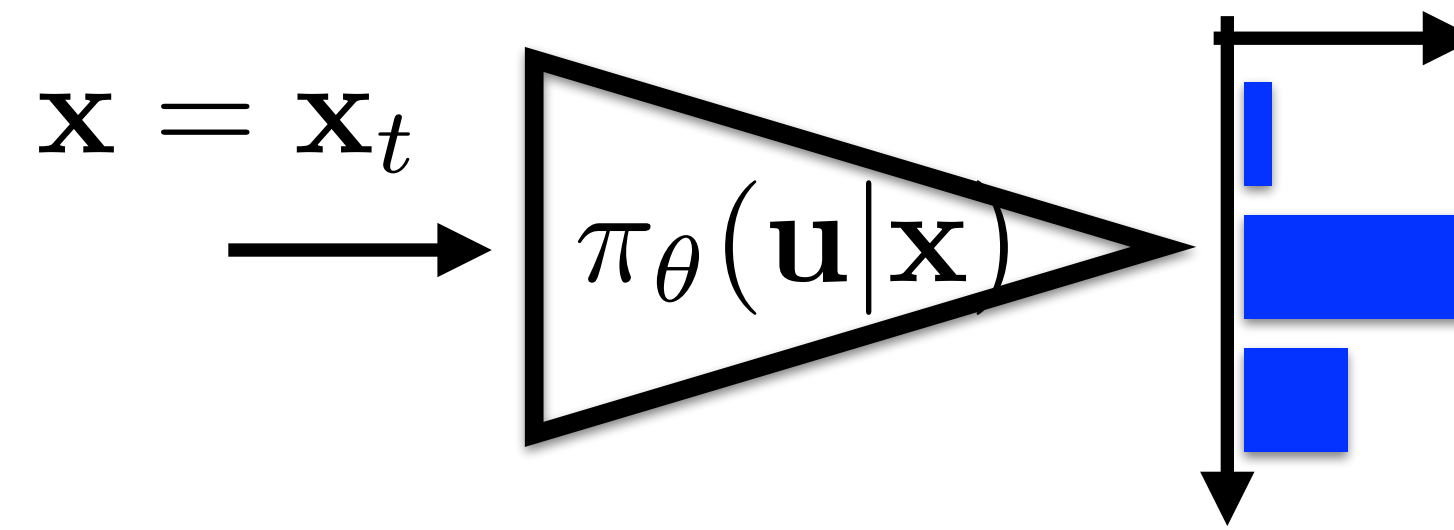
1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
2. Collect trajectories τ with policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
4. **Actor:** Update policy by policy gradient:

$$\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \underbrace{\left(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x}) \right)}_{A_\omega = Q - V}$$

$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$$

replace by approximation

5. Repeat from 2

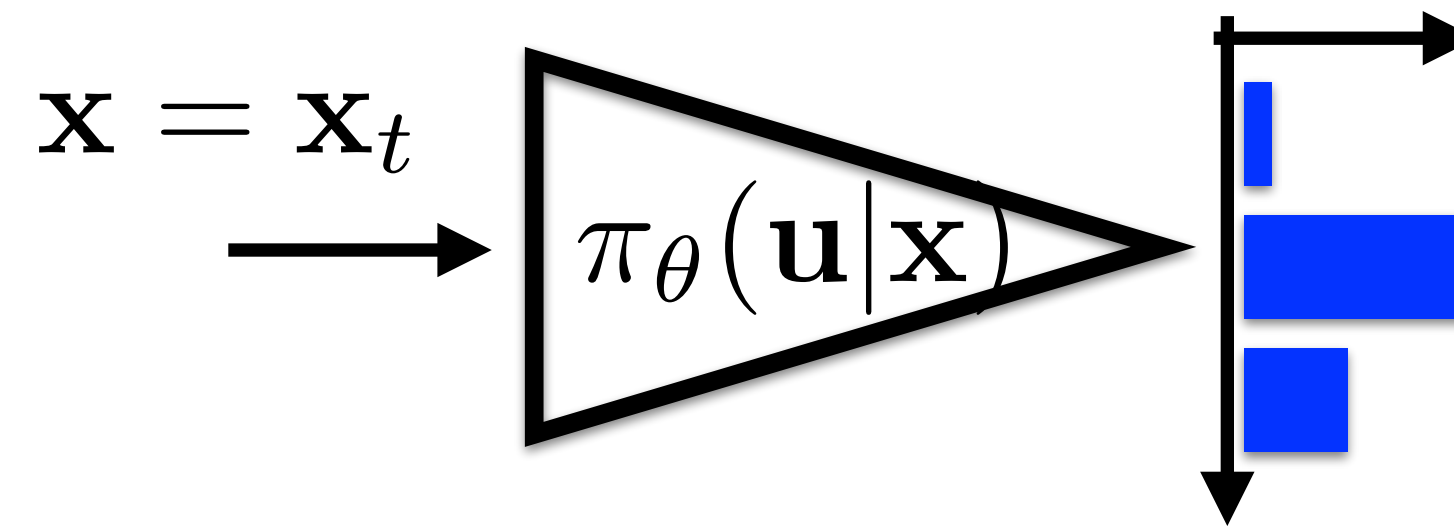


1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
2. Collect trajectories τ with policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
4. **Actor:** Update policy by policy gradient:
replace by criterion

$$\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \underbrace{\left(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x}) \right)}_{A_\omega = Q - V}$$

$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$$

5. Repeat from 2

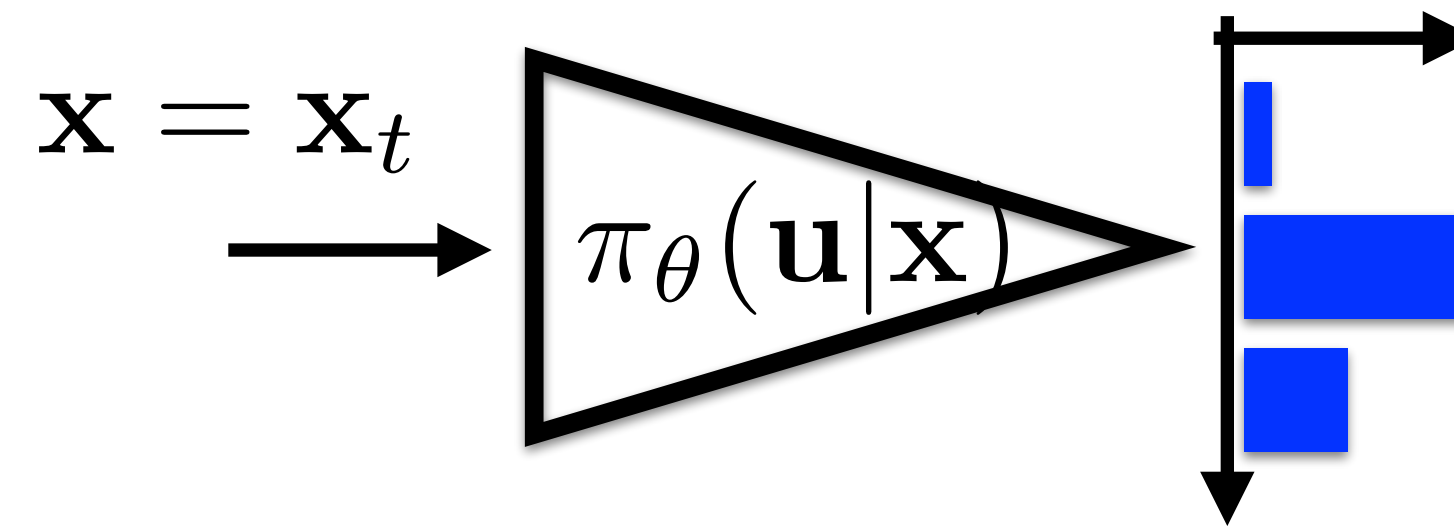


1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
2. Collect trajectories τ with policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
4. **Actor:** Update policy by policy gradient:

$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u}, \mathbf{x}, \mathbf{x}') \in \tau} \log \pi_\theta(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\left(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x}) \right)}_{A_\omega = Q - V}$$

$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$$

5. Repeat from 2



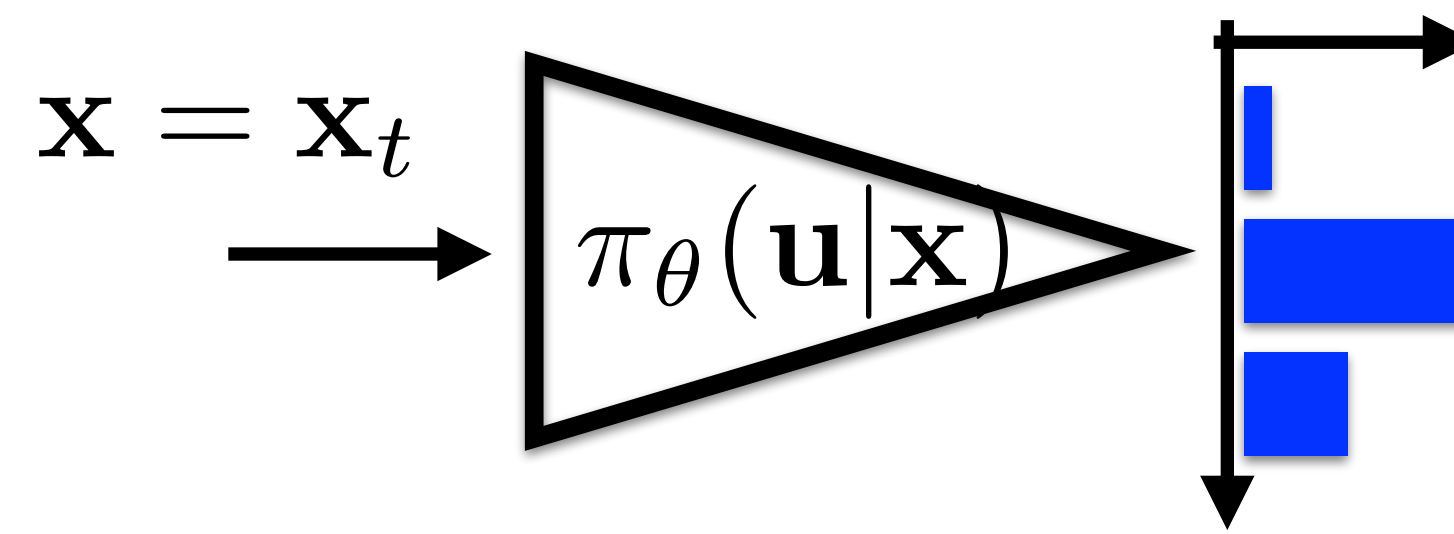
1. Initialize policy $\pi_{\theta}(\mathbf{u}|\mathbf{x})$, $V_{\omega}(\mathbf{x})$
2. Collect trajectories τ with policy $\pi_{\theta}(\mathbf{u}|\mathbf{x})$

4. **Actor:** Update policy by policy gradient:

$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u}, \mathbf{x}, \mathbf{x}') \in \tau} \log \pi_{\theta}(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\left(r + \gamma V_{\omega}(\mathbf{x}') - V_{\omega}(\mathbf{x}) \right)}_{A_{\omega} = Q - V}$$

$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$$

5. Repeat from 2



1. Initialize policy $\pi_{\theta}(\mathbf{u}|\mathbf{x})$, $V_{\omega}(\mathbf{x})$
2. Collect trajectories τ with policy $\pi_{\theta}(\mathbf{u}|\mathbf{x})$
3. **Critic:** Update value function to predict observed values: $V_{\omega}(\mathbf{x}) \leftarrow r + \gamma V_{\omega}(\mathbf{x}')$

$$\mathcal{L}_{\text{critic}}(\omega) = \left(\underbrace{r + \gamma V_{\omega}(\mathbf{x}') - V_{\omega}(\mathbf{x})}_{A_{\omega}} \right)^2$$

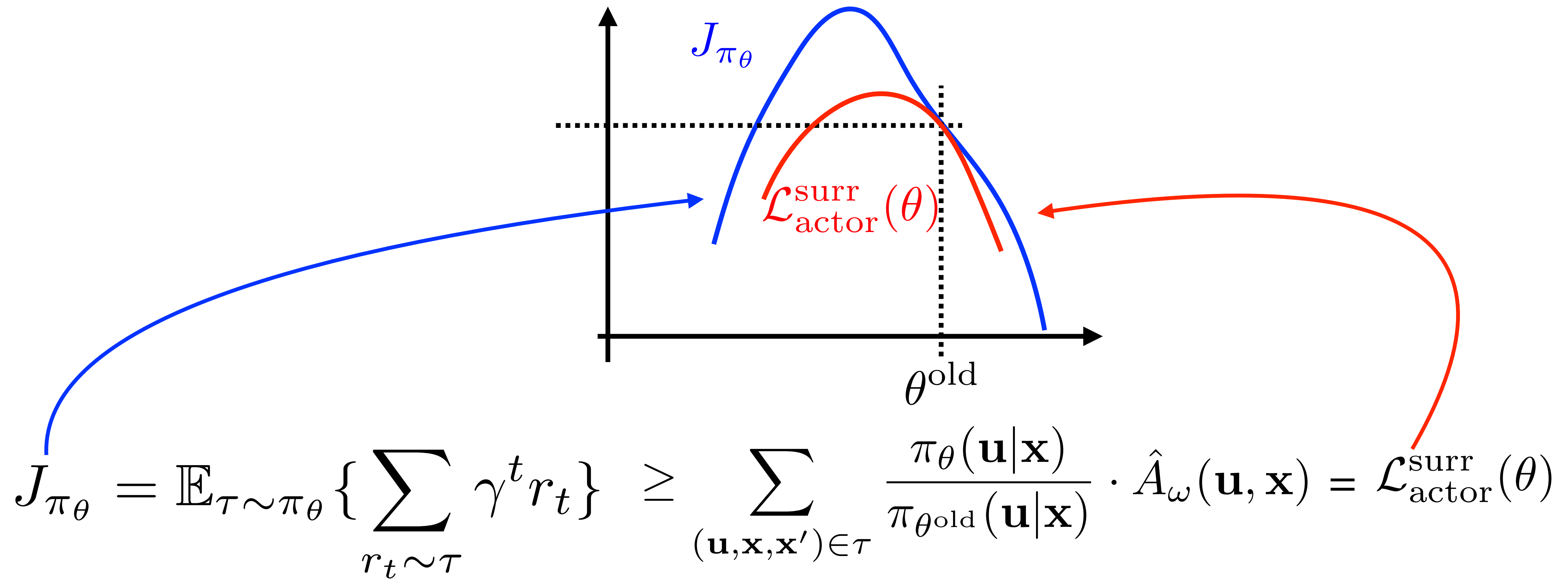
4. **Actor:** Update policy by policy gradient:

$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u}, \mathbf{x}, \mathbf{x}') \in \tau} \log \pi_{\theta}(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\left(r + \gamma V_{\omega}(\mathbf{x}') - V_{\omega}(\mathbf{x}) \right)}_{A_{\omega} = Q - V}$$

$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta} \quad \omega := \omega + \beta \frac{\partial \mathcal{L}_{\text{critic}}(\omega)}{\partial \omega}$$

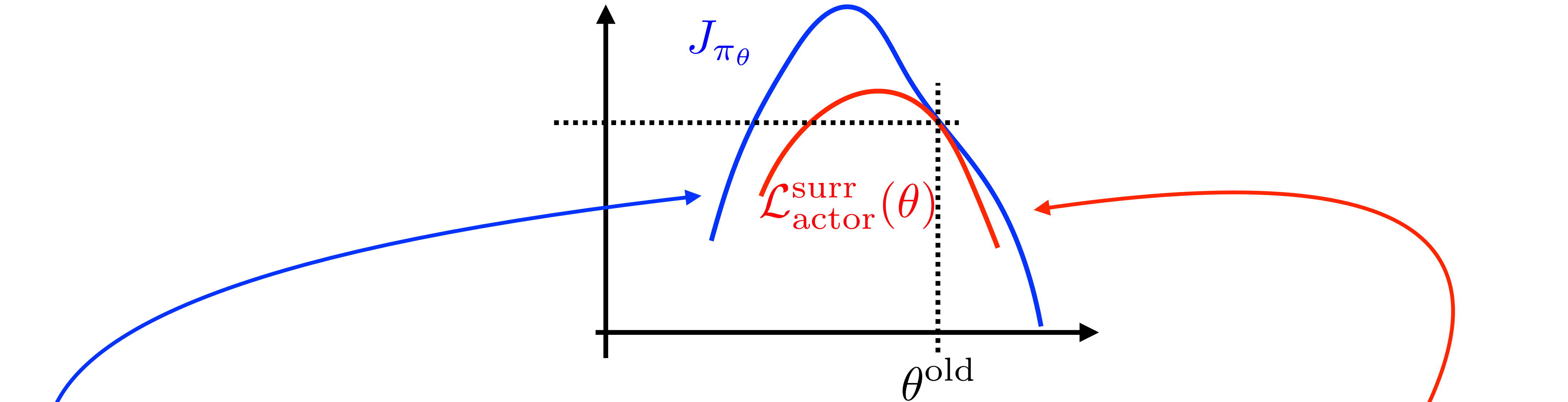
5. Repeat from 2

Proximal Policy Optimization PPO



Lower bound surrogate loss that **locally approximates** the true criterion

Trust Region Policy Optimization (TRPO)

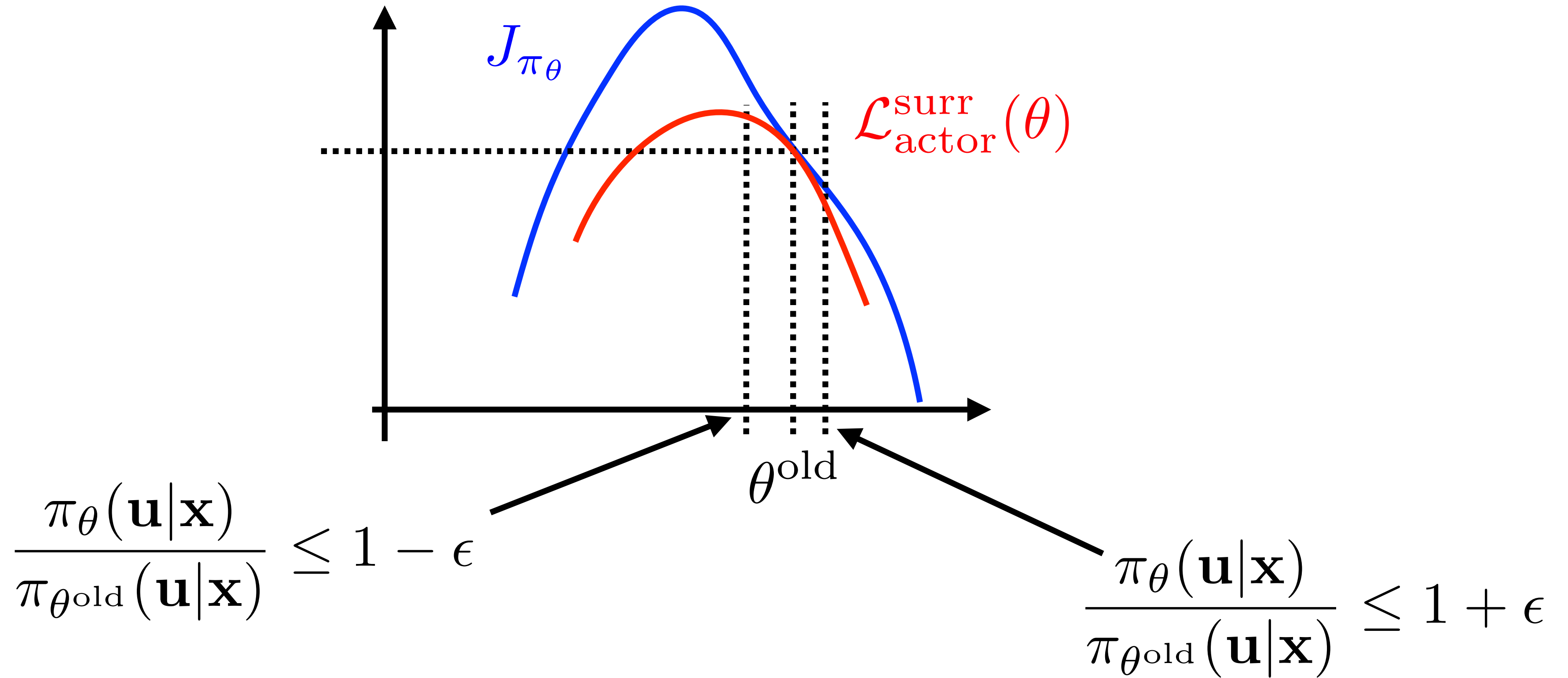


$$J_{\pi_{\theta}} = \mathbb{E}_{\tau \sim \pi_{\theta}} \left\{ \sum_{r_t \sim \tau} \gamma^t r_t \right\} \geq \sum_{(\mathbf{u}, \mathbf{x}, \mathbf{x}') \in \tau} \frac{\pi_{\theta}(\mathbf{u}|\mathbf{x})}{\pi_{\theta^{\text{old}}}(\mathbf{u}|\mathbf{x})} \cdot \hat{A}_{\omega}(\mathbf{u}, \mathbf{x}) = \mathcal{L}_{\text{actor}}^{\text{surr}}(\theta)$$

TRPO: $\arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\frac{\pi_{\theta}(\mathbf{u}|\mathbf{x})}{\pi_{\theta^{\text{old}}}(\mathbf{u}|\mathbf{x})} \cdot \hat{A}_{\omega}(\mathbf{u}, \mathbf{x}) \right]$

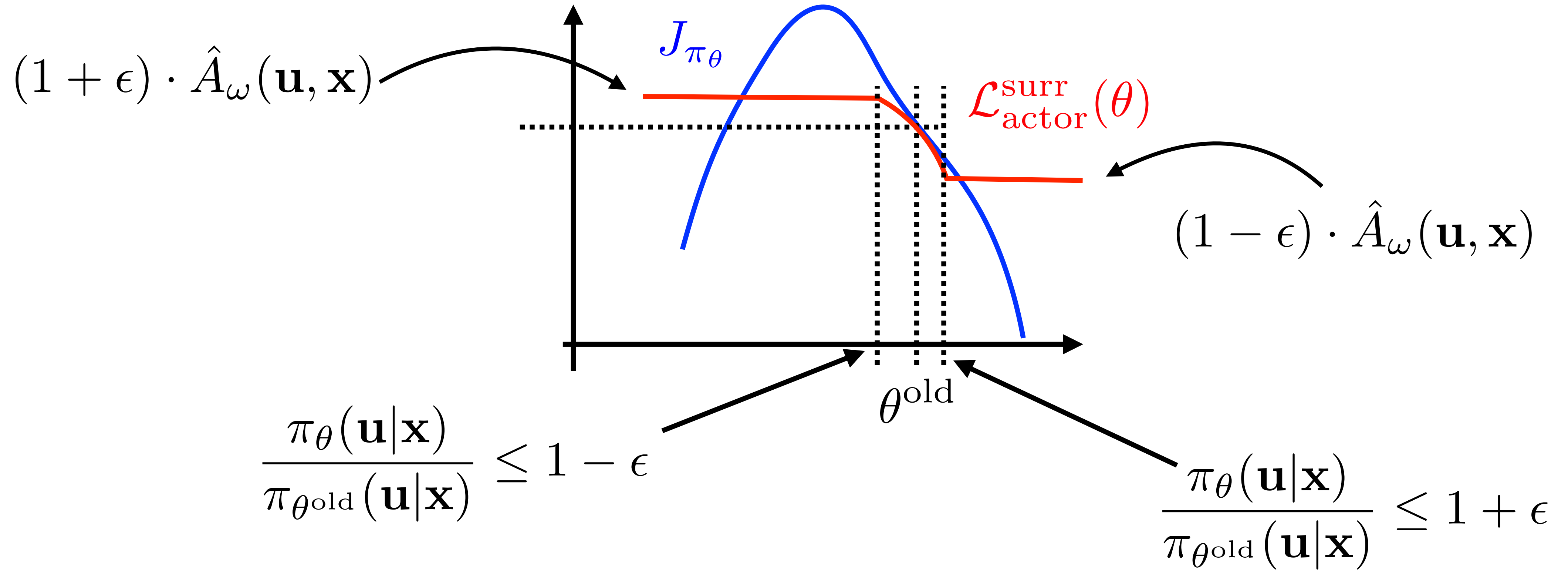
subject to: $\mathbb{E}_{\tau \sim \pi_{\theta}} \left[D_{\text{KL}}(\pi_{\theta}(\mathbf{u}|\mathbf{x}) \parallel \pi_{\theta^{\text{old}}}(\mathbf{u}|\mathbf{x})) \right] \leq \epsilon$

Proximal Policy Optimization PPO



TRPO: $\arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\frac{\pi_{\theta}(\mathbf{u}|\mathbf{x})}{\pi_{\theta^{\text{old}}}(\mathbf{u}|\mathbf{x})} \cdot \hat{A}_{\omega}(\mathbf{u}, \mathbf{x}) \right]$

Proximal Policy Optimization PPO



$$\arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\min \left\{ \frac{\pi_{\theta}(\mathbf{u}|\mathbf{x})}{\pi_{\theta^{\text{old}}}(\mathbf{u}|\mathbf{x})} \cdot \hat{A}_{\omega}(\mathbf{u}, \mathbf{x}), \quad \text{clip}\left(\frac{\pi_{\theta}(\mathbf{u}|\mathbf{x})}{\pi_{\theta^{\text{old}}}(\mathbf{u}|\mathbf{x})}, 1 - \epsilon, 1 + \epsilon\right) \cdot \hat{A}_{\omega}(\mathbf{u}, \mathbf{x}) \right\} \right]$$

Reinforcement learning baselines

<https://gym.openai.com/>

```
import gym
```

```
env = gym.make('CartPole-v1')
```

```
obs = env.reset()
```

```
for i in range(1000):
```

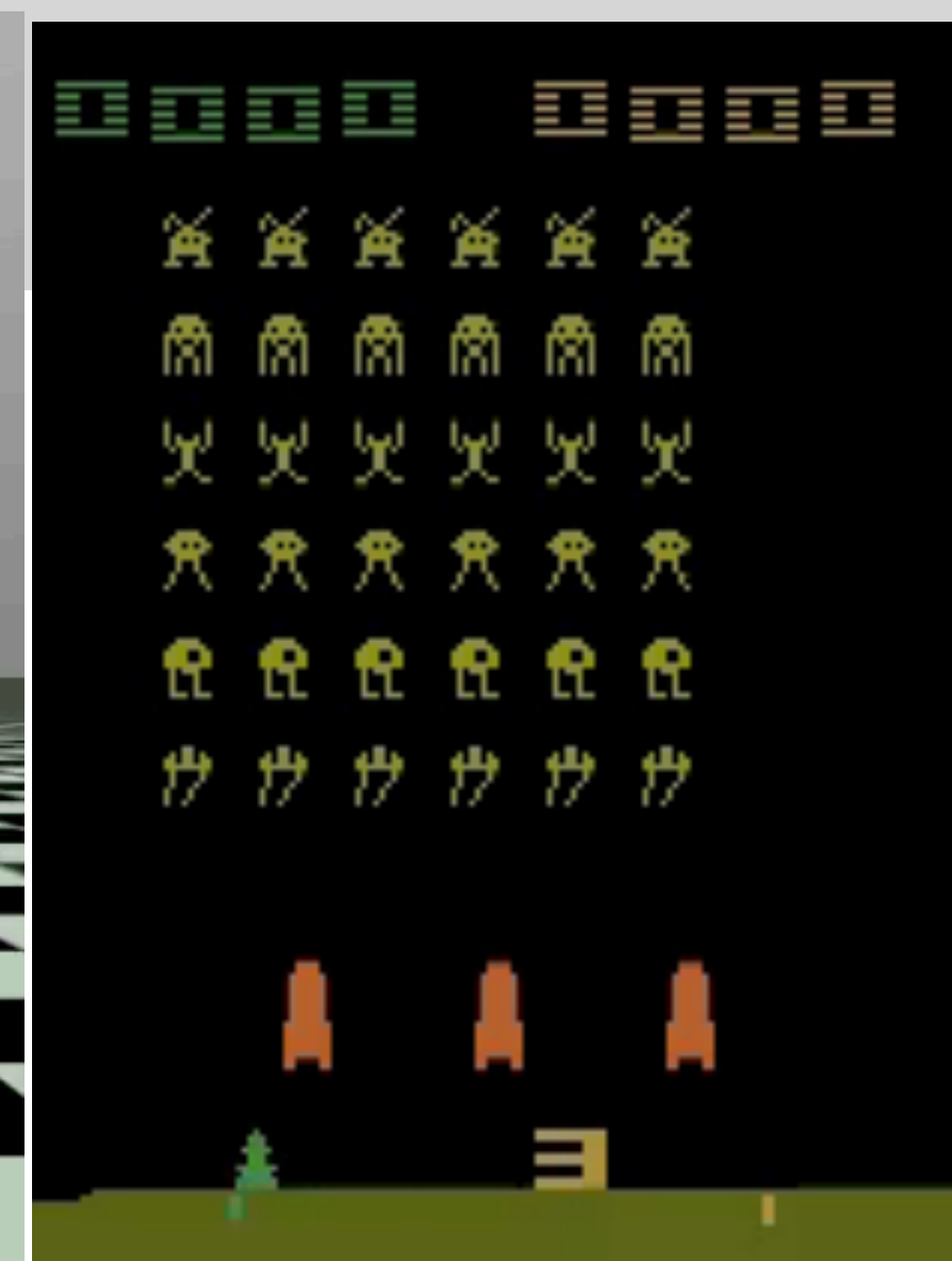
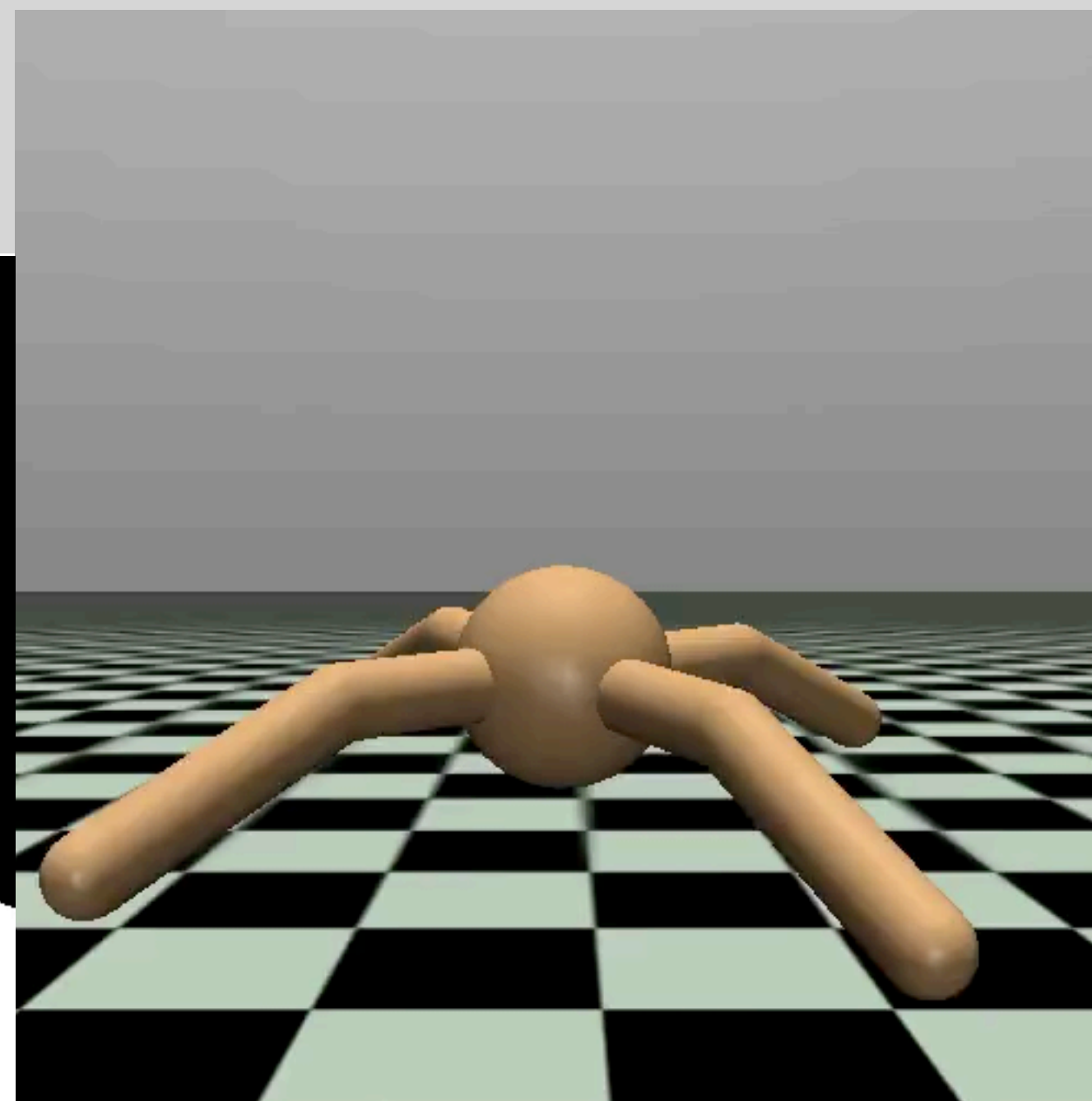
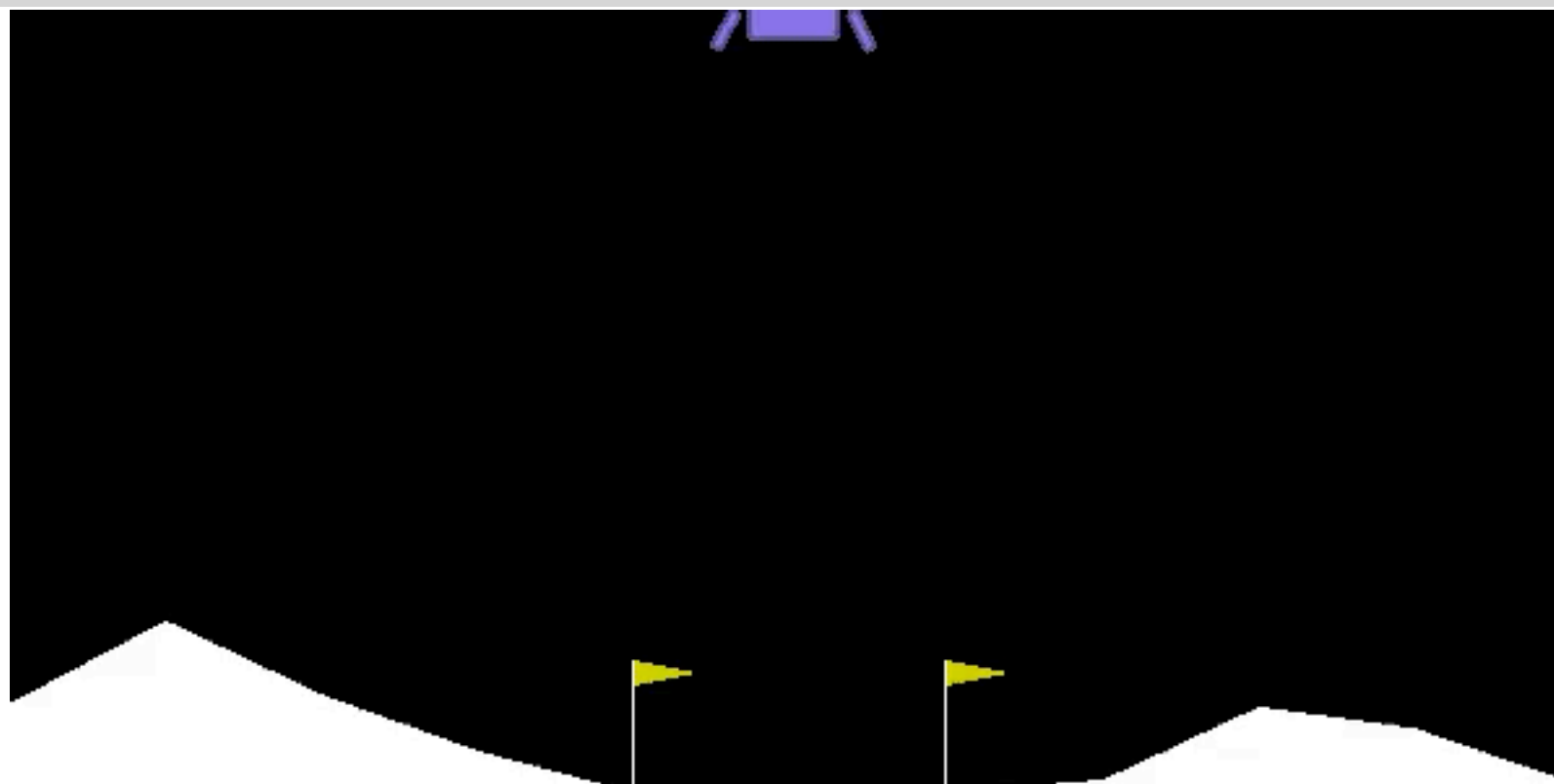
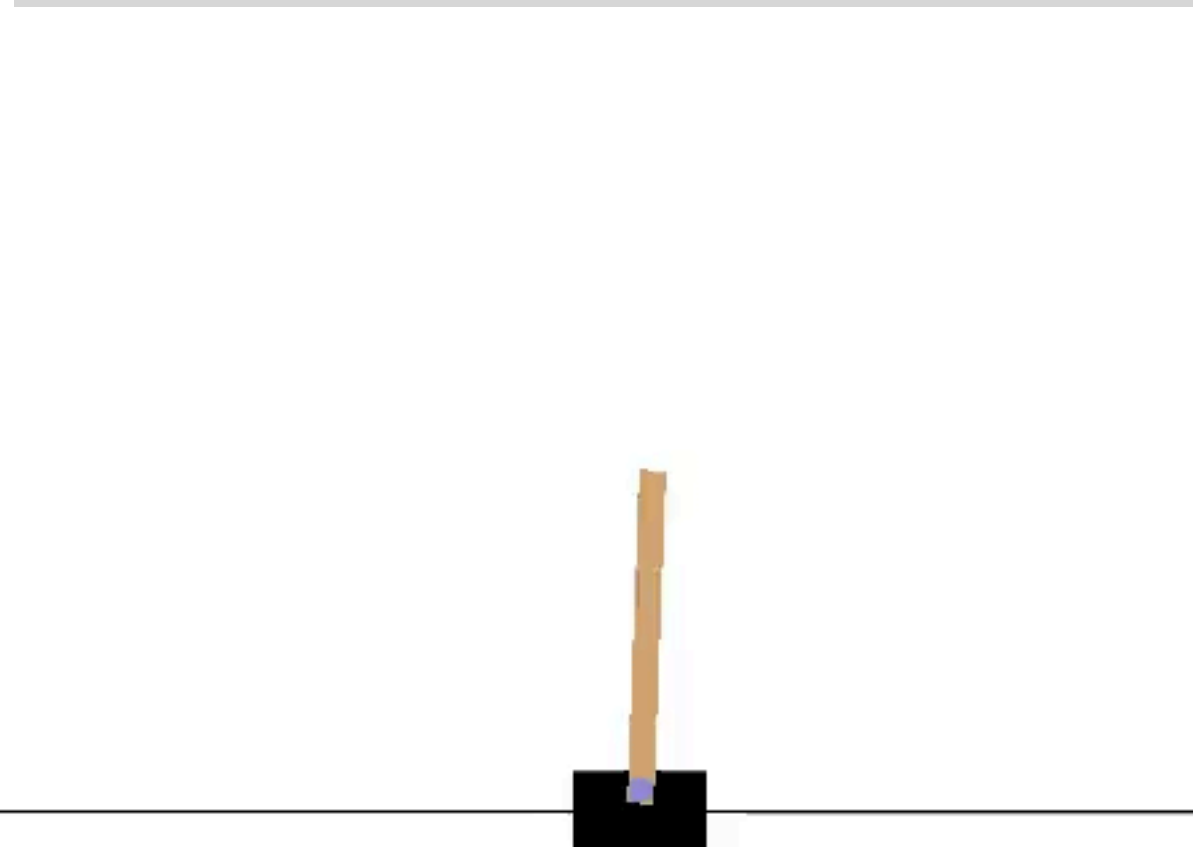
```
    action, _state = model.predict(obs, deterministic=True)
```

```
    obs, reward, done, info = env.step(action)
```

```
    env.render()
```

```
    if done:
```

```
        obs = env.reset()
```



Reinforcement learning baselines

<https://stable-baselines3.readthedocs.io/>

```
import gym

from stable_baselines3 import A2C

env = gym.make('CartPole-v1')

model = A2C('MlpPolicy', env, verbose=1)
model.learn(total_timesteps=10000)
```


Known successes of RL

- Computer games controlled from pixel inputs
 - Starcraft II (AlphaStar)
 - Atari 2D platformers (DQN)
 - Doom 2 - VizDoom [Wydemuch 2018]
<https://arxiv.org/abs/1809.03470>
 - Quake III - Arena capture the flag
 - DOTA 2 openAI+ bot <https://blog.openai.com/dota-2/>

Known successes of RL - Starcraft II

- Starcraft II (Deepmind AlphaStar beaten top-end professional human gamers 5:0)



- no single best strategy
- partially observable
- longterm planning (delayed rewards from upgrades)
- realtime
- large action space
- human gameplay dataset
- more-than-realtime implementation of the game for RL

<https://medium.com/mlmemoirs/deepminds-ai-alphastar-showcases-significant-progress-towards-agi-93810c94fbe9>

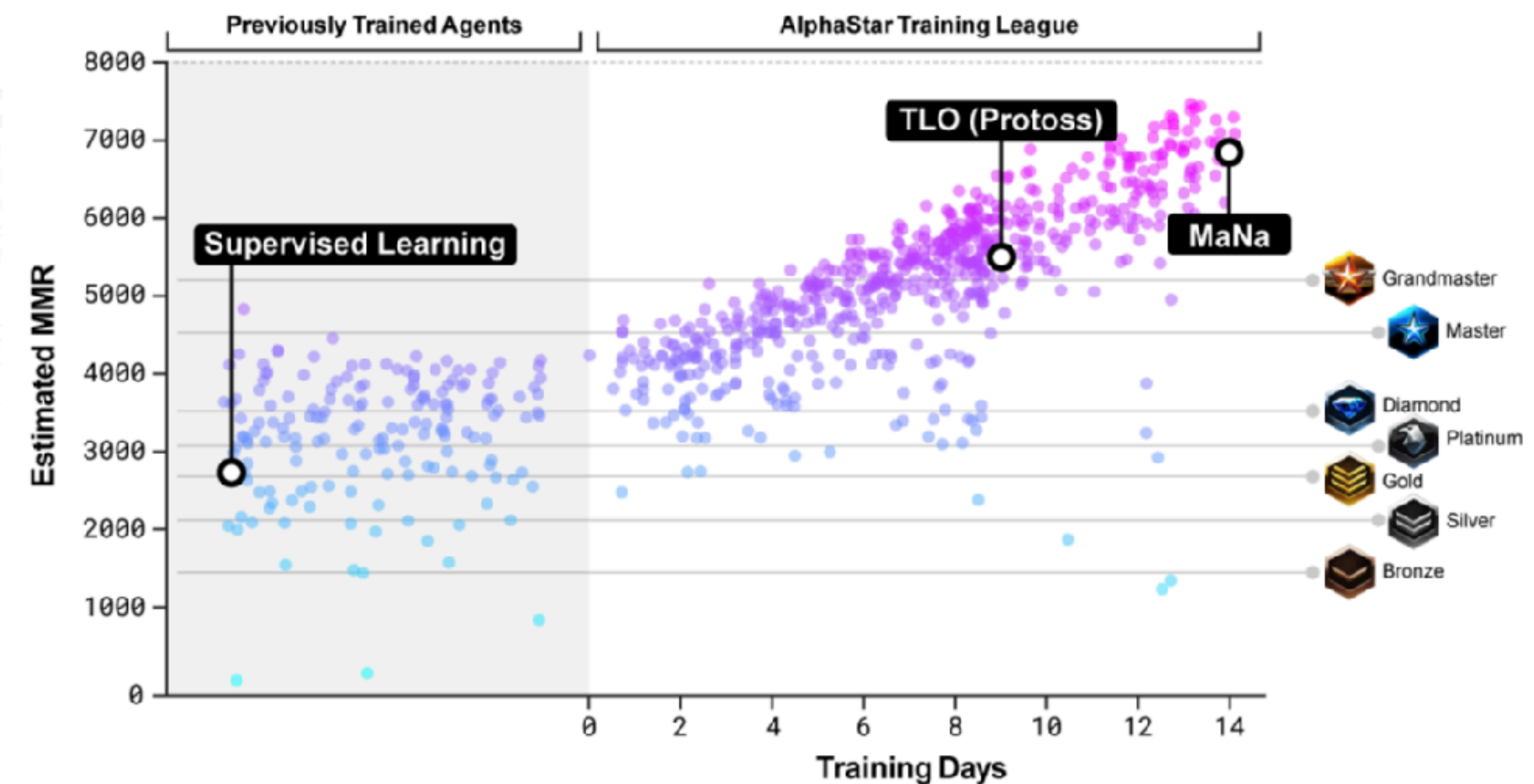
Known successes of RL - Starcraft II

Minigames allows for training small RL agents



Learning Starcraft II

- **Supervised learning** from human games dataset
(performance: (i) humans - gold level, (ii) AI - elite level)
- **Reinforcement learning:**
 - minigames (collect minerals, build marines, ...)
 - self-play
 - 14 days playing against two grand masters (TLO, MaNa)



Known successes of RL - Starcraft II

- **Reinforcement learning:** 14 days playing against two grand masters (TLO, MaNa)
 - Distributed Actor-Critic method IMPALA:
<https://arxiv.org/pdf/1802.01561.pdf>
 - $TD(\lambda)$ learning of $Q_\theta(\mathbf{x}, \mathbf{u})$
 - stochastic policy gradient:

$$\mathbb{E}_{\tau \sim p(\tau | \pi_\omega)} \left[\sum_{(\mathbf{x}, \mathbf{u}) \in \tau} \frac{\partial \log \pi_\omega(\mathbf{u} | \mathbf{x})}{\partial \omega} \cdot Q_\theta(\mathbf{x}, \mathbf{u}) \right] \approx$$
$$\approx \sum_k \frac{\partial \log \pi_\omega(\mathbf{u}_k | \mathbf{x}_k)}{\partial \omega} \cdot Q_\theta(\mathbf{x}_k, \mathbf{u}_k)$$

- recurrent policy architecture with LSTM blocks
- parallelized learning

Results:



TLO

ROUND

1. ALPHASTAR WINS

2. ALPHASTAR WINS

3. ALPHASTAR WINS

4. ALPHASTAR WINS

5. ALPHASTAR WINS

SCORE TLO 0 - 5 ALPHASTAR



GRZEGORZ 'MANA' KOMINCZ

ROUND

1. ALPHASTAR WINS

2. ALPHASTAR WINS

3. ALPHASTAR WINS

4. ALPHASTAR WINS

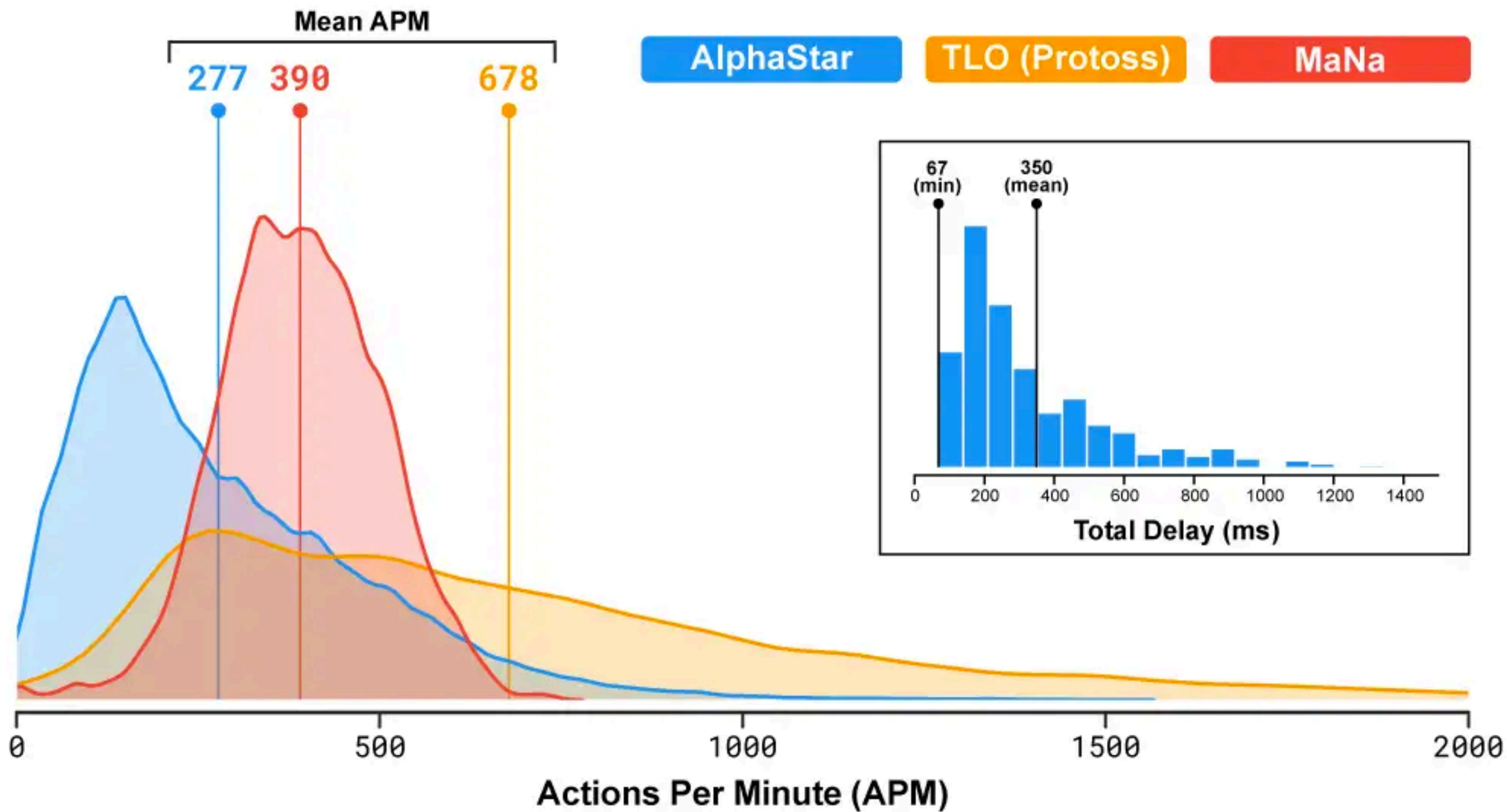
5. ALPHASTAR WINS

SCORE MANA 0 - 5 ALPHASTAR

Suggested explanations:

- higher APM
- smaller delays in counter-actions
- does not have to move camera
- fog of war

- ~~• higher APM~~

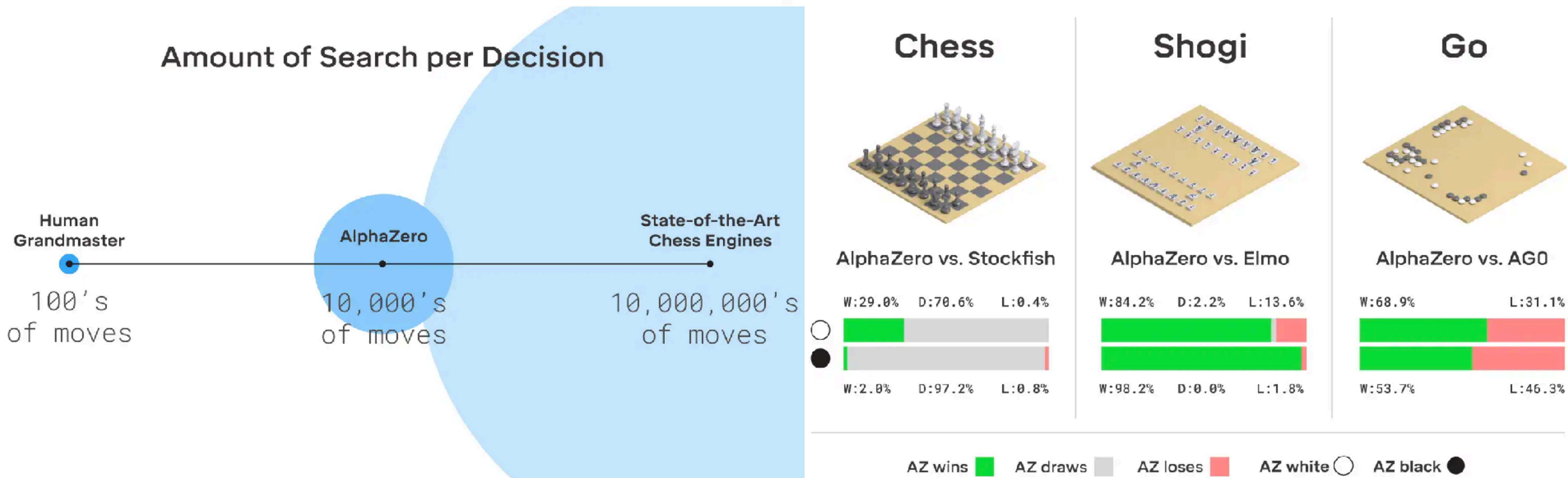


- ~~moving interface~~
- Of course, haze of war is used.



Known successes of RL - board games

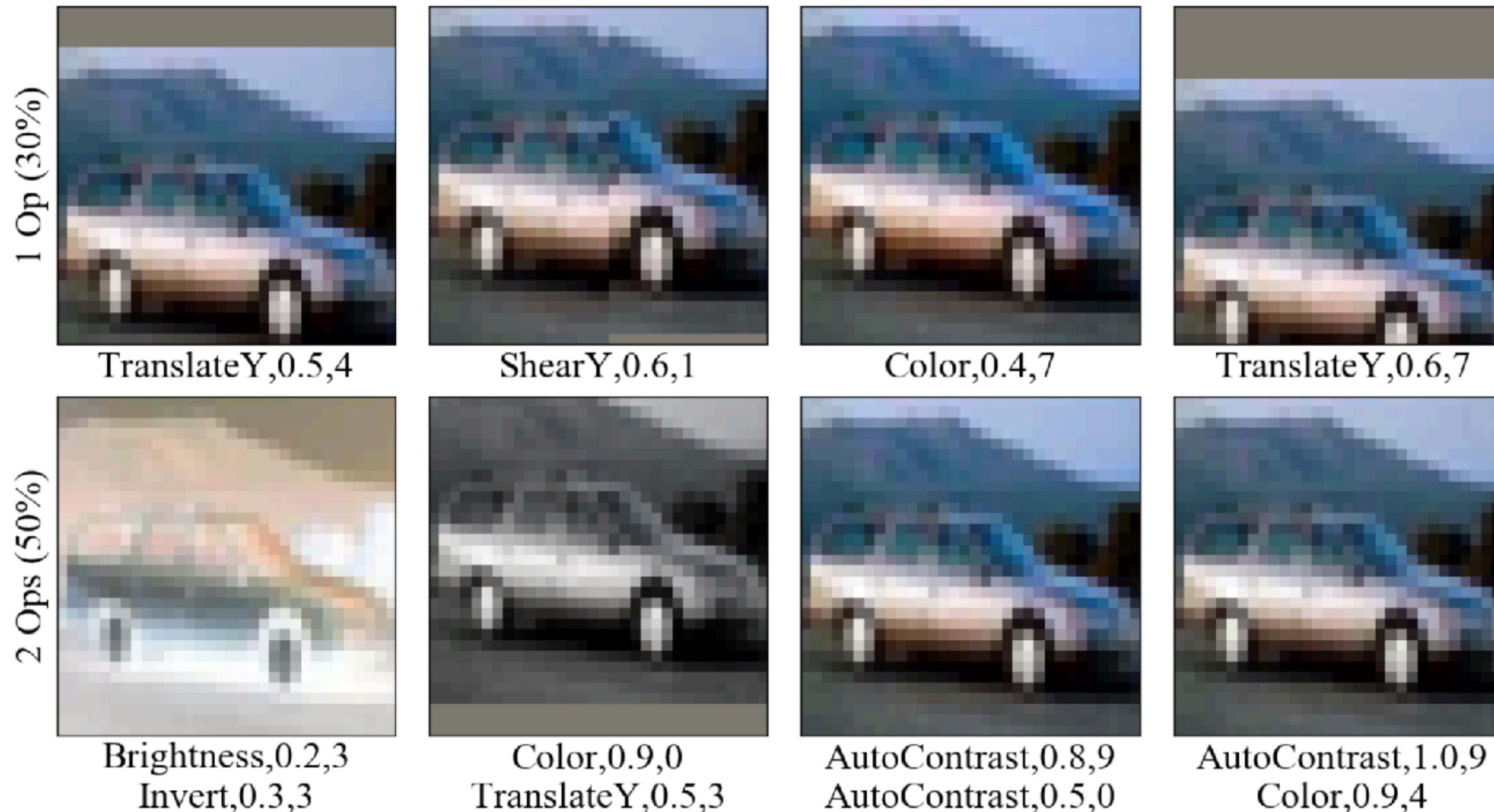
- Brute-force-search-based algorithms has no chance in huge state-action spaces
=> trained net guides the search efficiently



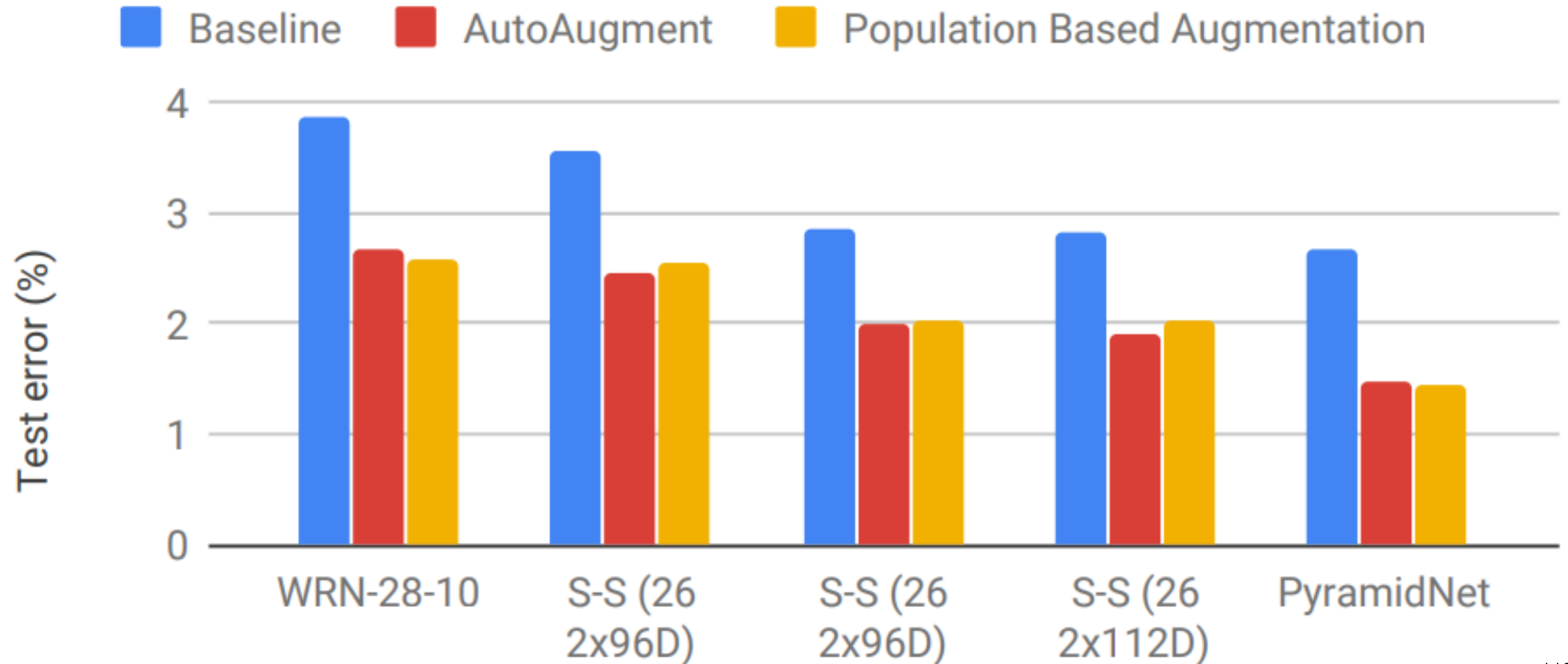
Known successes of RL

Learning to learn

- Training set augmentation
(jittering, mirroring, occlusions, brightness/contrast/color variations)
- Learn augmentation policy (AutoAugment, PBA), which provides good generalization

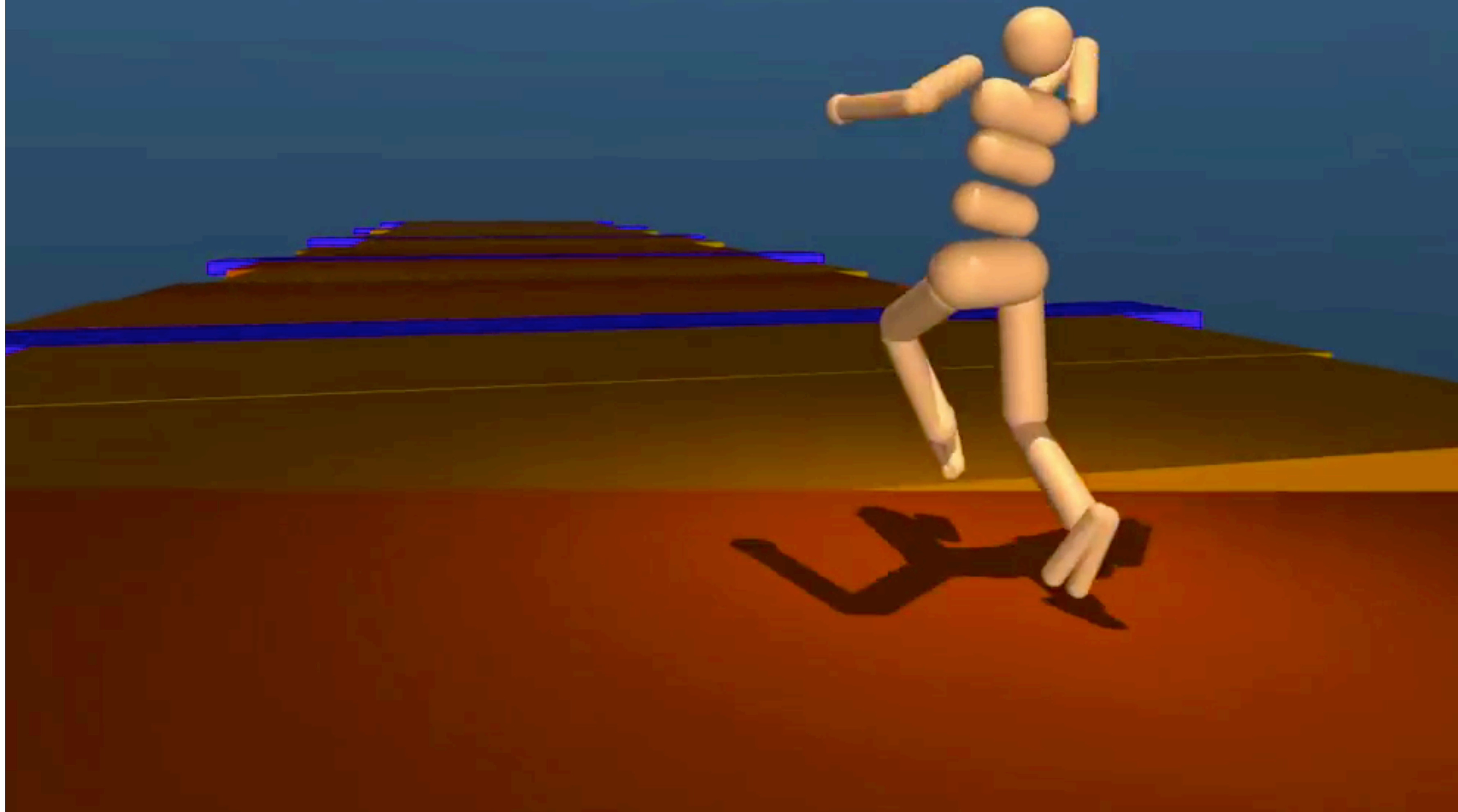


Known successes of RL - Learning to learn



Known successes of RL - learning complex motions in simulation
[Heess 2017] <https://arxiv.org/abs/1707.02286>

This agent, trained on several terrain types, has never seen the "see-saw" terrain.



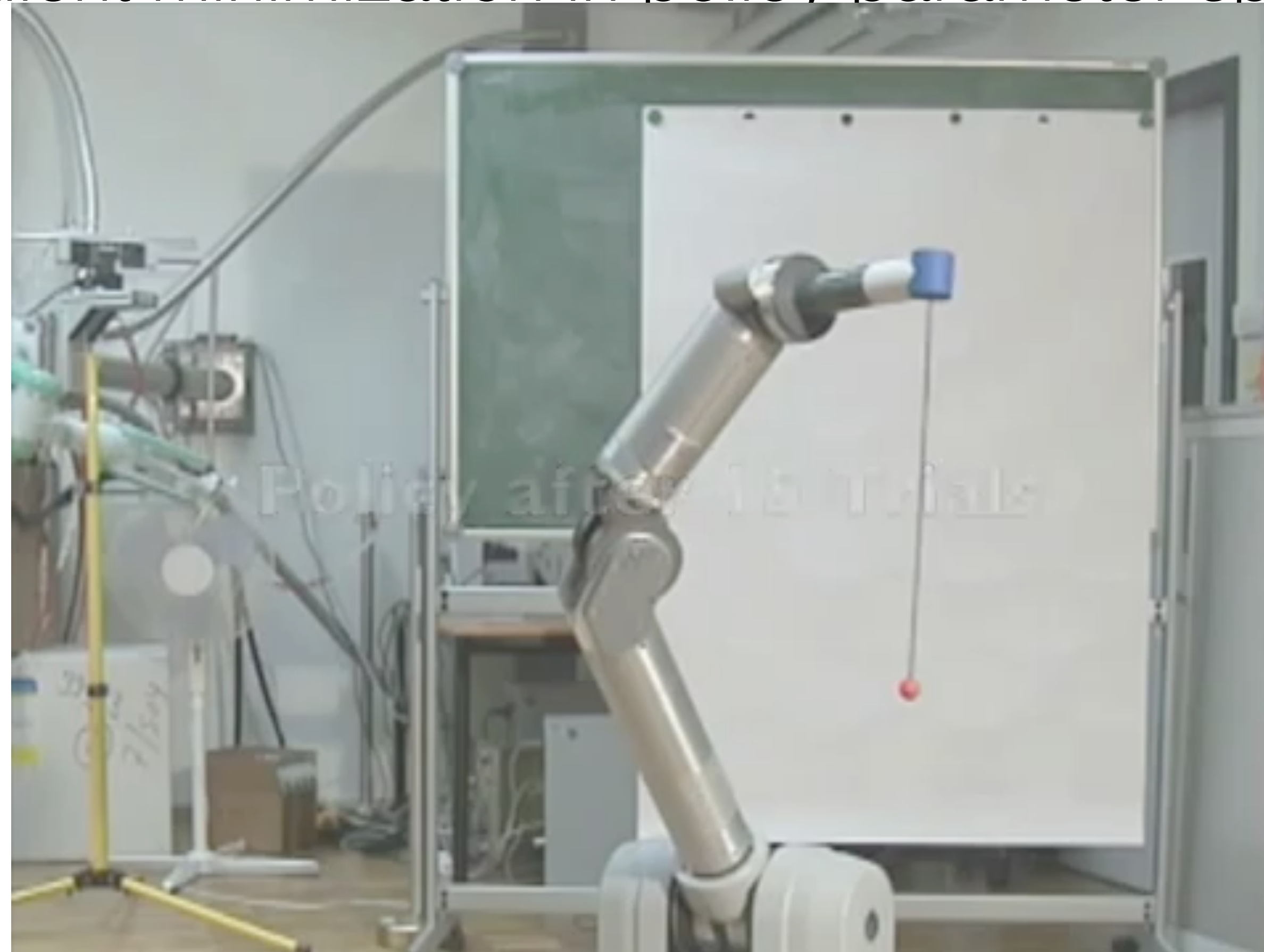
Source: Peter Pastor



Known successes of RL

learning complex motions in reality by manually designing low-dim policy

- imitation learning from human demonstration
- **state space:** joint+ball positions, velocities, acceler.
- **action space:** motor torques
- gradient minimization in policy parameter space



Peters et al. NOW 2013

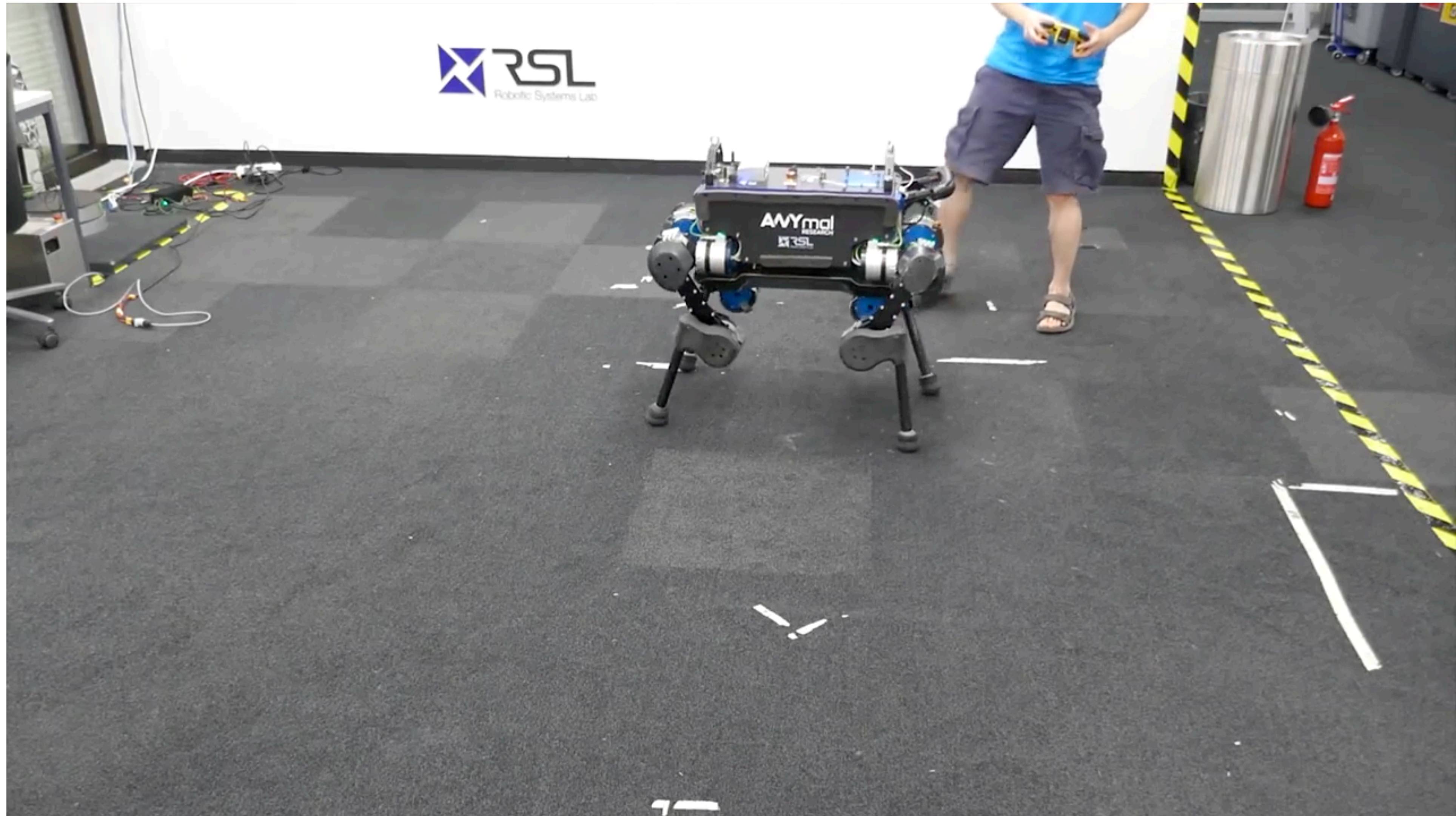
Motion and compliance control of flippers



[3] Pecka, Zimmermann, Svoboda, et al. **IROS/RAL/TIE(IF=6)**, 2015-2018

Known successes of RL

learning complex motions in reality by transferring policy from simulation
No visual inputs + flat terrain => simple domain transfer



[Hwangbo, ETH Zurich, Science Robotics, 2018]

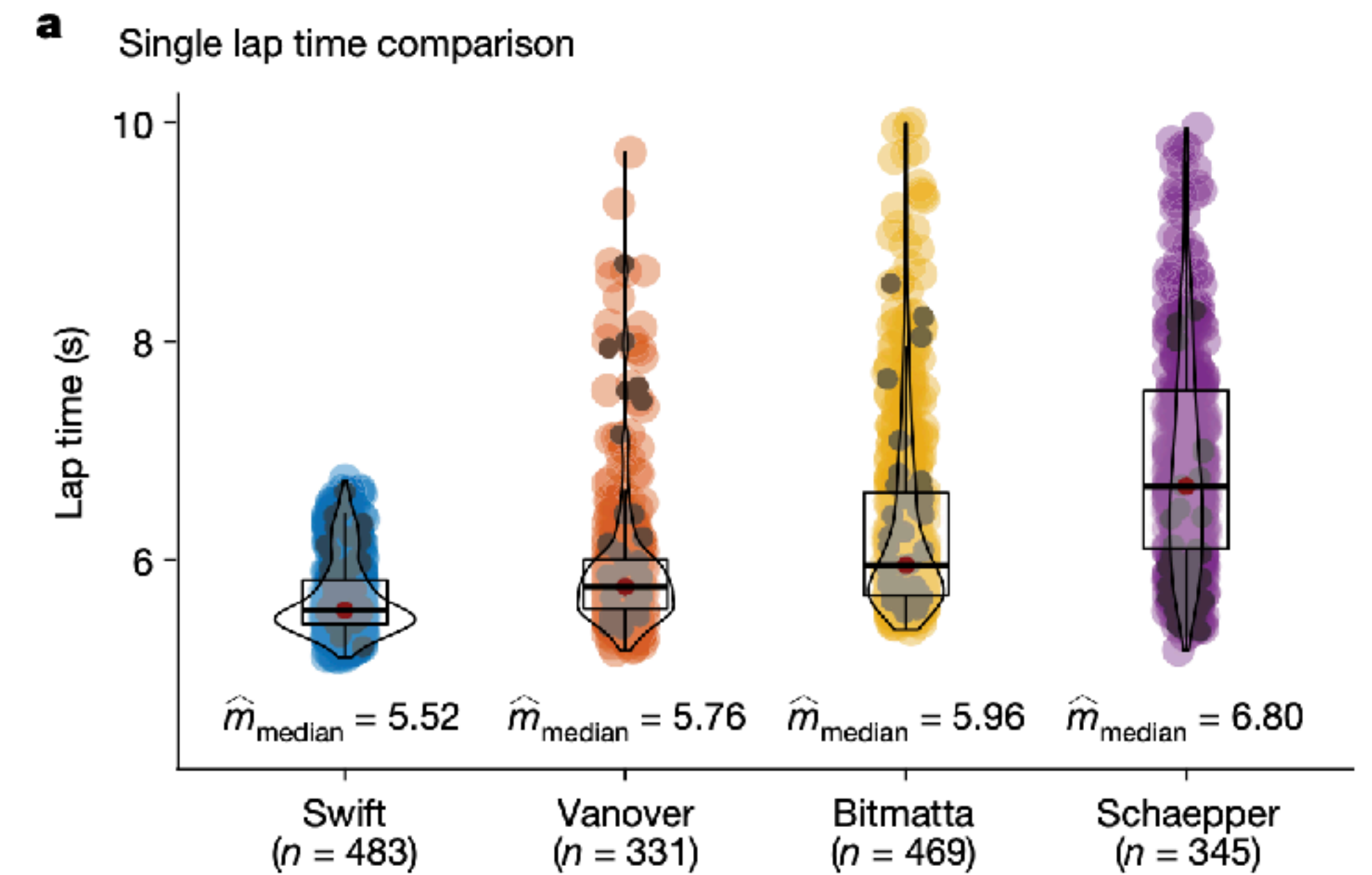
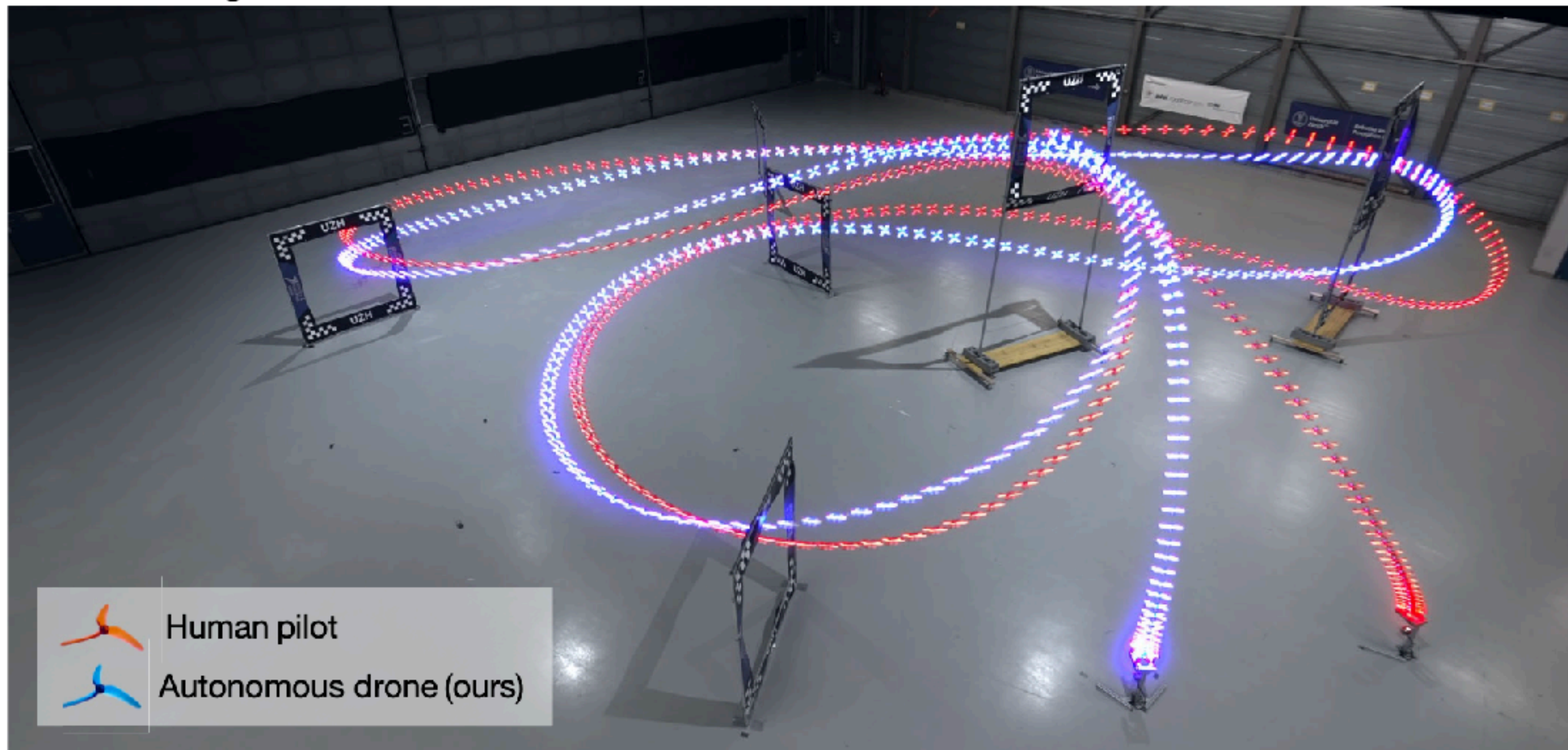
[Kumar 2020] Rapid Motor Adaptation for legged robot
<https://ashish-kmr.github.io/rma-legged-robots/>



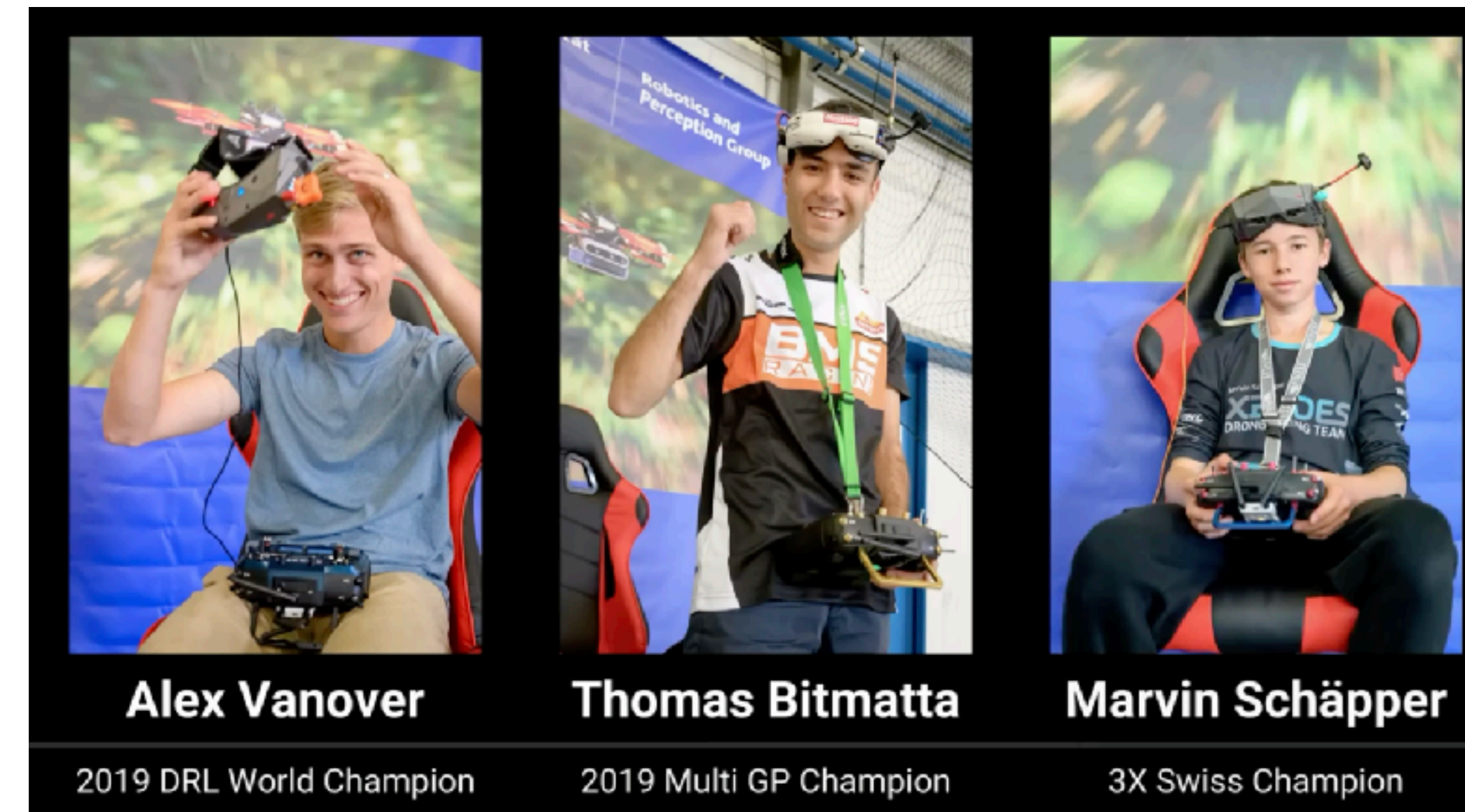
Rocky area next to river bed

[Kaufman Nature 2023] Champion-level drone racing using deep RL

a Drone racing: human versus autonomous



- RL runs in simulation
- Method = PPO
- Simulation+residual model



[Kaufman Nature 2023] Champion-level drone racing using deep RL



Boston dynamics - Big dog - NO RL AT ALL



Typical problems

τ
→

Model identification:

- given some trajectories
estimate model

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→

$p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Model predictive control / Planning

- given the model and reward
estimate optimal policy/plan

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ
→
 $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Reinforcement learning:

- given rewards and trajectories,
estimate optimal policy

$\pi^* = \arg \max_{\pi} J_{\pi}$
→

τ^*
→

Inverse reinforcement learning:

- given optimal trajectories
estimate reward function

$r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$
→

Rewards engineering

- Designing **RL-friendly** rewards is **hard**
- **Sparse** rewards are **easier to design** correctly (winning the race)
- **Dense** rewards are **easier to learn** (current speed)



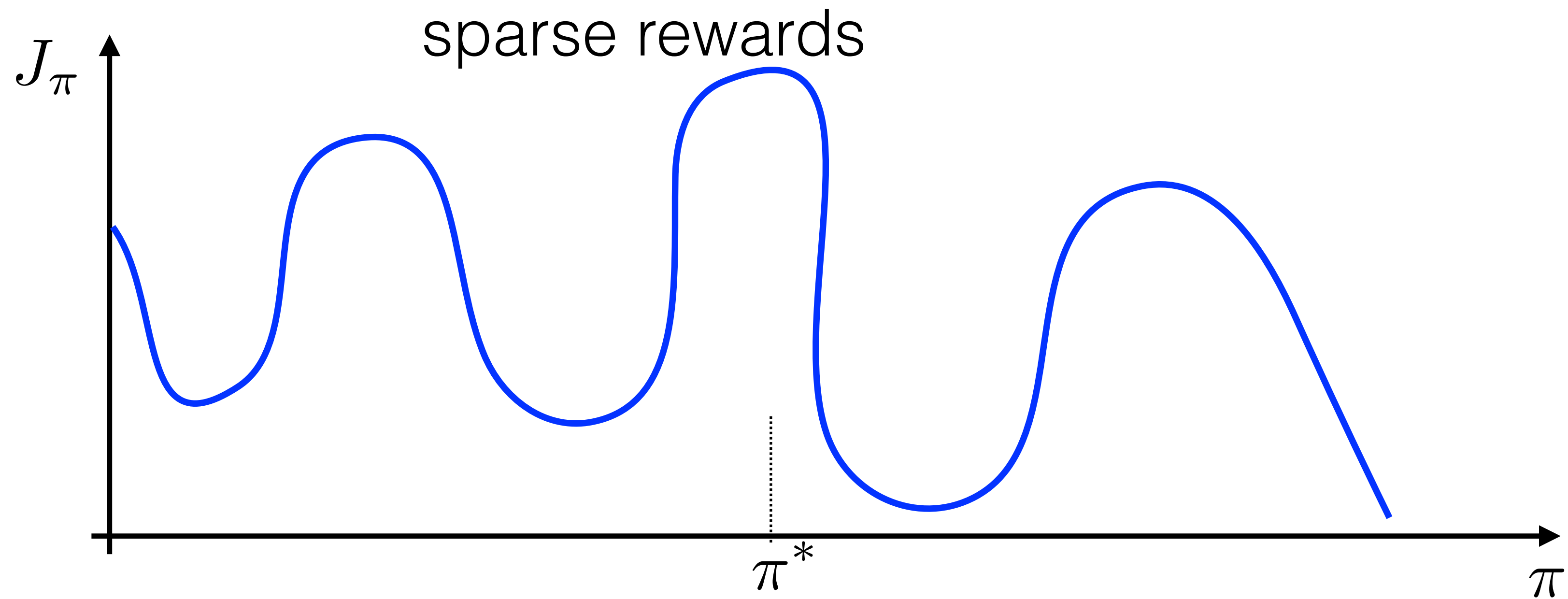
Rewards engineering

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn



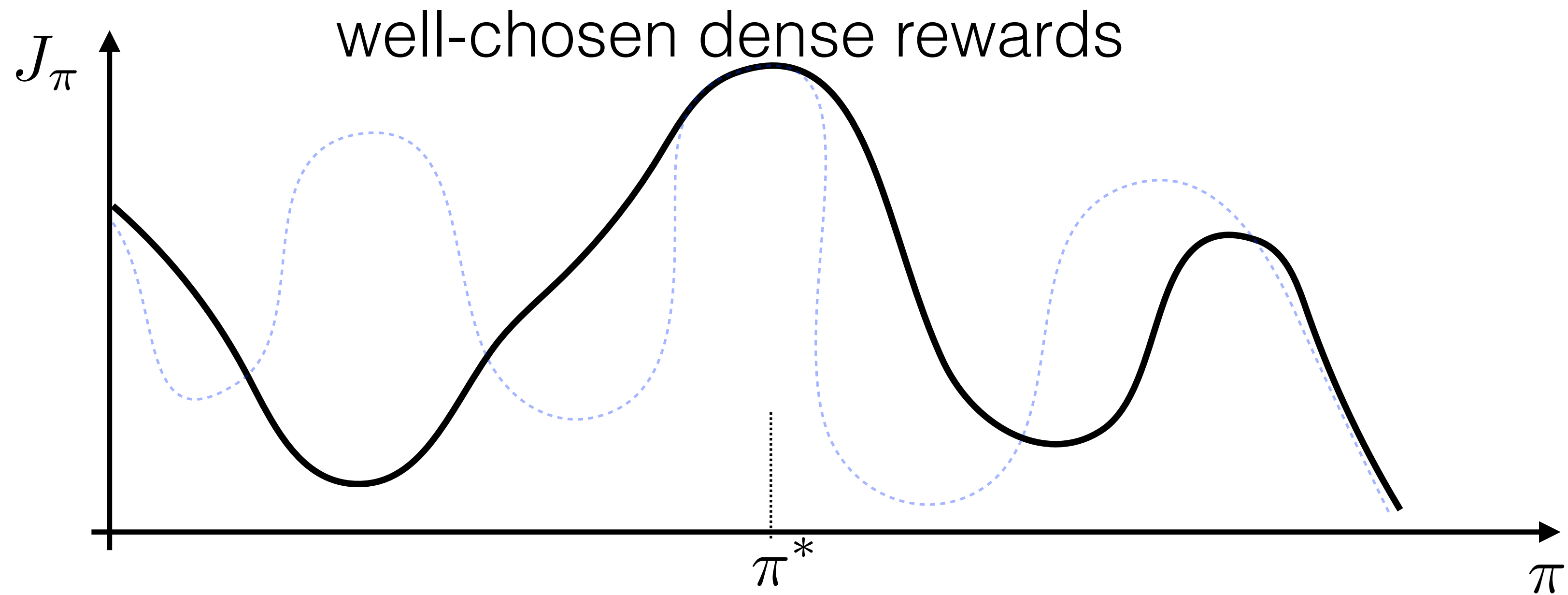
Rewards engineering

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn



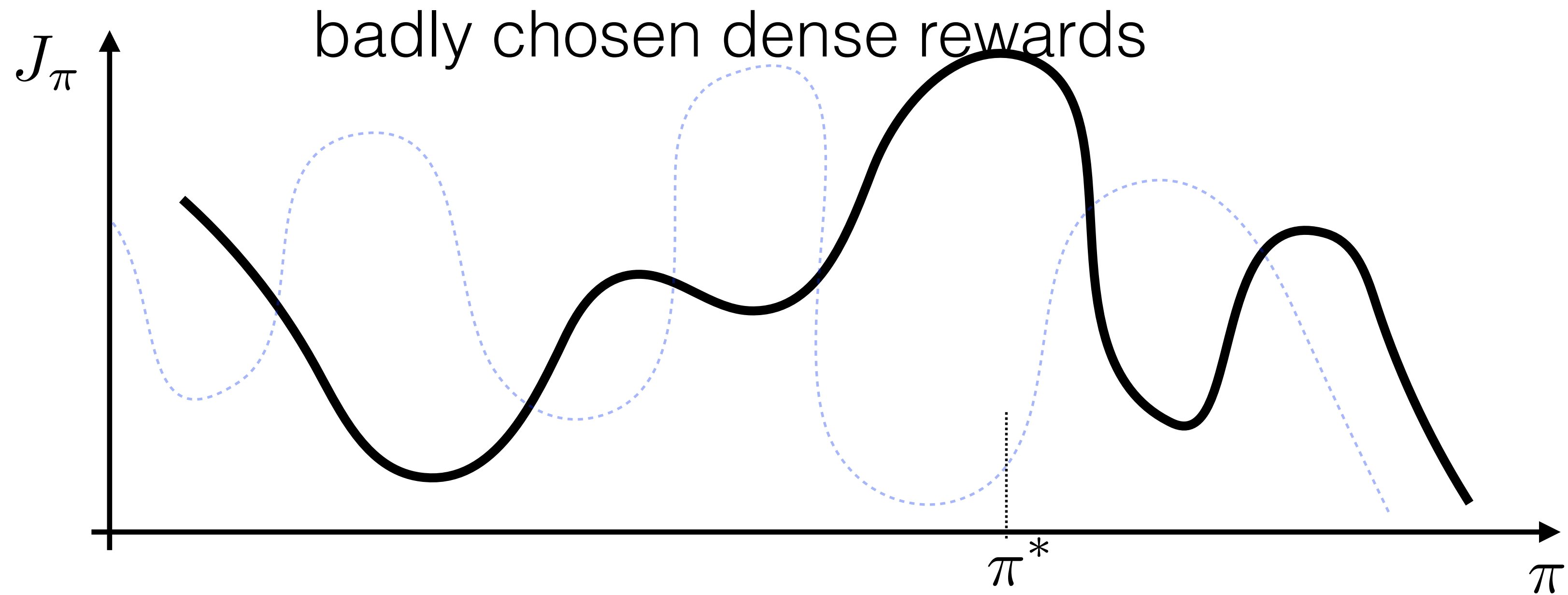
Rewards engineering

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn



Rewards engineering

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn

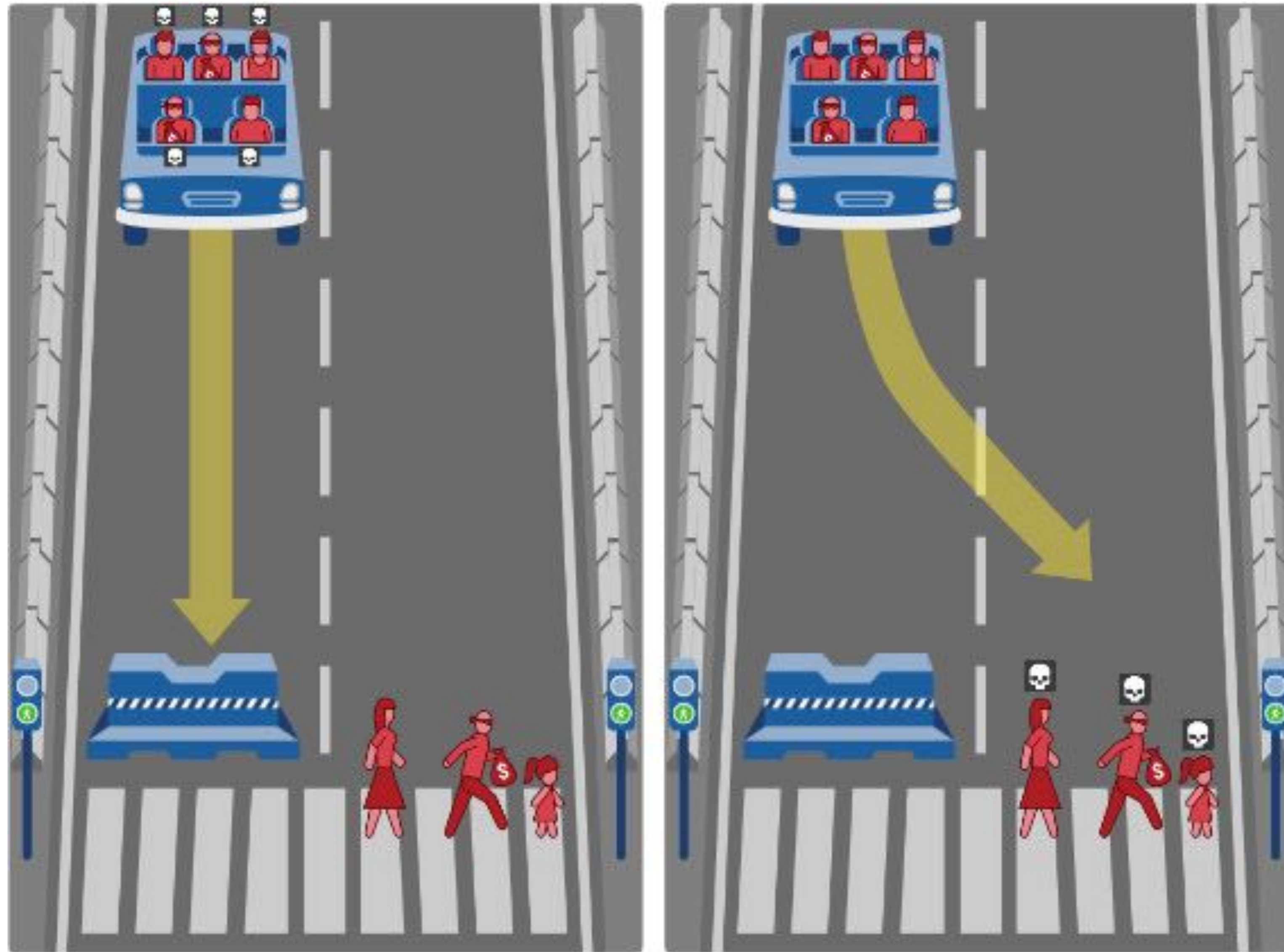


Rewards engineering

- Boat racing (bad dense rewards):
 - sparse rewards (winning the race)
 - dense rewards (collecting powerups, checkpoints ...)



Trolley problem



<https://www.nature.com/articles/s41586-018-0637-6>
[Moral Machine Experiment, Nature, 2018]

Trolley problem
estimated preference (normalized rewards) for life saving

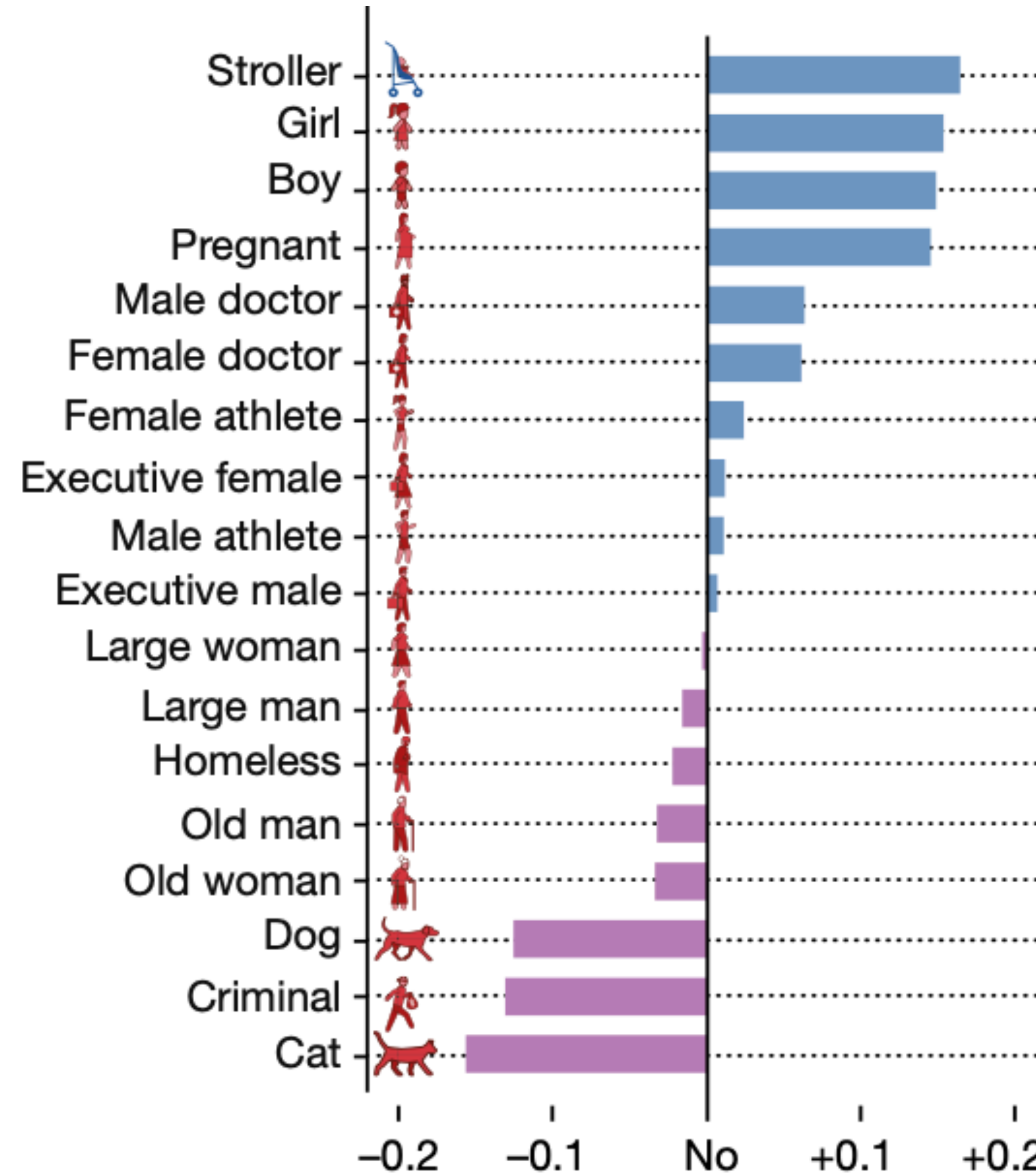
Male, Female, Cat, Dog, Criminal, Boy, Girl, Stroller, Homeless, ...

Guess your own preferences !

<https://www.nature.com/articles/s41586-018-0637-6>
[Moral Machine Experiment, Nature, 2018]

Trolley problem

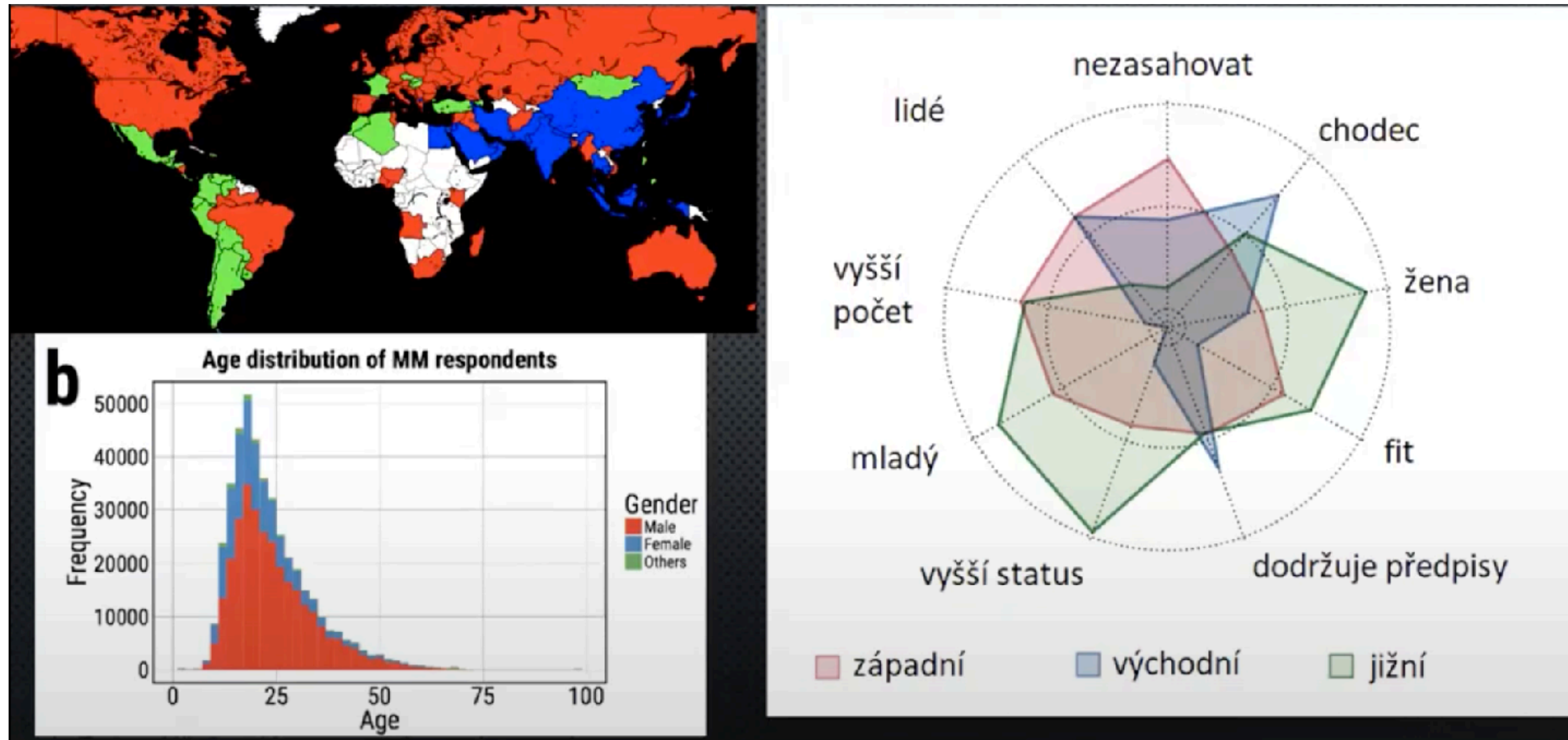
estimated preference (normalized rewards) for life saving



<https://www.nature.com/articles/s41586-018-0637-6>
[Moral Machine Experiment, Nature, 2018]

Trolley problem

spatial distribution of life-saving preferences



<https://www.moralmachine.net>

<https://www.nature.com/articles/s41586-018-0637-6>
[Moral Machine Experiment, Nature, 2018]

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.



Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find policy $\arg \min_{\theta} \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_{\theta}(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find policy $\arg \min_{\theta} \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_{\theta}(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$
- Inverse reinforcement learning setup

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find policy $\arg \min_{\theta} \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_{\theta}(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$
- Inverse reinforcement learning setup
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find reward function $r_{\mathbf{w}}$

$$\arg \min_{\mathbf{w}} \|\mathbf{w}\|_2^2$$

$$\text{subject to: } \sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^{\text{best}}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \leq \sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}')$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find policy $\arg \min_{\theta} \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_{\theta}(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$
- Inverse reinforcement learning setup
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find reward function $r_{\mathbf{w}}$

$$\arg \min_{\mathbf{w}} \|\mathbf{w}\|_2^2$$

$$\text{subject to: } \text{ReLU} \left(\sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^{\text{best}}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') - \sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \right) = 0$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find policy $\arg \min_{\theta} \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_{\theta}(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$
- Inverse reinforcement learning setup
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find reward function $r_{\mathbf{w}}$

$$\arg \min_{\mathbf{w}} \|\mathbf{w}\|_2^2 + \text{ReLU} \left(\sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^{\text{best}}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') - \sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \right)$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find policy $\arg \min_{\theta} \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_{\theta}(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$
- Inverse reinforcement learning setup
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find reward function $r_{\mathbf{w}}$

$$\arg \min_{\mathbf{w}} \|\mathbf{w}\|_2^2 + \text{ReLU} \left(\sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^{\text{best}}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') - \sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \right)$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find policy $\arg \min_{\theta} \sum_{(\mathbf{x}_i, \mathbf{a}_i) \in \tau^*} \|\pi_{\theta}(\mathbf{x}_i) - \mathbf{a}_i\|_2^2$
- Inverse reinforcement learning setup
 1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \dots$
 2. Find reward function $r_{\mathbf{w}}$

$$\arg \min_{\mathbf{w}} \|\mathbf{w}\|_2^2 + \text{ReLU} \left(\sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^{\text{best}}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') - \sum_{(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \right)$$

3. Solve underlying RL/control task

Abbeel et al. IJRR 2010

- inverse reinforcement learning
- **state space:** angular and euclidean position, velocity, acceleration
- **action space:** motor torques
- learning reward function from expert pilot





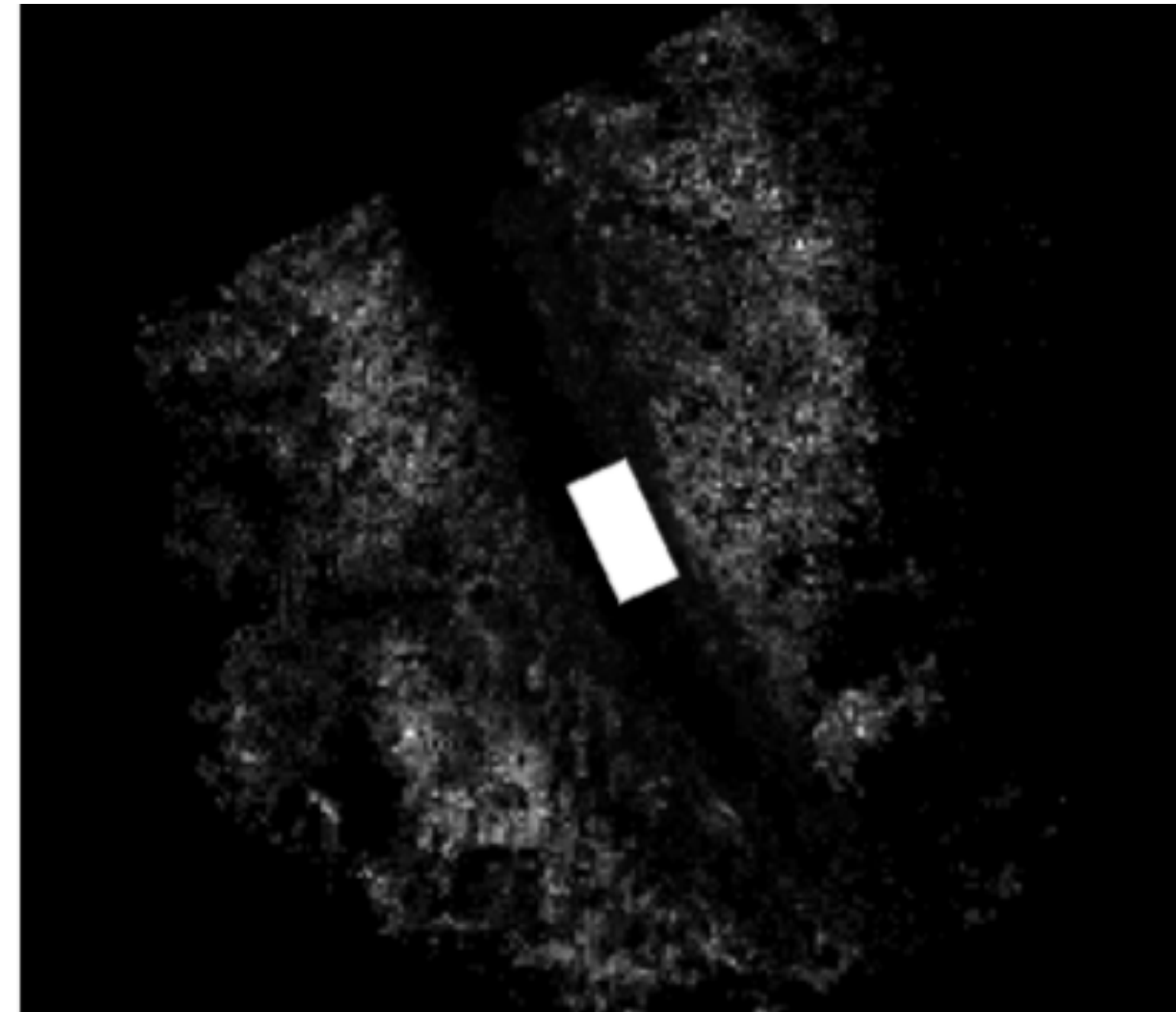


Similar to recent DARPA RACER
<http://www.dtic.mil/dtic/tr/fulltext/u2/a525288.pdf>

Silver et al. IJRR 2010



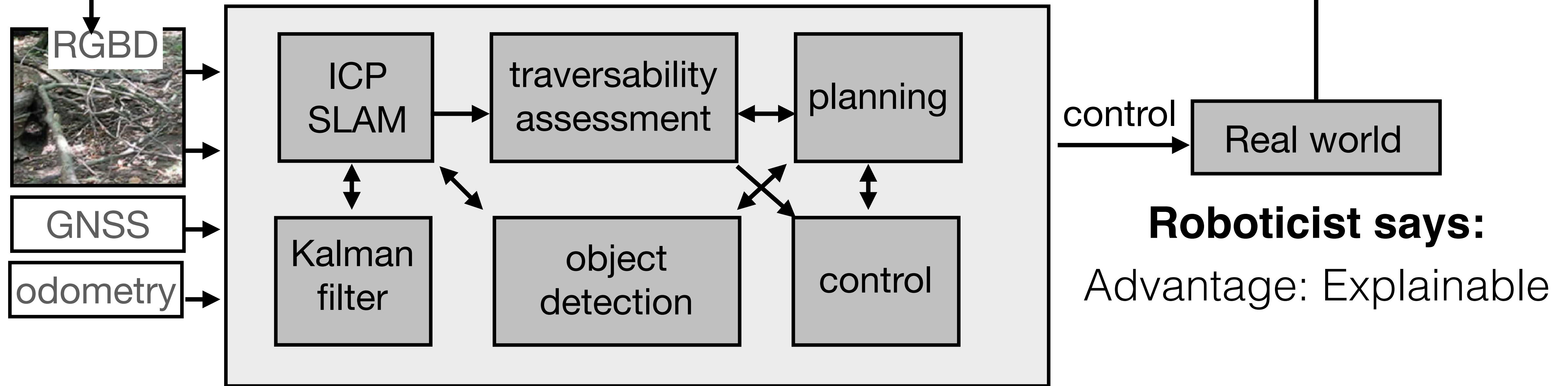
input image (state)



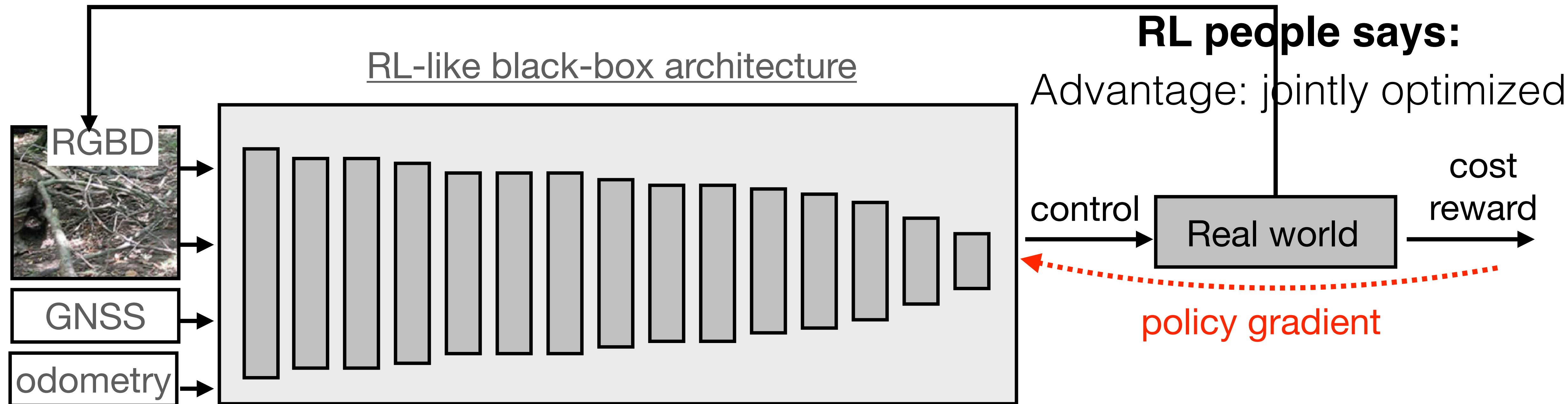
learned reward function
(traversability map)

<http://www.dtic.mil/dtic/tr/fulltext/u2/a525288.pdf>

Typical robotics white-box architecture

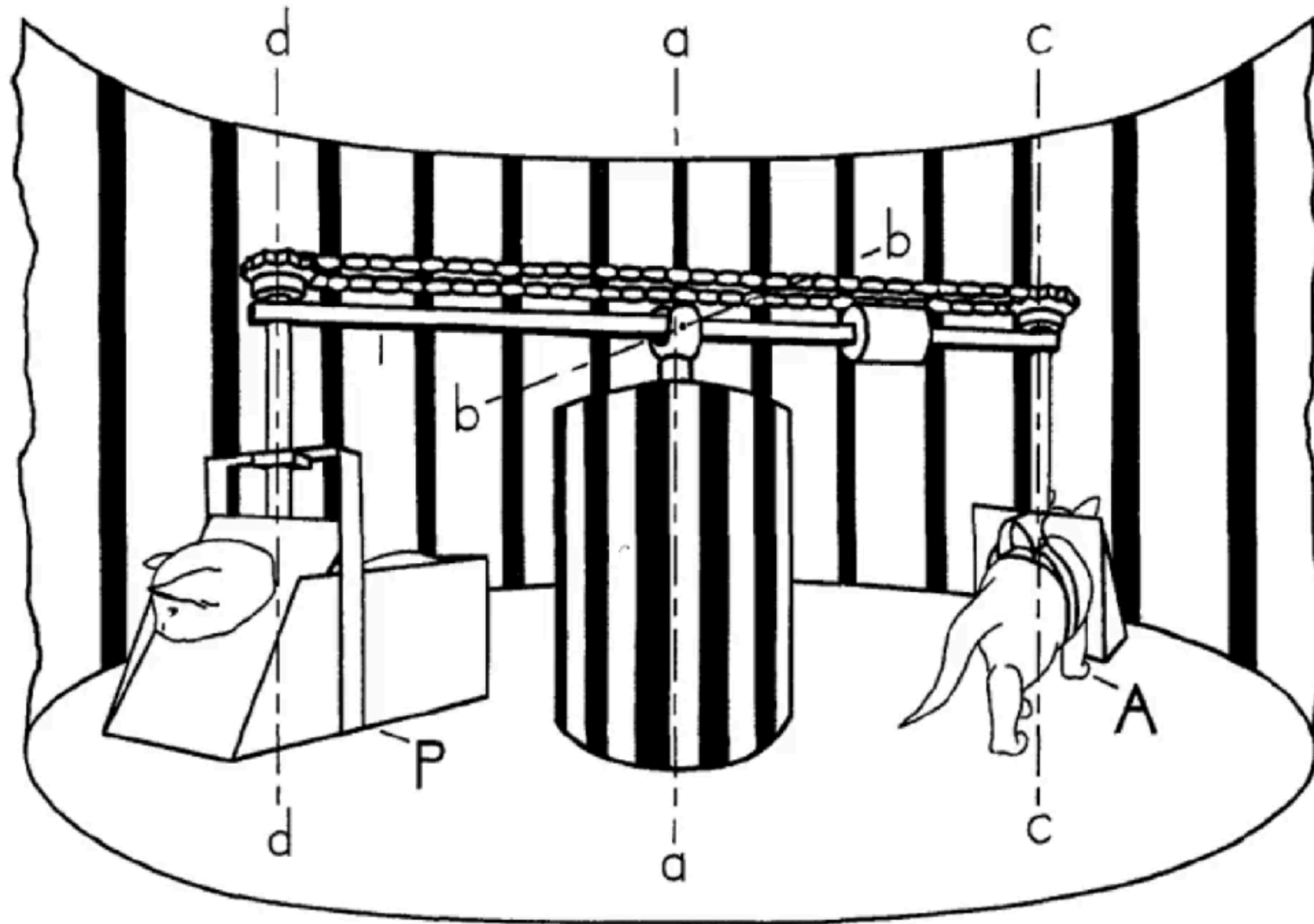


RL-like black-box architecture



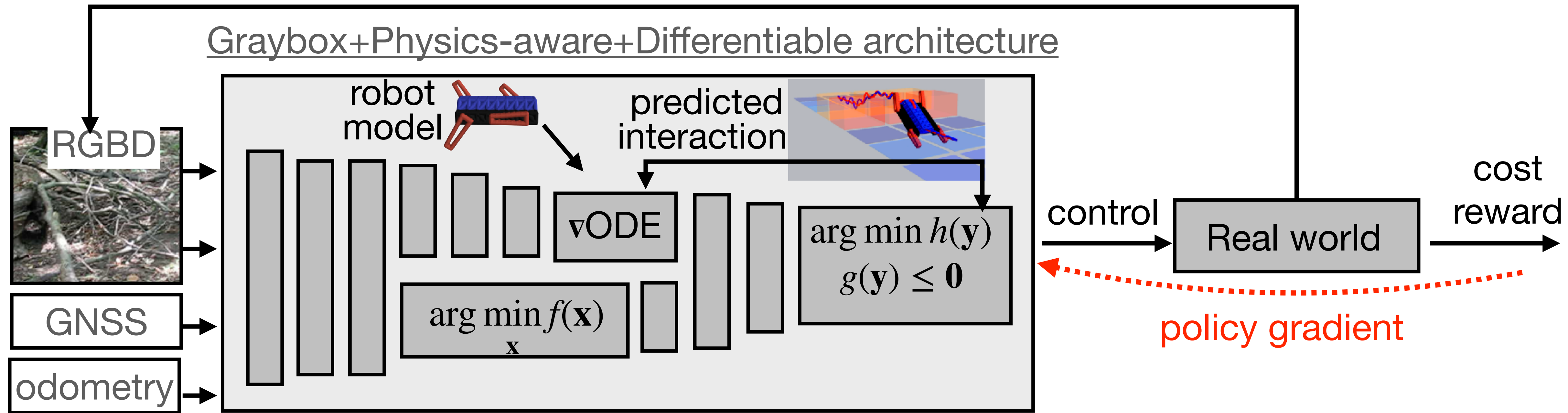
RL people says:

- Self-actuated movement is necessary in order to develop normal perception.
- => independent training of components is bad idea



[Held and Hein, J. of Comparative Psychology, 1963]

Longterm goal: Explainable + self-learnable architecture



Next lecture topic: implicit layers