Reinforcement learning in robotics

Karel Zimmermann

http://cmp.felk.cvut.cz/~zimmerk/

Vision for Robotics and Autonomous Systems
https://cyber.felk.cvut.cz/vras/

Center for Machine Perception
https://cmp.felk.cvut.cz

Department for Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

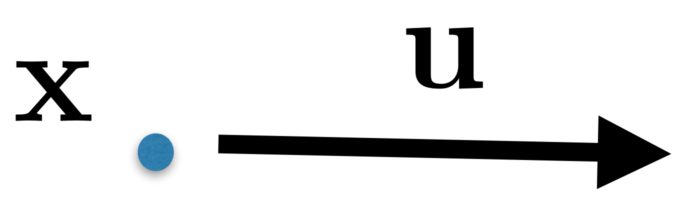
Problems often formalised as MDP

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

$\mathbf{x}$

Problems often formalised as MDP

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



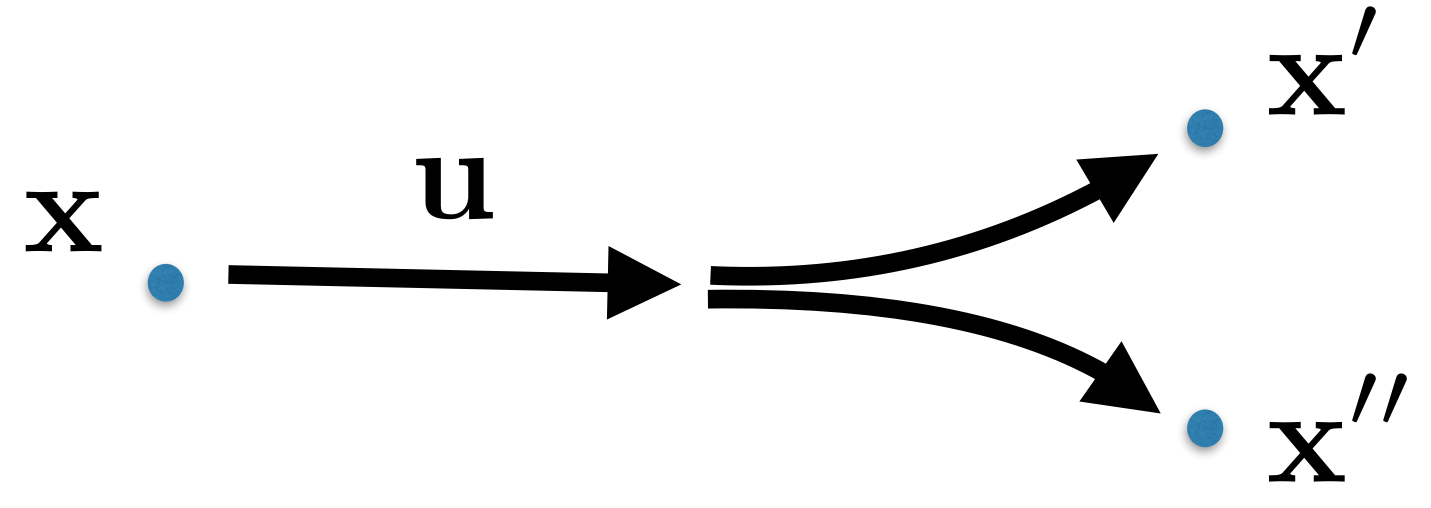
Actions: $\mathbf{u} \in \mathcal{R}^m$

Problems often formalised as MDP

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

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

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



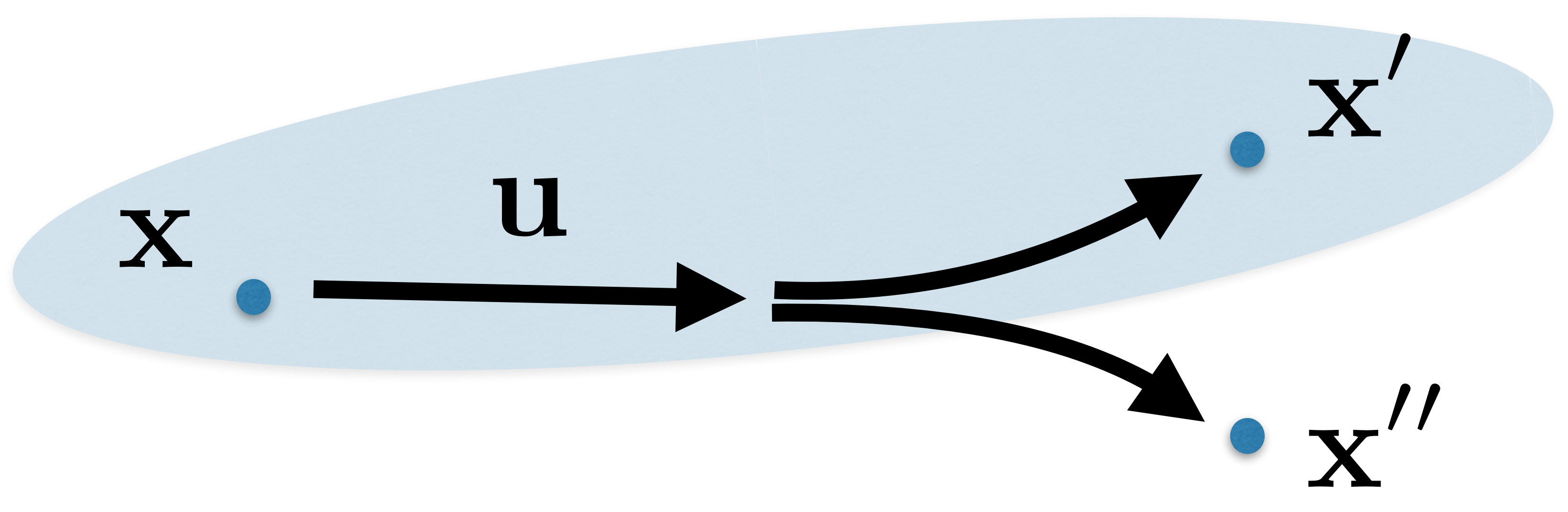
Problems often formalised as MDP

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

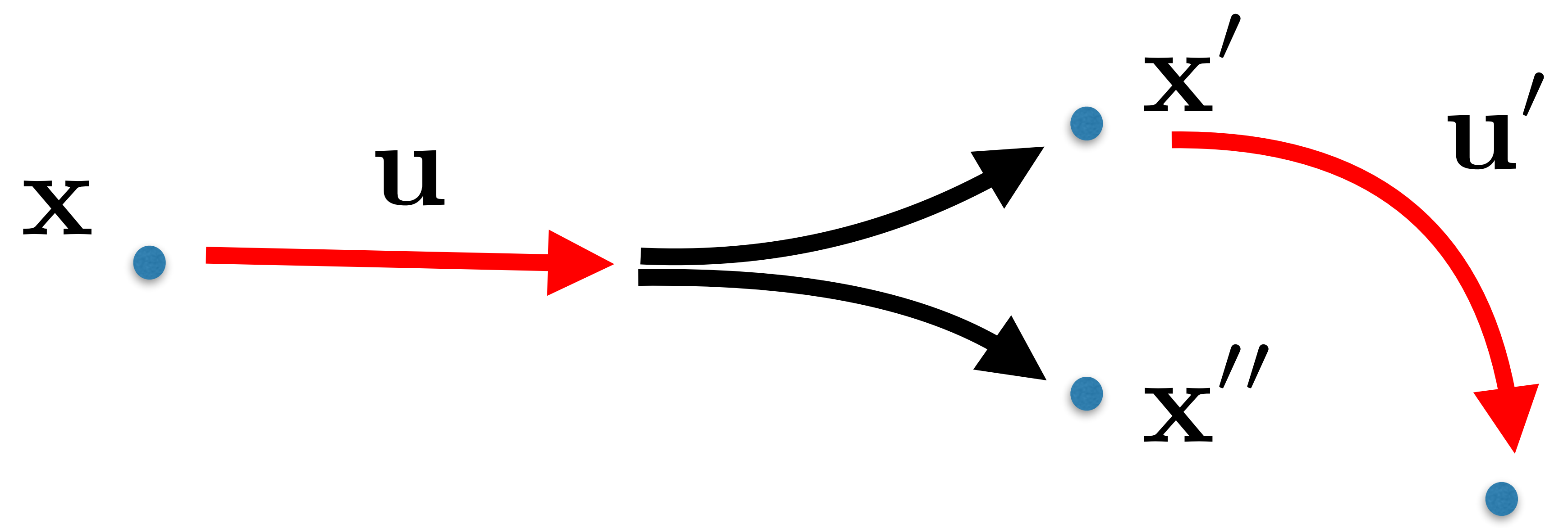
States: $\mathbf{x} \in \mathcal{R}^n$

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

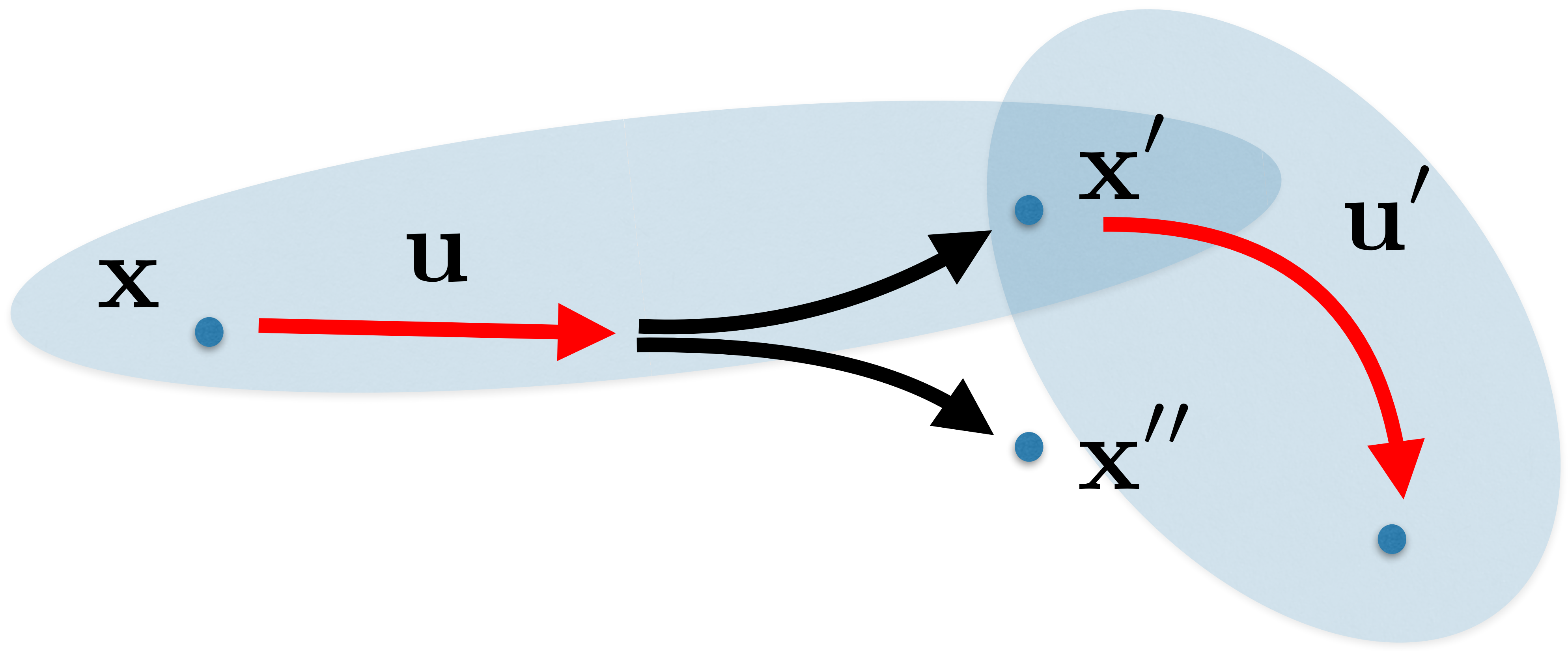
Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

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

Input	Problem	Output
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	$r(\mathbf{x},\mathbf{u},\mathbf{x}') \in \mathcal{R}$

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:

- Build physical engine and identify physical quantities
  - usually non-differentiable black-box model

- Learn (deep convolutional) network to predict following state $\mathbf{x}_{k+1} = p_\theta(\mathbf{x}_k, \mathbf{u}_k) + \mathcal{N}$

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

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

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$

θ

- More complex formulations: RNN or autoregressive model such as PixelCNN++

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

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

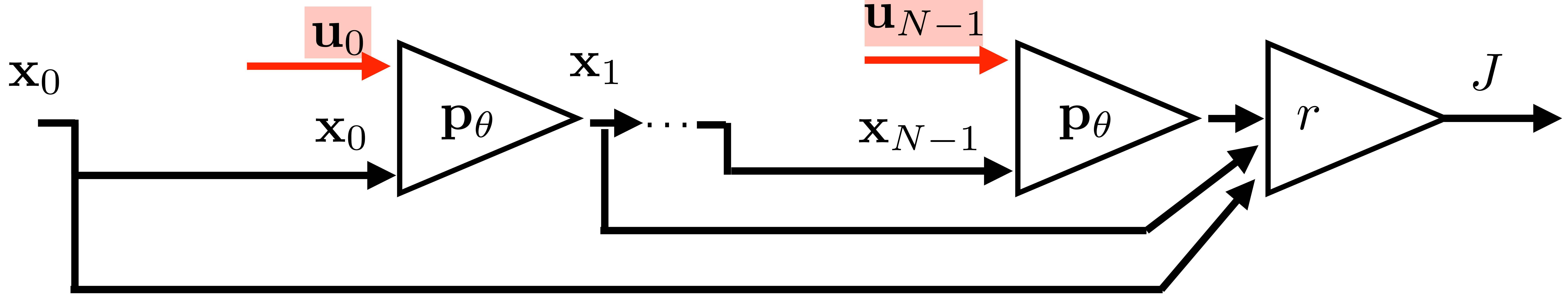
Planning actions

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots,$ ini: $\theta = \text{rand}, \omega = \text{rand}$
2. Estimate motion model

$$\underset{(\mathbf{x},\mathbf{u},\mathbf{x}')\in\tau}{\arg\min} \|\mathbf{x}' - p_\theta(\mathbf{x},\mathbf{u})\|$$

3. **Plan** policy (sequence of actions) maximizing the rewards on model-based trajectories

$$\underset{\mathbf{u}_0,\mathbf{u}_1,\ldots,\mathbf{u}_{N-1}}{\arg\max} \left\{ \sum_k r(\mathbf{x}_k, \mathbf{u}_k) \;\middle|\; \mathbf{x}_{k+1} = p_\theta(\mathbf{x}_k, \mathbf{u}_k) \right\}$$



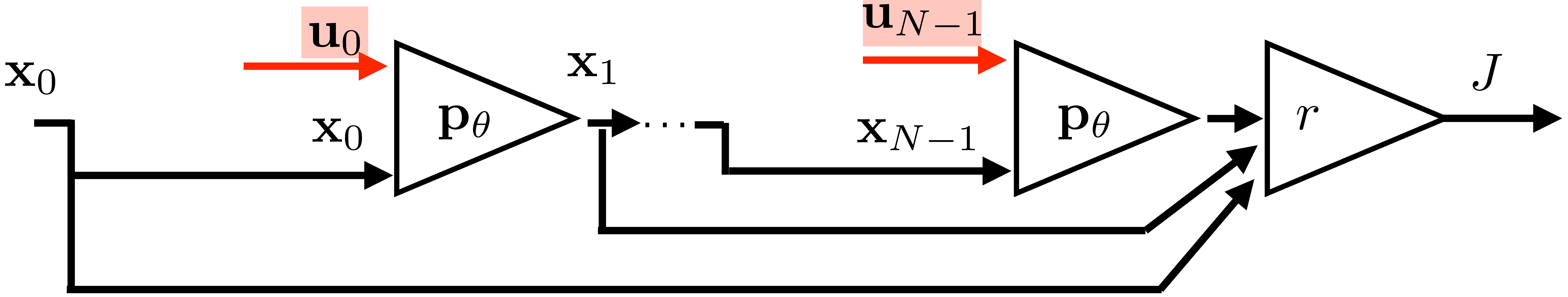
Planning actions

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots,$ ini: $\theta = \text{rand}, \omega = \text{rand}$
2. Estimate motion model

$$\arg\min_{(\mathbf{x},\mathbf{u},\mathbf{x}')\in\tau} \|\mathbf{x}' - p_\theta(\mathbf{x}, \mathbf{u})\|$$

3. **Plan** policy (sequence of actions) maximizing the rewards on model-based trajectories

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



- typically non-convex => gradient optimization inefficient
- BFS, Dijkstra, A*, RRT, … => **open loop control**

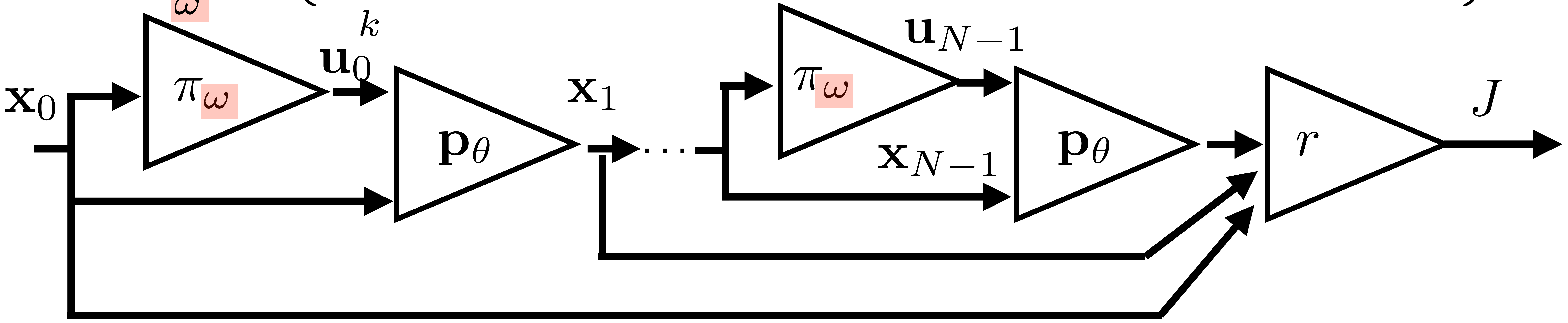
Learning policy

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$, ini: $\theta = \text{rand}, \omega = \text{rand}$
2. Estimate motion model

$$\arg\min_{(\mathbf{x},\mathbf{u},\mathbf{x}')\in\tau} \|\mathbf{x}' - p_\theta(\mathbf{x}, \mathbf{u})\|$$

3. **Learn** policy (e.g. deepnet) maximizing the rewards on model-based trajectories

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



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

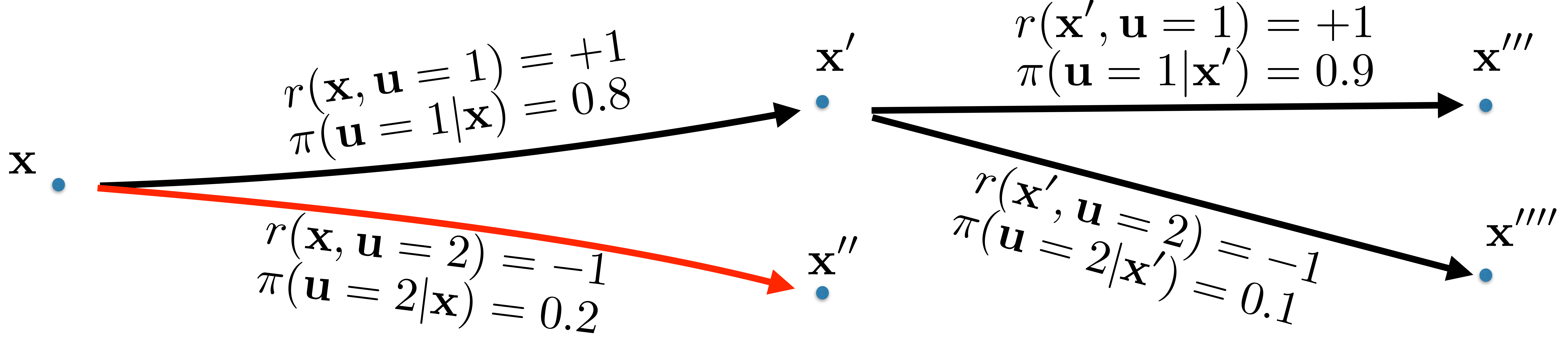
Reinforcement learning:
- given rewards and trajectories, estimate optimal policy

→ $\pi^* = \arg\max_\pi J_\pi$

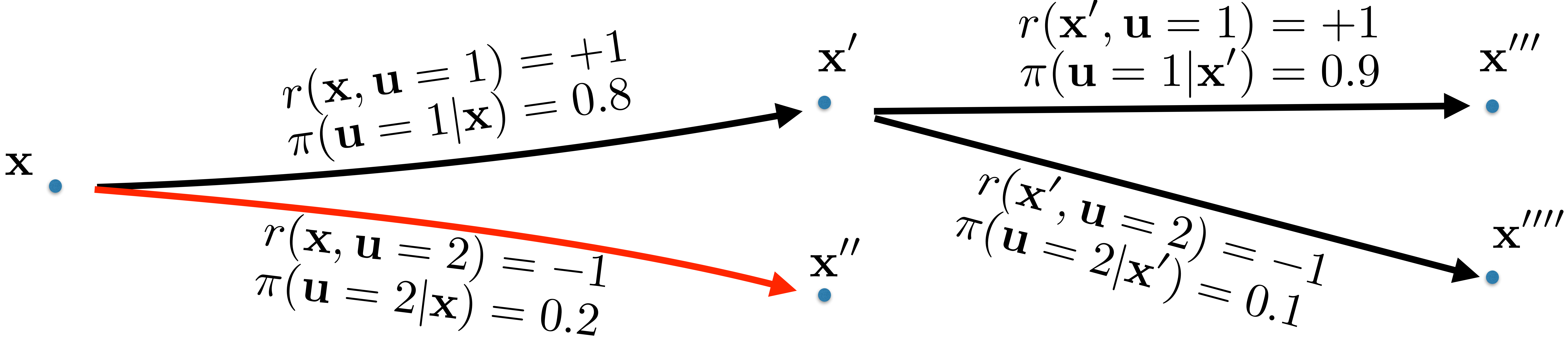
τ^* →

Inverse reinforcement learning:
- given optimal trajectories estimate reward function

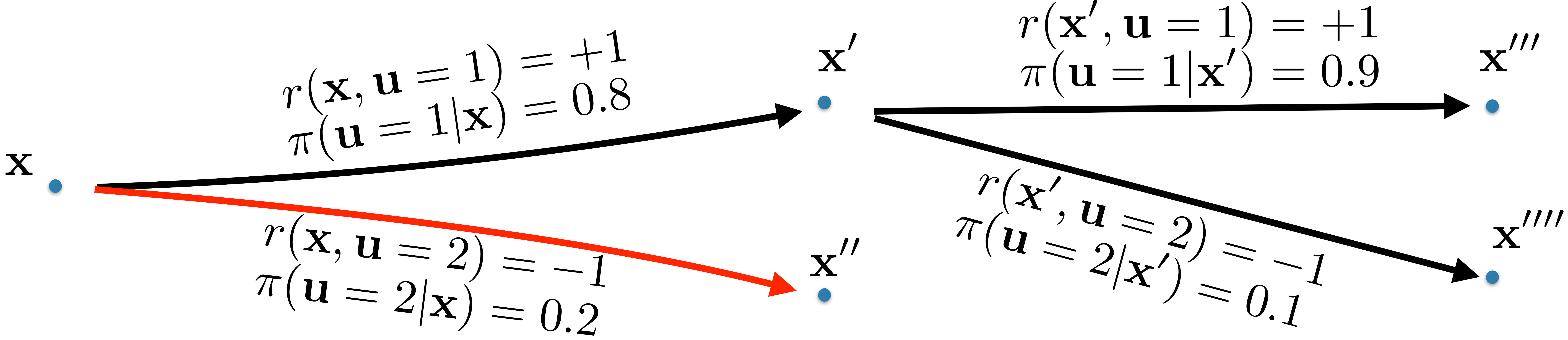
→ $r(\mathbf{x},\mathbf{u},\mathbf{x}') \in \mathcal{R}$



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

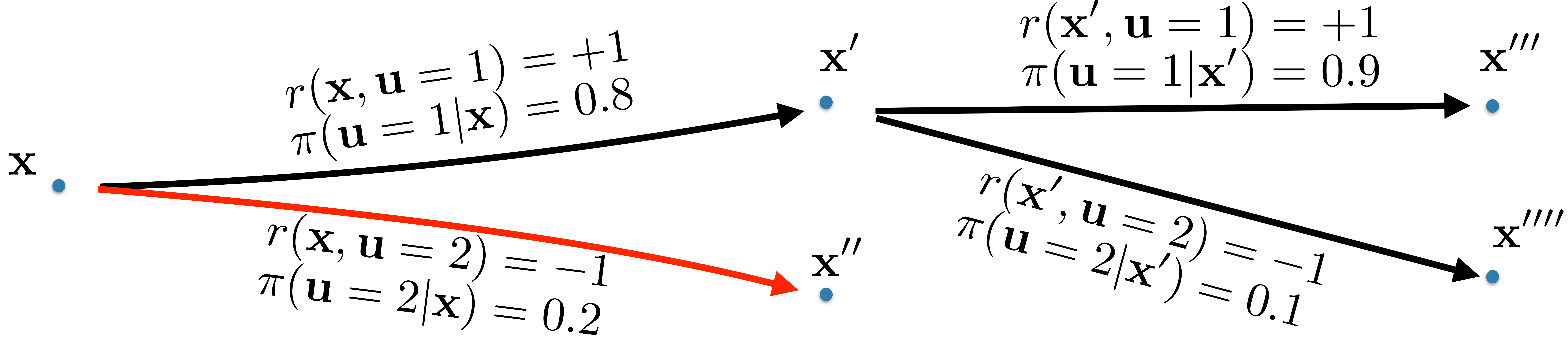


$$V^{\pi}(\mathbf{x}) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\right] = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}[r(\tau)] = \int_{\tau:\mathbf{x}_0=\mathbf{x}} p(\tau|\pi)r(\tau)$$



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

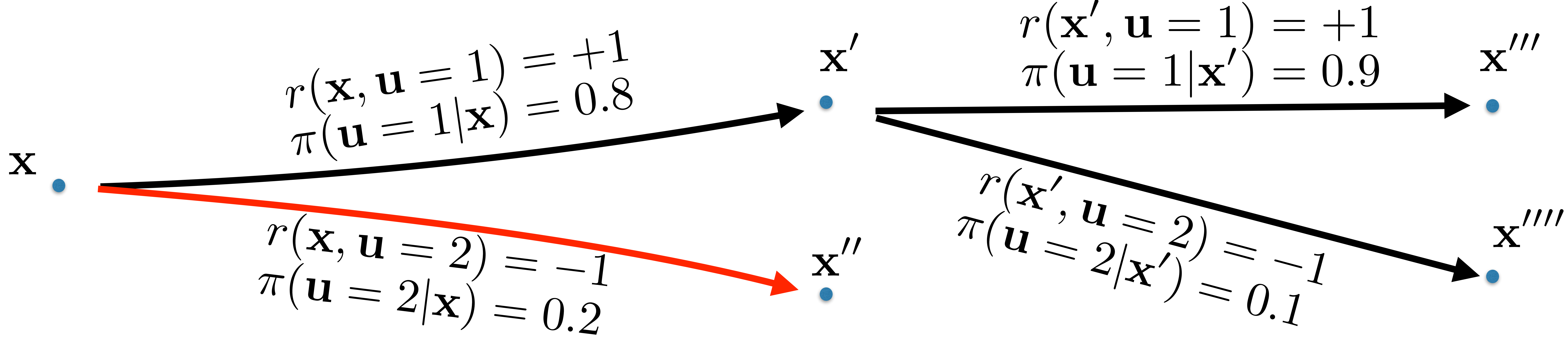
$$Q^{\pi}(\mathbf{x},\mathbf{u}) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x} \\ \mathbf{u}_0=\mathbf{u}}}\Big[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\Big] = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x} \\ \mathbf{u}_0=\mathbf{u}}}[r(\tau)] = \int_{\substack{\tau:\mathbf{x}_0=\mathbf{x}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



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

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

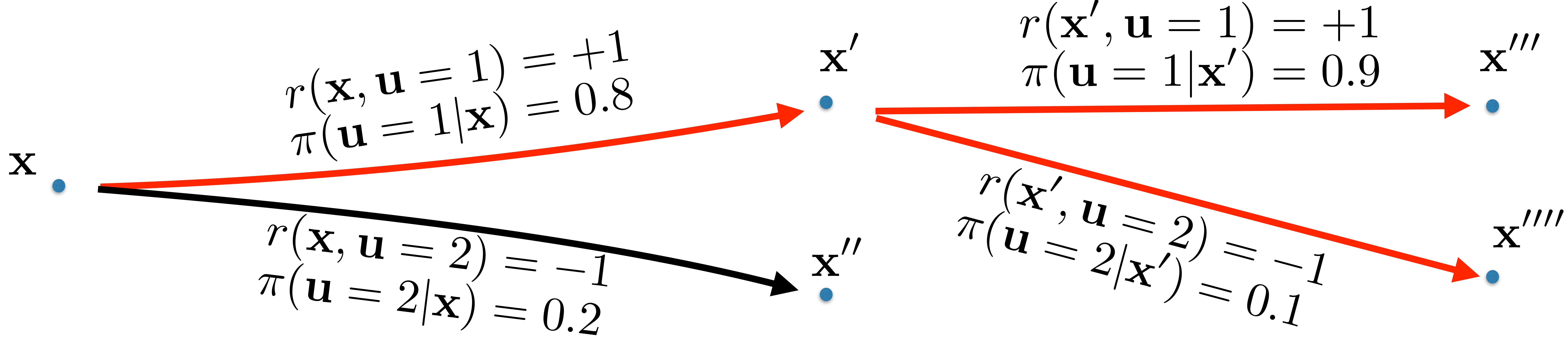
$$Q^{\pi}(\mathbf{x},\mathbf{u}=2) = \text{???}$$



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

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

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

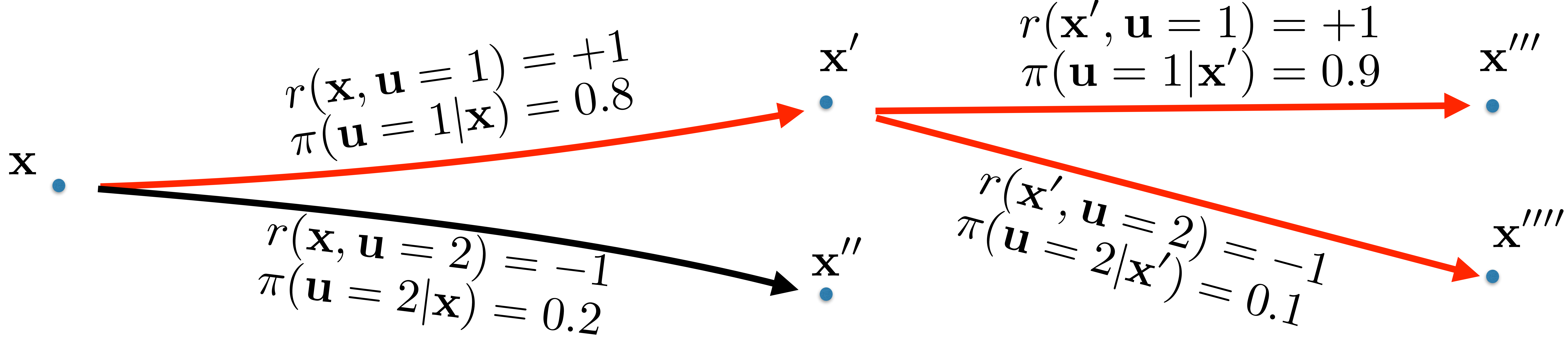


$$V^\pi(\mathbf{x}) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}] = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}[r(\tau)] = \int_{\tau:\mathbf{x}_0=\mathbf{x}} p(\tau|\pi) r(\tau)$$

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

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

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

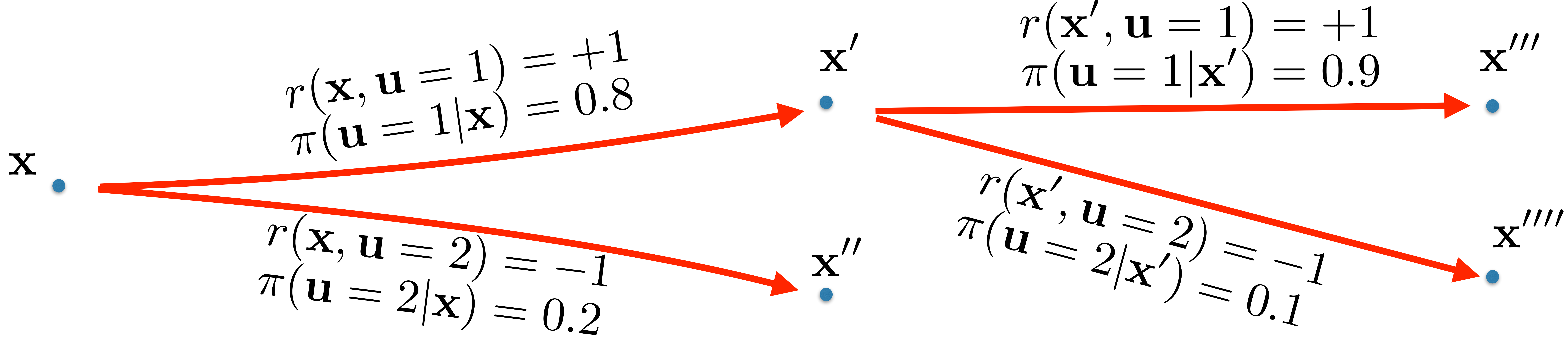


$$V^\pi(\mathbf{x}) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi\\ \mathbf{x}_0=\mathbf{x}}}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\right] = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi\\ \mathbf{x}_0=\mathbf{x}}}[r(\tau)] = \int_{\tau:\mathbf{x}_0=\mathbf{x}} p(\tau|\pi)r(\tau)$$

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

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

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



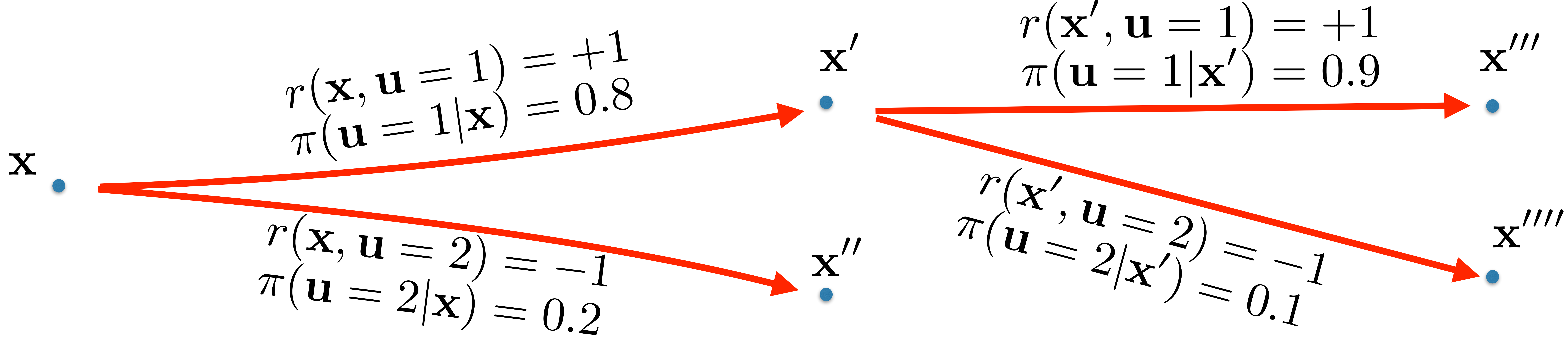
$$V^\pi(\mathbf{x}) = \underset{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}{\mathbb{E}}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\right] = \underset{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}{\mathbb{E}}[r(\tau)] = \int_{\tau:\mathbf{x}_0=\mathbf{x}} p(\tau|\pi)r(\tau)$$

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

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

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

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



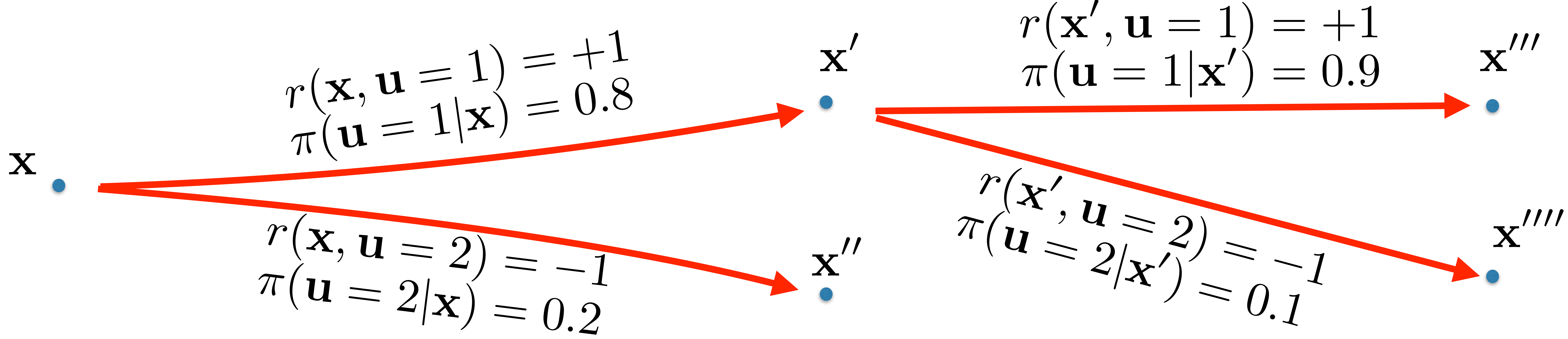
$$V^\pi(\mathbf{x}) = \underset{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}{\mathbb{E}}[\sum_{t=0}^{\infty}\gamma^t r_{t+1}] = \underset{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}{\mathbb{E}}[r(\tau)] = \int_{\tau:\mathbf{x}_0=\mathbf{x}} p(\tau|\pi)r(\tau)$$

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

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

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

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



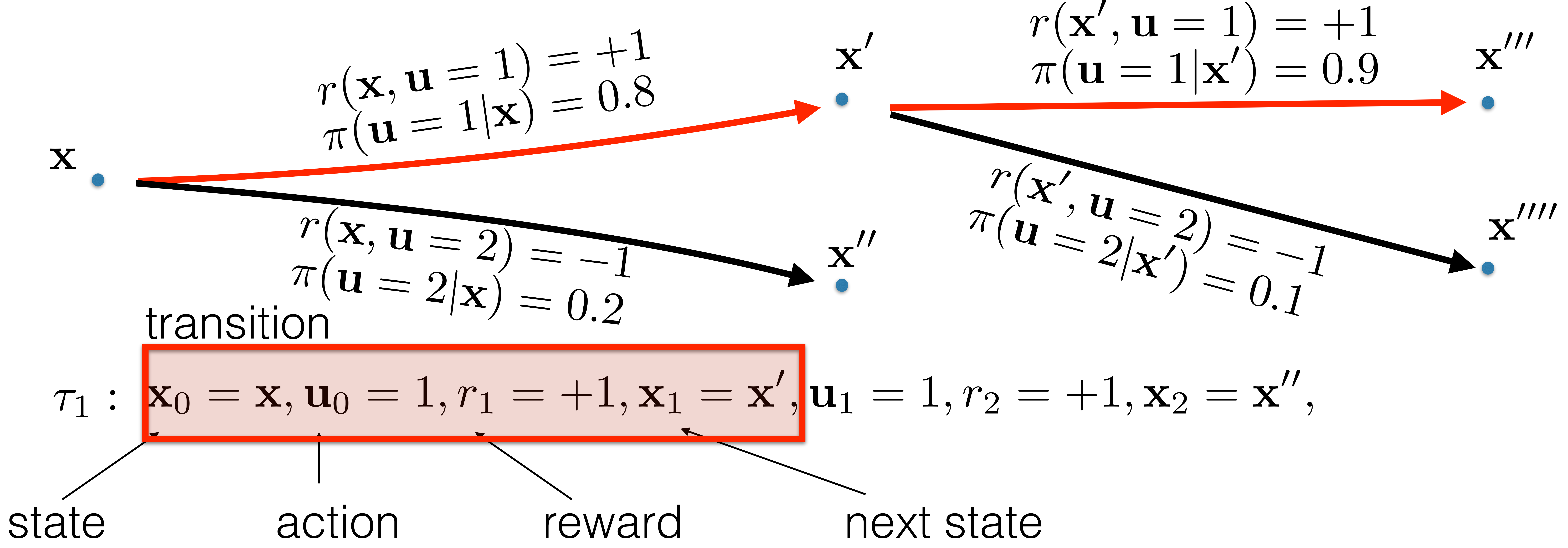
$$V^\pi(\mathbf{x}) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}] = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \mathbf{x}_0=\mathbf{x}}}[r(\tau)] = \int_{\tau:\mathbf{x}_0=\mathbf{x}} p(\tau|\pi) r(\tau)$$

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

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

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

$$V^\pi(\mathbf{x}) = 0.8 * (1 + 0.9 * 1 + 0.1 * (-1)) + 0.2 * (-1) = 1.24$$
$$A^\pi(\mathbf{x},\mathbf{u}=1) = Q^\pi(\mathbf{x},\mathbf{u}=1) - V^\pi(\mathbf{x}) = 1.8 - 1.24 = 0.56$$
$$A^\pi(\mathbf{x},\mathbf{u}=2) = Q^\pi(\mathbf{x},\mathbf{u}=2) - V^\pi(\mathbf{x}) = -1 - 1.24 = -2.24$$

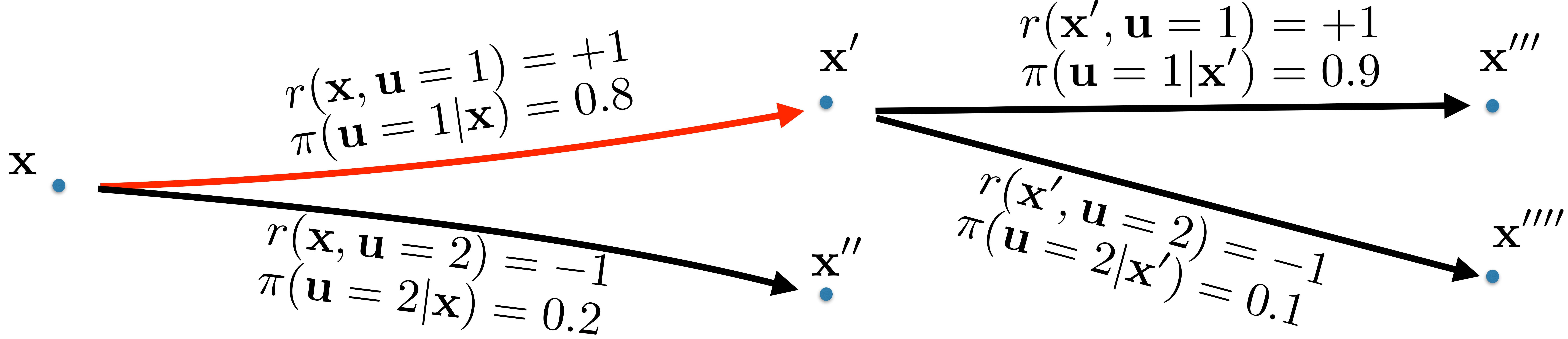


- Search for Q satisfying Bellman equation (for every transition):

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

- Once we find it, the optimal policy is:

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



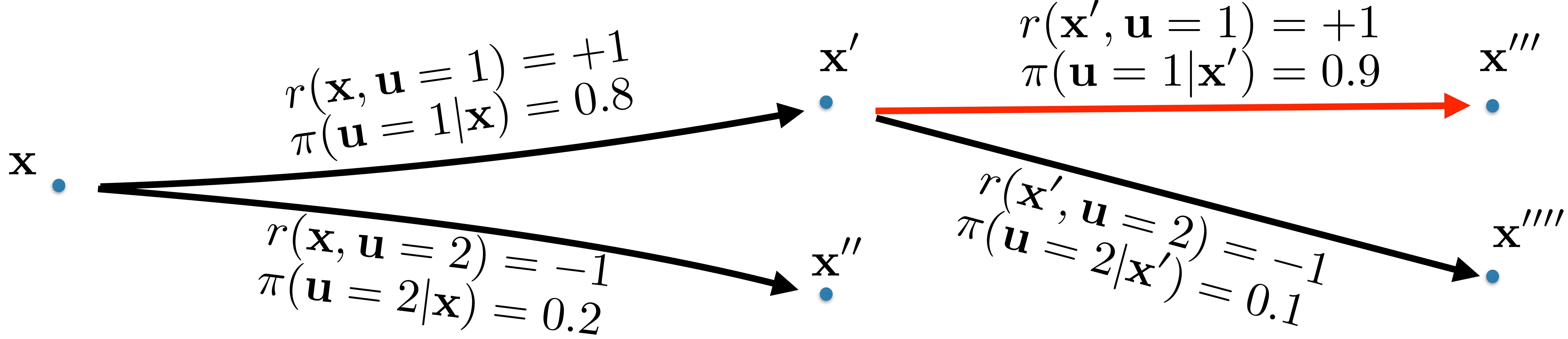
$$\tau_1 : \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1, r_1 = +1, \mathbf{x}_1 = \mathbf{x}', \mathbf{u}_1 = 1, r_2 = +1, \mathbf{x}_2 = \mathbf{x}'',$$

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

$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

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

Q	u=1	u=2
x	0	0
x’	0	0
x’’	0	0
x’’’	0	0
x’’’’	0	0



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

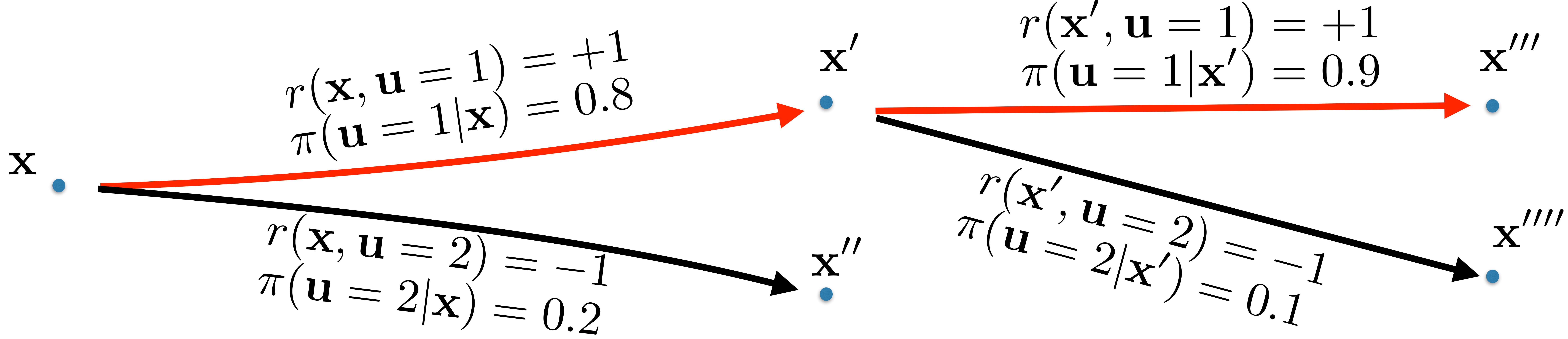
state $\mathbf{x}_1$ | action $\mathbf{u}_1$ | reward r_1 | next state $\mathbf{x}_2$

$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

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

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

Q	u=1	u=2
x	0	0
x’	0	0
x”	0	0
x’’’	0	0
x’’’’	0	0



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

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

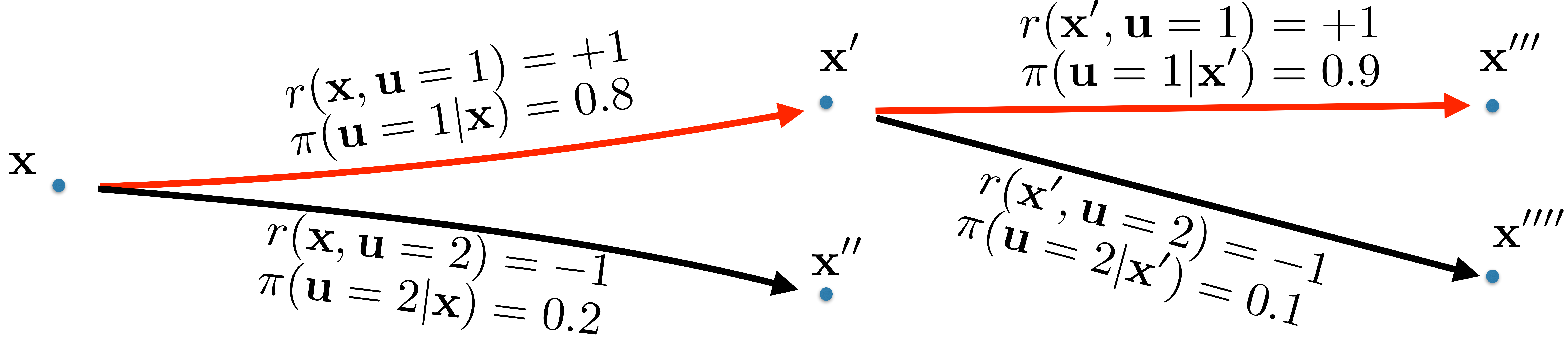
$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

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

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

Bellman equation is not satisfied

Q	u=1	u=2
x	0	0
x’	0	0
x’’	0	0
x’’’	0	0
x’’’’	0	0



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

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

$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

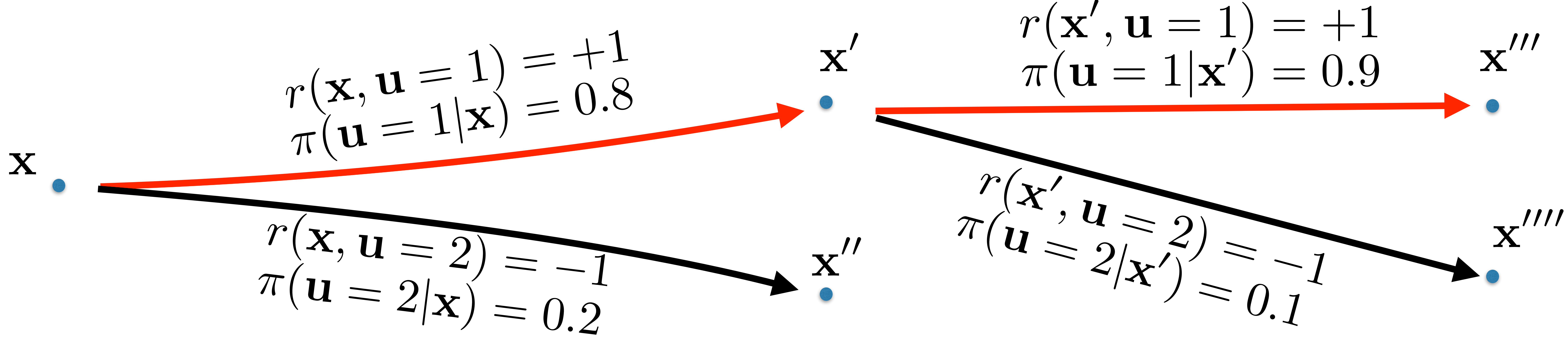
$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) \quad = +1$$

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

$\neq$

Q	u=1	u=2
x	0	0
x’	0	0
x”	0	0
x’’’	0	0
x’’’’	0	0

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 $\mathbf{x}_2$

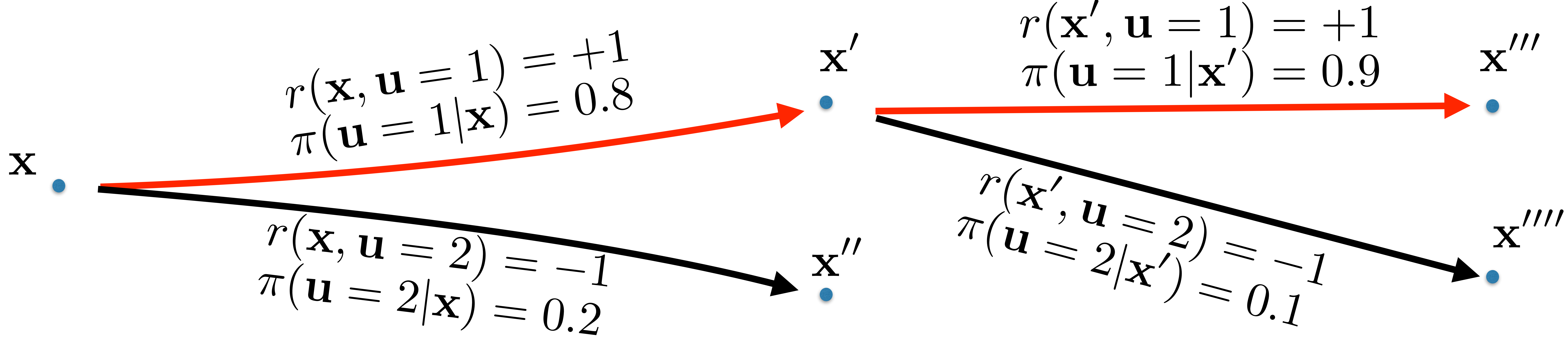
$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

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

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

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

Q	u=1	u=2
x	1	0
x’	0	0
x”	0	0
x’’’	0	0
x’’’’	0	0



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

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

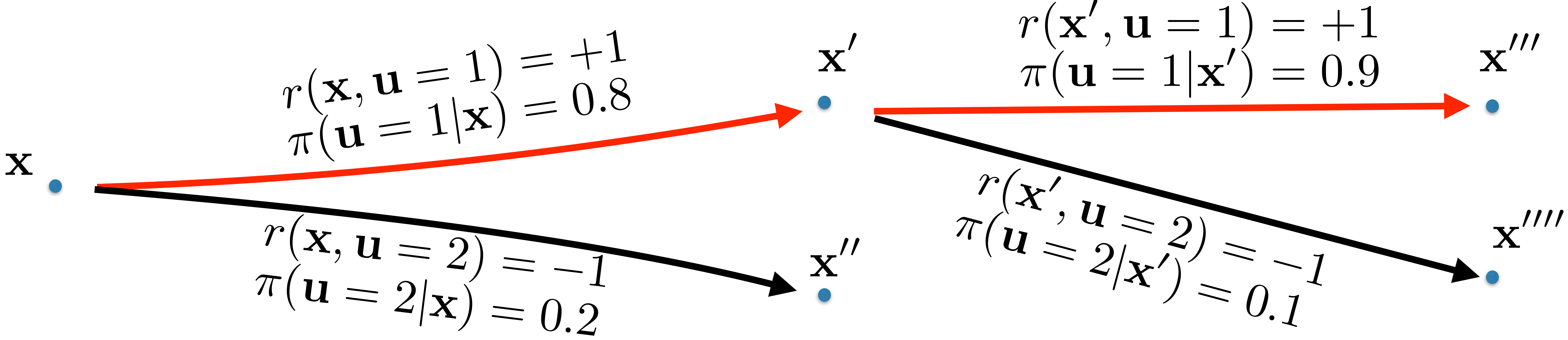
$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

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

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

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



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

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

$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

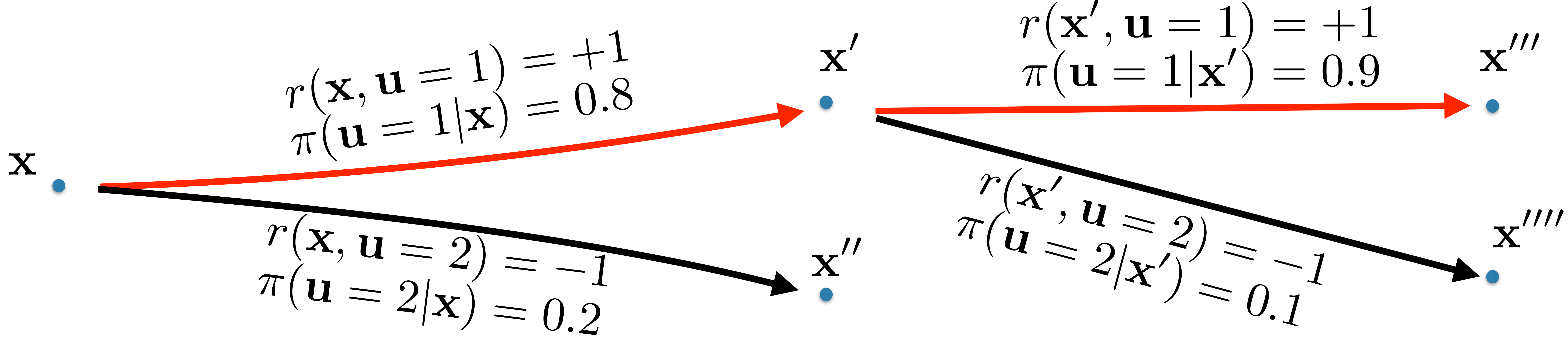
$$Q(\mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = 1) = +1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1 = \mathbf{x}', \mathbf{u}) \quad =+1.9$$

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

Recompute RHS

$\neq$

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



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

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

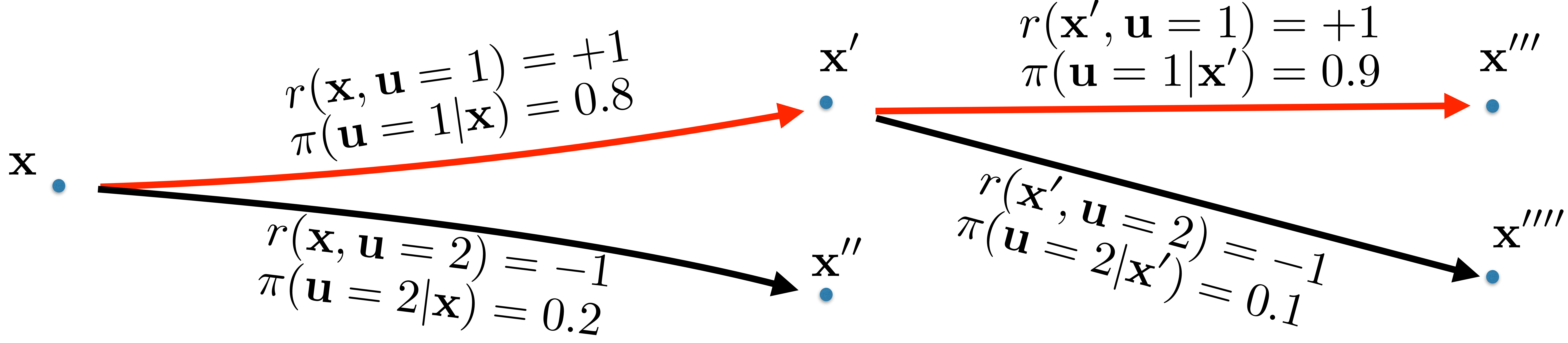
$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

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

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

Substitute of RHS to LHS.

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



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

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

$$Q(\mathbf{x}_0, \mathbf{u}_0) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_1, \mathbf{u})$$

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

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

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

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

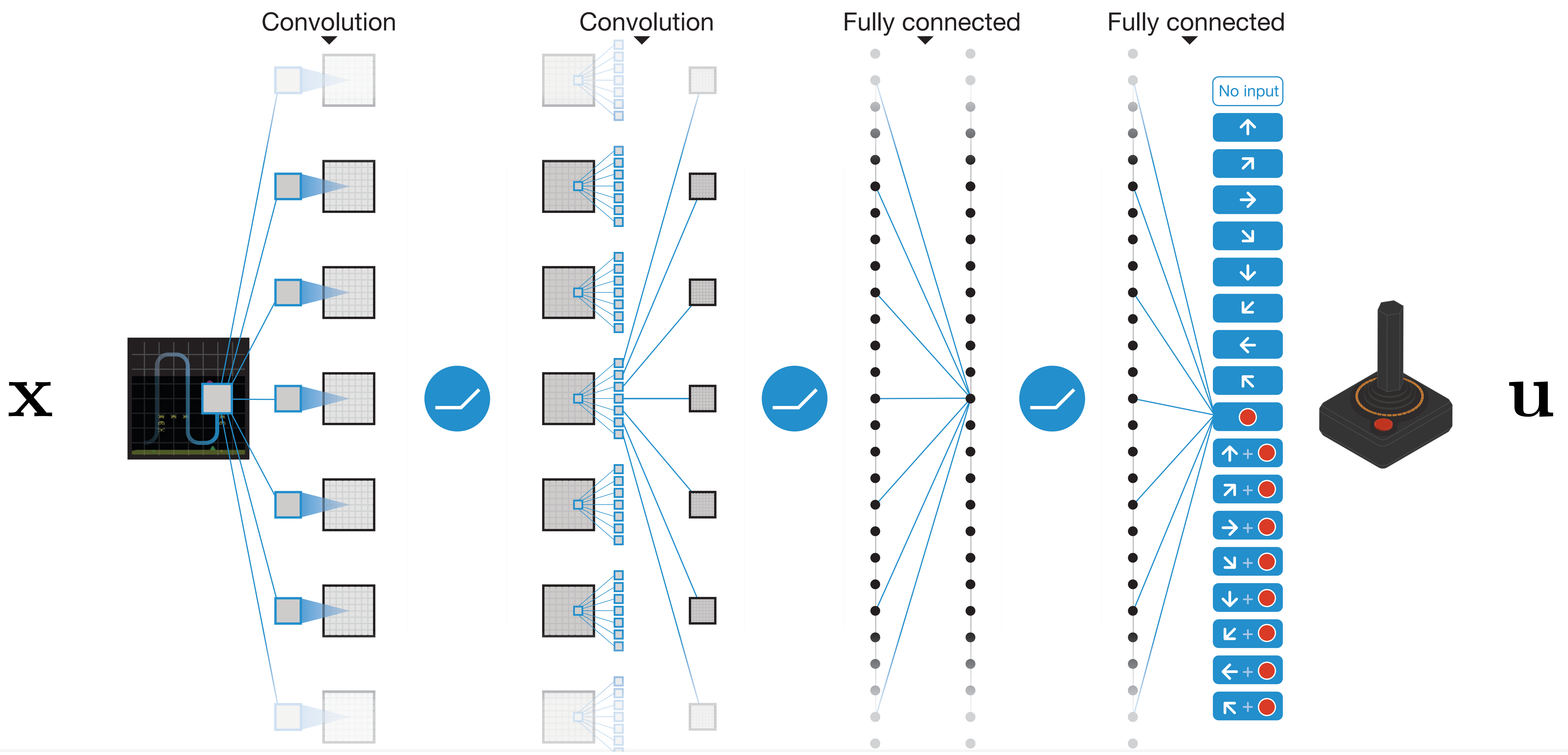
$$\arg\min_\theta \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

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
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

Q-learning

1. Collect transition
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.)

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

★ **Transitions are strongly correlated !**

4. 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)
```

https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

Q-learning

1. Collect transition
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 $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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 $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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 !

★ **Approximated Q-learning often diverges**

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 $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$ Target net (slowly upd.)
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.)

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

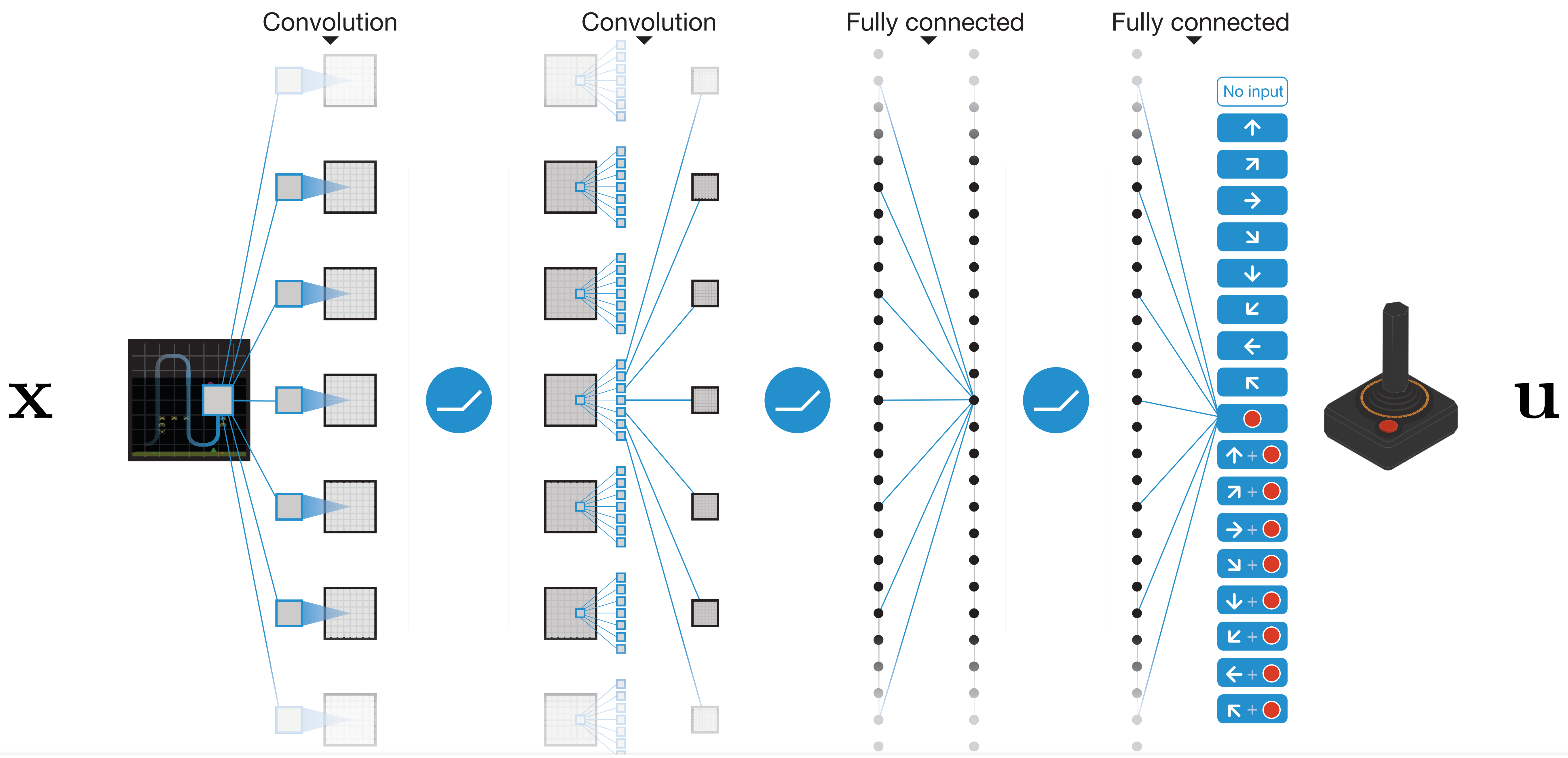
$$\arg\min_\theta \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$ Policy net (regularly upd.)

5. Update target network $\overline{\theta} := \alpha\theta + (1-\alpha)\overline{\theta}$
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)

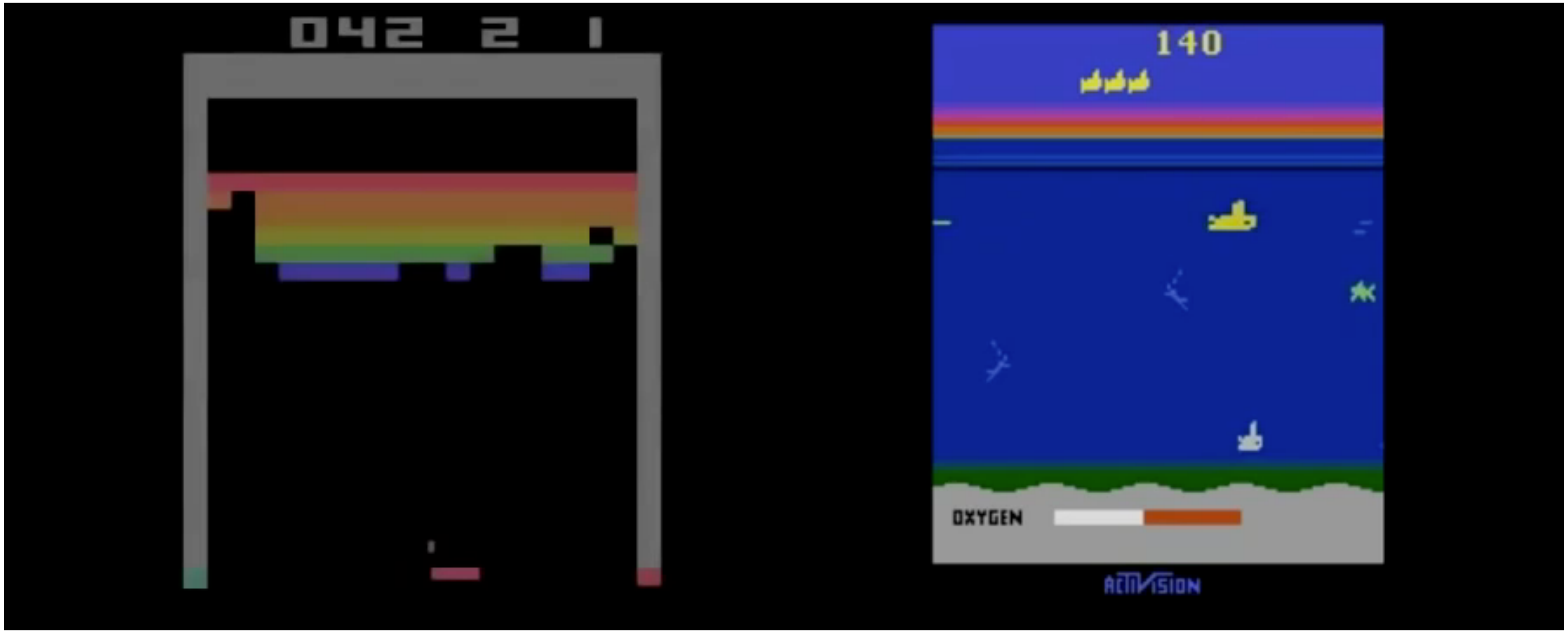
$$Q_\theta(\mathbf{x}, \mathbf{u})$$



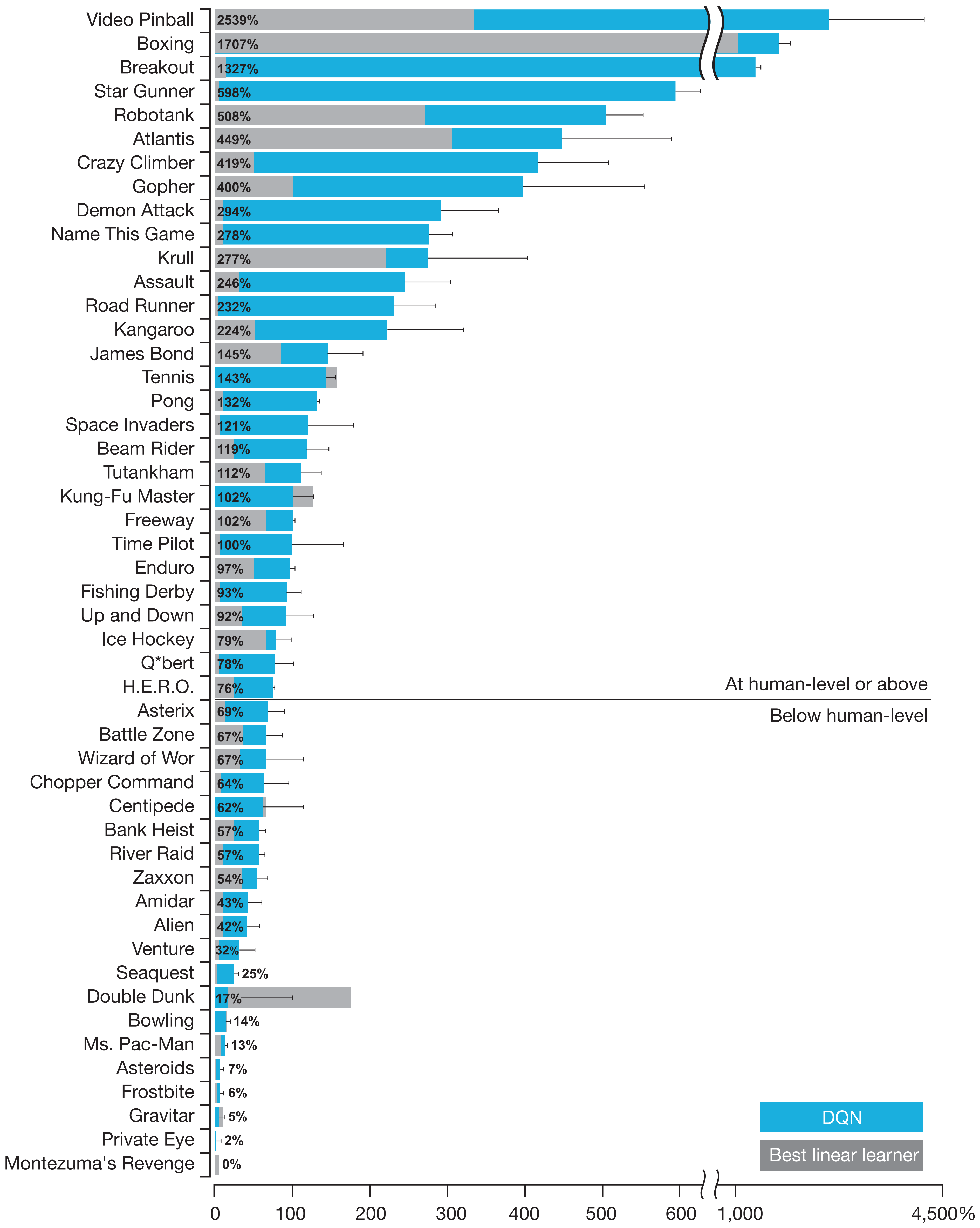
Mnih et al. Nature 2015

- replay buffer (decorrelates samples to be “more i.i.d”)
- two Q-networks (suppress oscilations)

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

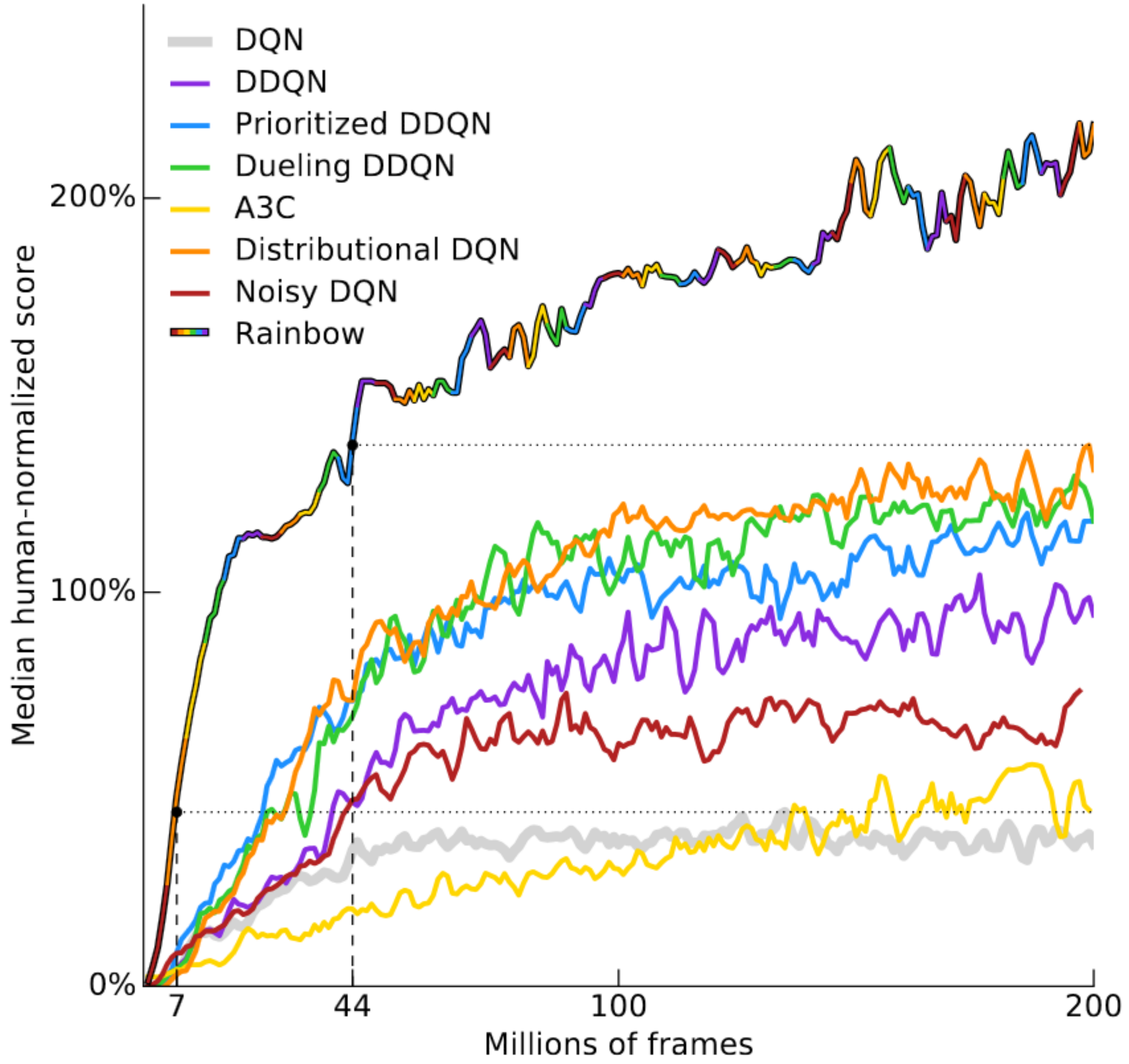


Mnih et al. Nature 2015



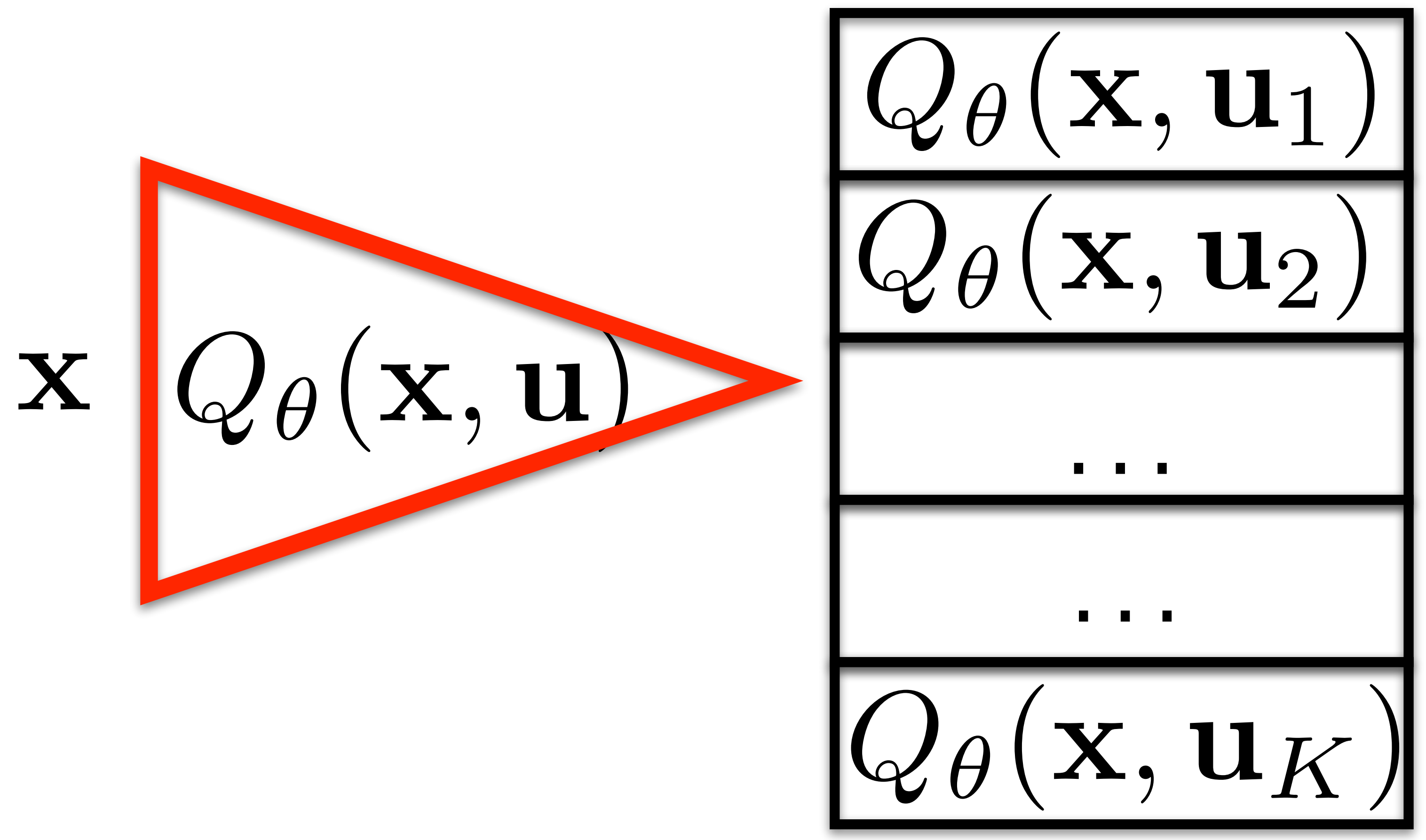
Hessel et. al Rainbow DQN, 2017

Ensemble of different RL methods



Main bottleneck of approximate Q-learning (DQN)

continuous
high-dimensional



6 legs, 3 motors/leg,
10 input levels/motor

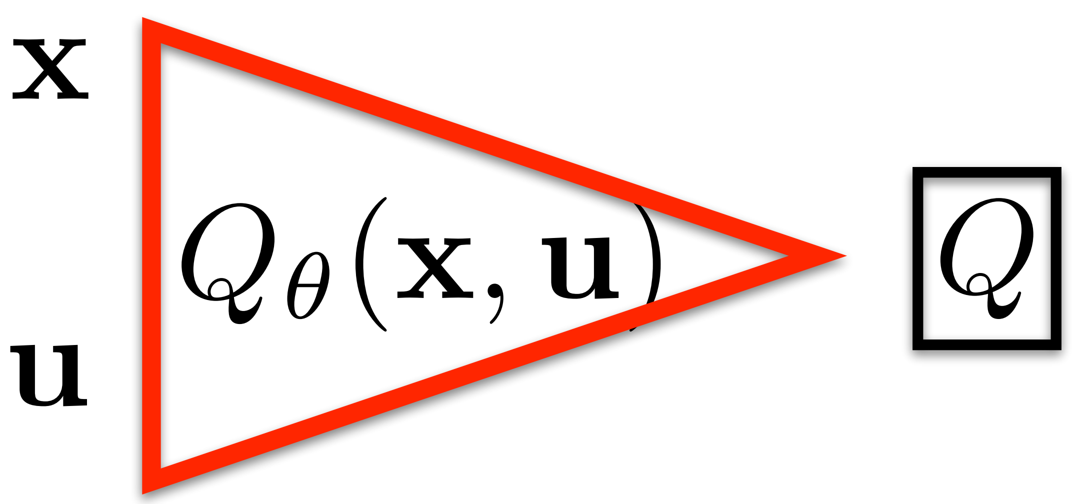
$\dim(\mathbf{u}) =$ **10^(3*6)**

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$
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 $\overline{\theta} := \alpha\theta + (1-\alpha)\overline{\theta}$
7. Repeat from 1

Main bottleneck of approximate Q-learning (DQN)

continuous
high-dimensional

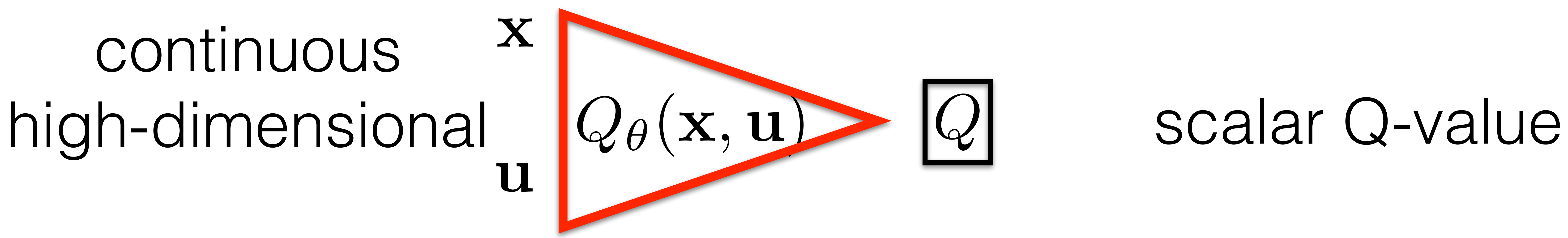


scalar Q-value

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$
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 $\overline{\theta} := \alpha\theta + (1-\alpha)\overline{\theta}$
7. Repeat from 1

Main bottleneck of approximate Q-learning (DQN)

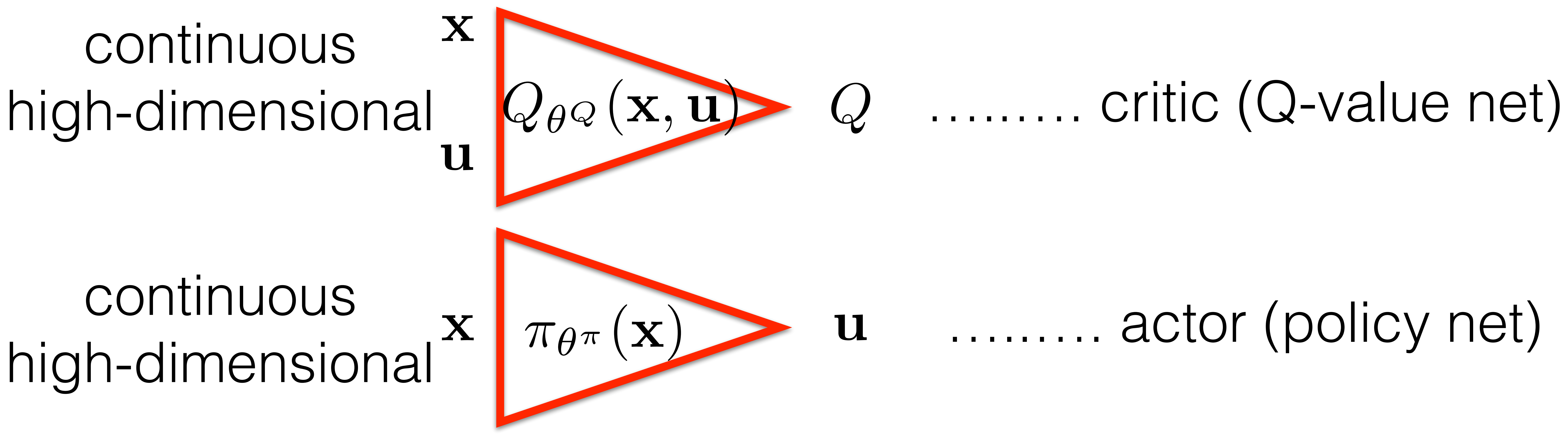


You cannot exhaustively maximize

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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 $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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)

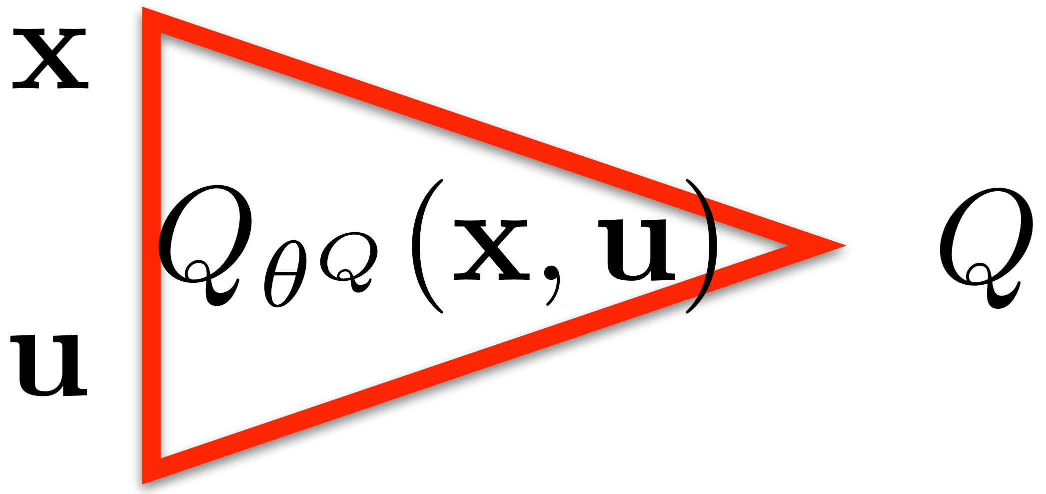
continuous high-dimensional $\mathbf{x}$, $\mathbf{u}$ → $Q_{\theta^Q}(\mathbf{x}, \mathbf{u})$ → Q ……… critic (Q-value net)

continuous high-dimensional $\mathbf{x}$ → $\pi_{\theta^\pi}(\mathbf{x})$ → $\mathbf{u}$ ……… actor (policy net)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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

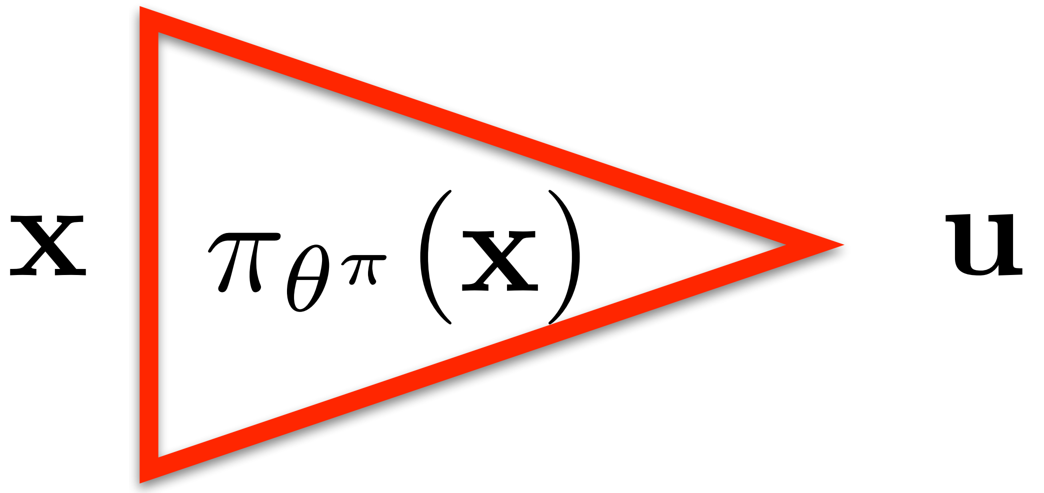
Deep Deterministic Policy Gradient (DDPG)

continuous
high-dimensional



……… critic (Q-value net)

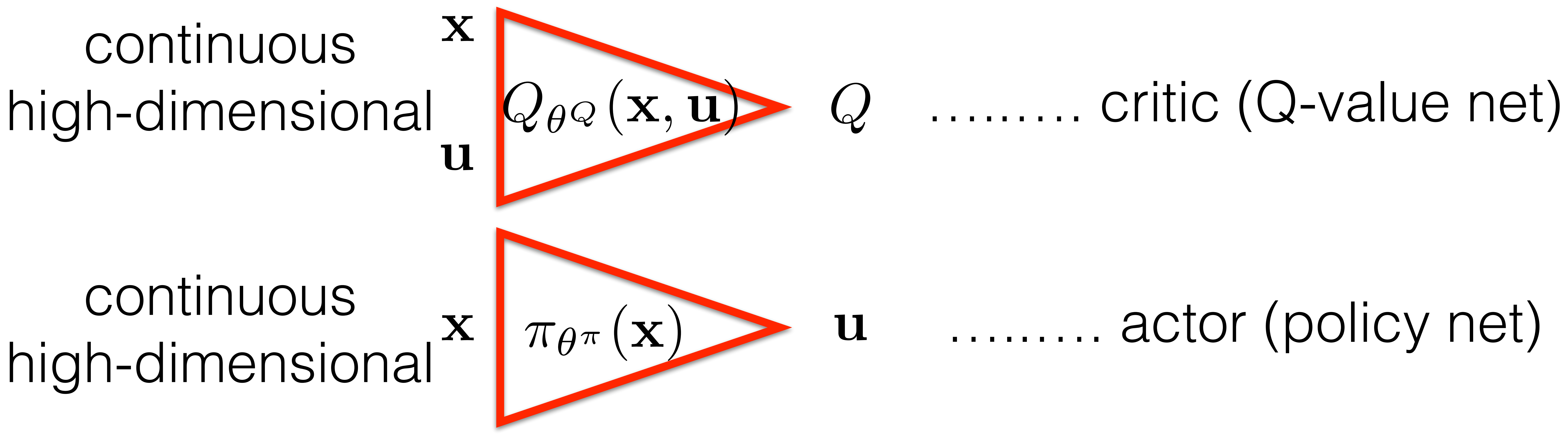
continuous
high-dimensional



……… actor (policy net)

1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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

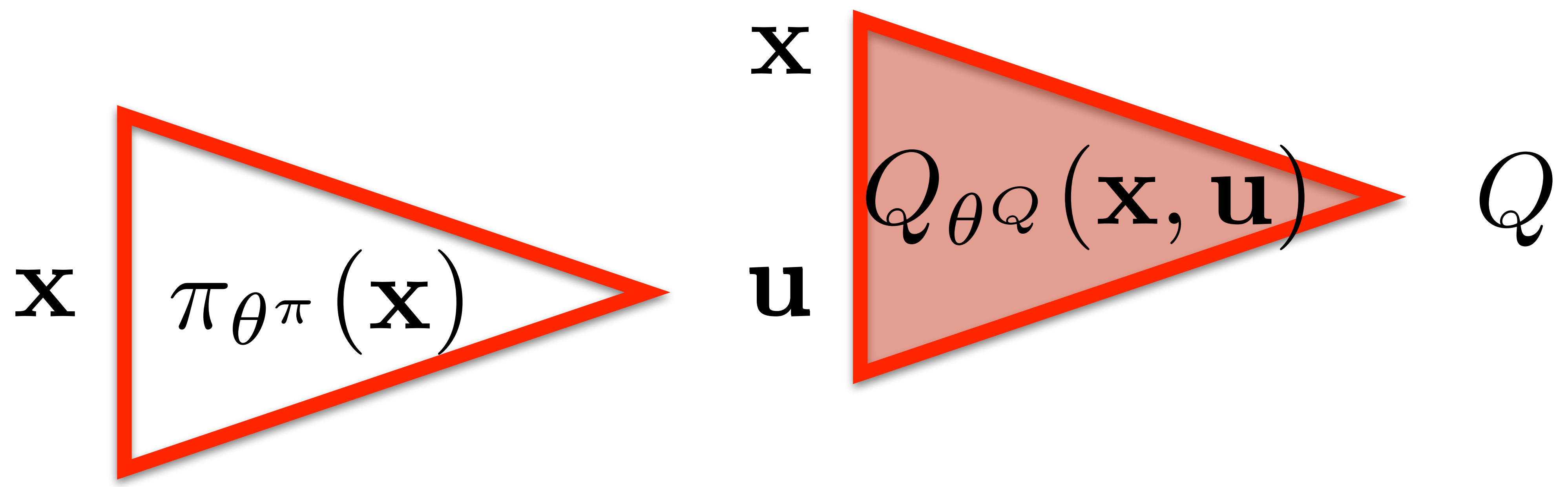
Deep Deterministic Policy Gradient (DDPG)



1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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

Deep Deterministic Policy Gradient (DDPG)

Update critic



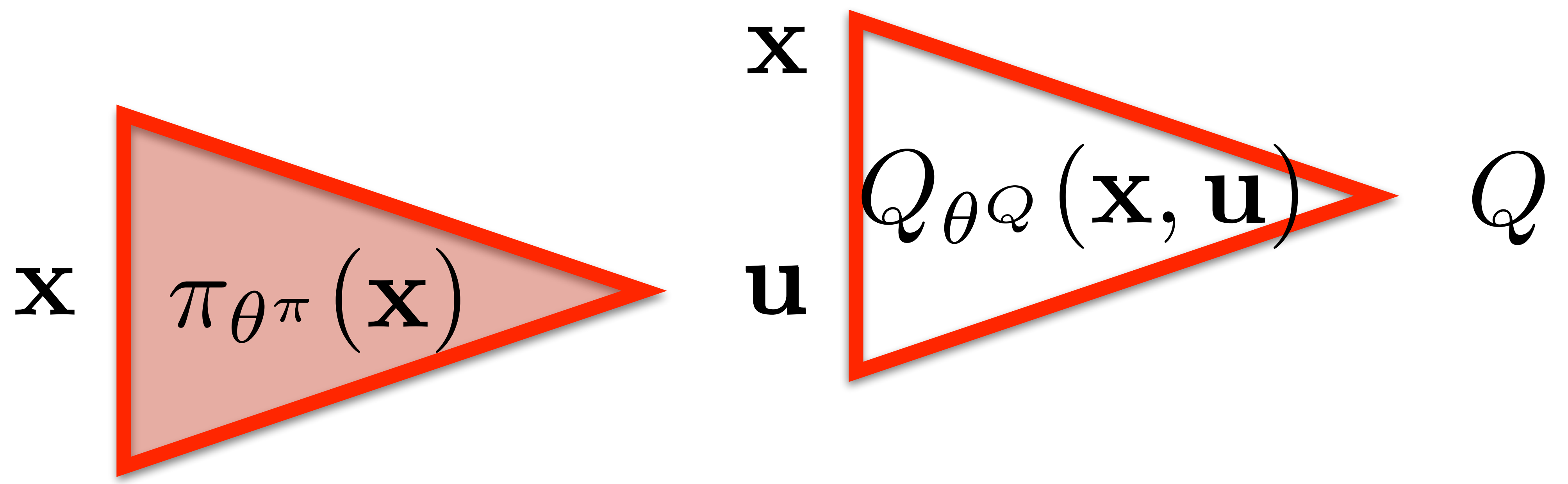
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma Q_{\overline{\theta^Q}}(\mathbf{x}', \pi_{\overline{\theta^\pi}}(\mathbf{x}'))$
4. Update critic $\arg\min_{\theta^Q} \sum_{\mathbf{x},\mathbf{u},\mathbf{y}}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

Deep Deterministic Policy Gradient (DDPG)

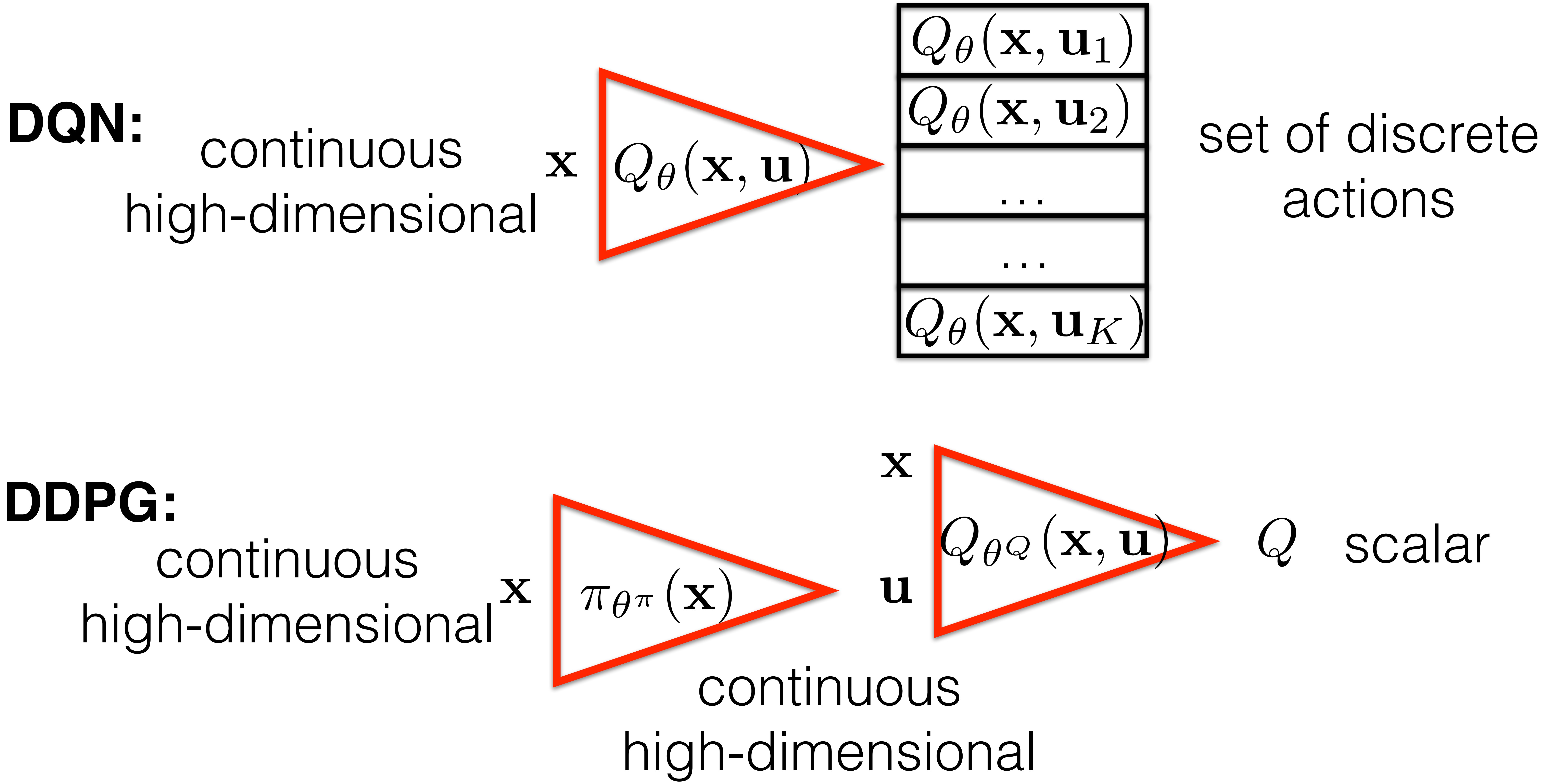
Update actor



1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> 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)
  => Can use ReplayMemory
  => Can use deterministic policy (exploration by synth.noise)



Summary

- DQN and DDPG are off-policy algorithms
  (can learn from transitions collected by a different policy)
  - => Can use ReplayMemory (which includes outdated transitions)
  - => Can learn deterministic policy (while using synth.noise for exploration)

- Replay memory helps to decorrelate samples.
- Exploration with a slowly updating target network suppresses oscillations.
- Ensemble of different algorithms helps a lot.

Summary

- DQN and DDPG are off-policy algorithms
  (can learn from transitions collected by a different policy)
  - => Can use ReplayMemory (which includes outdated transitions)
  - => Can learn deterministic policy (while using synth.noise for exploration)

- Replay memory helps to decorrelate samples.
- Exploration with a slowly updating target network suppresses oscillations.
- Ensemble of different algorithms helps a lot.

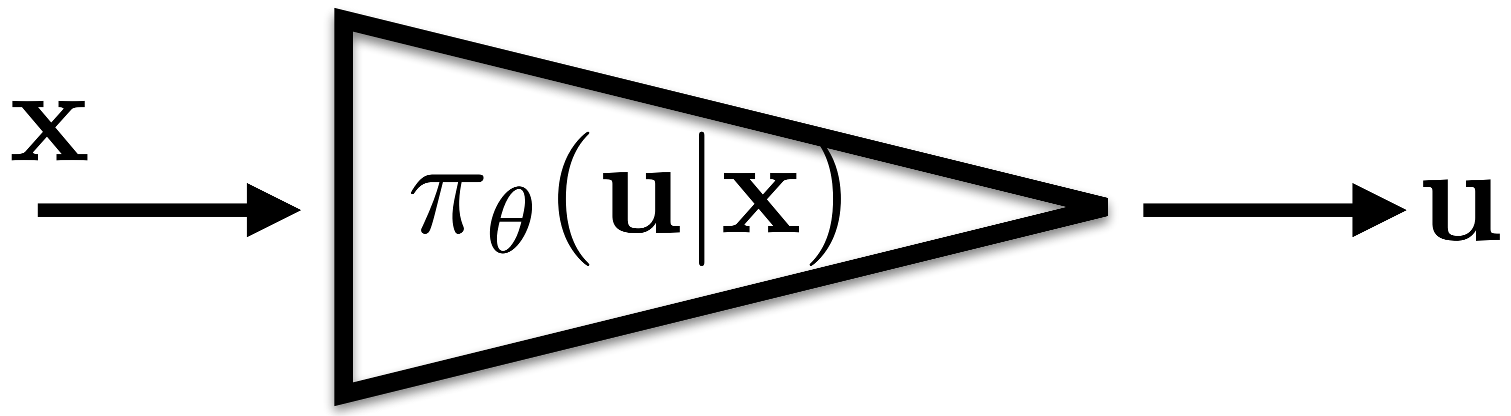
- Learning value function (Q,V,A) does not directly minimize

$$J(\theta) = \mathbb{E}_{r_k \sim \pi_\theta} \left[\sum_k \gamma^{k-1} r_k \right]$$

- Next: On-policy methods with stochastic gradient

Deterministic vs stochastic policy

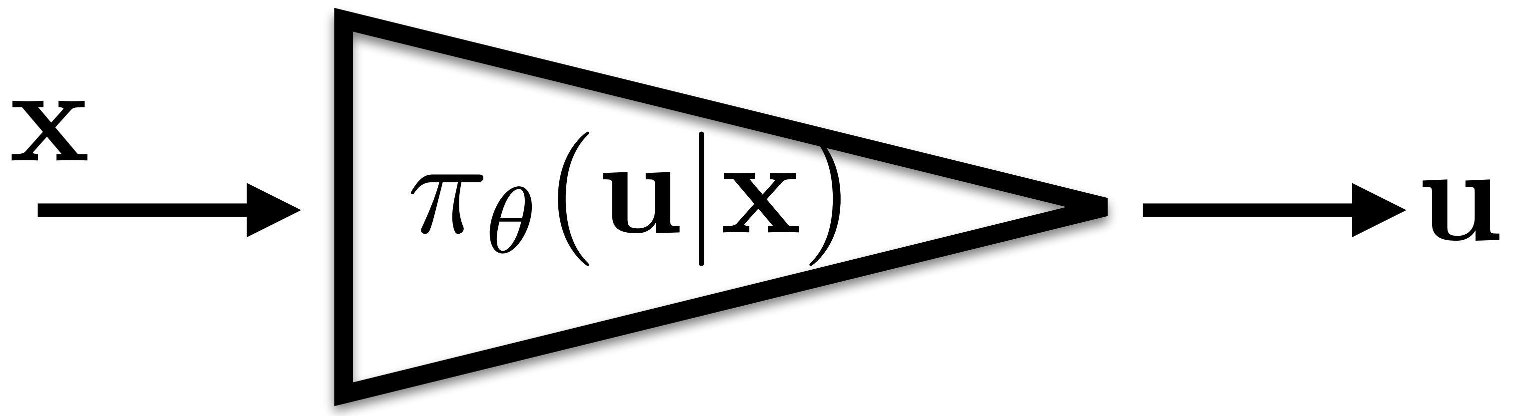
Deterministic policy for continuous control:



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

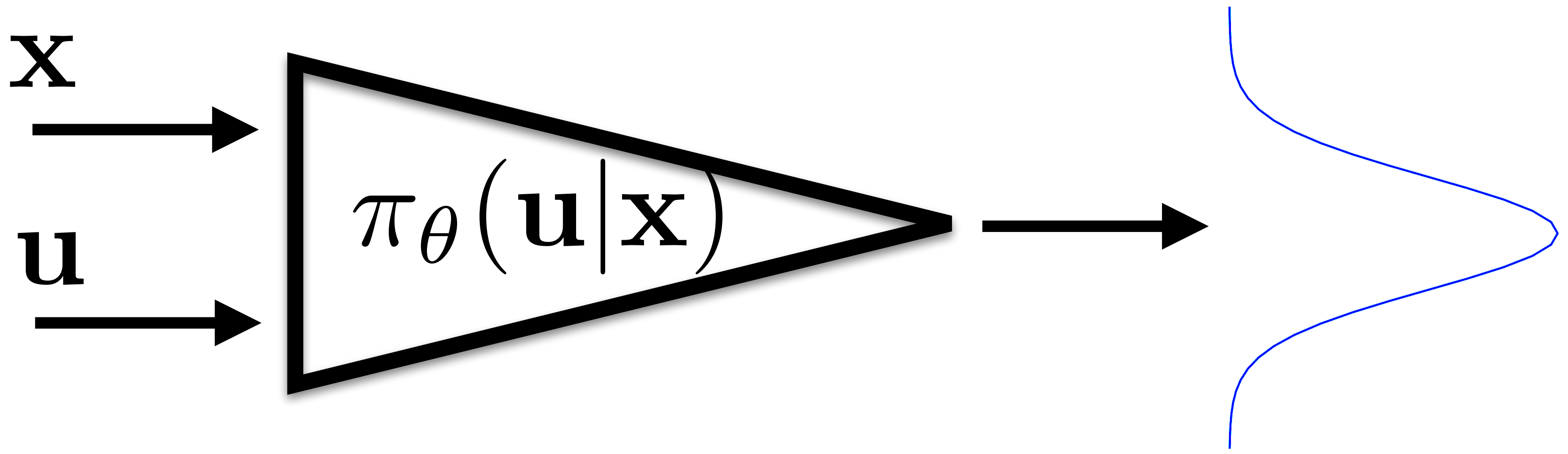
Deterministic vs stochastic policy

Deterministic policy for continuous control:



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

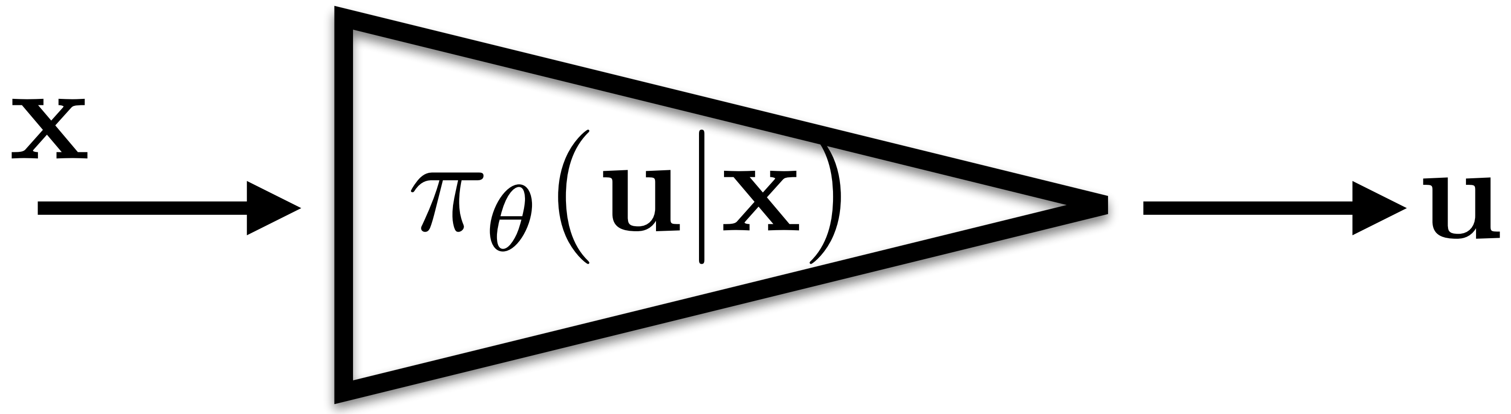
Stochastic policy for continuous control:



$$\pi_\theta(\mathbf{u}|\mathbf{x}) = C \cdot \exp\left(- \frac{(f(\mathbf{x}, \theta_\mu) - \mathbf{u})^2}{\theta_\sigma^2} \right)$$

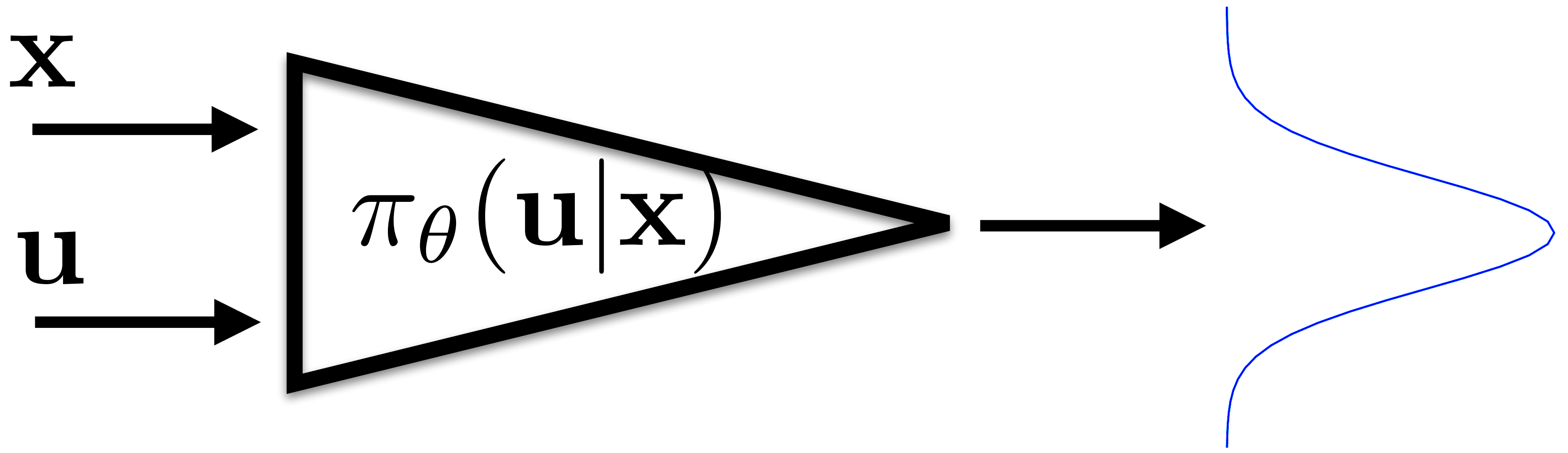
Deterministic vs stochastic policy

Deterministic policy for continuous control:



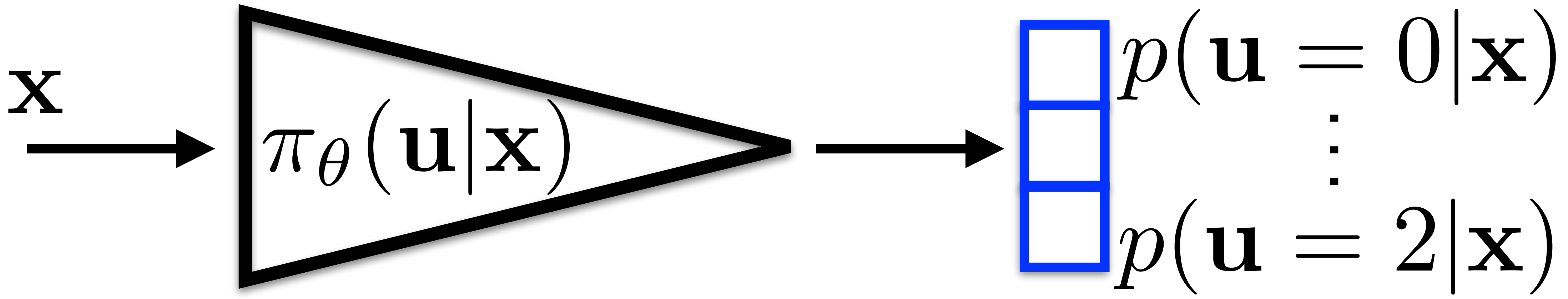
$$\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$

Stochastic policy for continuous control:



$$\pi_\theta(\mathbf{u}|\mathbf{x}) = C \cdot \exp\left(- \frac{(f(\mathbf{x}, \theta_\mu) - \mathbf{u})^2}{\theta_\sigma^2} \right)$$

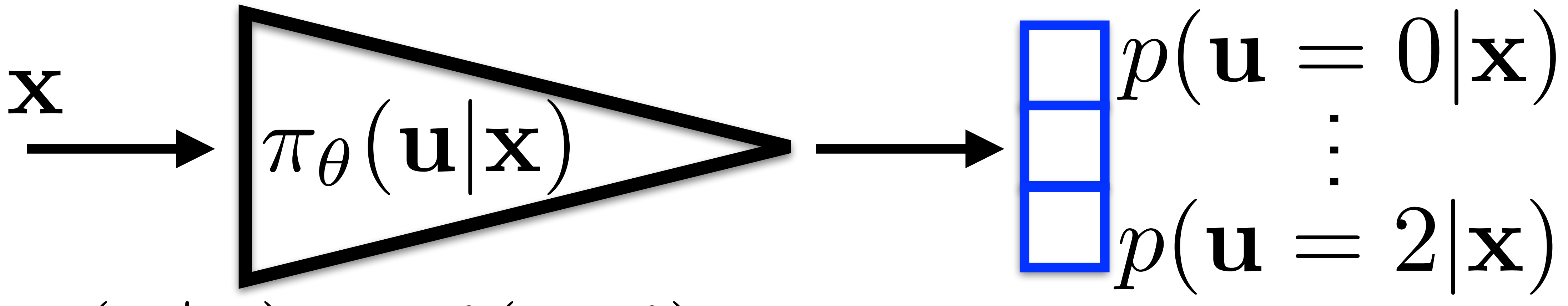
Stochastic policy for discrete control:



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

REINFORCE

Stochastic policy for discrete control:

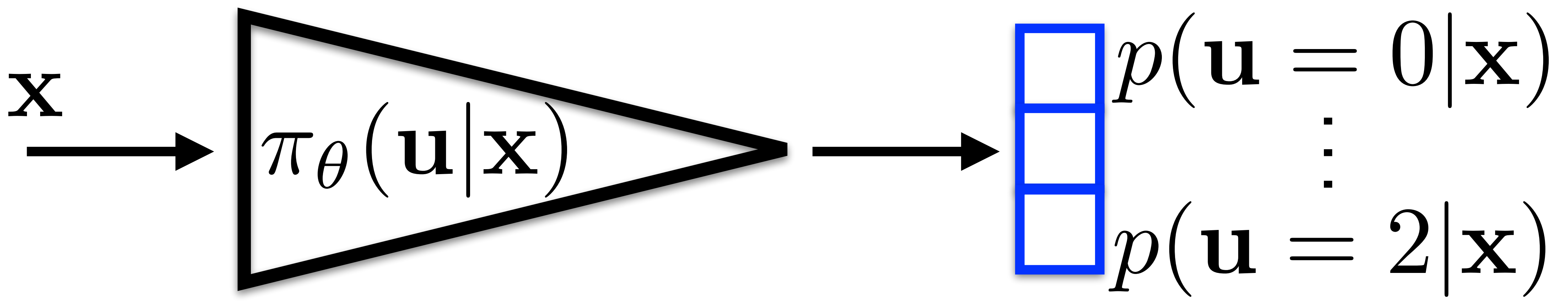


$$\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$

1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$
2. Collect trajectories τ with policy π_θ
3. Define criterion: $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\{\underbrace{\sum_{r_t \sim \tau} \gamma^t r_t}_{r(\tau)}\} \approx \frac{1}{N}\sum_{\tau} r(\tau)$
4. Optimize criterion: $\theta := \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$
5. Repeat from 2

REINFORCE

Stochastic policy for discrete control:



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

1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$
2. Collect trajectories τ with policy π_θ
3. Define criterion: $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\{\underbrace{\sum_{r_t\sim\tau} \gamma^t r_t}_{r(\tau)}\} \approx \frac{1}{N}\sum_{\tau} r(\tau)$
4. Optimize criterion: $\theta := \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$ **What is the gradient???**
5. Repeat from 2

What is the gradient???

- REINFORCE theorem:

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

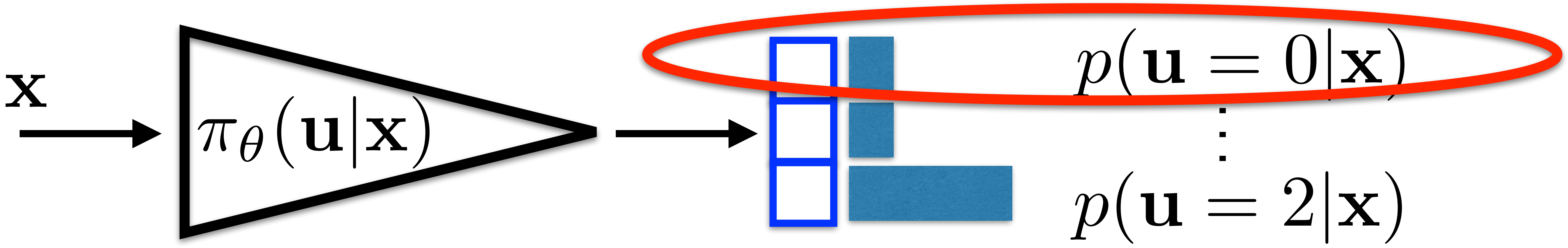
What is the gradient???

- REINFORCE theorem:

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

Gradient is the weighted sum of directions (in θ-space), which increases probability of performed actions.

The weights are sum of rewards along the resulting trajectory.



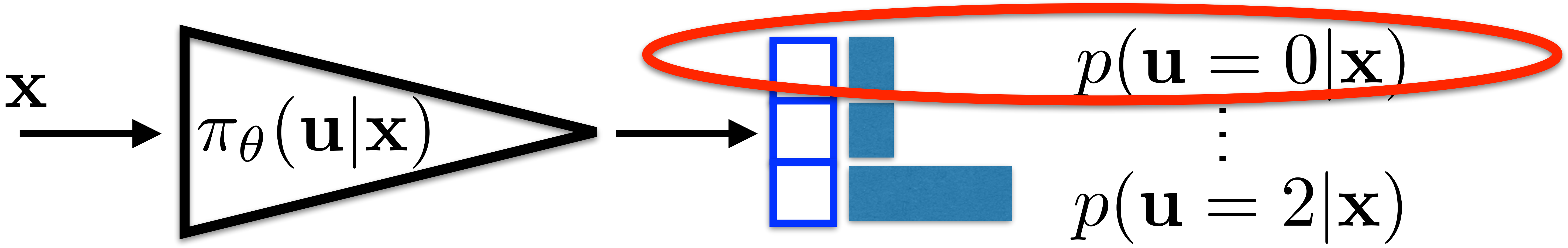
What is the gradient???

- REINFORCE theorem:

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

Gradient is the weighted sum of directions (in θ-space), which increases probability of performed actions.

The weights are sum of rewards along the resulting trajectory.



$r(\tau)$ =+100

Learning means increasing probability of predicting the actions, that have yielded high sum of rewards.

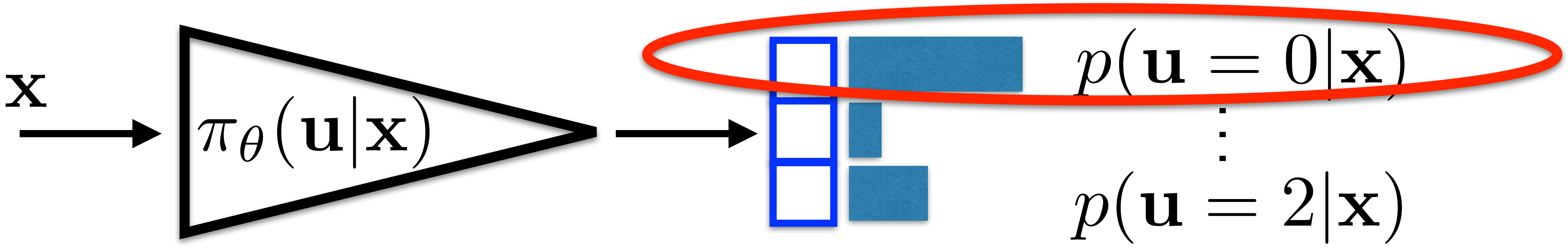
What is the gradient???

- REINFORCE theorem:

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

Gradient is the weighted sum of directions (in θ-space), which increases probability of performed actions.

The weights are sum of rewards along the resulting trajectory.



$r(\tau)$ =+100

Learning means increasing probability of predicting the actions, that have yielded high sum of rewards.

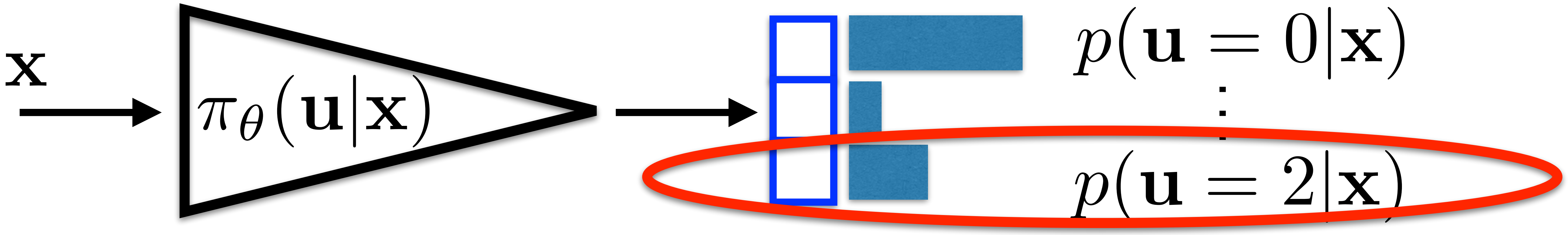
What is the gradient???

- REINFORCE theorem:

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

Gradient is the weighted sum of directions (in θ-space), which increases probability of performed actions.

The weights are sum of rewards along the resulting trajectory.



$r(\tau)$ =-100

Learning means increasing probability of predicting the actions, that have yielded high sum of rewards.

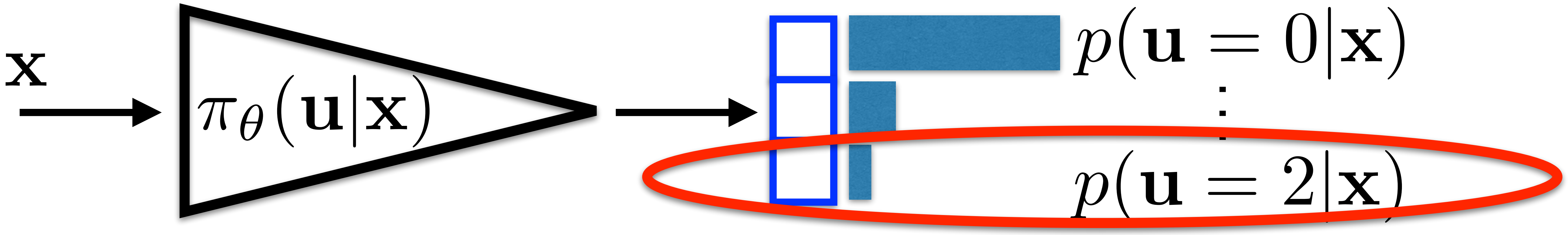
What is the gradient???

- REINFORCE theorem:

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

Gradient is the weighted sum of directions (in θ-space), which increases probability of performed actions.

The weights are sum of rewards along the resulting trajectory.

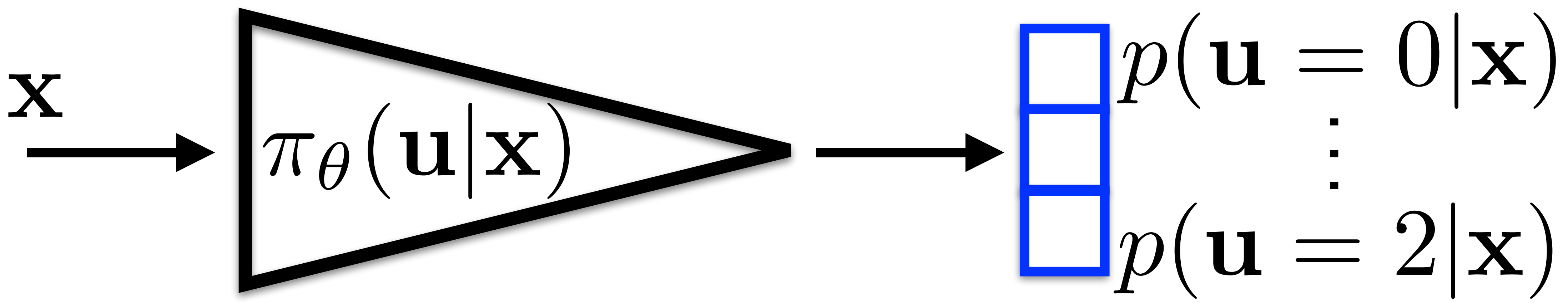


$r(\tau)$ =-100

Learning means increasing probability of predicting the actions, that have yielded high sum of rewards.

REINFORCE

Stochastic policy for discrete control:



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

4. Update policy (actor):

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

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

5. Repeat from 2

Policy gradient derivation

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

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

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

Policy gradient derivation

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

$$\frac{\partial J(\theta)}{\partial\theta} = \int_T \frac{\partial p(\tau|\pi_\theta)}{\partial\theta}r(\tau)d\tau = \int_T p(\tau|\pi_\theta)\frac{\partial \log p(\tau|\pi_\theta)}{\partial\theta}r(\tau)d\tau = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\frac{\partial \log p(\tau|\pi_\theta)}{\partial\theta}r(\tau)]$$

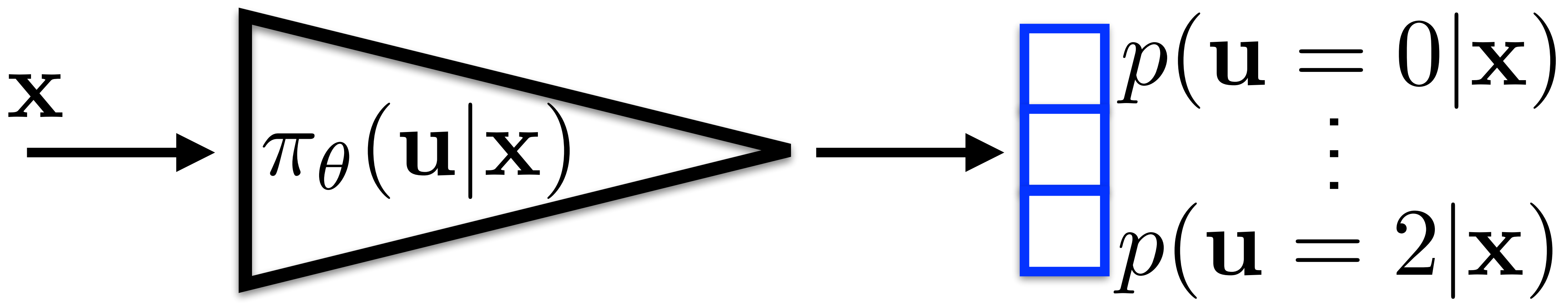
$$= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}\frac{\partial \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}{\partial\theta}r(\tau)] \approx \frac{1}{N}\sum_{\tau\in\mathcal{T}}\sum_{t=0}^{T}\frac{\partial \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}{\partial\theta}r(\tau)$$

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

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

REINFORCE

Stochastic policy for discrete control:



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

4. **Actor:** Update policy:

$$\frac{\partial J(\theta)}{\partial \theta} \approx \frac{1}{N} \sum_{\tau \in \mathcal{T}} \sum_{t=0}^{T} \frac{\partial \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}{\partial \theta} r(\tau)$$

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

5. Repeat from 2

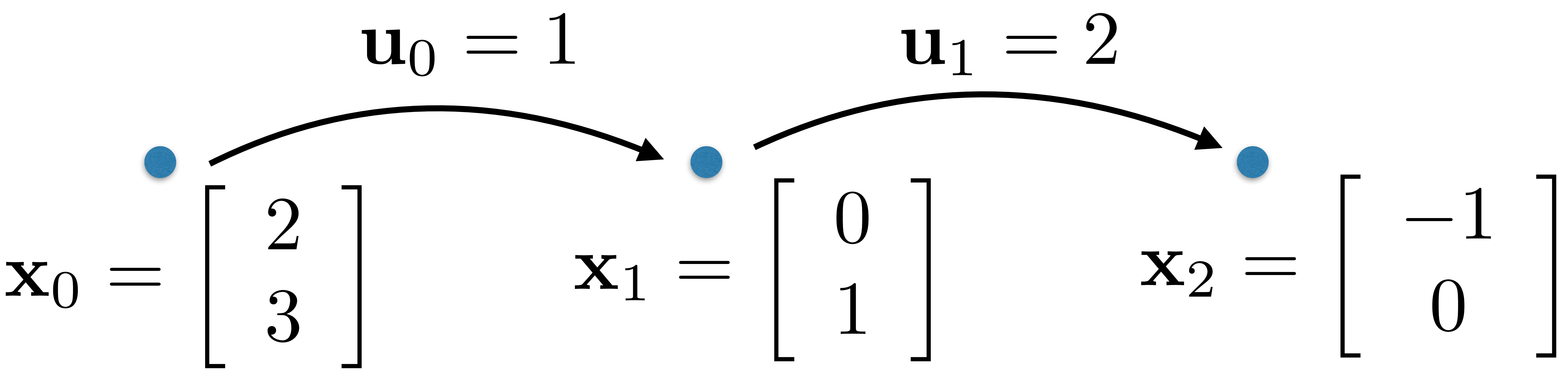
trajectory:

$$\mathbf{x}_0 = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \xrightarrow{\mathbf{u}_0 = 1} \mathbf{x}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \xrightarrow{\mathbf{u}_1 = 2} \mathbf{x}_2 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

policy: $\pi_\theta(\mathbf{u}|\mathbf{x}) = \mathbf{s}\left(\begin{bmatrix} \theta_1^\top \mathbf{x} \\ \theta_2^\top \mathbf{x} \end{bmatrix}\right)$

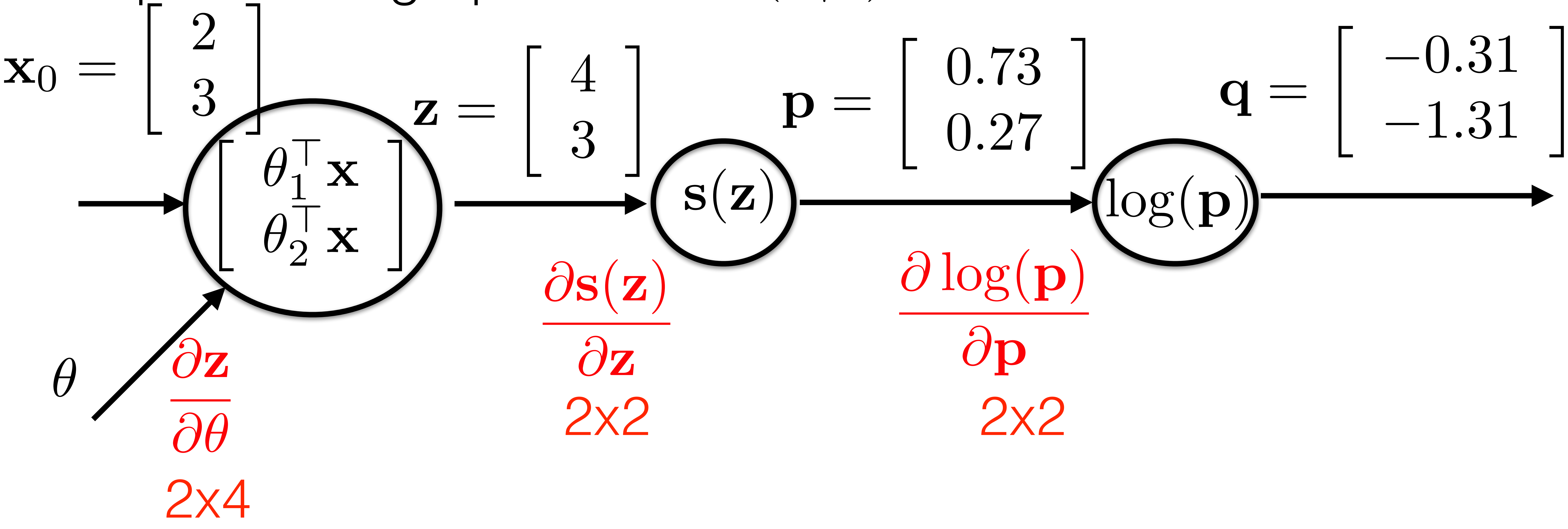
parameters: $\theta_1^\top = [2,\ 0]$, $\theta_2^\top = [0,\ 1]$

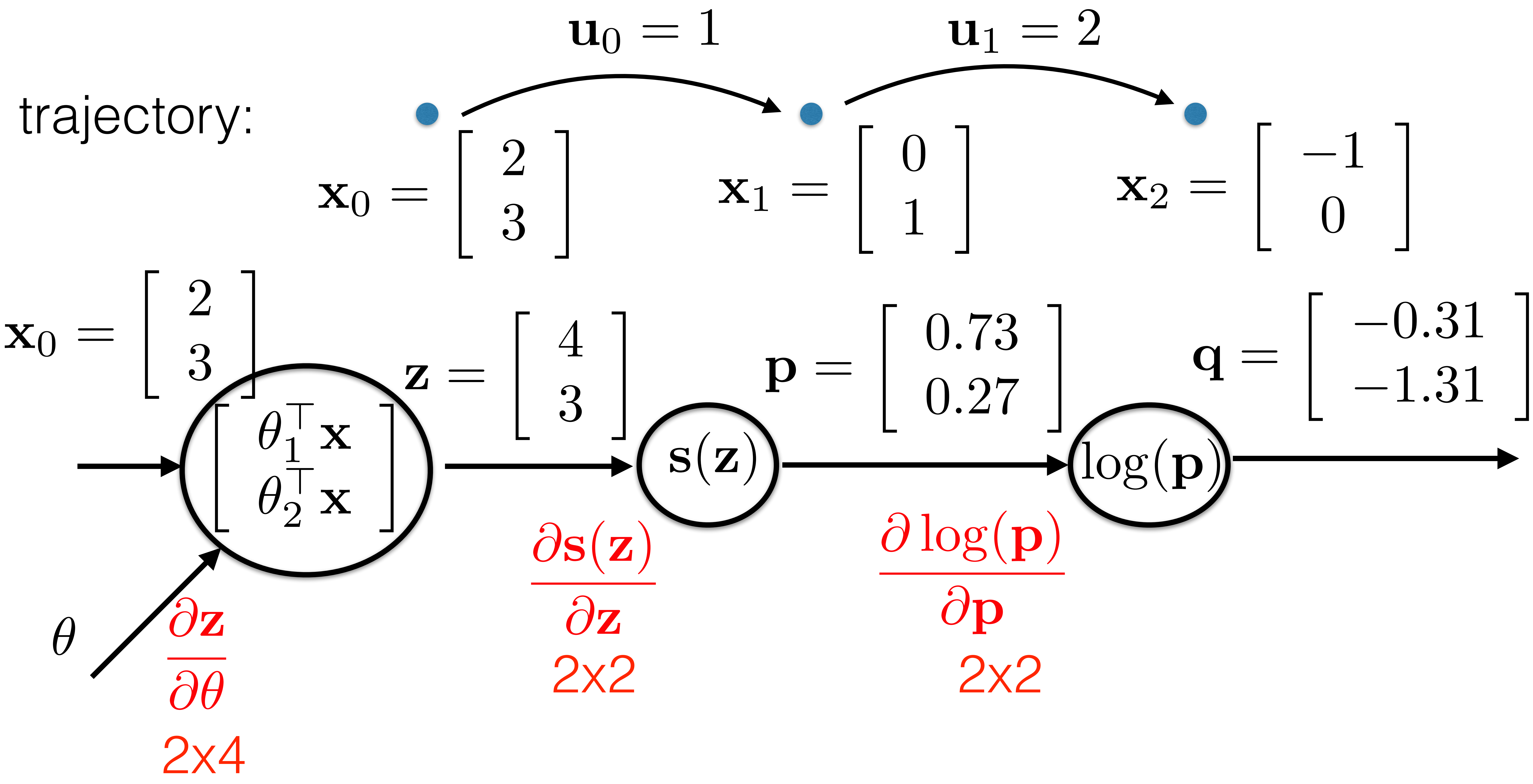
trajectory:



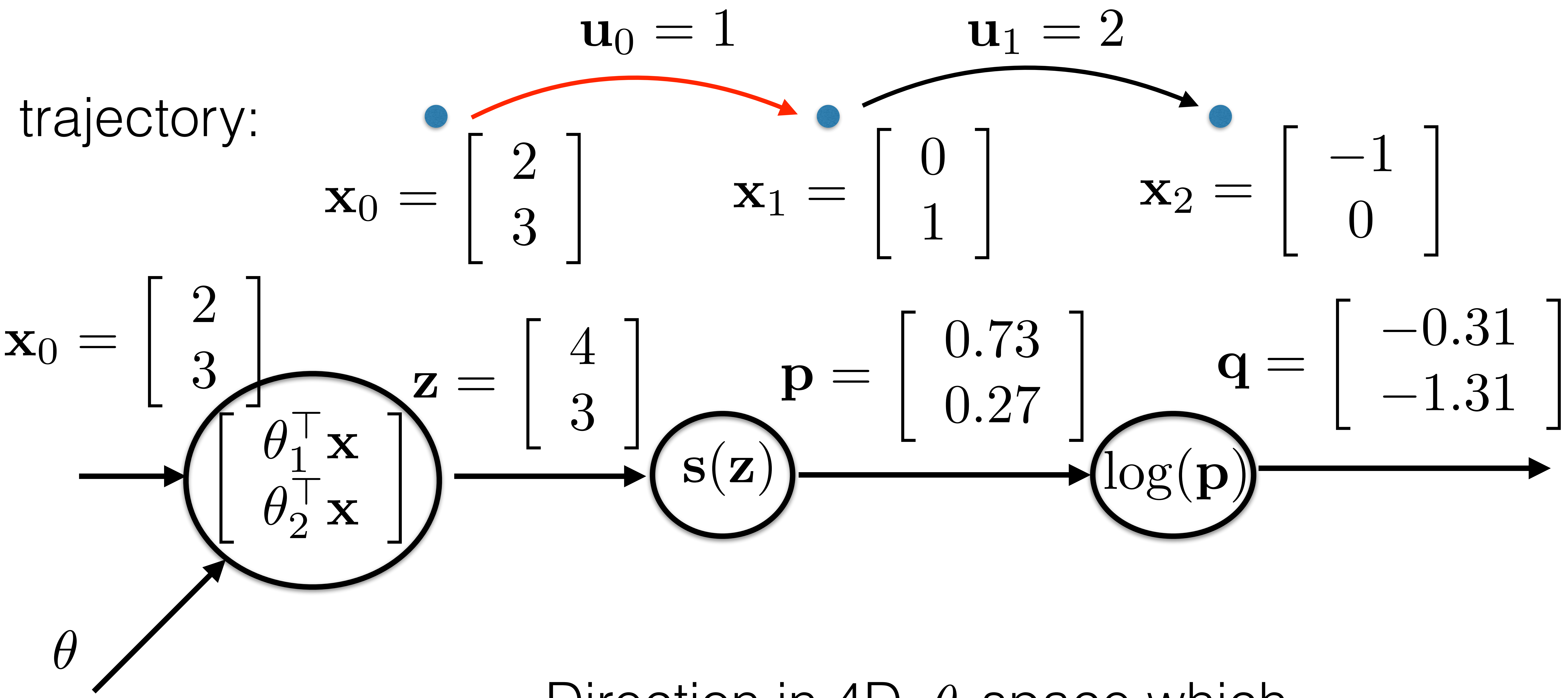
policy: $\pi_\theta(\mathbf{u}|\mathbf{x}) = \mathbf{s}\left(\begin{bmatrix} \theta_1^\top \mathbf{x} \\ \theta_2^\top \mathbf{x} \end{bmatrix}\right)$ parameters: $\theta_1^\top = [2,\ 0]$, $\theta_2^\top = [0,\ 1]$

computational graph of $\log \pi_\theta(\mathbf{u}|\mathbf{x})$:





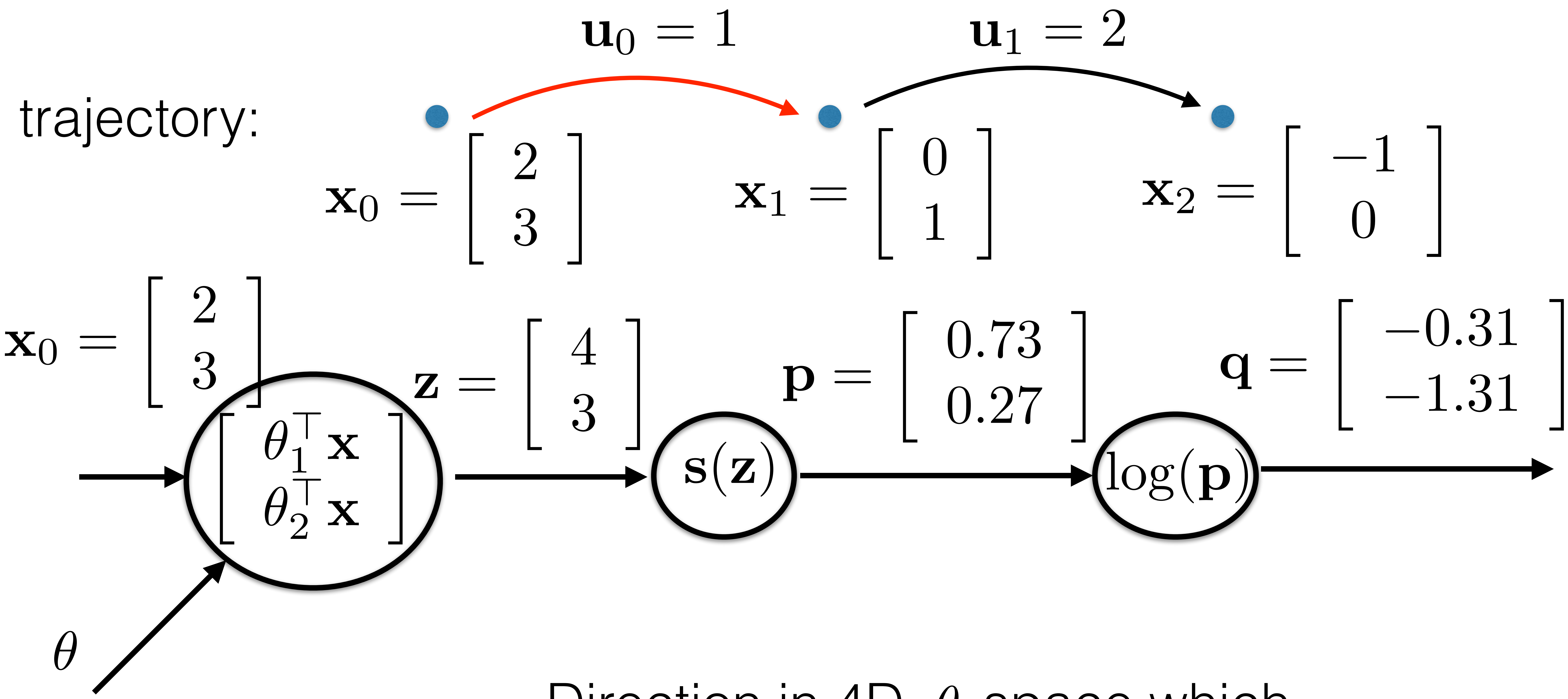
$$\frac{\partial \log \pi_\theta(\mathbf{u}|\mathbf{x})}{\partial \theta} = \textbf{???}$$



Direction in 4D θ-space which increases prob. of choosing control $\mathbf{u} = 1$

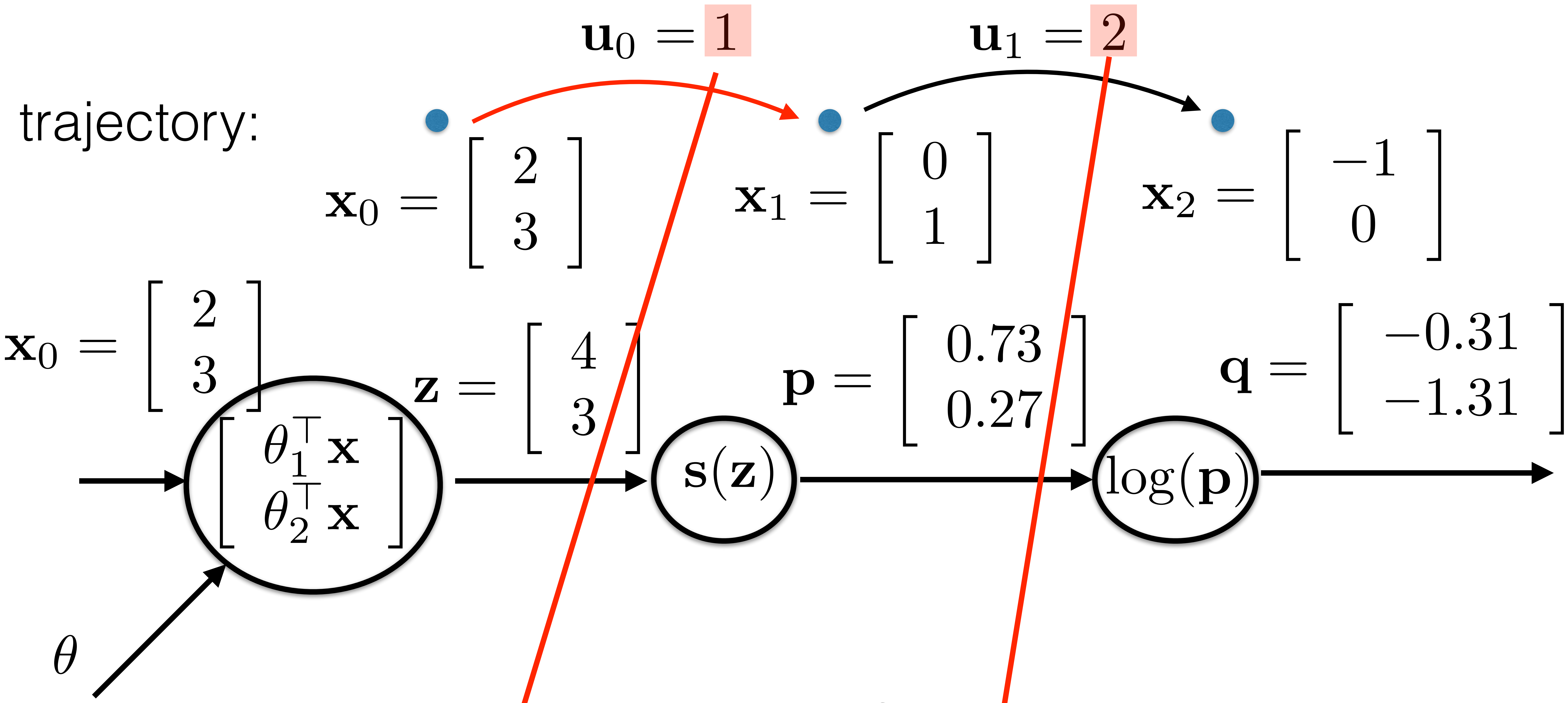
$$\frac{\partial \log \pi_\theta(\mathbf{u}|\mathbf{x})}{\partial \theta} = \frac{\partial \log(\mathbf{p})}{\partial \mathbf{p}} \frac{\partial \mathbf{s}(\mathbf{z})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \theta} = \begin{bmatrix} \mathbf{g}_1^\top(\mathbf{x}) \\ \quad \end{bmatrix}$$

2x2 2x2 2x4 2x4



Direction in 4D θ-space which increases prob. of choosing control $\mathbf{u} = 2$

$$\frac{\partial \log \pi_\theta(\mathbf{u}|\mathbf{x})}{\partial \theta} = \underset{2\text{x}2}{\frac{\partial \log(\mathbf{p})}{\partial \mathbf{p}}} \; \underset{2\text{x}2}{\frac{\partial \mathbf{s}(\mathbf{z})}{\partial \mathbf{z}}} \; \underset{2\text{x}4}{\frac{\partial \mathbf{z}}{\partial \theta}} = \underset{2\text{x}4}{\begin{bmatrix} \mathbf{g}_1^\top(\mathbf{x}) \\ \mathbf{g}_2^\top(\mathbf{x}) \end{bmatrix}}$$



By substituting actions and states from the trajectory into the policy gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial \log \pi_\theta(\mathbf{u}_0|\mathbf{x}_0)}{\partial \theta} \cdot r(\tau) + \frac{\partial \log \pi_\theta(\mathbf{u}_1|\mathbf{x}_1)}{\partial \theta} \cdot r(\tau) + \dots$$

$$= \boxed{\mathbf{g}_1^\top(\mathbf{x}_0)} \cdot r(\tau) + \boxed{\mathbf{g}_2^\top(\mathbf{x}_1)} \cdot r(\tau) + \dots$$

$$\mathbf{u}_0 = 1 \qquad \mathbf{u}_1 = 2$$

trajectory:

$$\mathbf{x}_0 = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \qquad \mathbf{x}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

By substituting controls and states from the trajectory into the policy gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial \log \pi_\theta(\mathbf{u}_0|\mathbf{x}_0)}{\partial \theta} \cdot r(\tau) + \frac{\partial \log \pi_\theta(\mathbf{u}_1|\mathbf{x}_1)}{\partial \theta} \cdot r(\tau) + \dots$$

$$= \boxed{\mathbf{g}_1^\top(\mathbf{x}_0)} \cdot r(\tau) + \boxed{\mathbf{g}_2^\top(\mathbf{x}_1)} \cdot r(\tau) + \dots$$

we obtain $r(\tau)$-weighted mean of directions in θ-space.

If trajectories are good, then $r(\tau)$-weights are big and this direction in 4D θ-space is more preferred.

Consequently, policy parameters are changed in the direction, which generates good trajectories

Policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot r(\tau)$$

Policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \left(\sum_{k=1}^{T} \gamma^{k-1} r(\mathbf{u}_k, \mathbf{x}_k) \right)$$

Policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \Big(\sum_{k=1}^{T} \gamma^{k-1} r(\mathbf{u}_k, \mathbf{x}_k) \Big)$$

Policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \Big(\sum_{k=t}^{T} \gamma^{k-1} r(\mathbf{u}_k, \mathbf{x}_k) \Big)$$

Policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \Big(\sum_{k=t}^{T} \gamma^{k-1} r(\mathbf{u}_k, \mathbf{x}_k) \Big)$$

- state-action function (policy gradient theorem):

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot Q(\mathbf{u}_t, \mathbf{x}_t)$$

Policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \Big(\sum_{k=t}^{T} \gamma^{k-1} r(\mathbf{u}_k, \mathbf{x}_k) \Big)$$

- state-action function (policy gradient theorem):

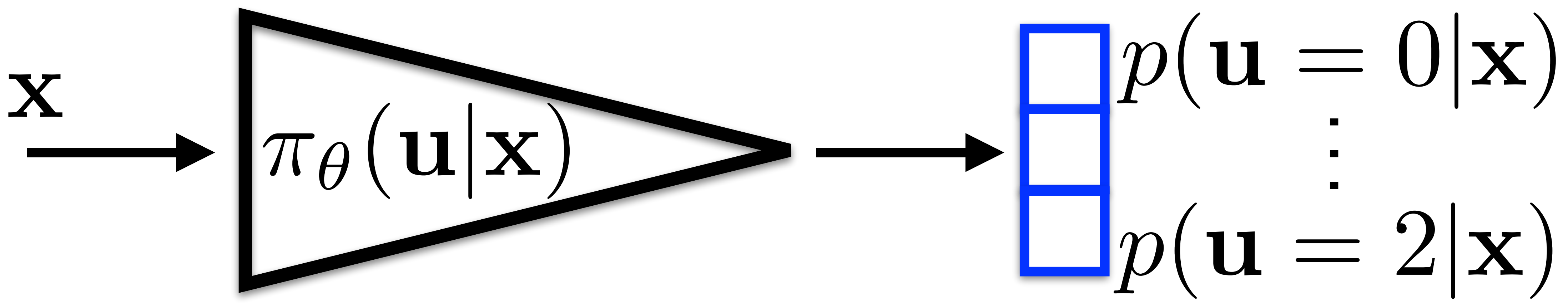
$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot Q(\mathbf{u}_t, \mathbf{x}_t)$$

- arbitrary baseline can be subtracted (Q-function => A-function)

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{(\mathbf{u}_t, \mathbf{x}_t) \in \tau} \frac{\partial \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)}{\partial \theta} \cdot \underbrace{\Big(Q(\mathbf{u}_t, \mathbf{x}_t) - V(\mathbf{x}_t) \Big)}_{A(\mathbf{u}_t, \mathbf{x}_t)}$$

REINFORCE

Stochastic policy for discrete control:



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

4. **Actor:** Update policy:

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

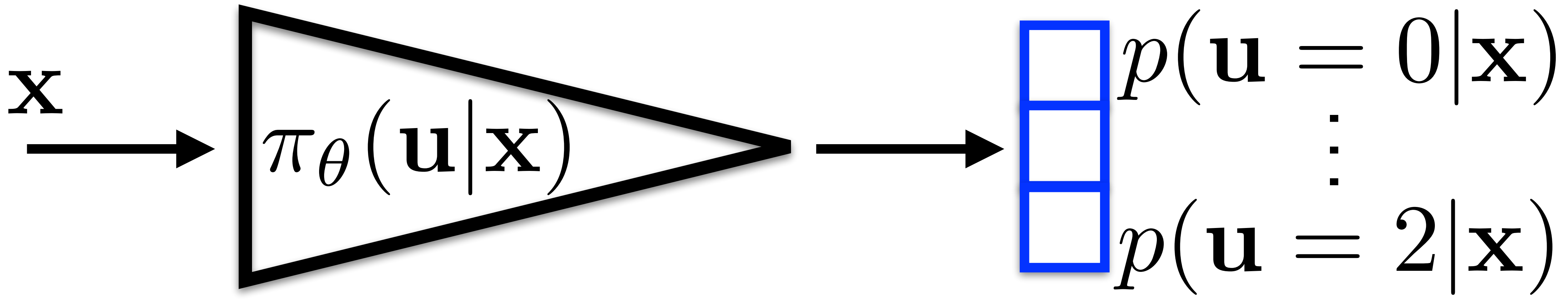
$$\theta := \theta + \alpha \frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$$

5. Repeat from 2

Several equivalent ways to express the quality of trajectory

Advantage Actor Critic (A2C)

Stochastic policy for discrete control:



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:

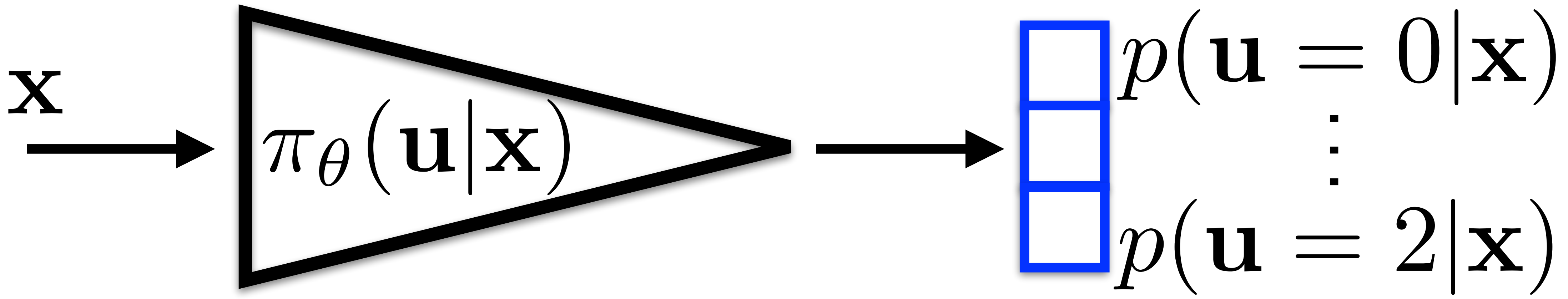
$$\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta} = \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \frac{\partial \log \pi_\theta(\mathbf{u}|\mathbf{x})}{\partial \theta} \cdot \underbrace{\Big(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x})\Big)}_{A_\omega = Q - V}$$

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

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for discrete control:



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:

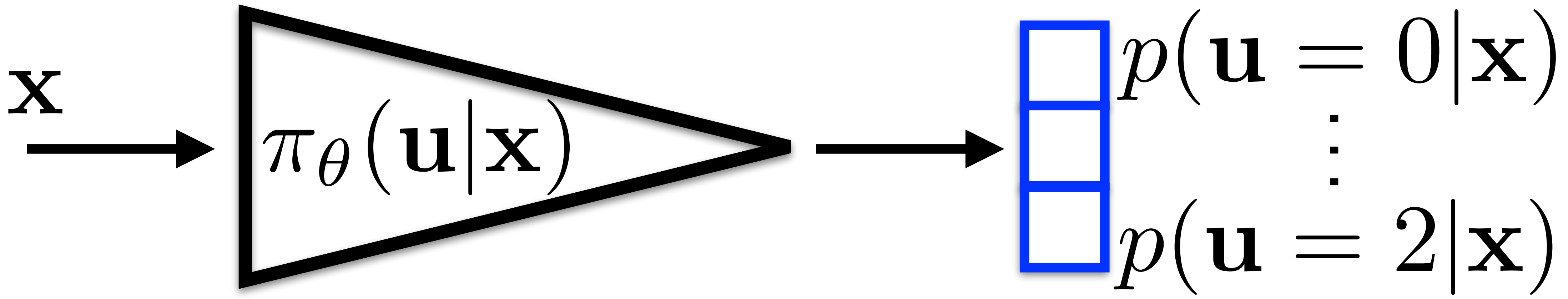
$$\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta} = \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \frac{\partial \log \pi_\theta(\mathbf{u}|\mathbf{x})}{\partial \theta} \cdot \underbrace{\left(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x})\right)}_{A_\omega = Q - V}$$

Use arbitrary optimizer (e.g. Adam) which makes use of $\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for discrete control:



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:

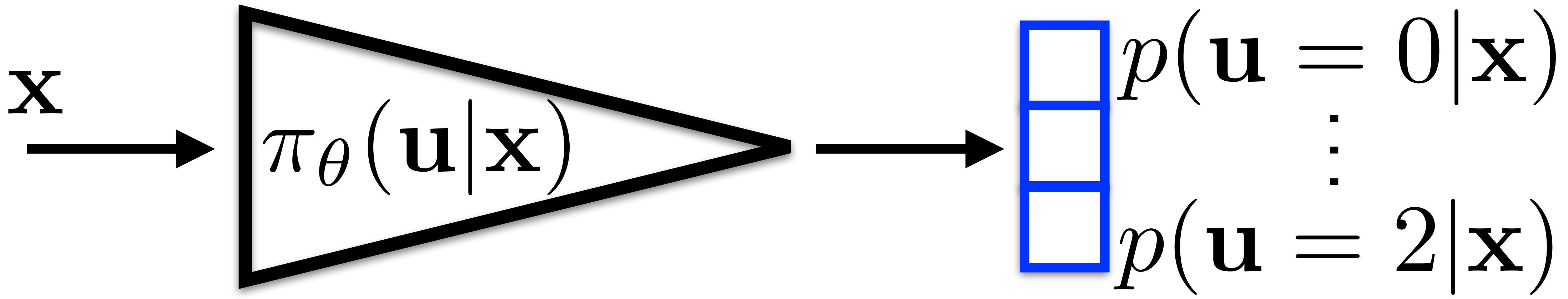
$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \log \pi_\theta(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\Big(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x})\Big)}_{A_\omega = Q - V}$$

Use arbitrary optimizer (e.g. Adam) which makes use of $\dfrac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for discrete control:



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: $V_\omega(\mathbf{x}) \leftarrow r + \gamma V_\omega(\mathbf{x}')$
4. **Actor:** Update policy by policy gradient:

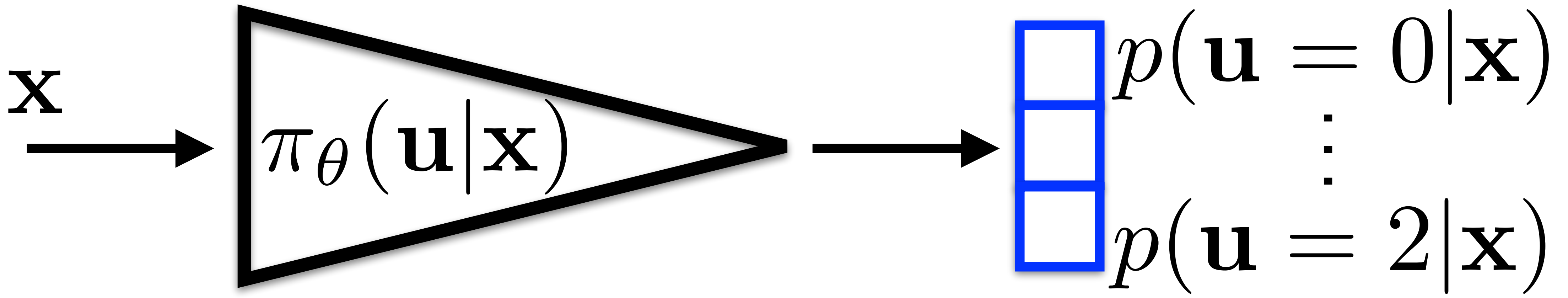
$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \log \pi_\theta(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\Big(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x})\Big)}_{A_\omega = Q - V}$$

Use arbitrary optimizer (e.g. Adam) which makes use of $\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$, $\frac{\partial \mathcal{L}_{\text{critic}}(\omega)}{\partial \omega}$

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for discrete control:



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: $V_\omega(\mathbf{x}) \leftarrow r + \gamma V_\omega(\mathbf{x}')$

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

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

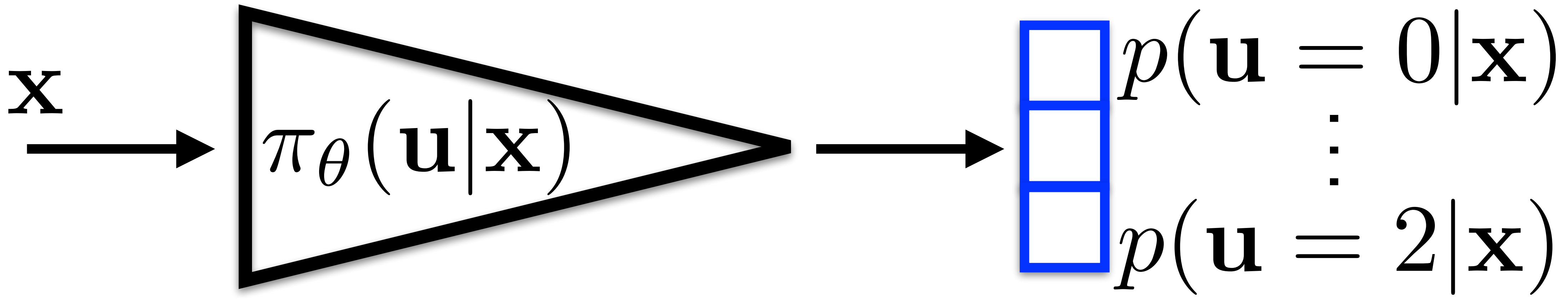
$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \log \pi_\theta(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\Big(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x})\Big)}_{A_\omega = Q - V}$$

Use arbitrary optimizer (e.g. Adam) which makes use of $\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$, $\frac{\partial \mathcal{L}_{\text{critic}}(\omega)}{\partial \omega}$

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for discrete control:



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

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

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

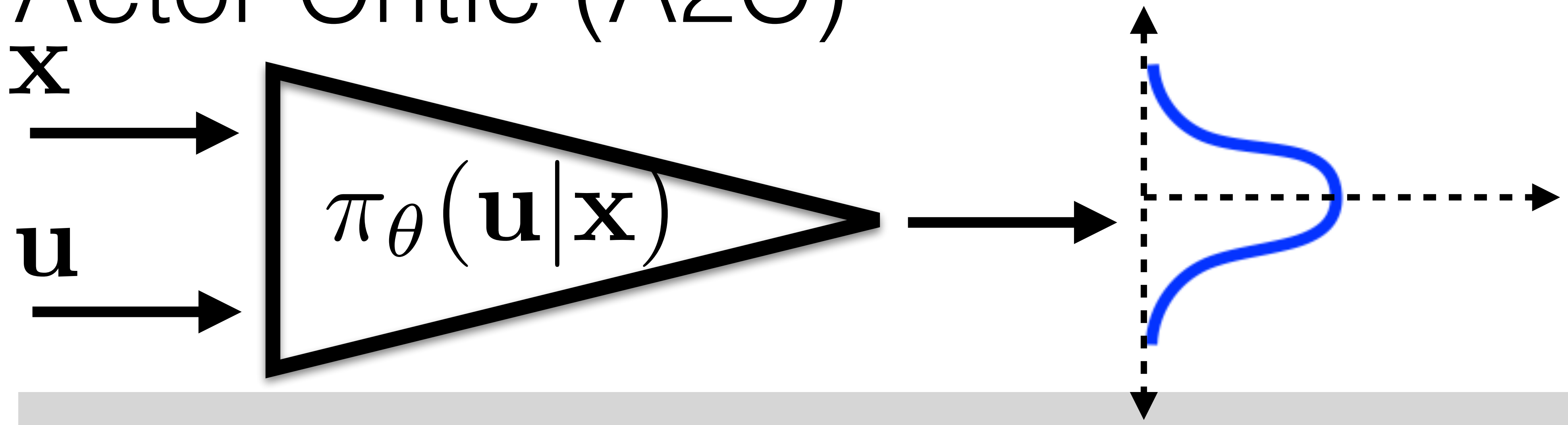
$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \log \pi_\theta(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\Big(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x})\Big)}_{A_\omega = Q - V}$$

Use arbitrary optimizer (e.g. Adam) which makes use of $\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$, $\frac{\partial \mathcal{L}_{\text{critic}}(\omega)}{\partial \omega}$

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for continuous control:



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

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

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

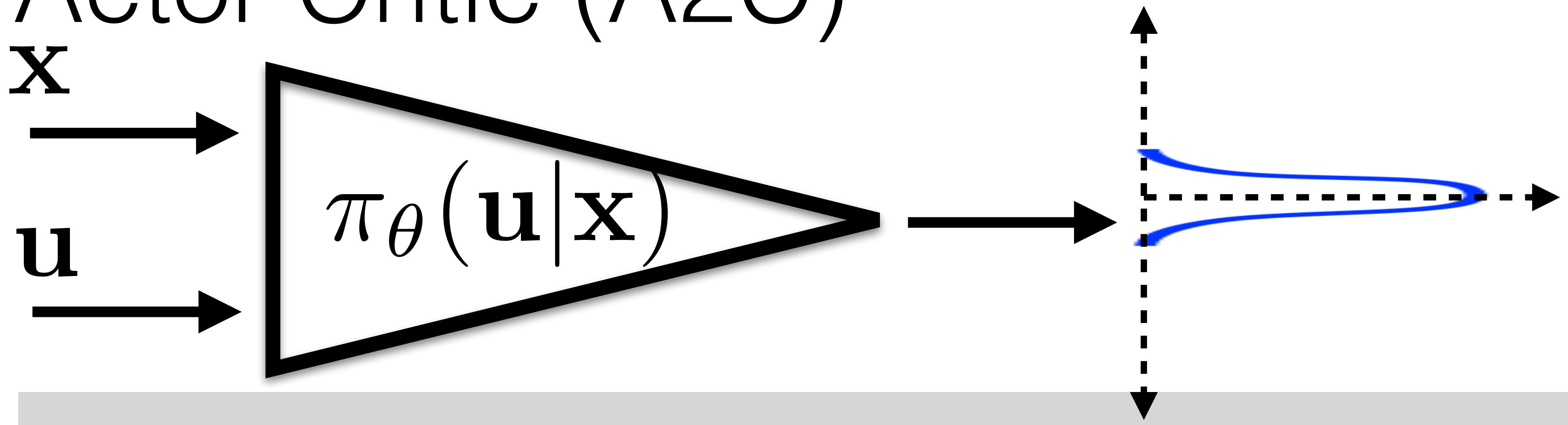
$$\mathcal{L}_{\text{actor}}(\theta) = \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \log \pi_\theta(\mathbf{u}|\mathbf{x}) \cdot \underbrace{\Big(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x})\Big)}_{A_\omega = Q - V}$$

Use arbitrary optimizer (e.g. Adam) which makes use of $\dfrac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$, $\dfrac{\partial \mathcal{L}_{\text{critic}}(\omega)}{\partial \omega}$

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for continuous control:



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

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

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

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

Use arbitrary optimizer (e.g. Adam) which makes use of $\frac{\partial \mathcal{L}_{\text{actor}}(\theta)}{\partial \theta}$, $\frac{\partial \mathcal{L}_{\text{critic}}(\omega)}{\partial \omega}$

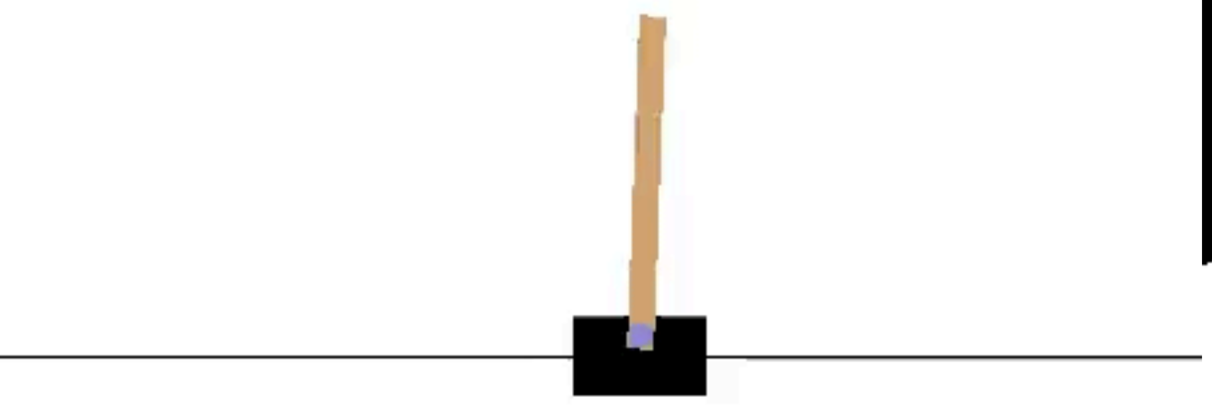
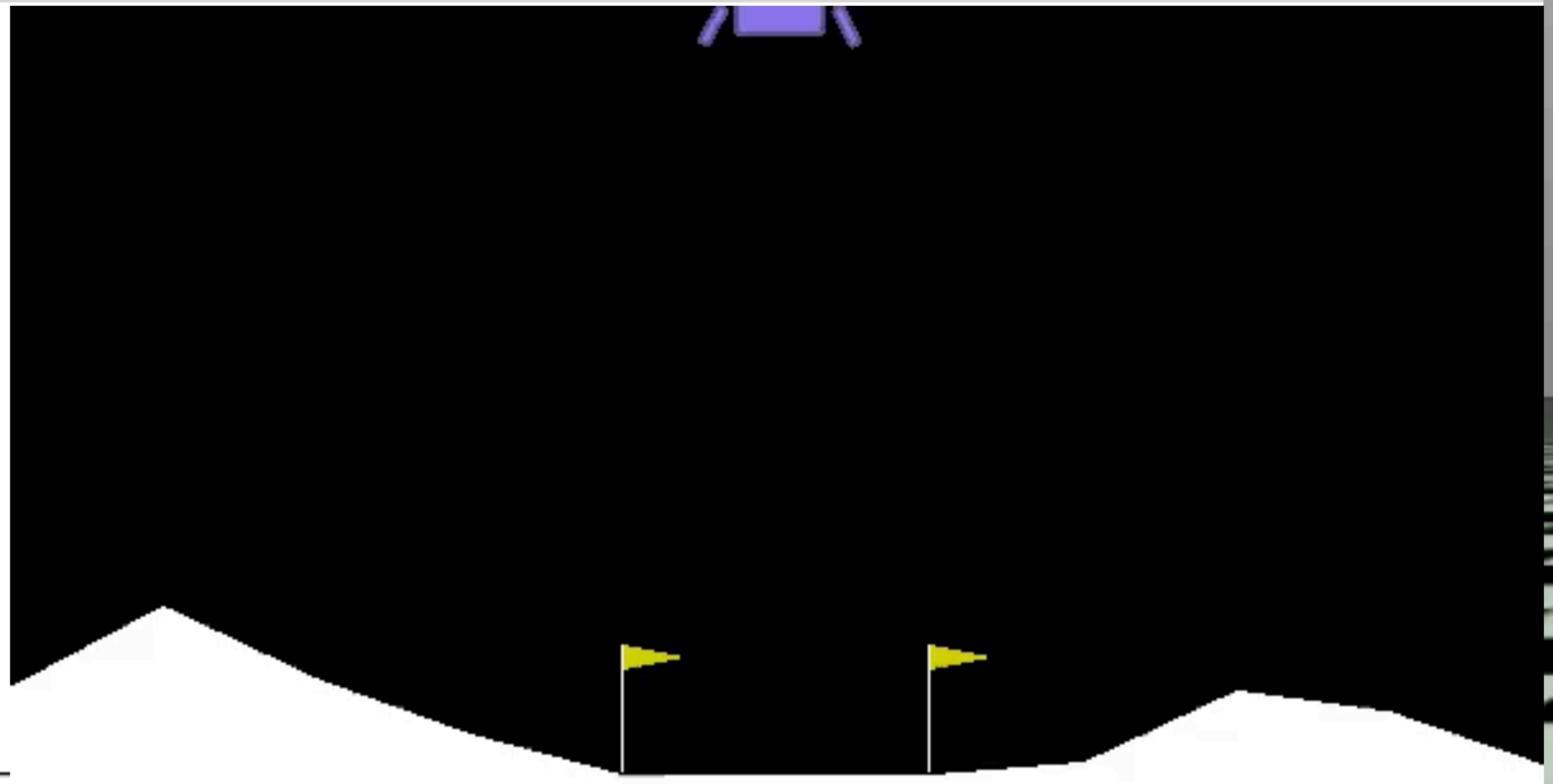
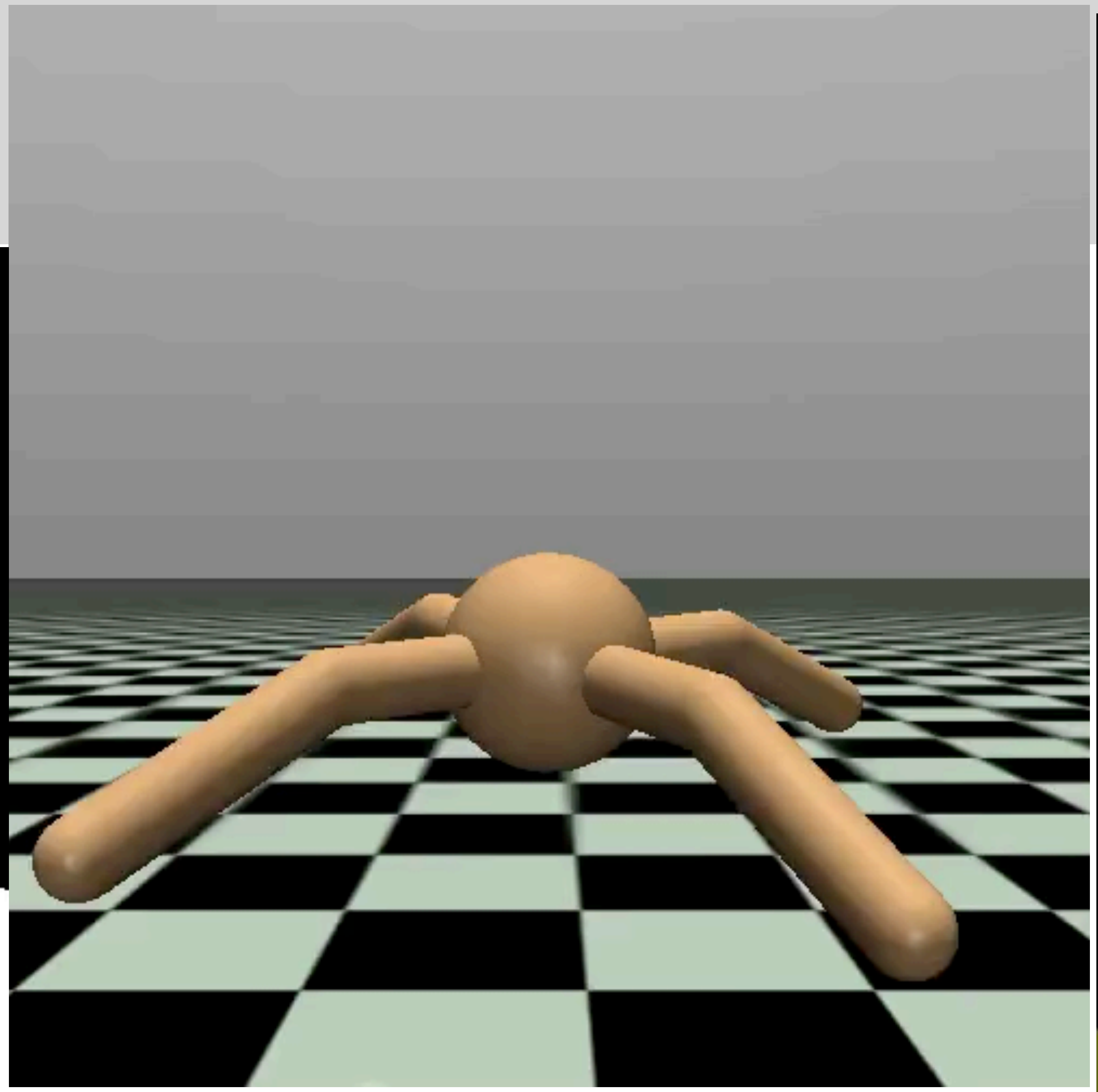
5. Repeat from 2

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/

```python
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 AlphaStart beaten top-end professional human gamers 5:0)

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

Known successes of RL - Starcraft II

- **Starcraft II game**
  - no single best strategy
  - imperfect information (unlike fully observable chess)
  - longterm planning (significantly delayed rewards for upgrades)
  - realtime (unlike traditional board games)
  - large action space (hundreds of buildings and possible locations, units and commands, upgrades)
- **Starcraft II client + dataset** of anonymised game plays:
- https://github.com/Blizzard/s2client-proto#replay-packs
- [DeepMind + Blizzard 2017] joint paper: https://kstatic.googleusercontent.com/files/8f5c46f2ca6f2dc1944e86fe852ecfa2072cc3729ceb6af4dc84307a939b60ac8915c82ead4e7e4d4862d0436a8a329a6f06a4d538b741219e85c207c5e04f62

Known successes of RL - Starcraft II
Minigames allows for traing small RL agents

Known successes of RL - Starcraft II

Learning consists of two phases:

- **Supervised learning** from anonymised human games (performance: (i) humans - gold level, (ii) AI - elite level
- **Reinforcement learning**: 14 days playing against two grand masters (TLO, MaNa)

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

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:

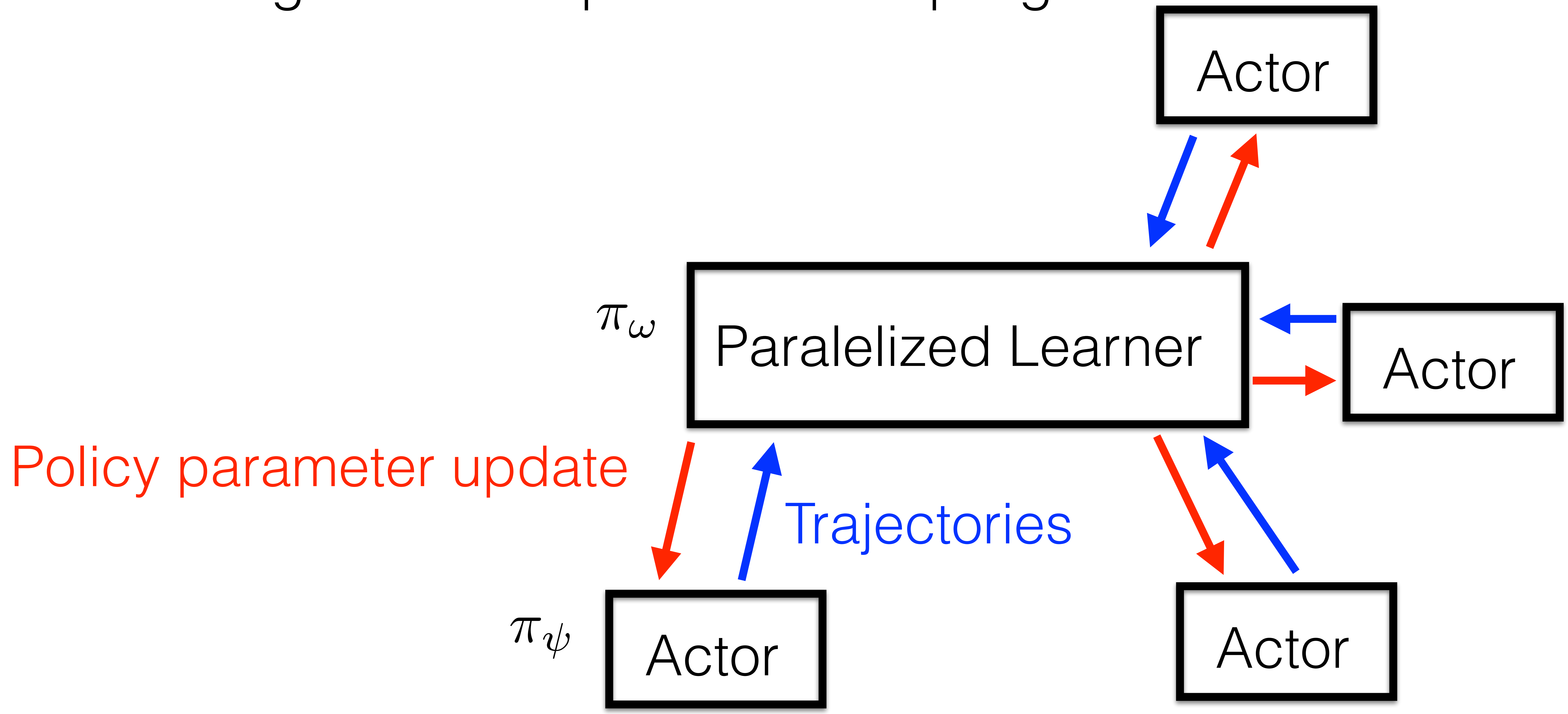
$$\mathbb{E}_{\tau \sim p(\tau|\pi_\omega)} \left[\sum_{(\mathbf{x},\mathbf{u}) \in \tau} \frac{\partial \log \pi_\omega(\mathbf{u}|\mathbf{x})}{\partial \omega} \cdot Q_\theta(\mathbf{x}, \mathbf{u}) \right] \approx$$

$$\approx \sum_k \frac{\partial \log \pi_\omega(\mathbf{u}_k|\mathbf{x}_k)}{\partial \omega} \cdot Q_\theta(\mathbf{x}_k, \mathbf{u}_k)$$

  - recurrent policy architecture with LSTM blocks
  - paralelized learning

paralelized learning

- Actors delayed wrt learner => policy being updated π_ω is different from the one which collected trajectories π_ψ
- rewards re-weighted via importance sampling



paralelized learning

- importance sampling

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

π_ω …current

π_ψ …old

paralelized learning

- importance sampling

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

π_ω …current

π_ψ …old

$$\mathbb{E}_{\tau\sim p(\tau|\pi_\omega)}[g(\tau)] = \int_T p(\tau|\pi_\omega)g(\tau) = \int_T \frac{p(\tau|\pi_\omega)}{p(\tau|\pi_\psi)}p(\tau|\pi_\psi)g(\tau) =$$

$$= \mathbb{E}_{\tau\sim p(\tau|\pi_\psi)}\left[g(\tau)\frac{p(\tau|\pi_\omega)}{p(\tau|\pi_\psi)}\right]$$

recalibrated estimate

paralelized learning

- importance sampling

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

π_ω …current

π_ψ …old

$$\mathbb{E}_{\tau\sim p(\tau|\pi_\omega)}\left[g(\tau)\right]=\int_T p(\tau|\pi_\omega)g(\tau)=\int_T\frac{p(\tau|\pi_\omega)}{p(\tau|\pi_\psi)}p(\tau|\pi_\psi)g(\tau)=$$

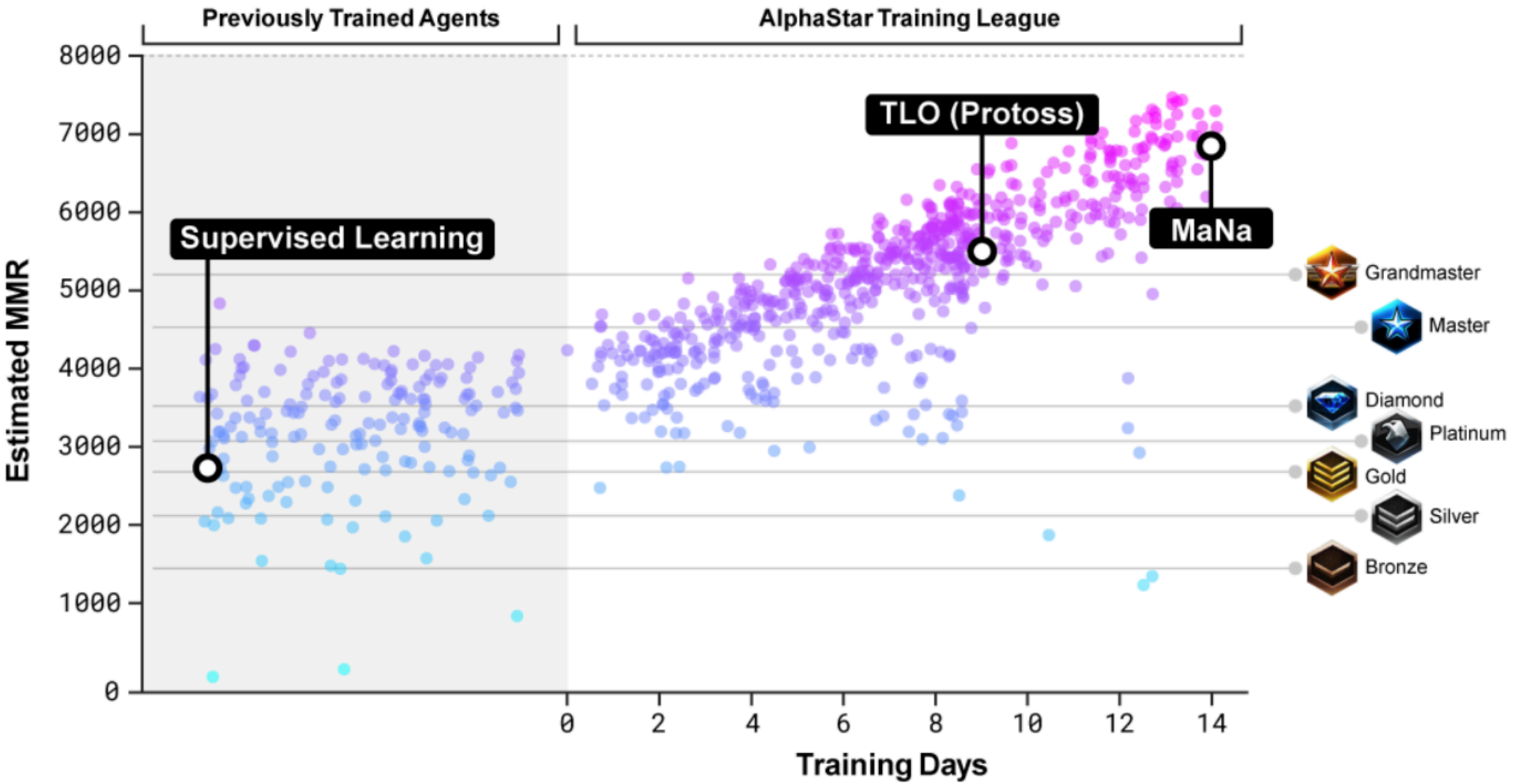
$$=\mathbb{E}_{\tau\sim p(\tau|\pi_\psi)}\left[g(\tau)\frac{p(\tau|\pi_\omega)}{p(\tau|\pi_\psi)}\right]$$

- recalibrated policy gradient estimate

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

Known successes of RL - Starcraft II

- Supervised training on Blizzards database + 14 days self-play against RL agents (faster binary => approx 200 years)

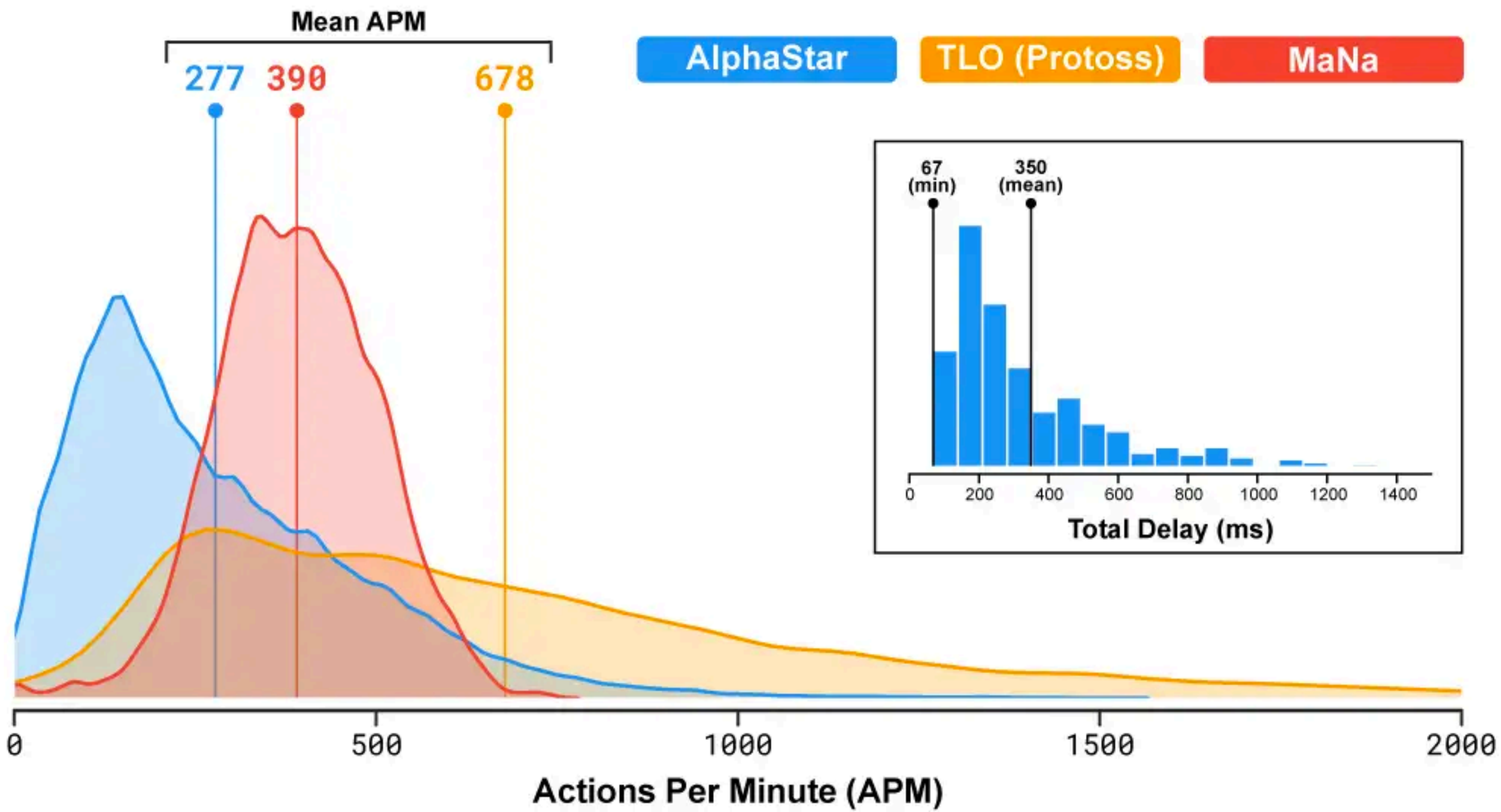


Render of Agent's view
AlphaStar
MaNa
Outcome Prediction
Win
Draw
Lose
Raw Observations
Neural Network Activations
Considered Location
Considered Build/Train
Nexus
Pylon
Assimilator
Gateway
Forge
FleetBeacon
TwilightCouncil
PhotonCannon
Stargate
TemplarArchive
DarkShrine
RoboticsBay
RoboticsFacility
CyberneticsCore
ShieldBattery
Mothership
Zealot
Stalker
HighTemplar
DarkTemplar
Sentry
Adept
Phoenix
Carrier
VoidRay
Oracle
Tempest
WarpPrism
Observer
Colossus
Immortal
Probe
Disruptor


ROUND
REPLAY
1. ALPHASTAR WINS
2. ALPHASTAR WINS
3. ALPHASTAR WINS
4. ALPHASTAR WINS
5. ALPHASTAR WINS
SCORE
TLO 0 - 5 ALPHASTAR
TLO


ROUND
REPLAY
1. ALPHASTAR WINS
2. ALPHASTAR WINS
3. ALPHASTAR WINS
4. ALPHASTAR WINS
5. ALPHASTAR WINS
SCORE
MANA 0 - 5 ALPHASTAR
GRZEGORZ 'MANA' KOMINCZ

Known successes of RL - Starcraft II

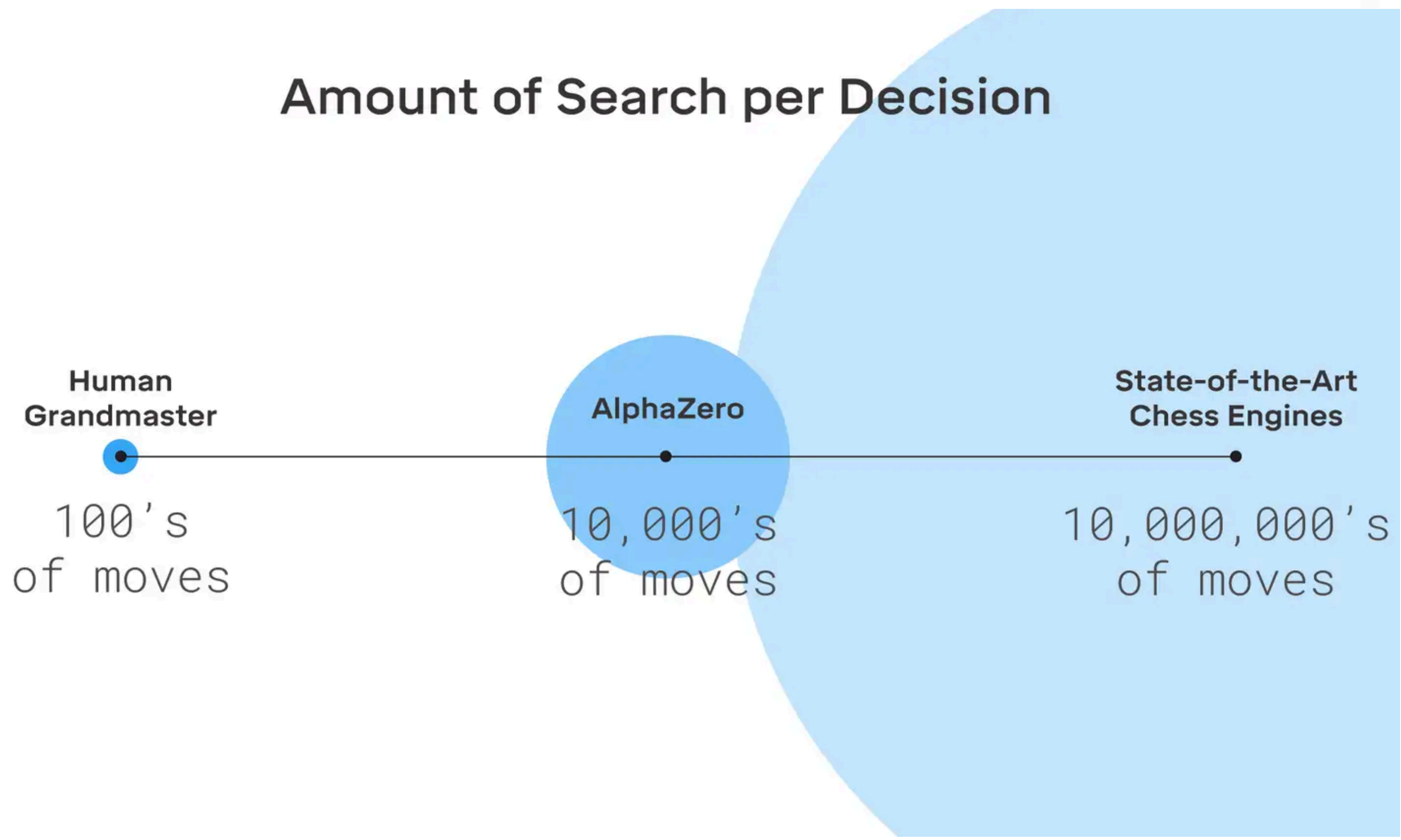


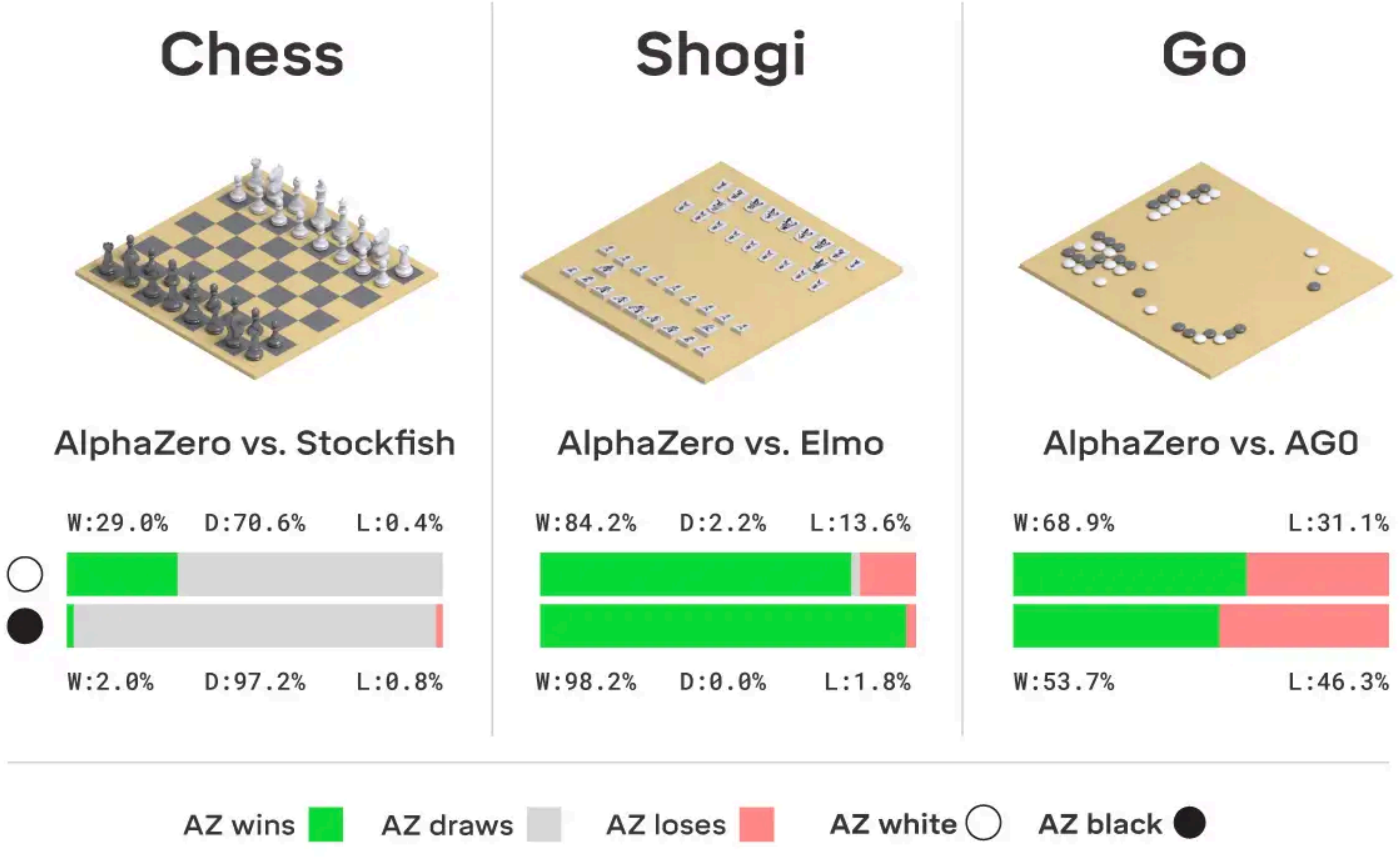
Known successes of RL - Starcraft II

- AlphaStar does not move camera (uses zoomed-out raw interface).
- Of course, haze of war is used.

Known successes of RL - board games

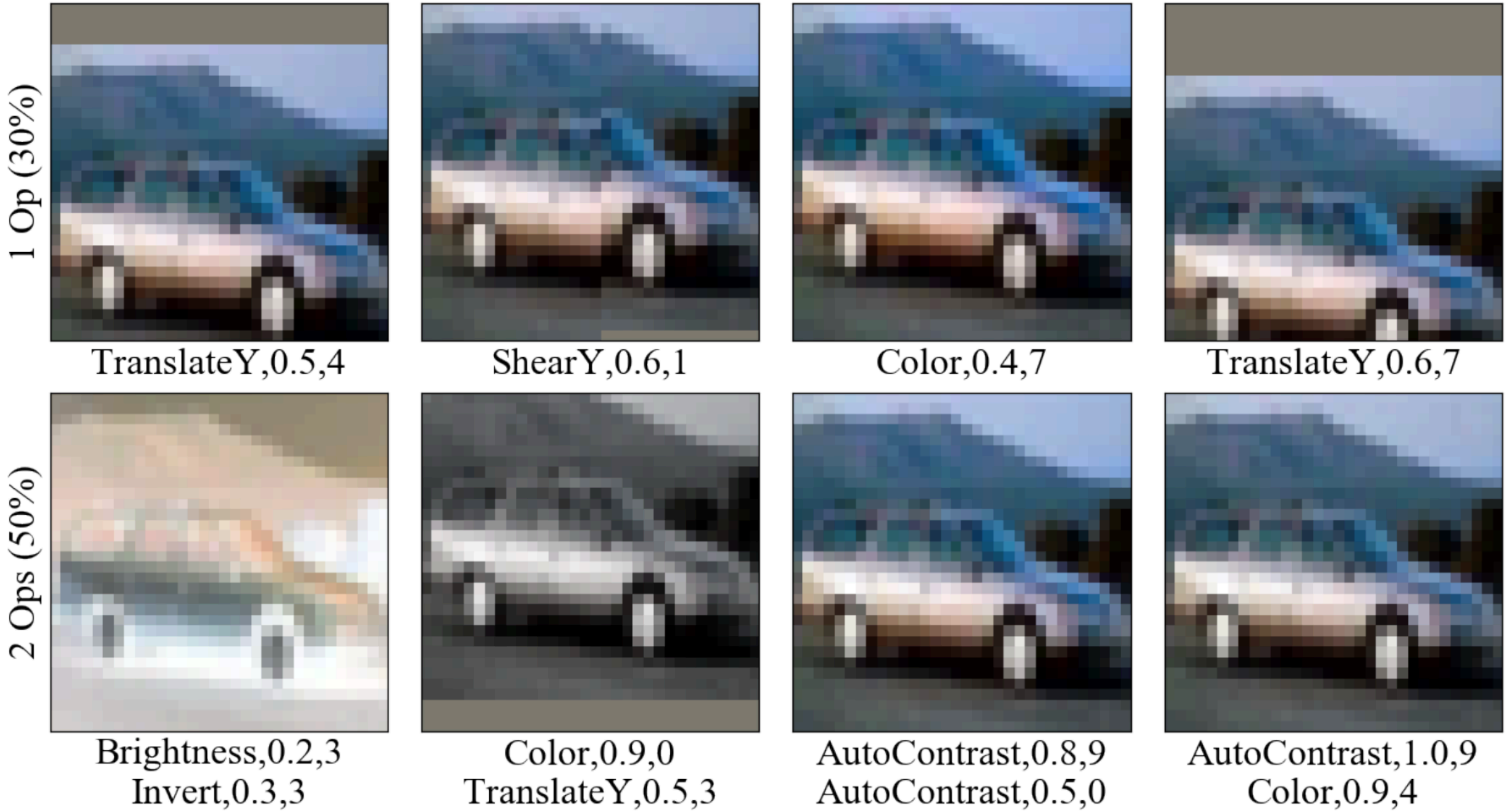
- Brute-force search-based algorithms has no chance in huge state-action spaces => trained net guides the search efficiently





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



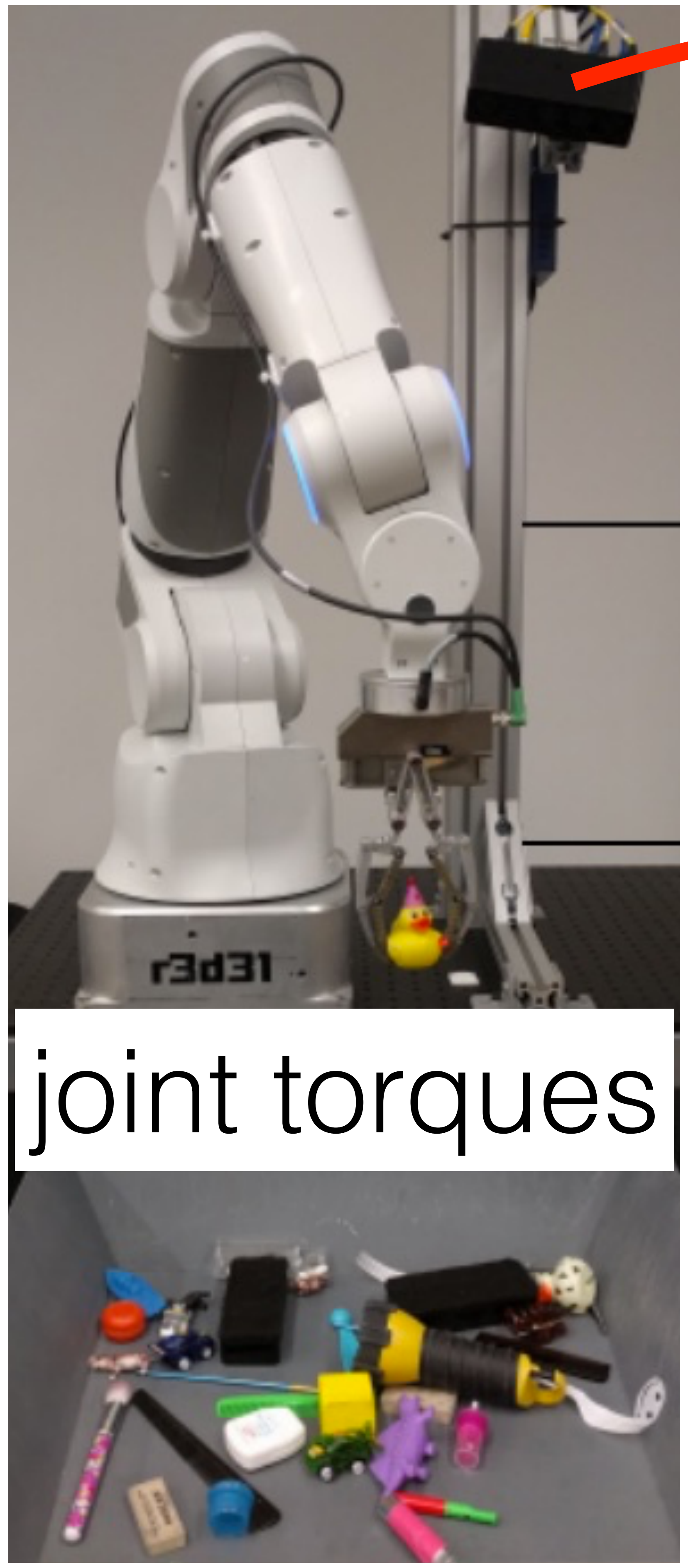
https://arxiv.org/pdf/1905.05393.pdf

Known successes of RL - Learning to learn

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

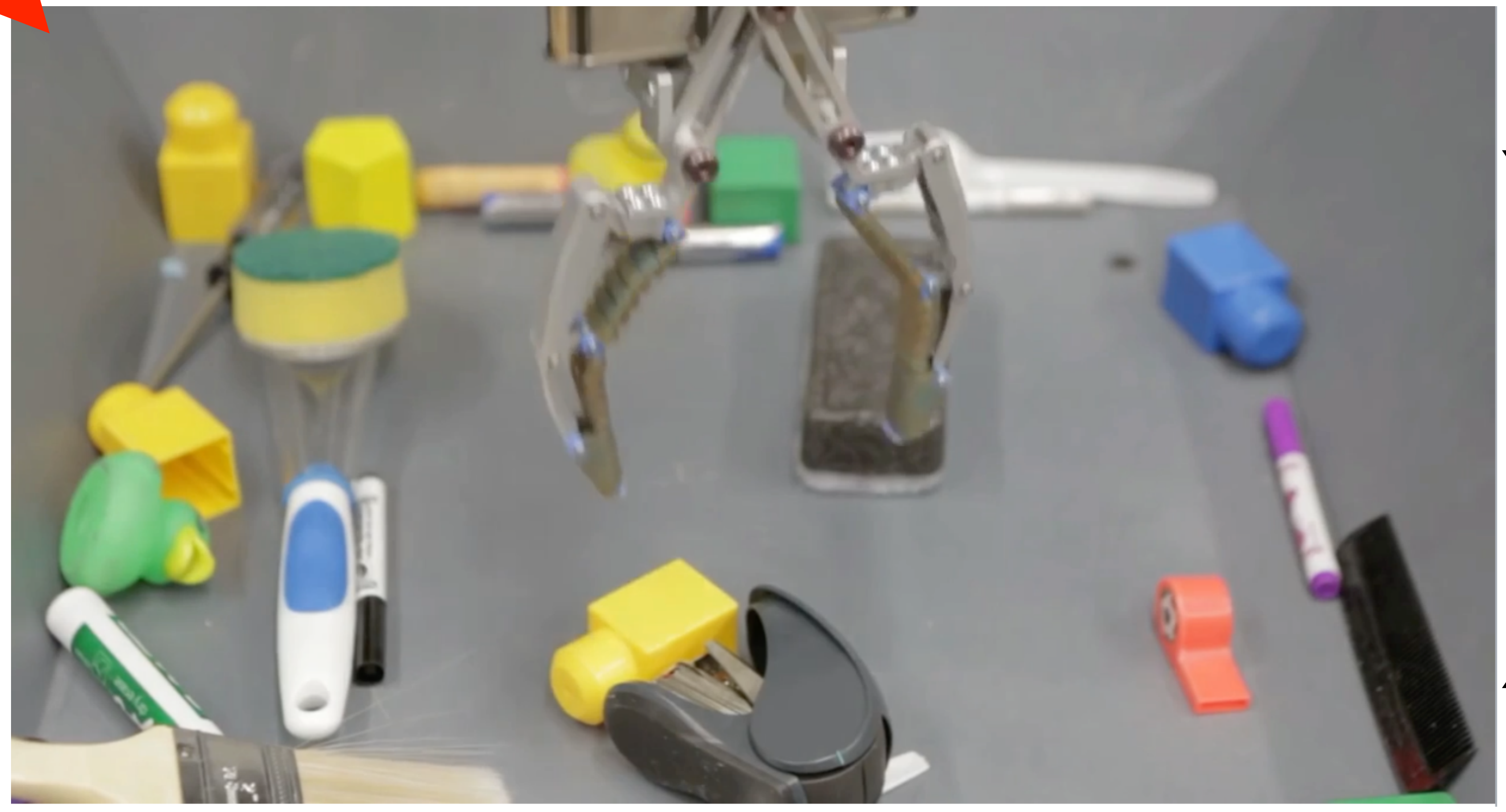
Known successes of RL
Learning complex motions in reality by paralelizing and automatizing rewards

manipulator+ RGB camera



image

$$= \pi_\theta(\ \ )$$



Continues motion control from RGB(D)

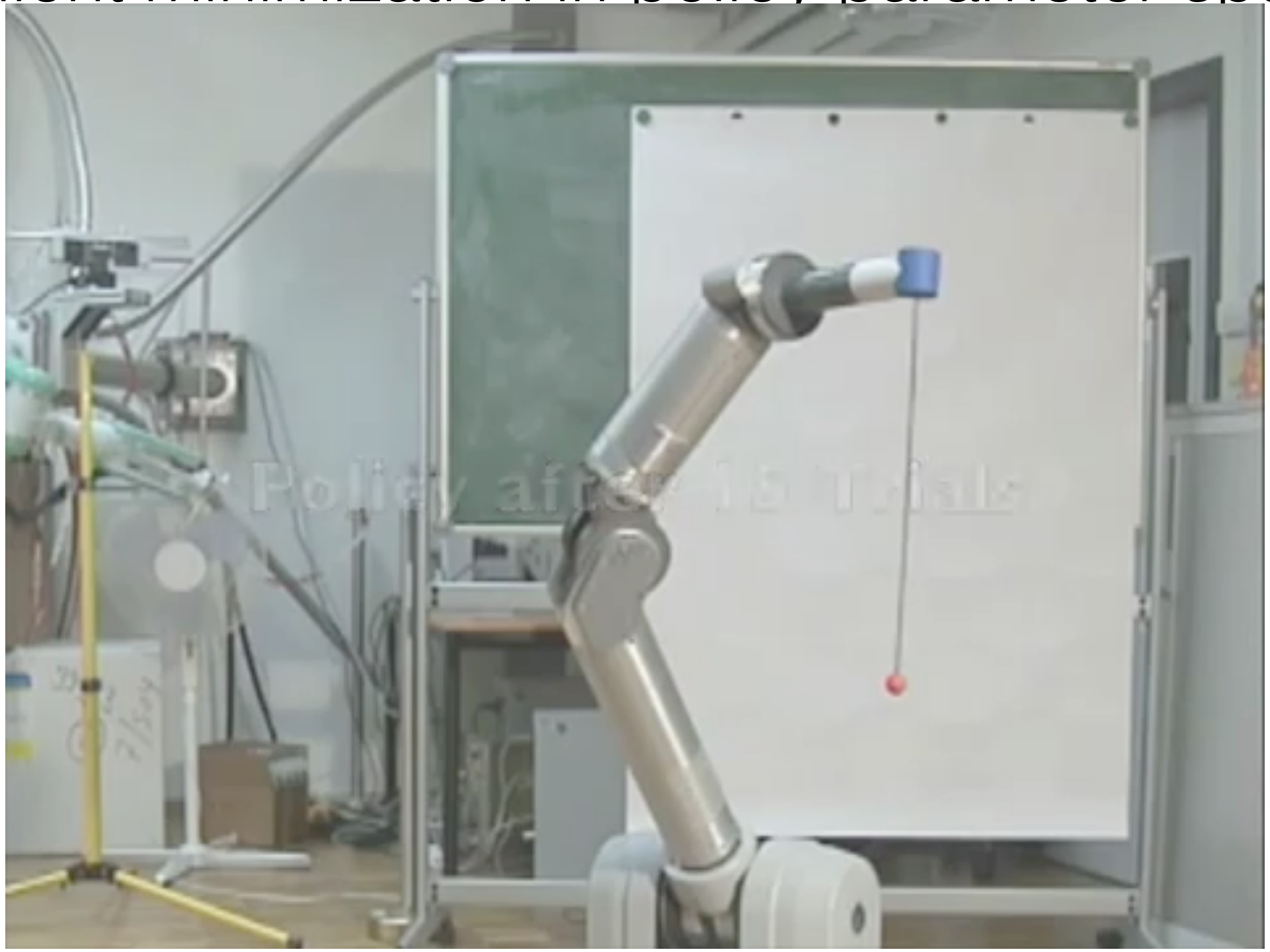
[Levine IJRR 2017] https://arxiv.org/abs/1603.02199


Source: Peter Pastor
r3d11
r3d24
r3d32

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

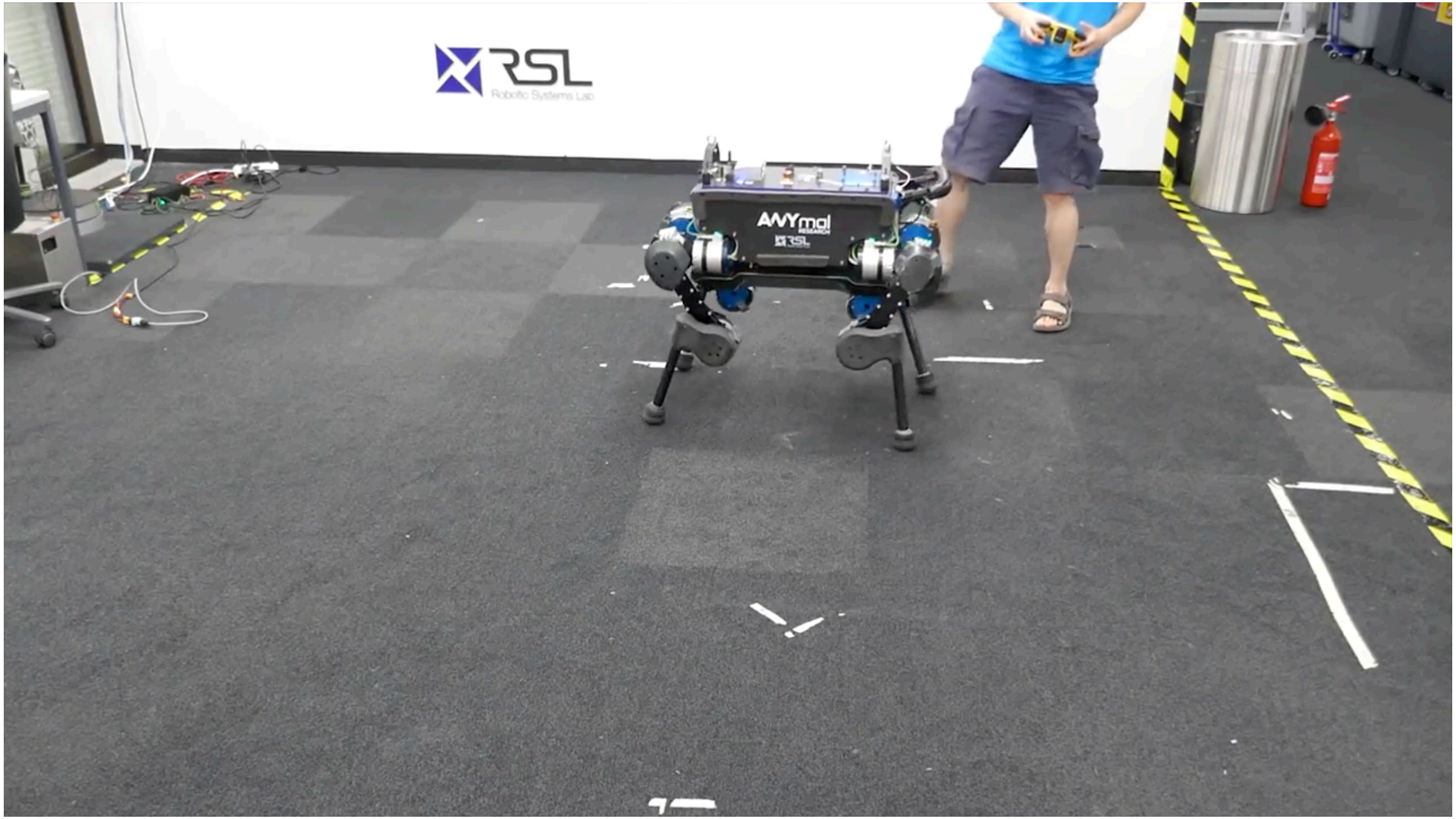
Motion and compliance control of flippers

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

Known successes of RL

learning complex motions in reality by transfering 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/

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

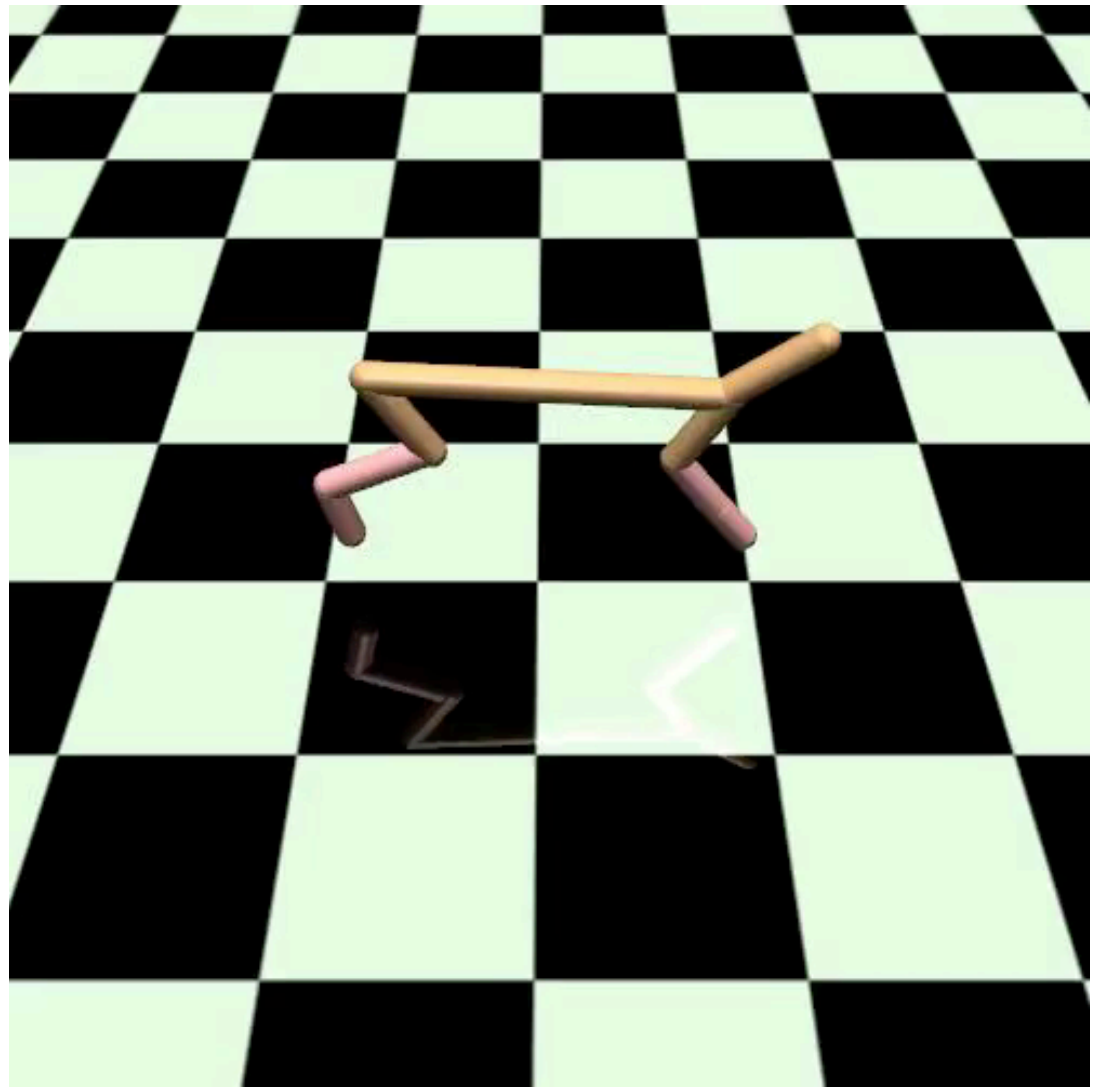
→ $\pi^* = \arg\max_{\pi} J_\pi$

τ^* → **Inverse reinforcement learning:**
- given optimal trajectories estimate reward function

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

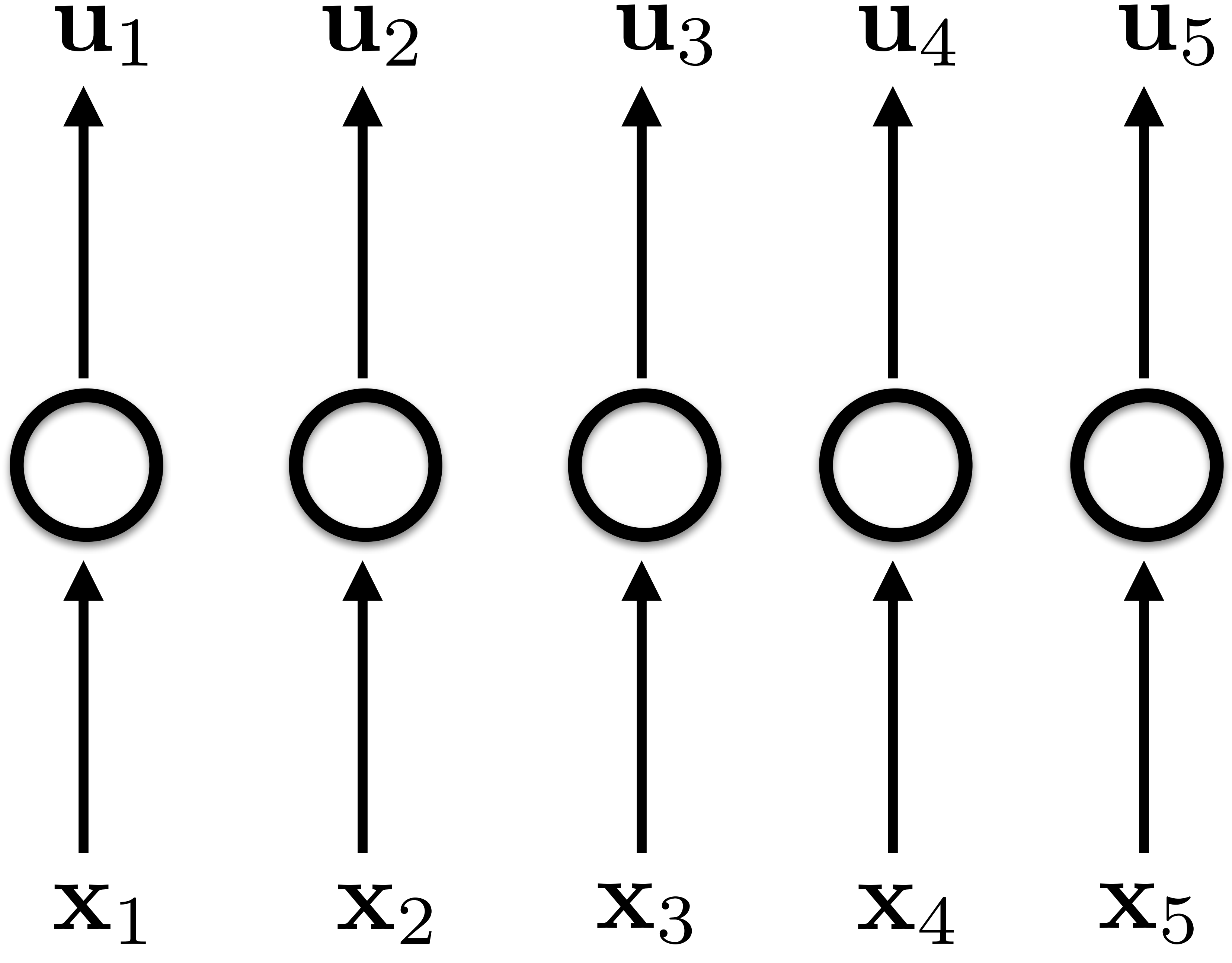
Rewards engineering

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)



Rewards engineering

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



Sparse rewards	0	0	0	0	10
Dense rewards	2	-2	4	2	2

Rewards engineering

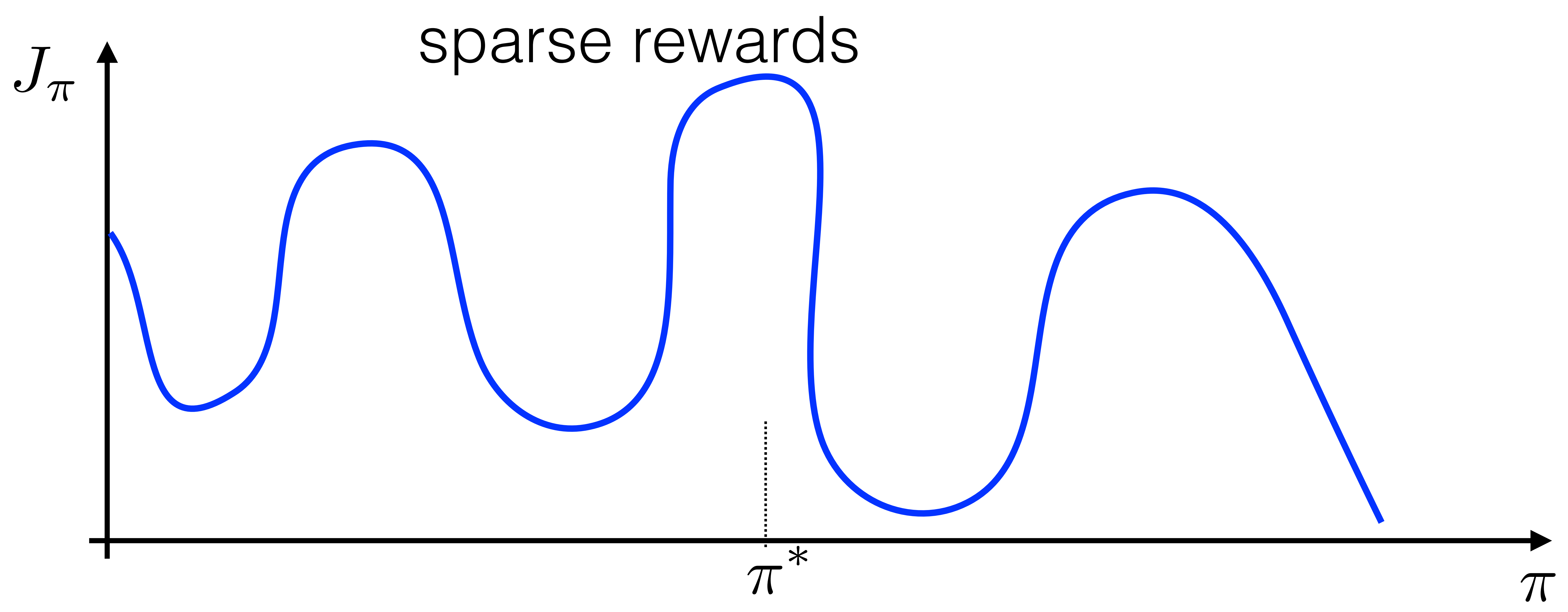
- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn

Rewards engineering

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

Rewards engineering

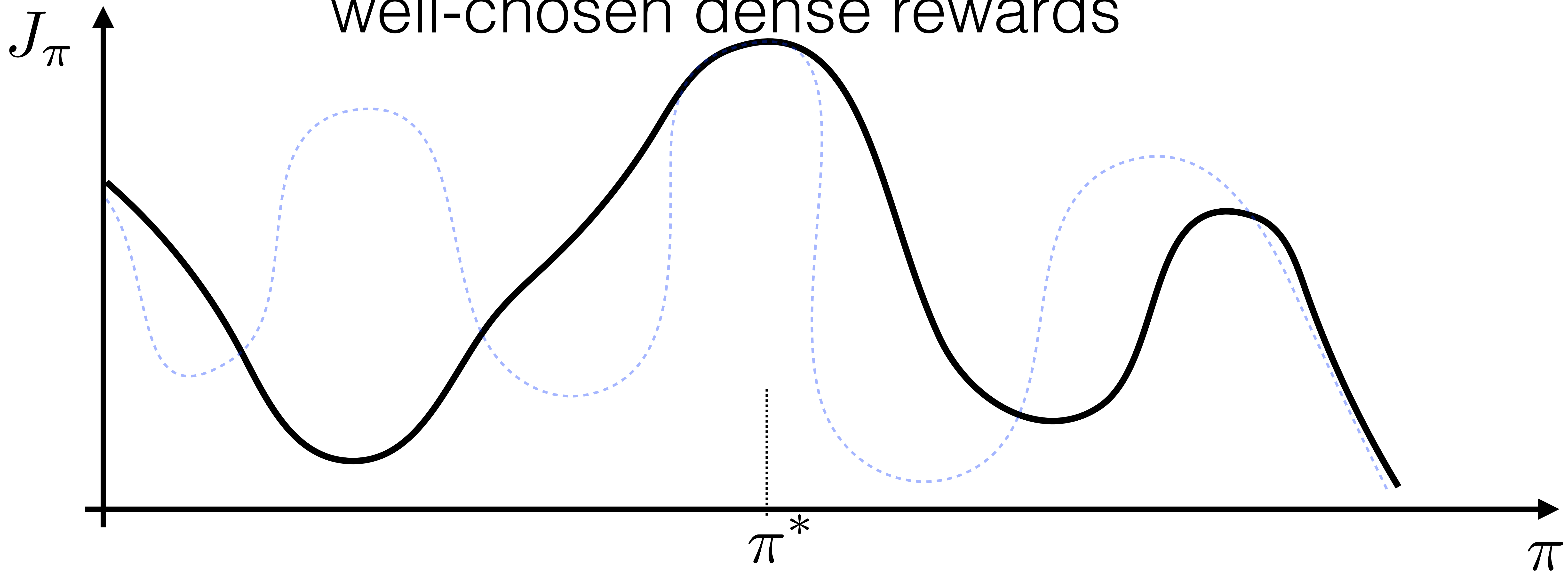
- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn



Rewards engineering

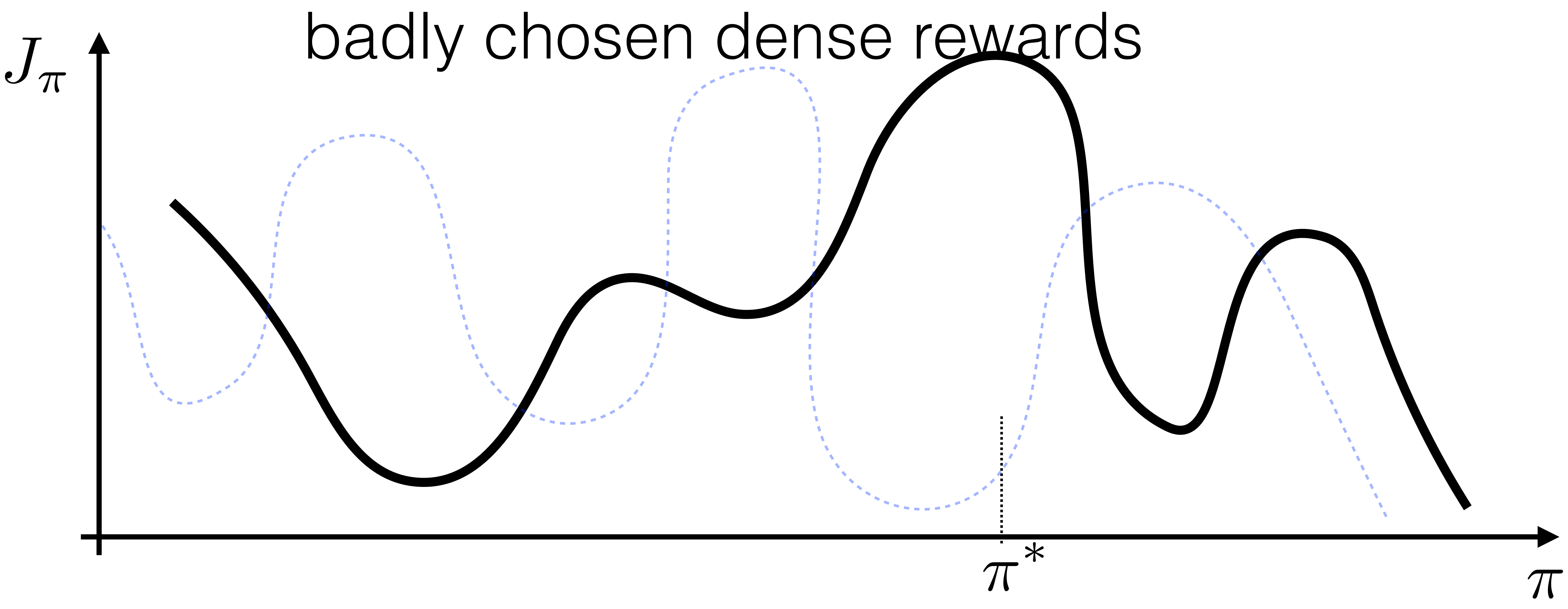
- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn

well-chosen dense rewards



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 …)

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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_{\mathbf{w}}$

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

$$\text{subject to:} \quad \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \{\mathcal{T} \setminus \tau^*\}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \leq \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}')$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_\mathbf{w}$

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

$$\text{subject to: } \mathrm{ReLU}\Big(\sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \{\mathcal{T} \setminus \tau^*\}} r_\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{x}') - \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \tau^*} r_\mathbf{w}(\mathbf{x}, \mathbf{u}, \mathbf{x}')\Big) = 0$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_{\mathbf{w}}$

$$\arg\min_{\mathbf{w}} \|\mathbf{w}\|_2^2 + \mathrm{ReLU}\Big(\sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \{\mathcal{T} \setminus \tau^*\}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') - \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \Big)$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_{\mathbf{w}}$

$$\arg\min_{\mathbf{w}} \|\mathbf{w}\|_2^2 + \text{ReLU}\Big(\sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \{\mathcal{T} \setminus \tau^*\}} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') - \sum_{(\mathbf{x},\mathbf{u},\mathbf{x}') \in \tau^*} r_{\mathbf{w}}(\mathbf{x}, \mathbf{u}, \mathbf{x}') \Big)$$

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_\mathbf{w}$

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

Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.
- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  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 $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_\mathbf{w}$

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

  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



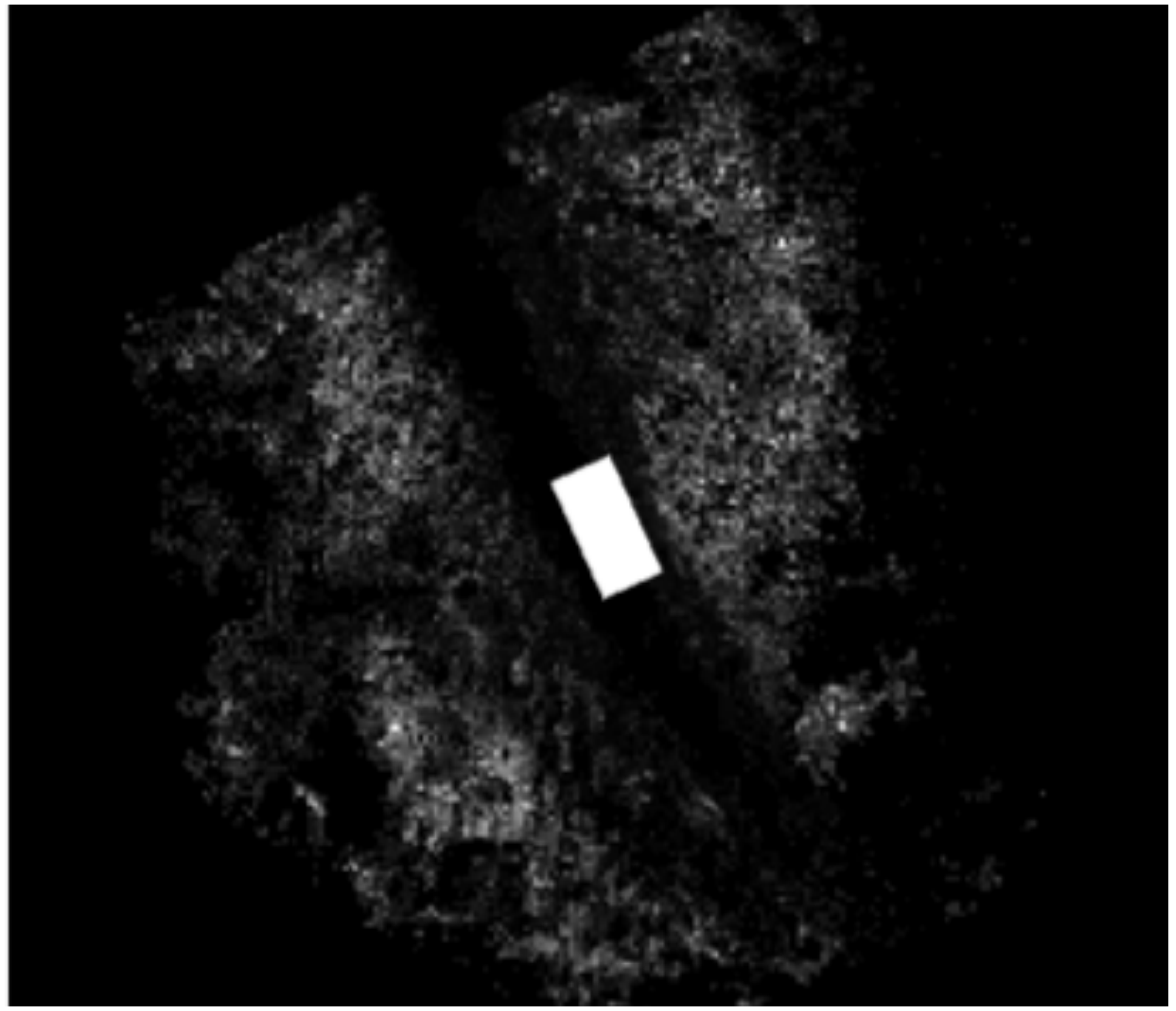
Abbeel et al. IJRR 2010

Silver et al. IJRR 2010

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

Silver et al. IJRR 2010

input image (state)

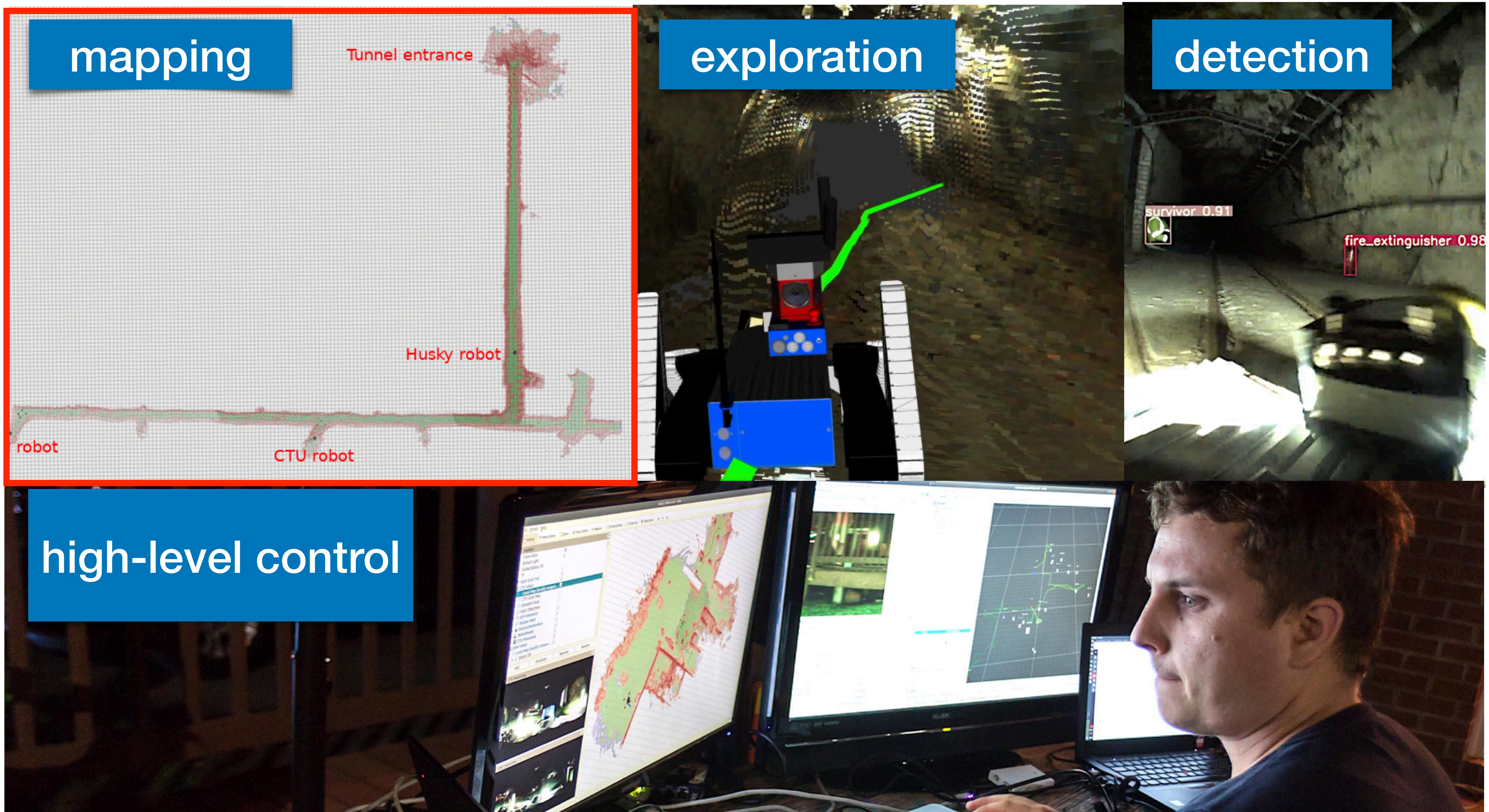


learned reward function
(traversability map)

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

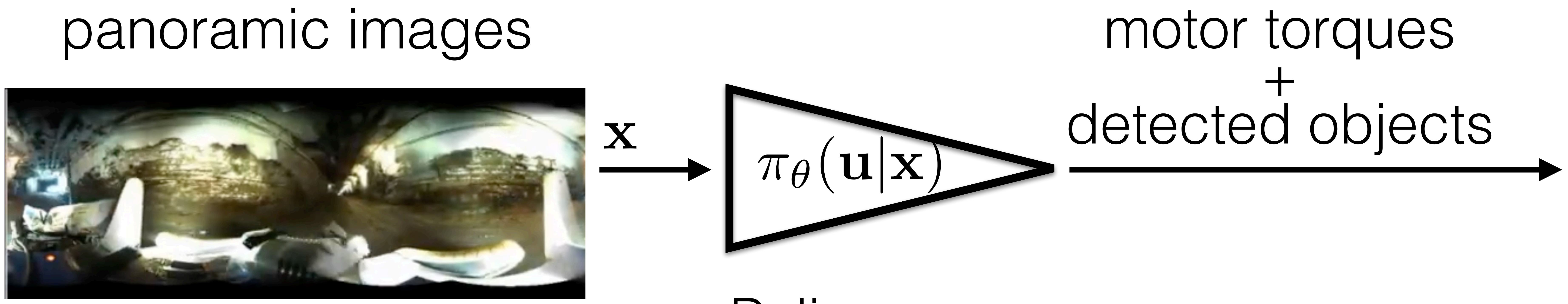
Going back to DARPA

- Should we keep building pipelines or should we rather train all-in-once??



Going back to DARPA

- Should we keep building pipelines or should we rather train all-in-once??



Policy:

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

PROS all-in-one approach

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

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

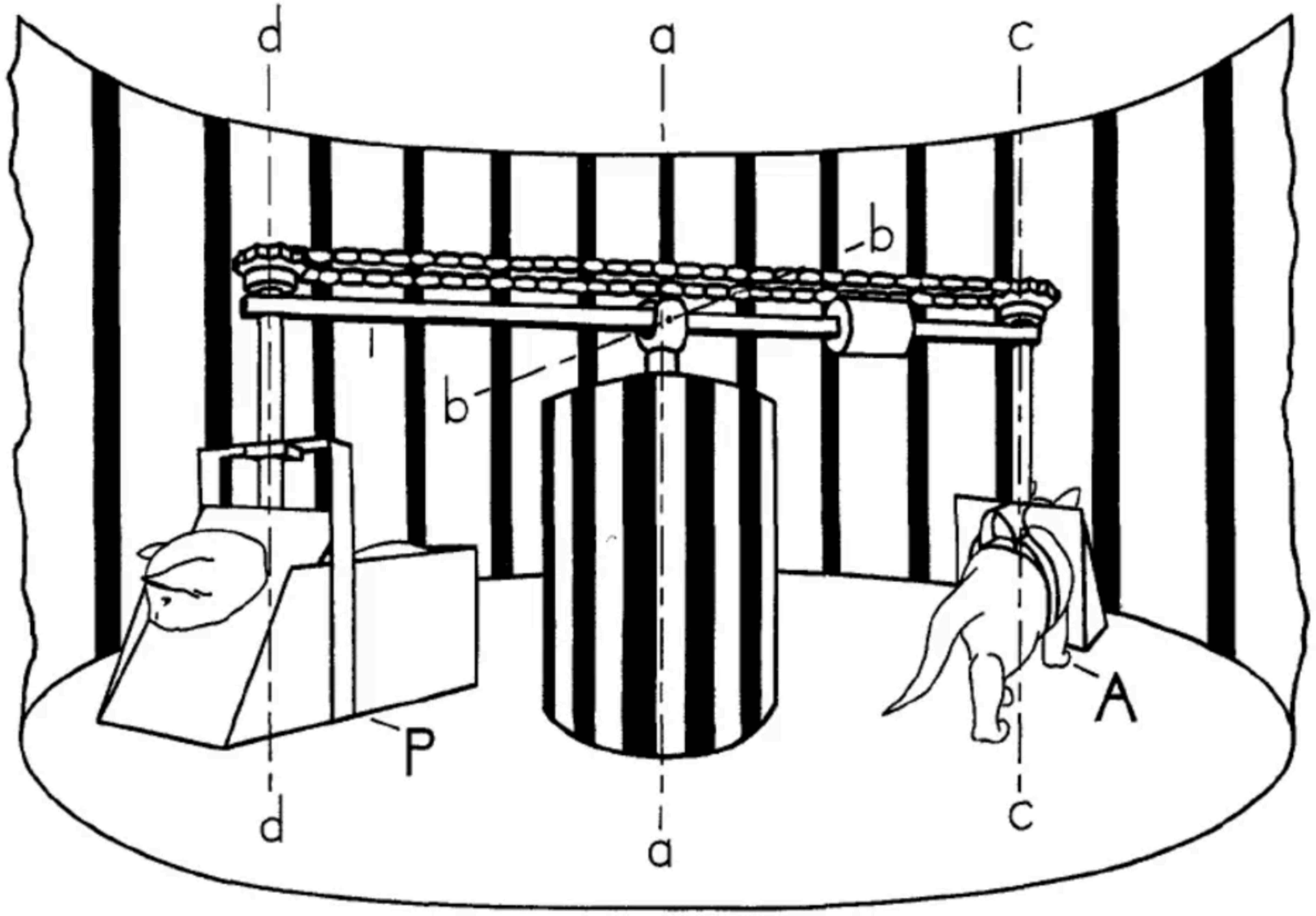


FIG. 1. Apparatus for equating motion and consequent visual feedback for an actively moving (A) and a passively moved (P) *S*.

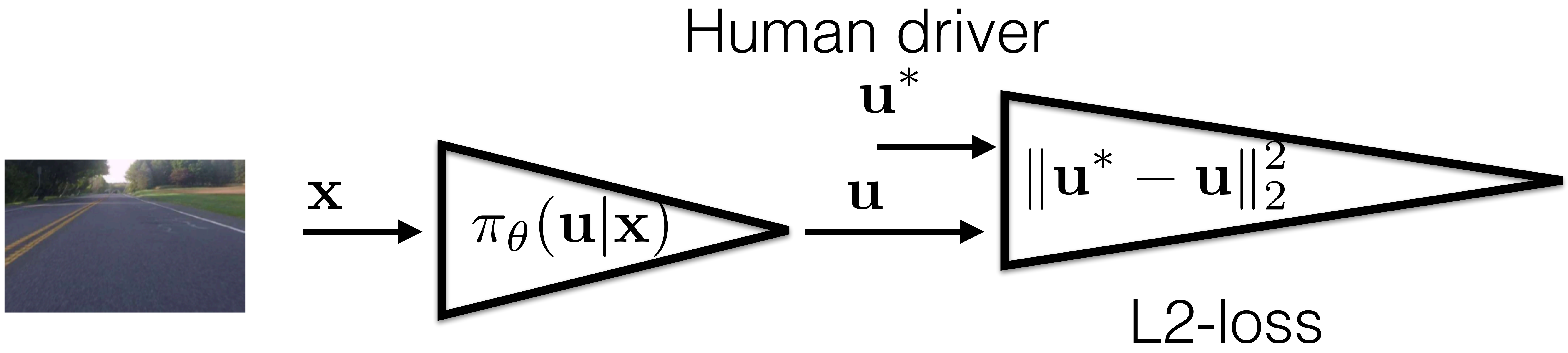
CONS all-in-one approach

- RL is sample inefficient (>200M transitions required for atari games)
- Real robot can easily break.
- Learning from simulator suffers from simulation bias (e.g. vision)
- Even if you learn an all-in-one network, the behaviour not interpretable.

https://waymo.com/open/data/perception/

[NVidia, CVPR, 2016]
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

Straightforward driving of autonomous car by a deep net?



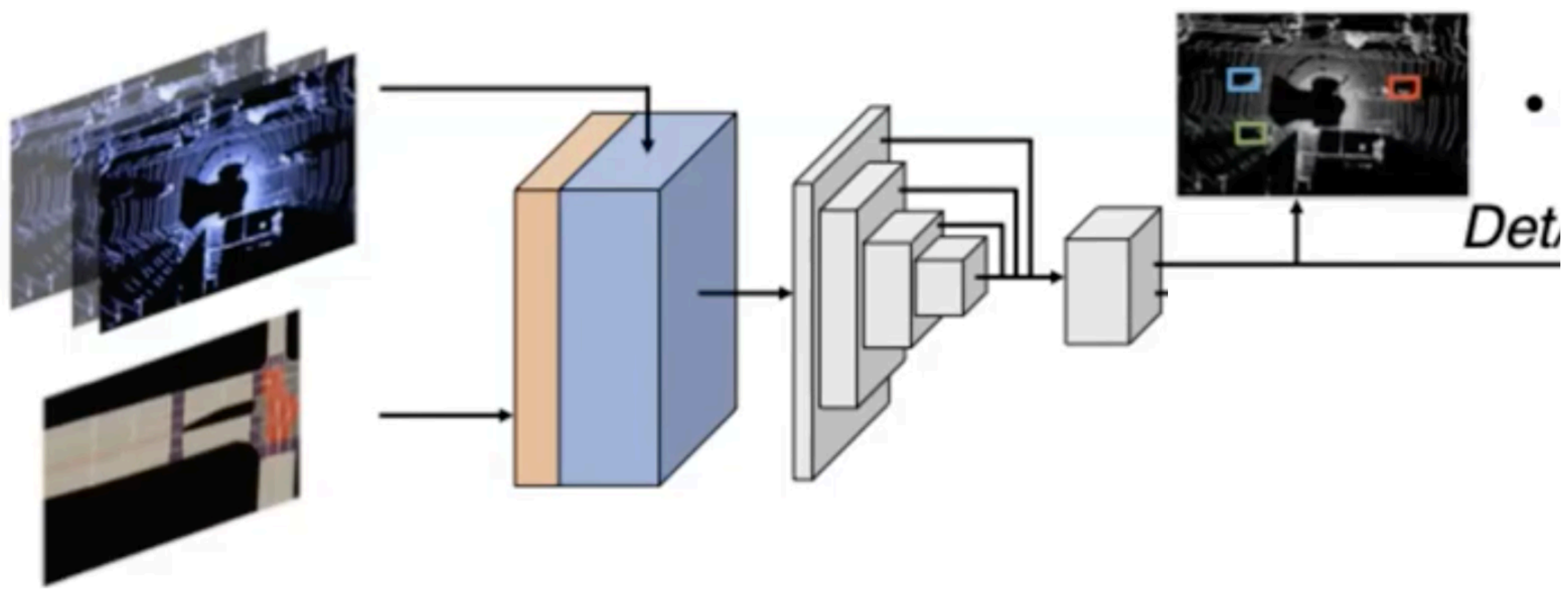
Policy:
$$\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$

- **Reliable? Explainable? Managable?**

Interpretable motion planning
[Zeng,.. Urtasun from Uber, CVPR, 2019]

Lidar scans

t=1

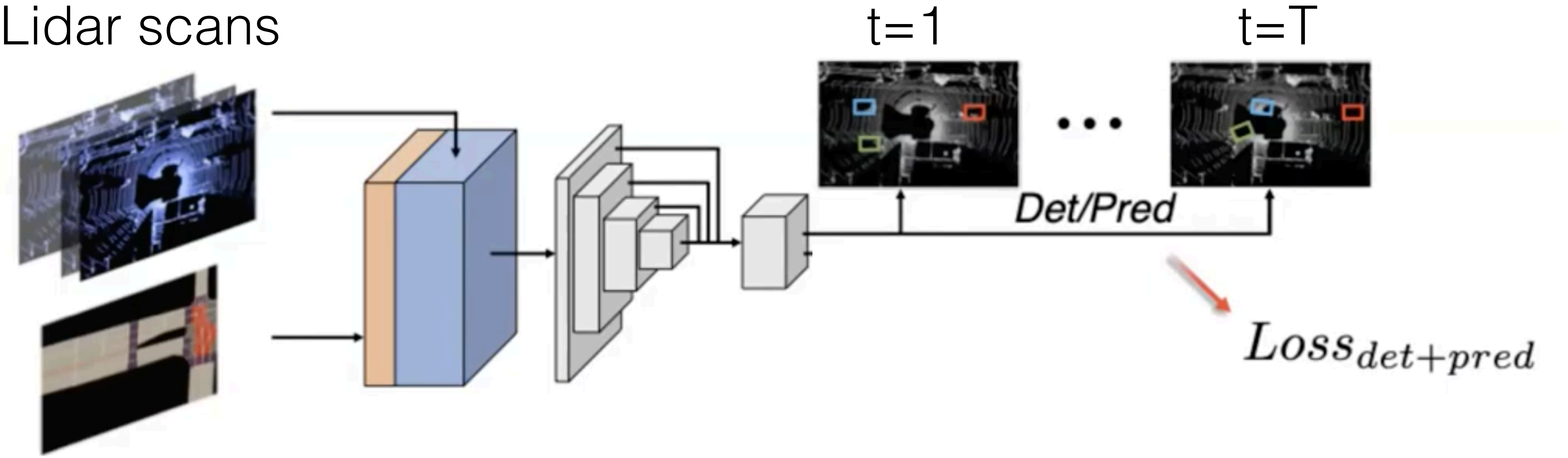


lanes
+
traffic signs
+
traffic lights

http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun from Uber, CVPR, 2019]

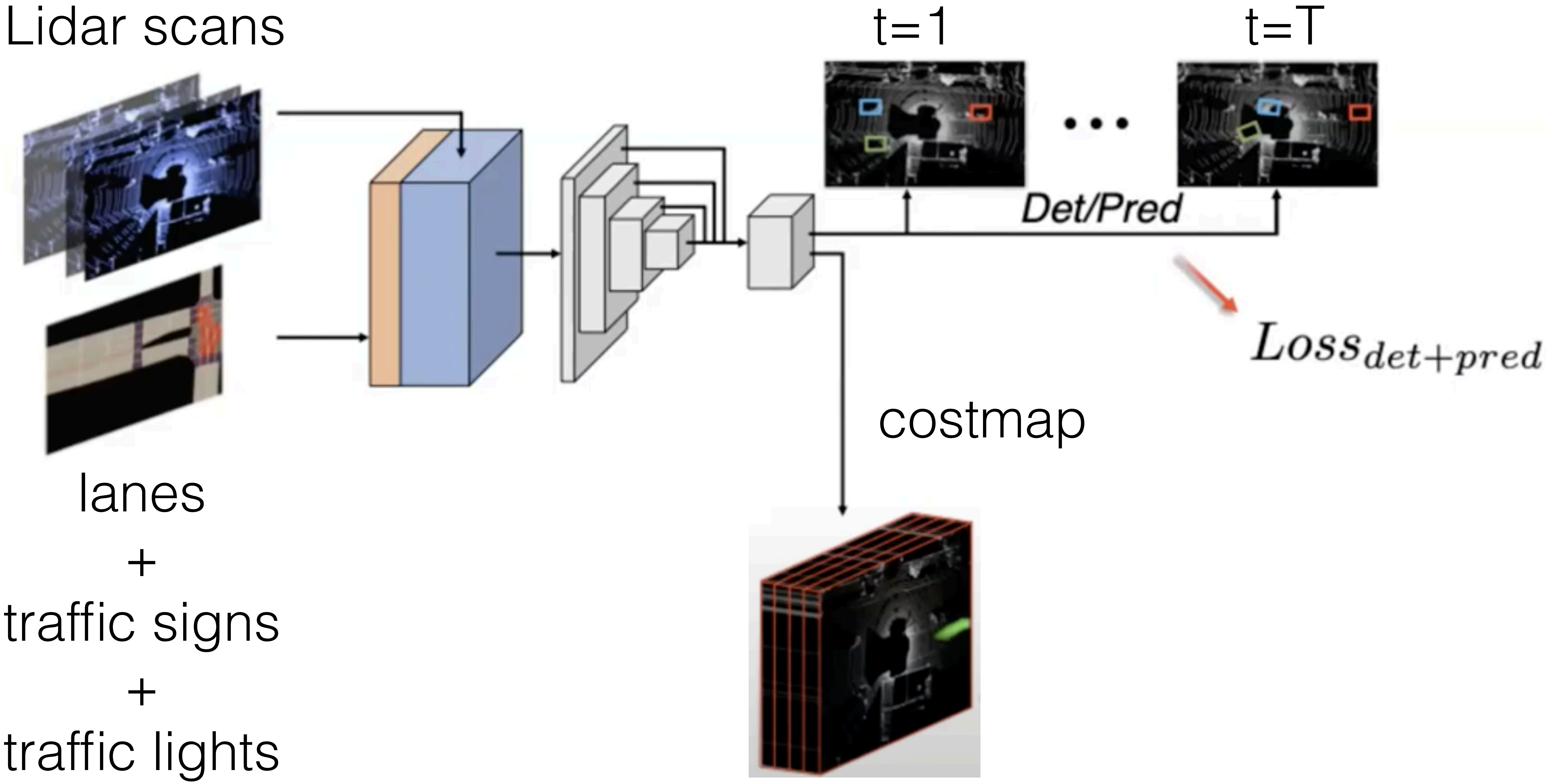
Lidar scans



lanes
+
traffic signs
+
traffic lights

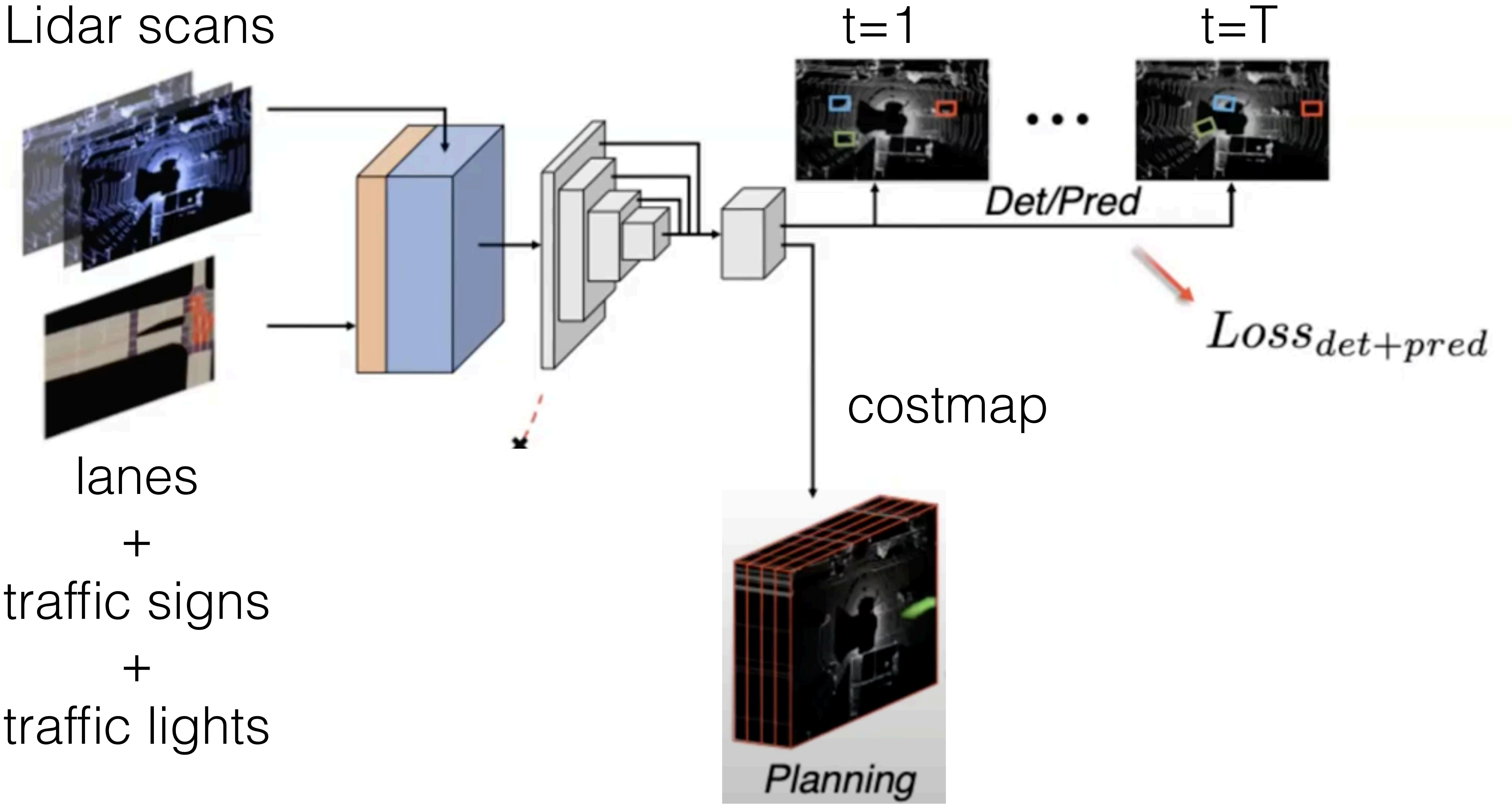
http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun from Uber, CVPR, 2019]



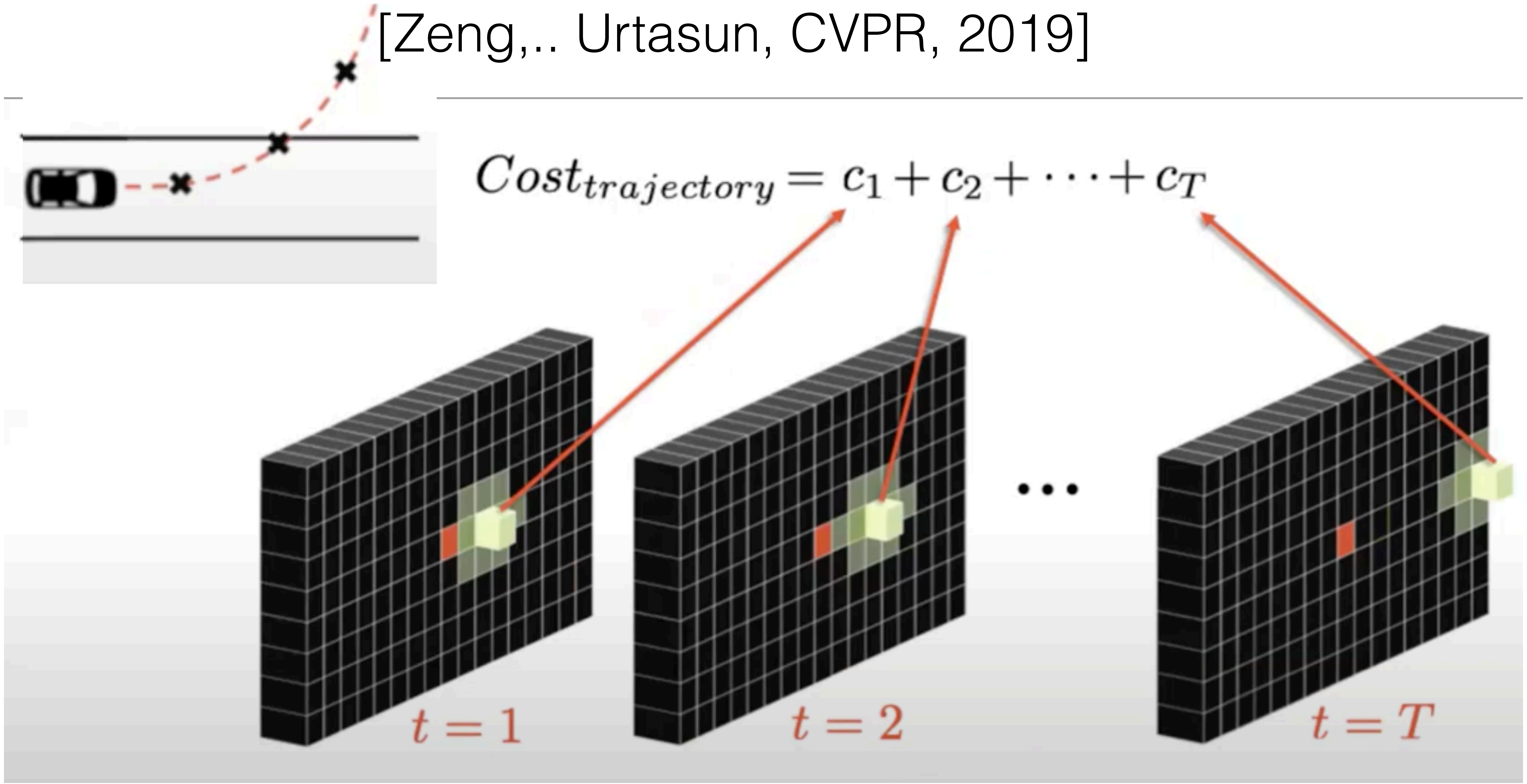
http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun from Uber, CVPR, 2019]



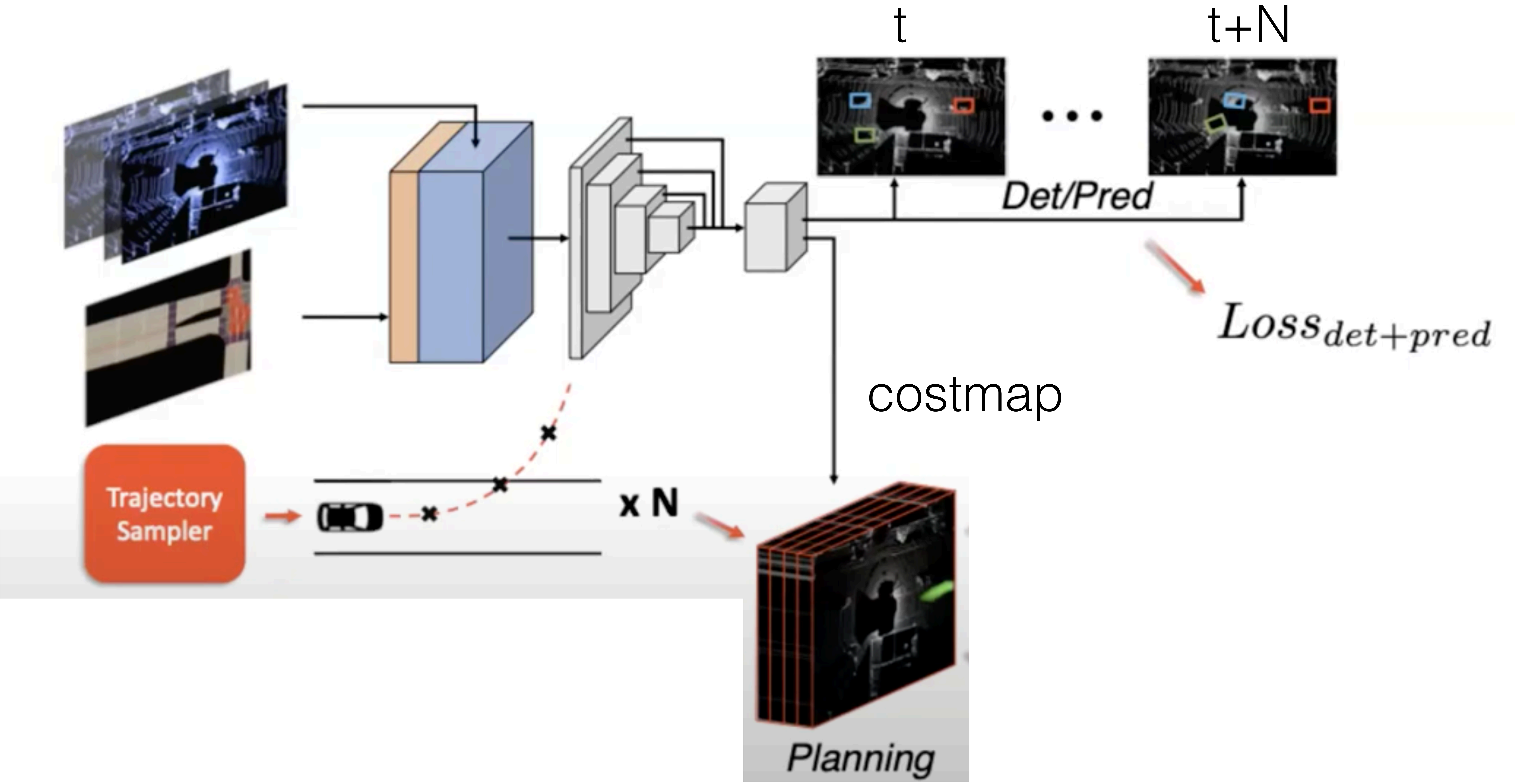
http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun, CVPR, 2019]



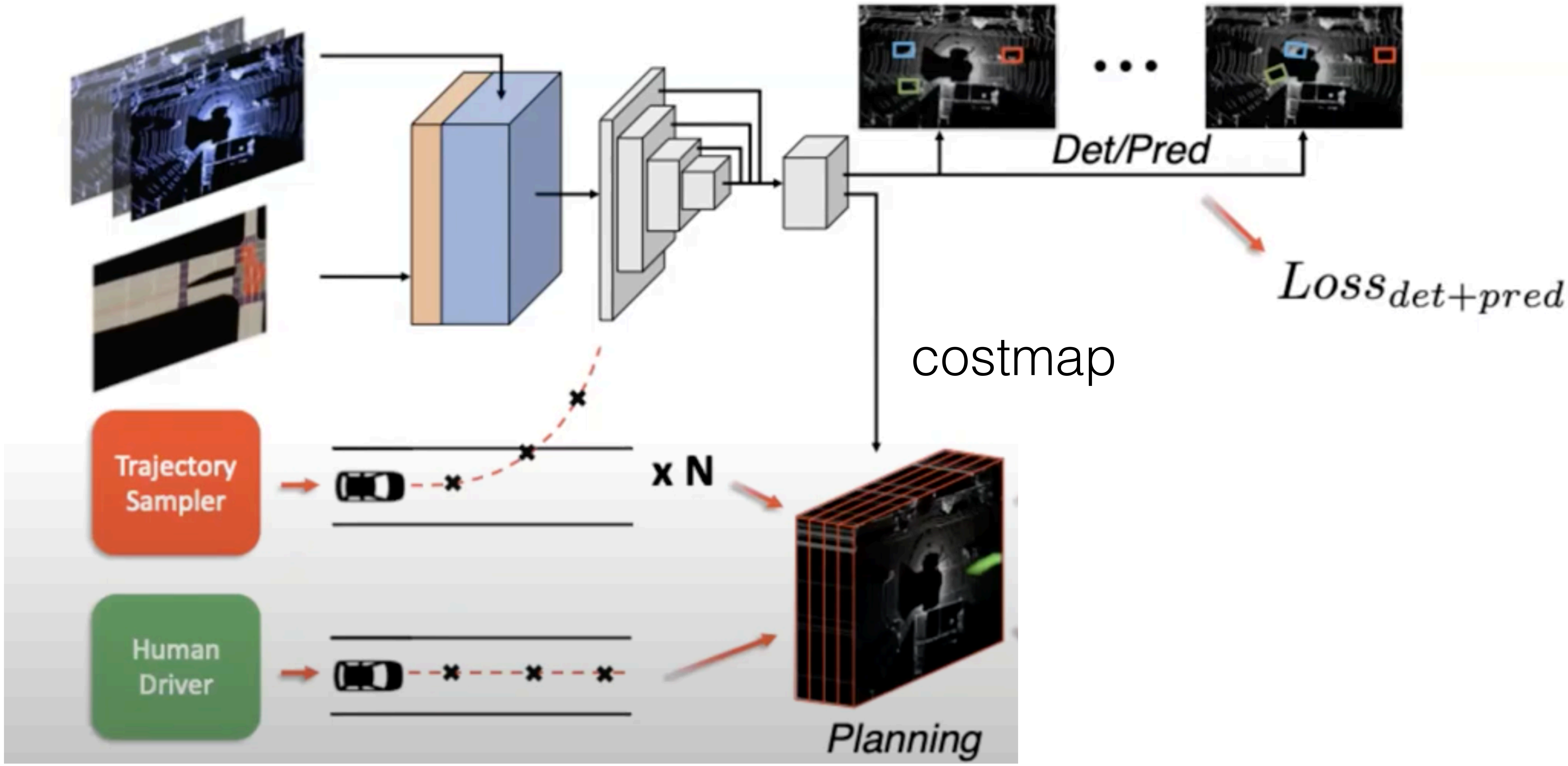
http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun from Uber, CVPR, 2019]



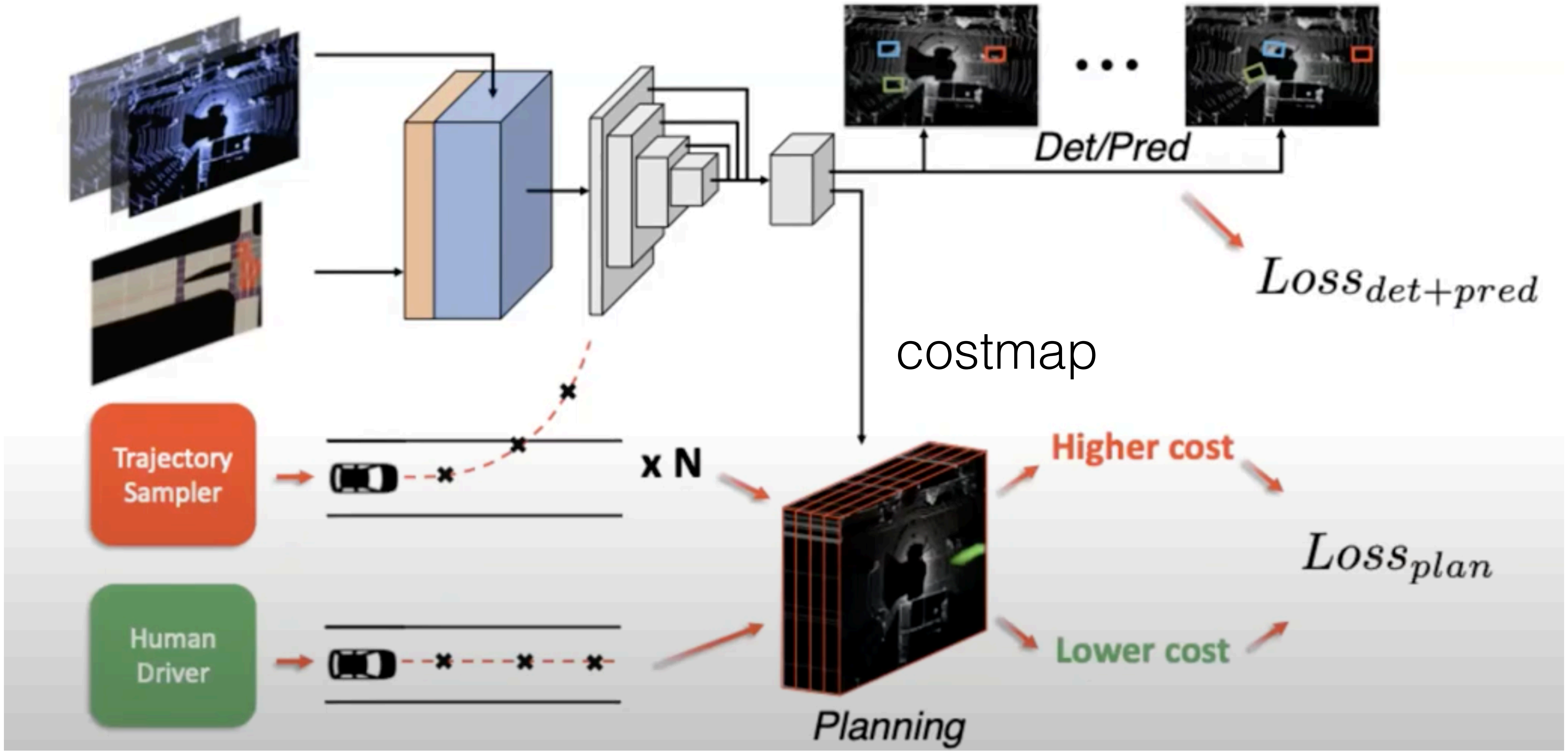
http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun from Uber, CVPR, 2019]



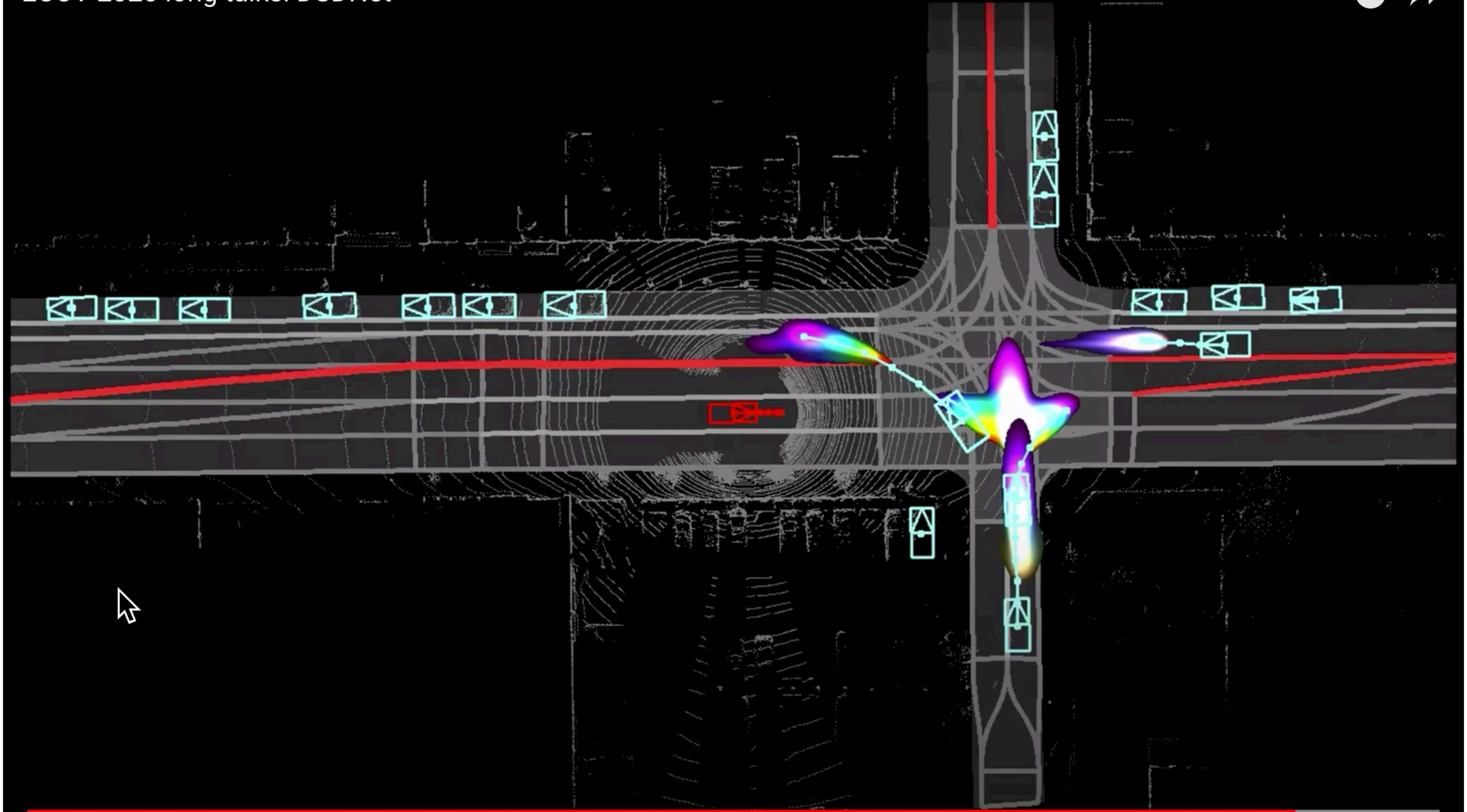
http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun from Uber, CVPR, 2019]

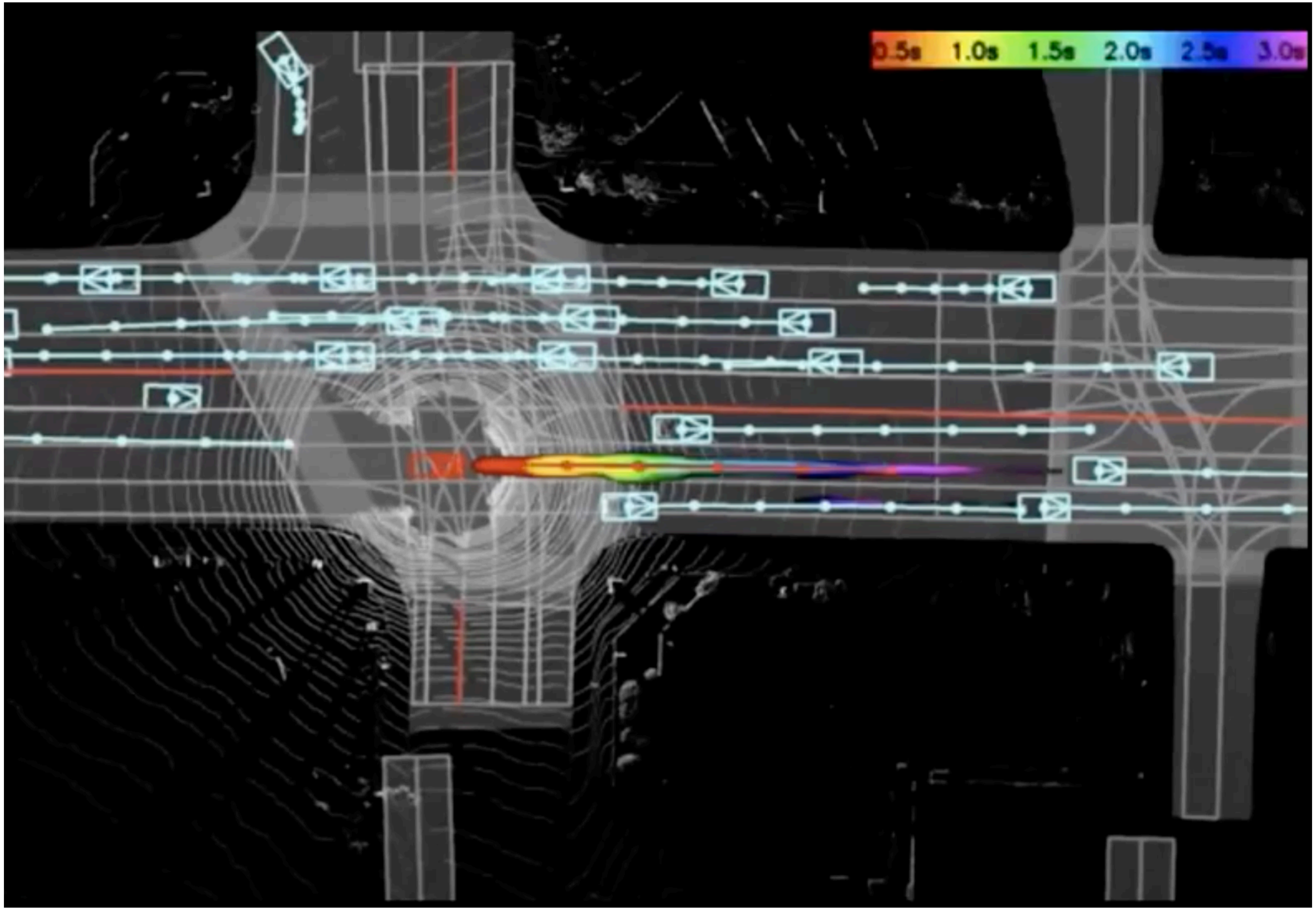


http://www.cs.toronto.edu/~wenjie/

ECCV 2020 long talks: DSDNet

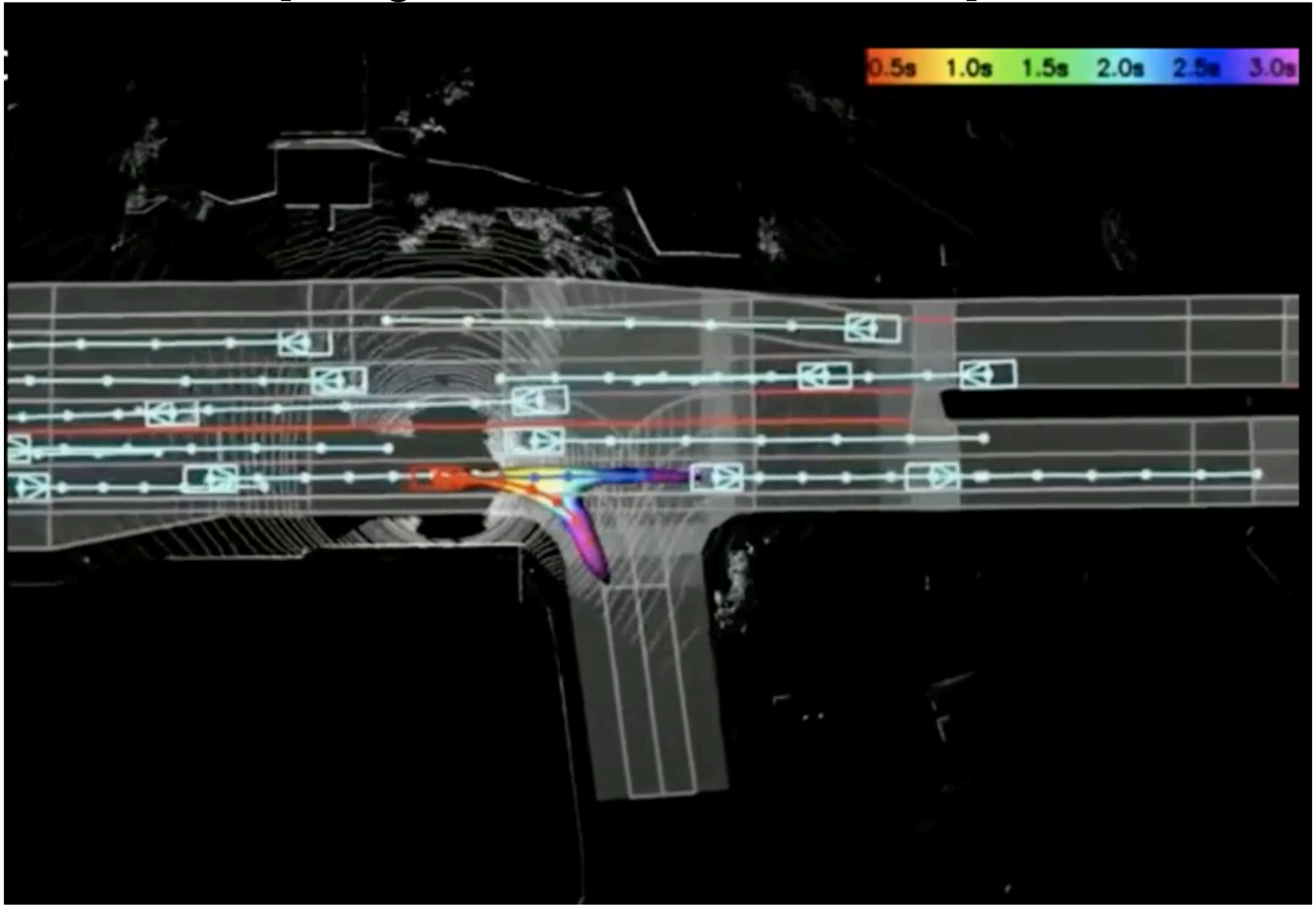


Interpretable motion planning
[Zeng,.. Urtasun, CVPR, 2019]



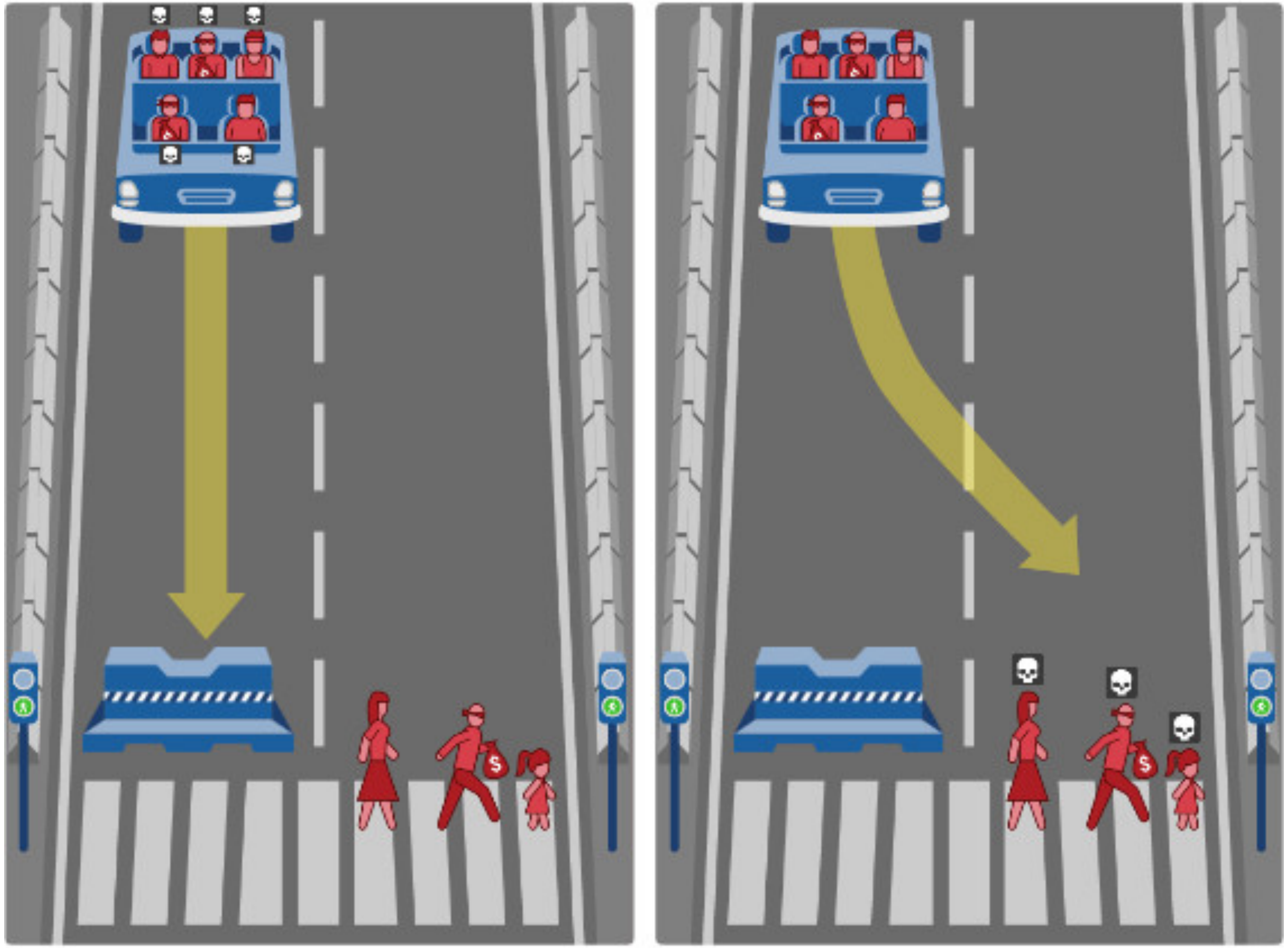
http://www.cs.toronto.edu/~wenjie/

Interpretable motion planning
[Zeng,.. Urtasun, CVPR, 2019]



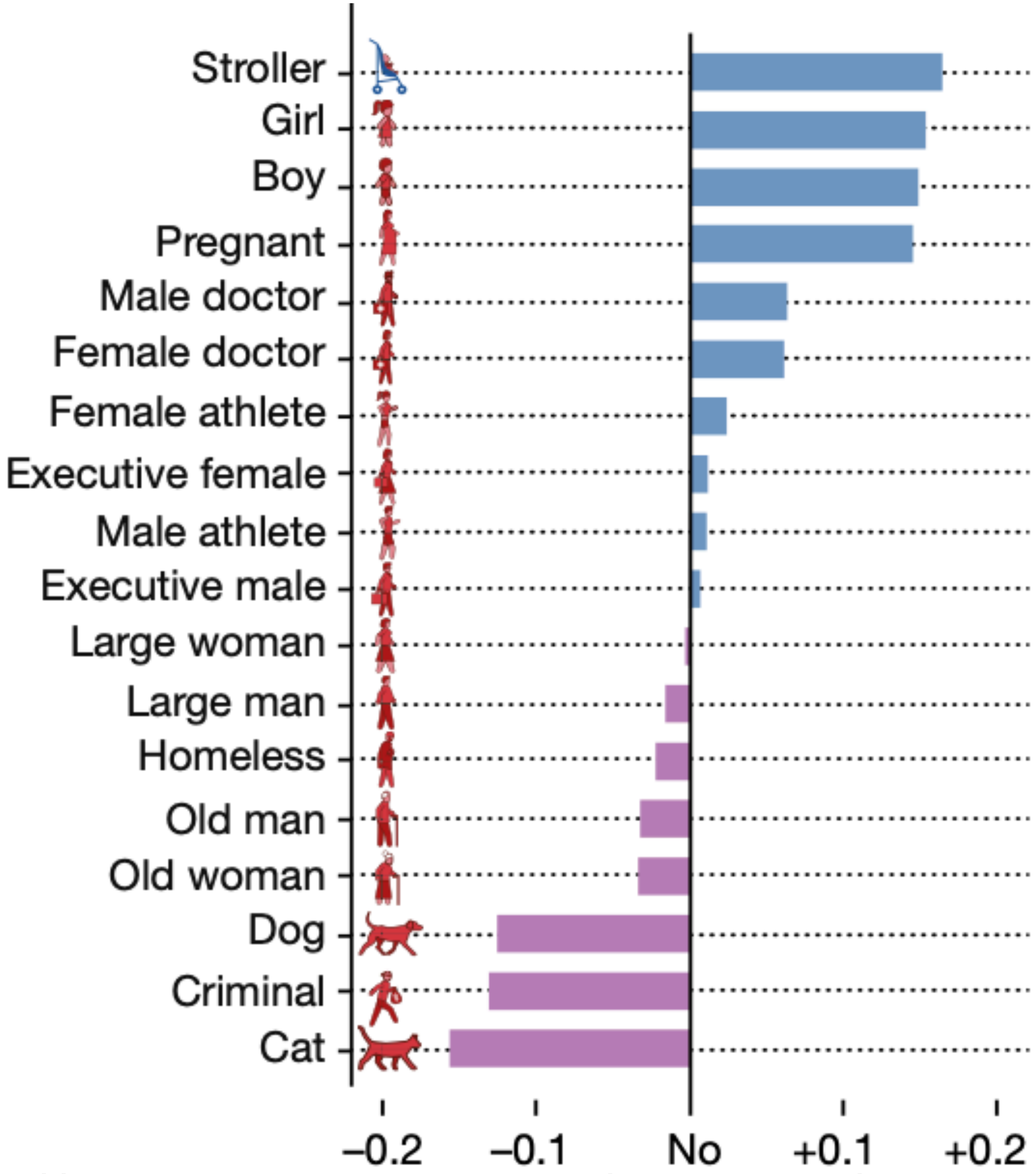
http://www.cs.toronto.edu/~wenjie/

Trolley problem



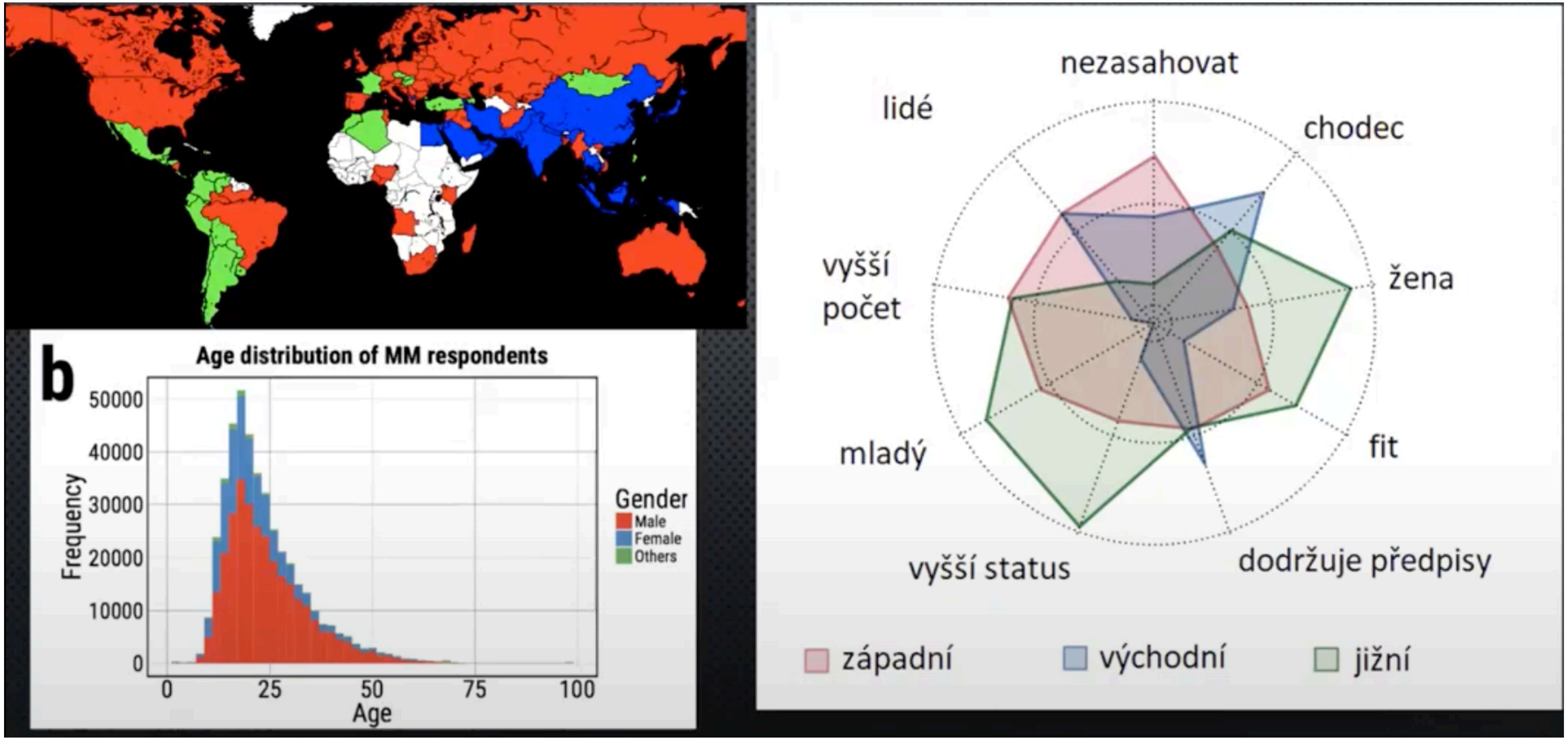
https://www.nature.com/articles/s41586-018-0637-6
[Moral Machine Experiment, Nature, 2018]

Trolley problem
estimated preference (normalized rewards) for life saving



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

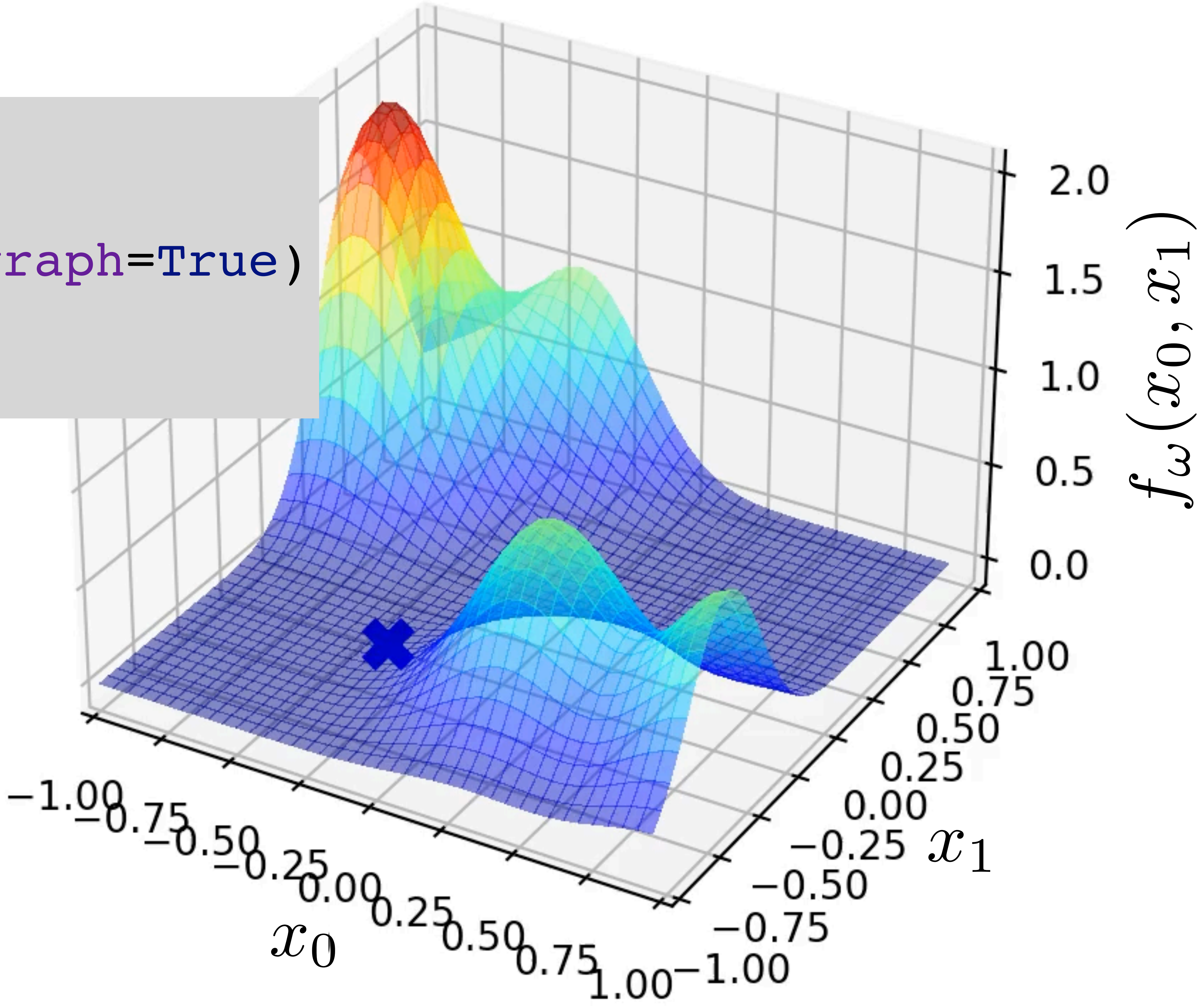
https://www.nature.com/articles/s41586-018-0637-6
[Moral Machine Experiment, Nature, 2018]

Can we backpropagate through optimization problem?

Can we backpropagate through optimization problem?

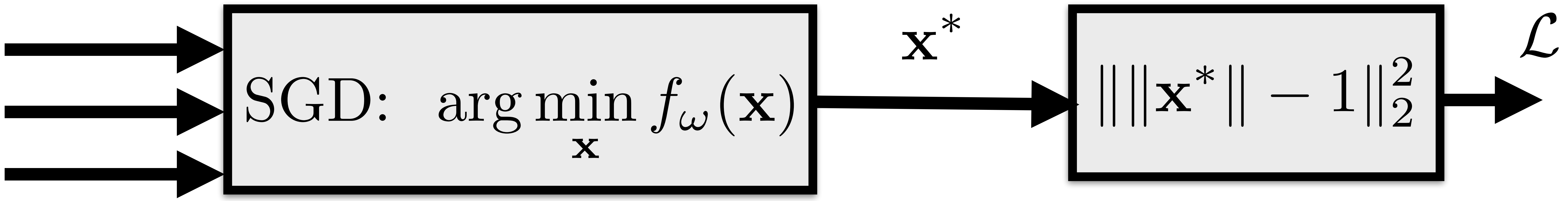
PyTorch:

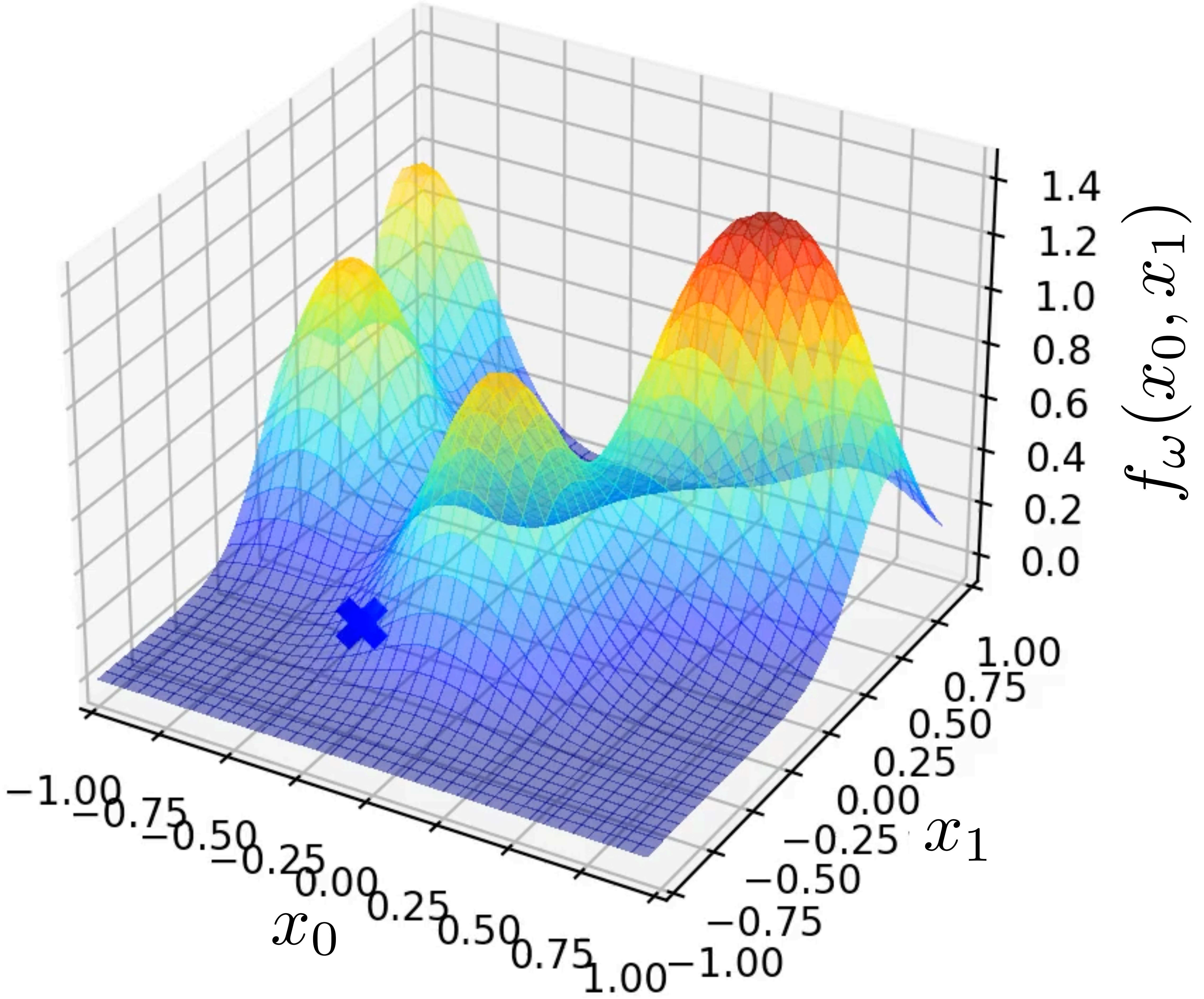
```
for i in range(iter):
    f = …
    x = x - torch.autograd.grad(f(x), x, retain_graph=True)
loss = x.norm()
loss.backward()
```



Can we backpropagate through optimization problem?

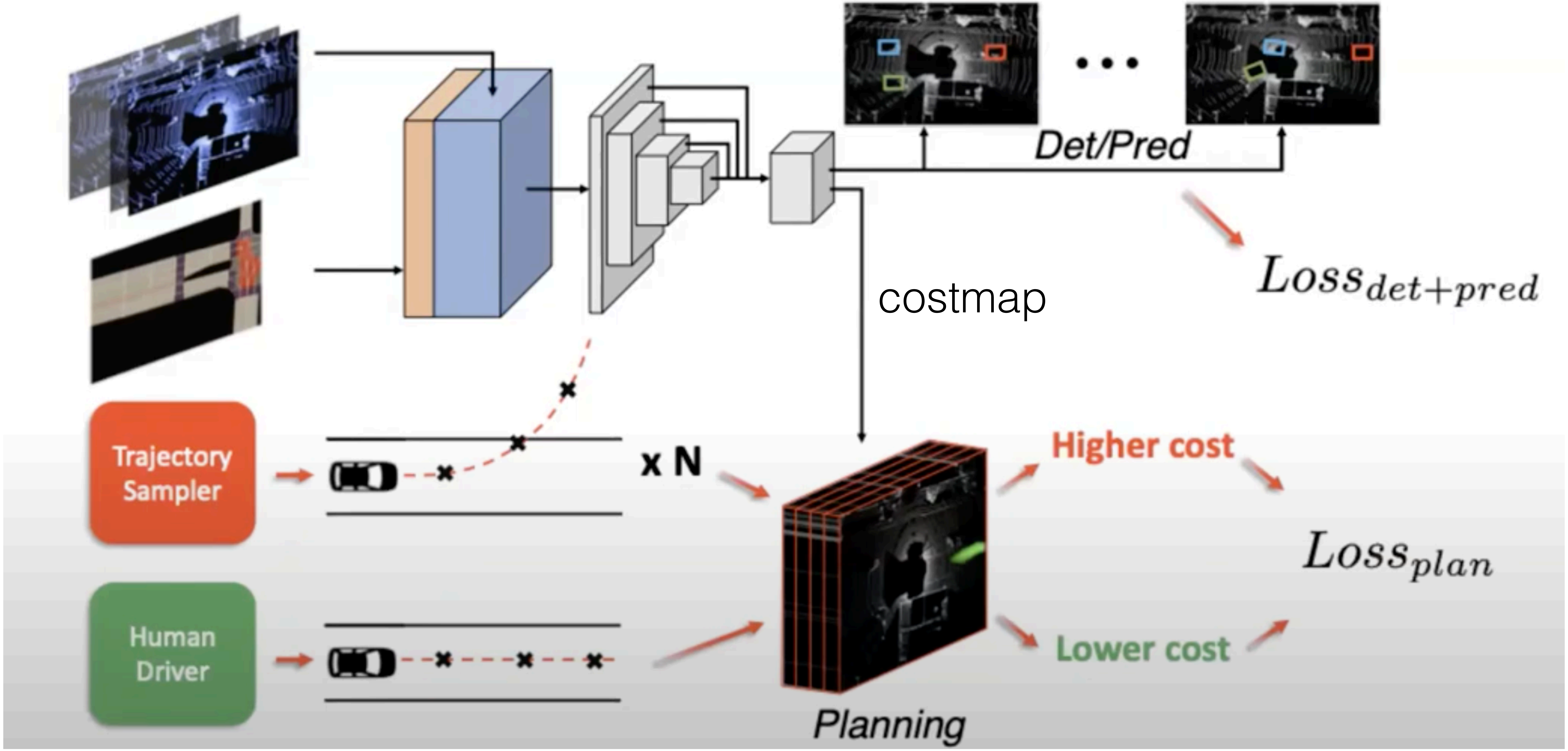
$\mathbf{x}_0$… initial point
ω… criterion params
α, β… optimizer params





Examples of differentiable optimization problems related to robotics

Interpretable motion planning
[Zeng,.. Urtasun, Uber, CVPR, 2019]
http://www.cs.toronto.edu/~wenjie/

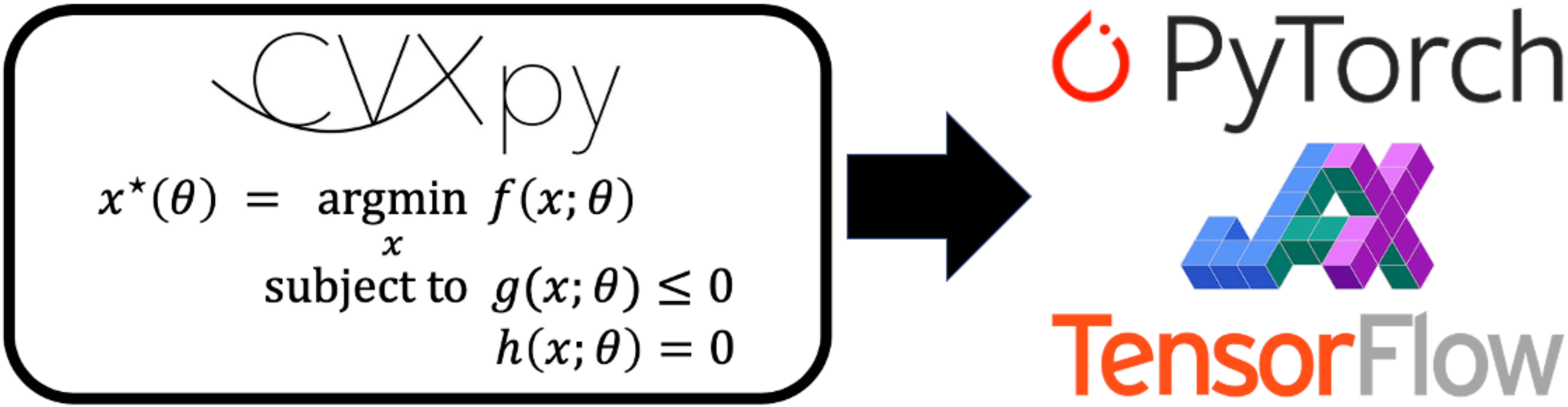


PyTorch embedded modeling language for convex optimization problemns

https://www.cvxpy.org/

[Boyd, et al, NIPS, 2019]

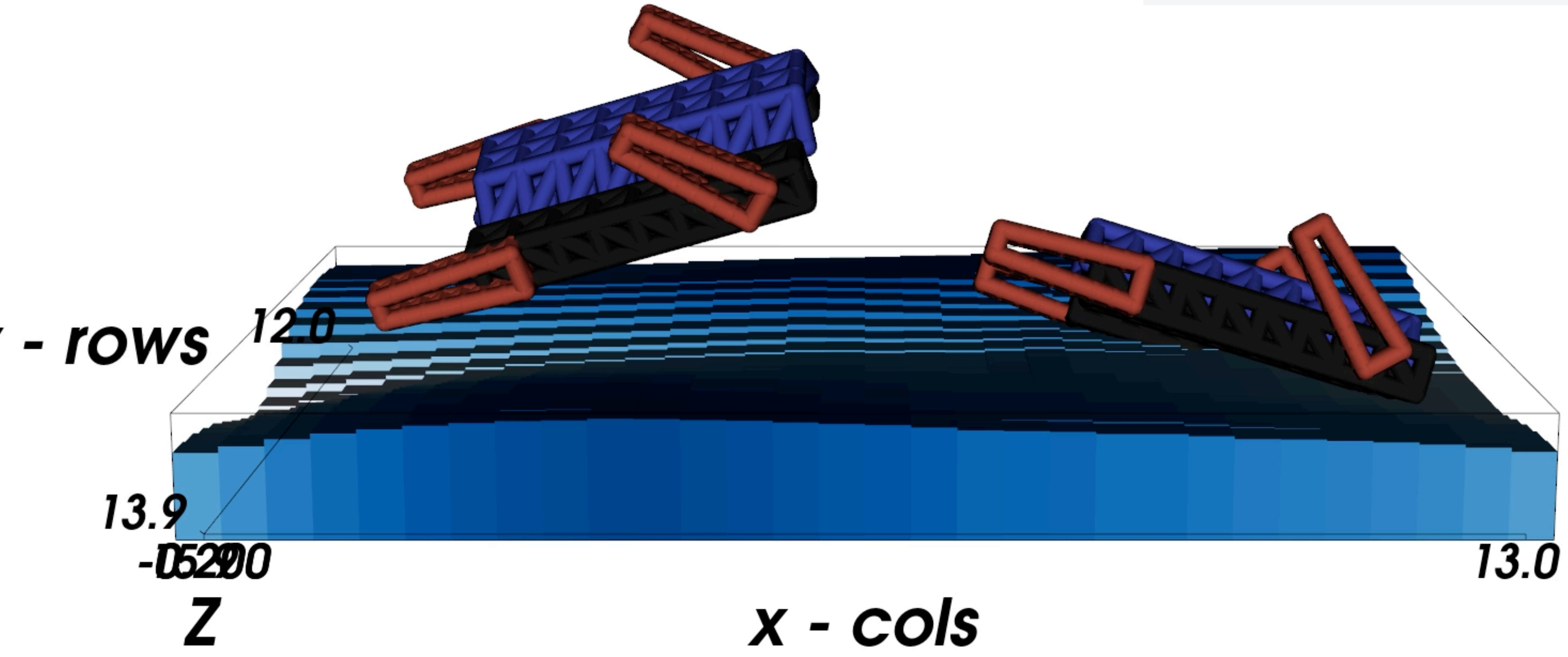
```
cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])
solution, = cvxpylayer(A, b) # feed-forward pass (solve the problem)
solution.sum().backward()    # backward pass (how A,b influnce solution)
```



https://www.cvxpy.org/

Pose consistency KKT-loss for weakly supervised learning of robot-terrain interaction model
[Salansky, Zimmermann, Petricek, Svoboda, RAL, 2021]

```
loss = loss_kkt(robot_pose, net(sparse_input), robot_model)
loss.backward()
```



ground truth robot pose Φ^*

$$\underset{\Phi=[\boldsymbol{\alpha},\mathbf{t}]}{\operatorname{argmin}} \sum_i m_i \cdot g \cdot [\mathbf{R}(\boldsymbol{\alpha}) \cdot \mathbf{p}_i + \mathbf{t}]_z$$

$$\hat{h}_i - [\mathbf{R}(\boldsymbol{\alpha}) \cdot \mathbf{p}_i + \mathbf{t}]_z \leq 0 \quad \forall_i$$

$\rightarrow \Phi$

heightmap $\hat{\mathbf{h}}$ predicted by convnet

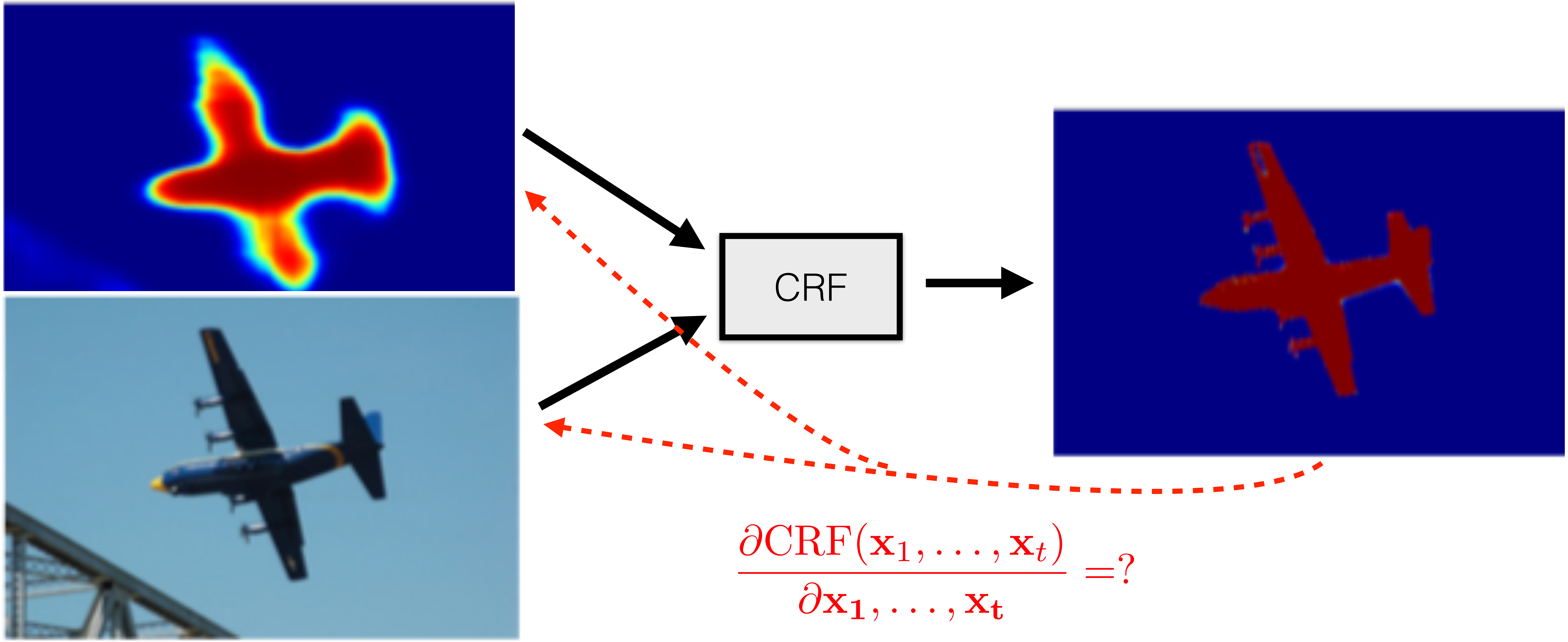
$$\frac{\partial \mathcal{L}_{\text{kkt}}(\boldsymbol{\phi}, \hat{\mathbf{h}})}{\partial \hat{\mathbf{h}}} = \frac{\partial}{\partial \hat{\mathbf{h}}} \min_{\boldsymbol{\lambda}} \left\{ \left\| \sum_i (m_i g - \lambda_i) \frac{\partial [\mathbf{R}(\boldsymbol{\alpha}) \cdot \mathbf{p}_i + \mathbf{t}]_z}{\partial \boldsymbol{\alpha}, \mathbf{t}} \right\|_2^2 + \right.$$
$$\left. + \sum_i \Big(\lambda_i \cdot (\hat{h}_i - [\mathbf{R}(\boldsymbol{\alpha}) \cdot \mathbf{p}_i + \mathbf{t}]_z) \Big)^2 + C \cdot \Big(\max\{0, \hat{h}_i - [\mathbf{R}(\boldsymbol{\alpha}) \cdot \mathbf{p}_i + \mathbf{t}]_z\} \Big)^2 \;\middle|\; \boldsymbol{\lambda} \geq 0 \right\}$$

Differentiable MPC control, [Amos et al. NIPS 2018]

https://arxiv.org/abs/1810.13400

PyTorch implementation of differentiable ConvCRF layer
[Teichmann & Cipolla, BMVC, 2019]
https://arxiv.org/pdf/1805.04777.pdf

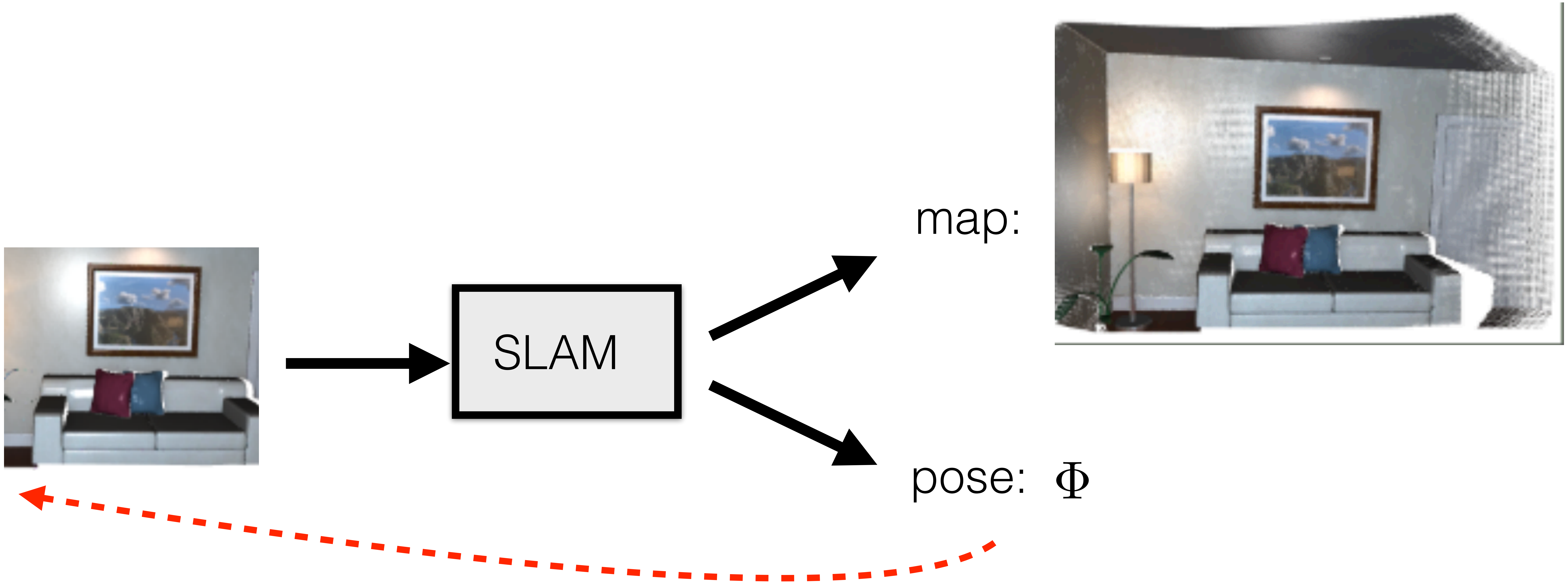
```
pred = gausscrf.forward(unary=unary_var, img=img_var)
```



$$\frac{\partial \mathrm{CRF}(\mathbf{x}_1,\ldots,\mathbf{x}_t)}{\partial \mathbf{x_1},\ldots,\mathbf{x_t}} = ?$$

Grad SLAM [Murthy, ICRA, 2021]

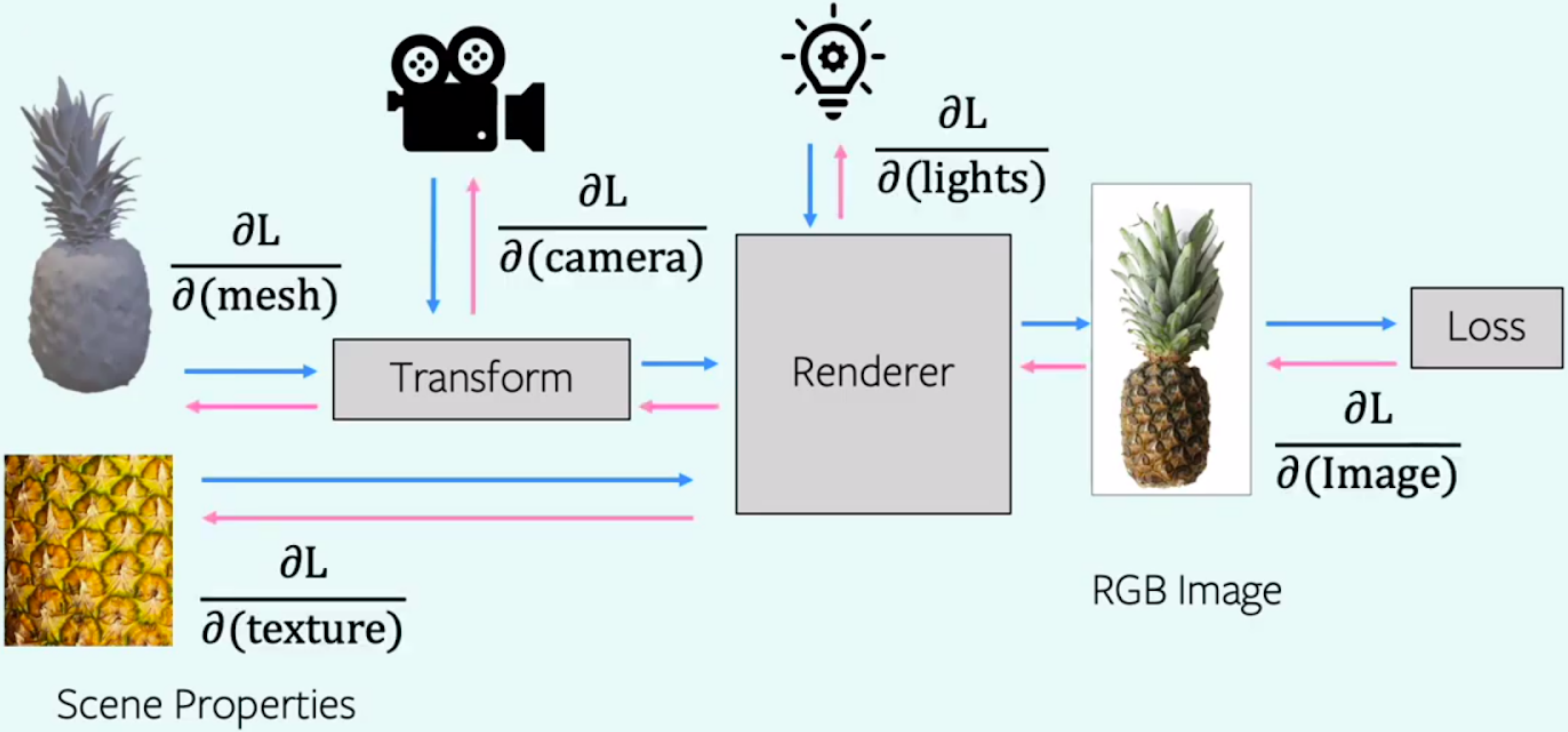
https://gradslam.github.io/



$$\frac{\partial \mathrm{SLAM}(\mathbf{x}_1, \ldots, \mathbf{x}_t)}{\partial \mathbf{x_1}, \ldots, \mathbf{x_t}} = ?$$

PyTorch3D
https://pytorch3d.org/

DIFFERENTIABLE RENDERING



Summary

- If accurate differentiable motion model and reward functions are known, than the optimal control is straightforward optimization problem (MPC)
- If model is not available, the RL can backpropagate through MDP, however sparse rewards makes action-reward-association problem very hard
- **Well engineered piece-wise architecture (object detection=> tracking=> planning/control) seems to be a better solution for typical robotic applications (explainable & manageable)**
- **Domain transfer is main bottleneck for real application !!!!**