Reinforcement learning in robotics

Karel Zimmermann

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

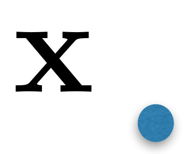
Vision for Robotics and Autonomous Systems
https://cyber.felk.cvut.cz/vras/

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

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

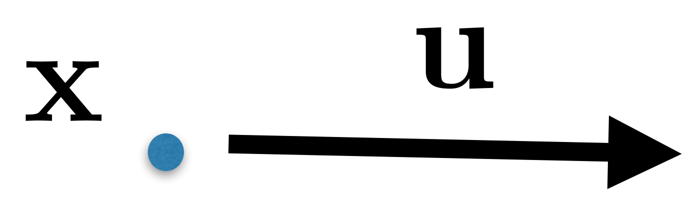
Problems often formalised as MDP

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



Problems often formalised as MDP

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



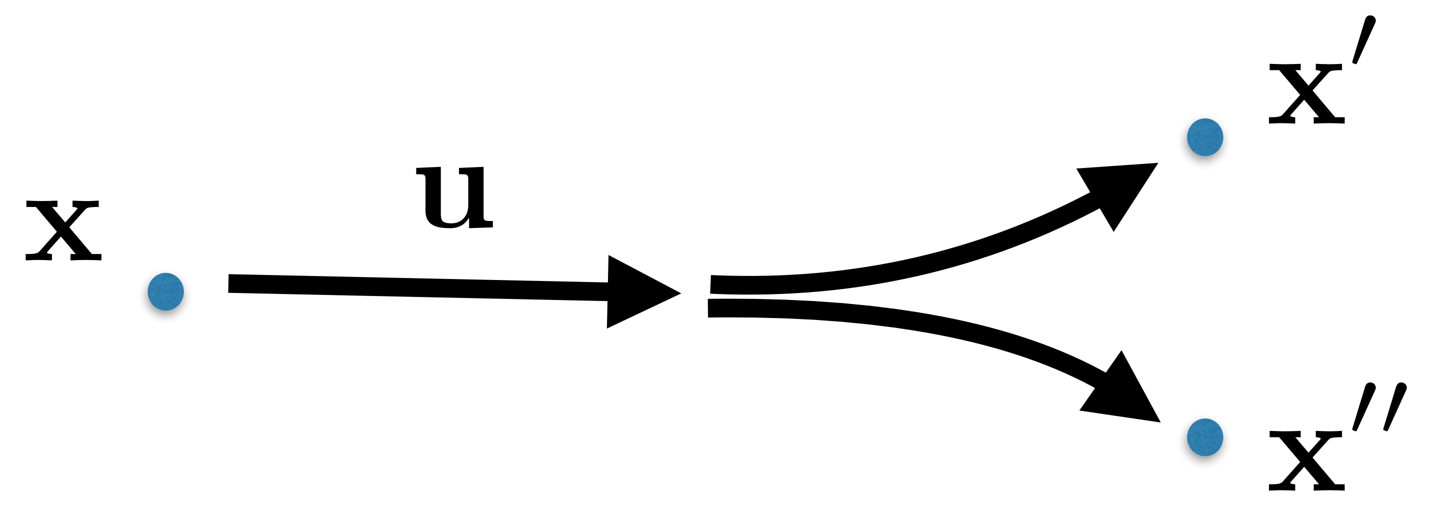
Actions: $\mathbf{u} \in \mathcal{R}^m$

Problems often formalised as MDP

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

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

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



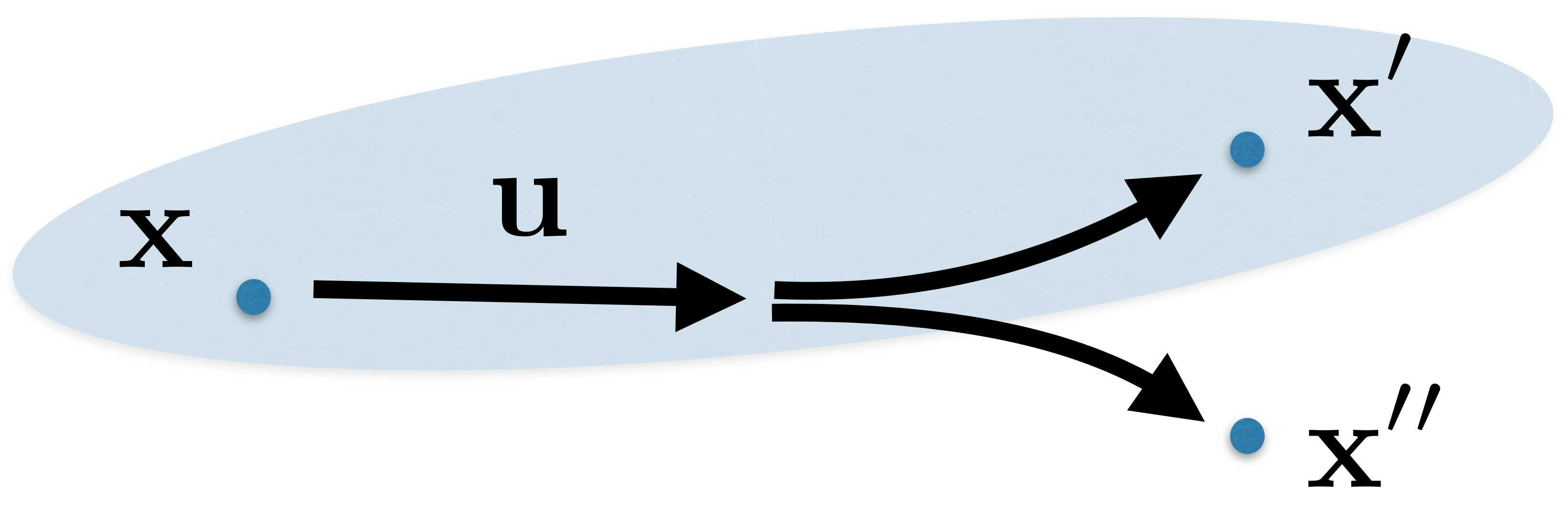
Problems often formalised as MDP

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

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

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

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



Problems often formalised as MDP

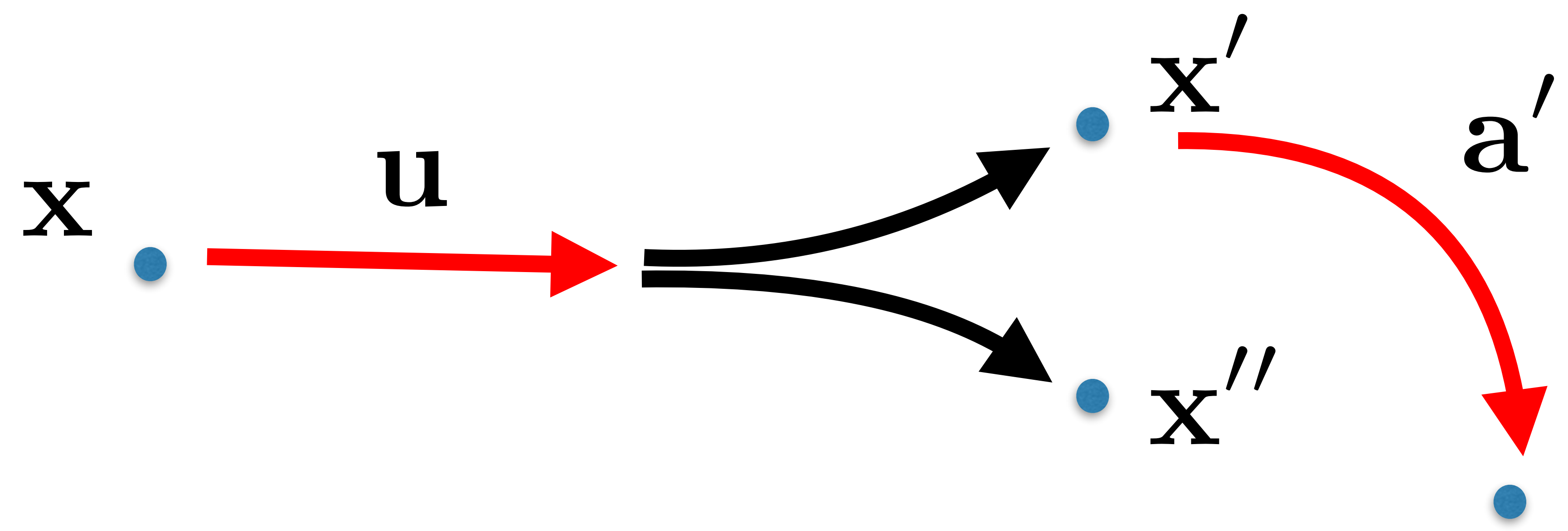
States: $\mathbf{x} \in \mathcal{R}^n$

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

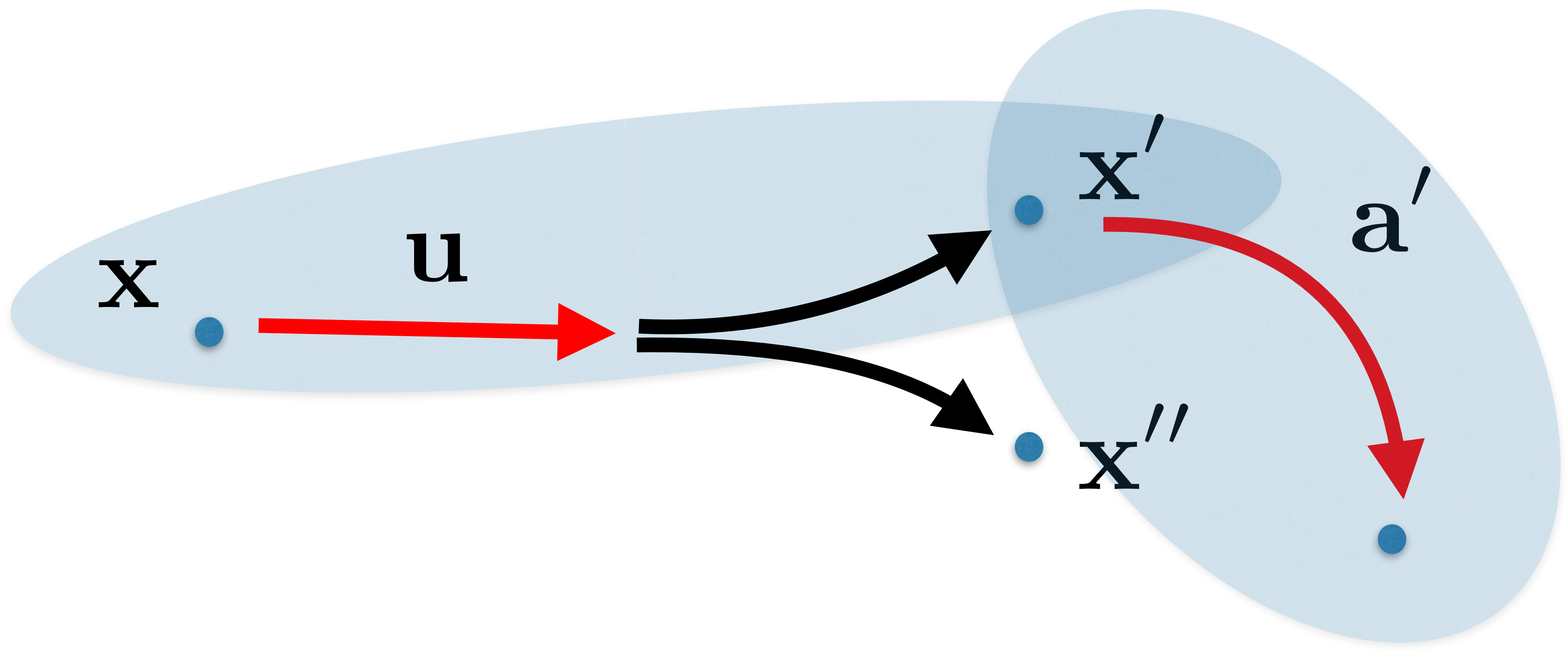
Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

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

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



Problems often formalised as MDP



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

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

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

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

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

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

Problems often formalised as MDP

States:	$\mathbf{x} \in \mathcal{R}^n$	incomplete, noisy
Actions:	$\mathbf{u} \in \mathcal{R}^m$	continuous high-dimensional
Model:	$p(\mathbf{x}'\|\mathbf{x}, \mathbf{u})$	inaccurate model
Rewards:	$r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$	hard to engineer
Policy:	$\pi(\mathbf{u}\|\mathbf{x})$	execution endanger the robot
Goal:	$\pi^* = \arg\max_{\pi} J_\pi$	(e.g. $J_\pi = \mathbb{E}_{\tau\sim\pi}\{\sum_{r_t\sim\tau} \gamma^t r_t\}$)

Typical problems

τ → **Model identification:**
- given some trajectories estimate model

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

$p(\mathbf{x}'|\mathbf{x},\mathbf{u})$, $r(\mathbf{x},\mathbf{u},\mathbf{x}')$ → **Model predictive control / Planning**
- given the model and reward estimate optimal policy/plan

→ $\pi^* = \arg\max_{\pi} J_\pi$

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 (→ $\mathbf{A}$, $\mathbf{B}$)

- 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

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

Model Predictive Control/Planning

- Given **model** and **reward function**, the criterion can be explicitly formulated and optimized

- You can either
  - **plan actions** (BFS, Dijkstra, A*, RRT, …)
    or
  - directly **optimize policy**, which generate actions (LQR)
- to maximize a reward function.

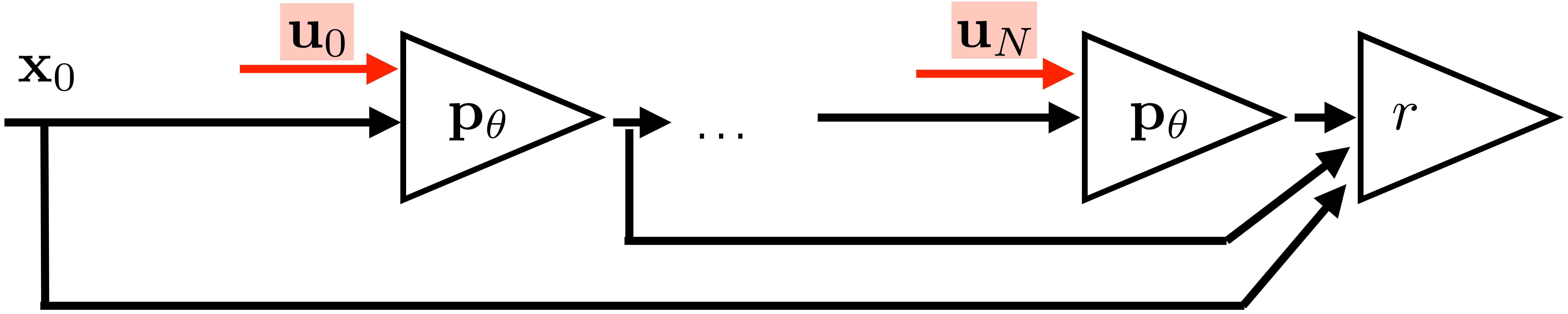
Planning actions

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**

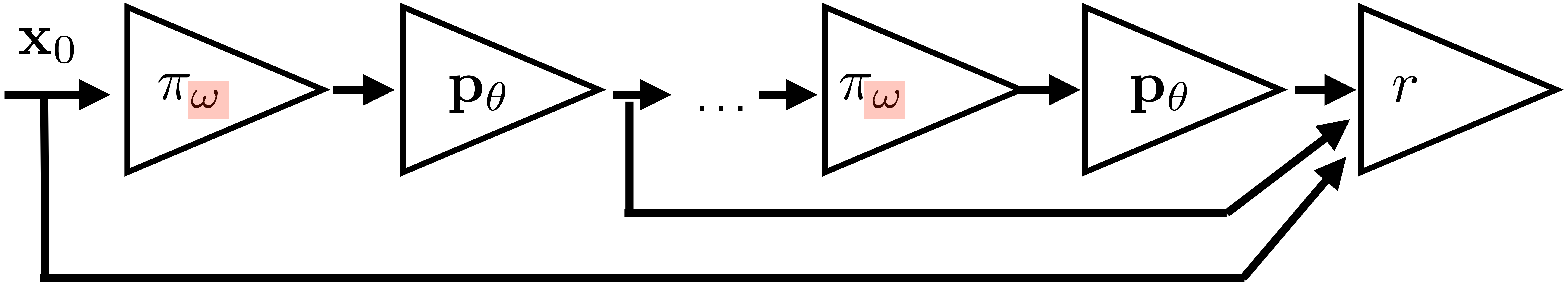
Learning policy

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/plan policy maximizing the rewards on model-based trajectories

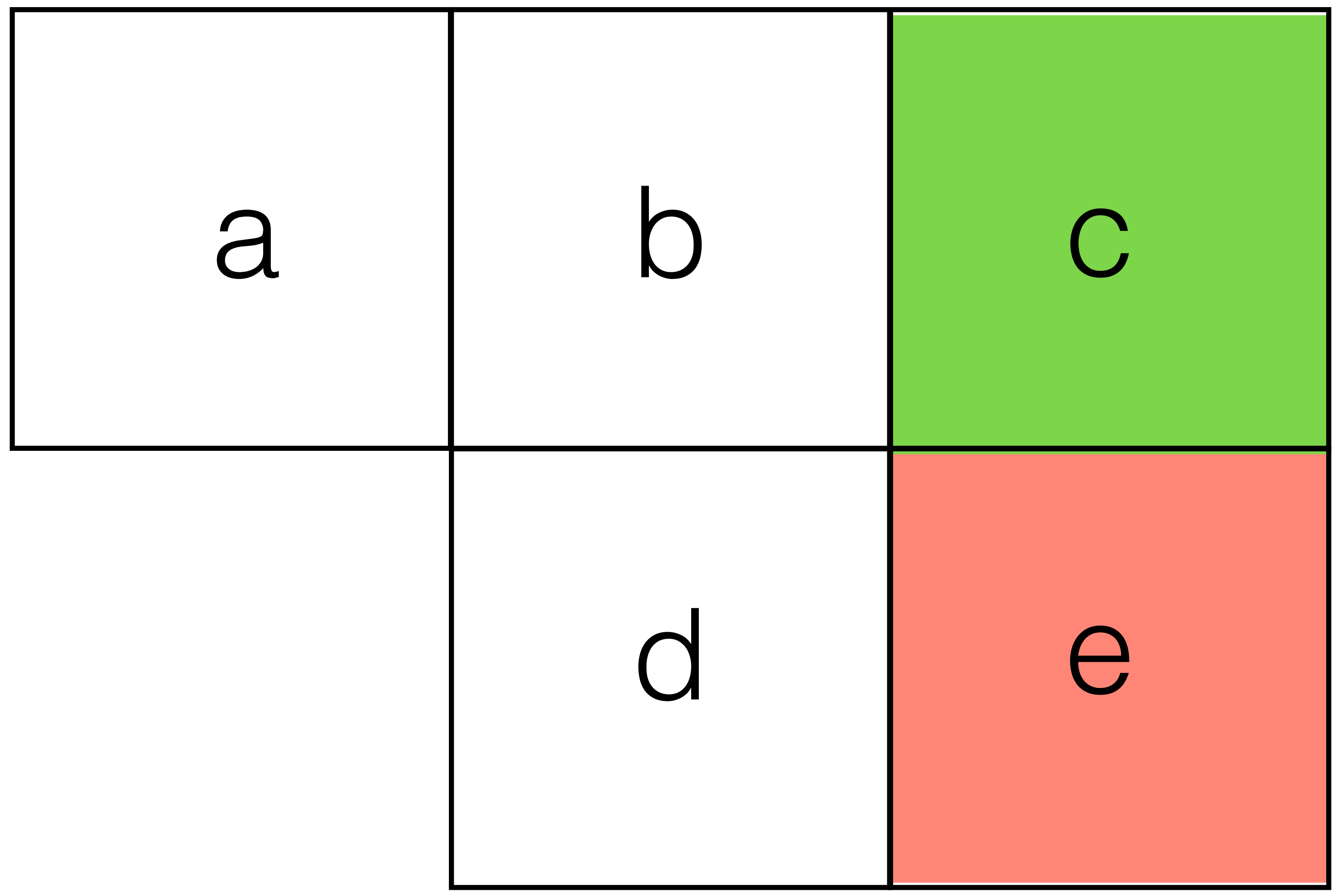
$$\arg\max_{\omega} \left\{ \sum_k r(\mathbf{x}_k, \mathbf{u}_k) \;\middle|\; \mathbf{x}_{k+1} = p_\theta(\mathbf{x}_k, \mathbf{u}_k),\ \mathbf{u}_k = \pi_\omega(\mathbf{x}_k) \right\}$$

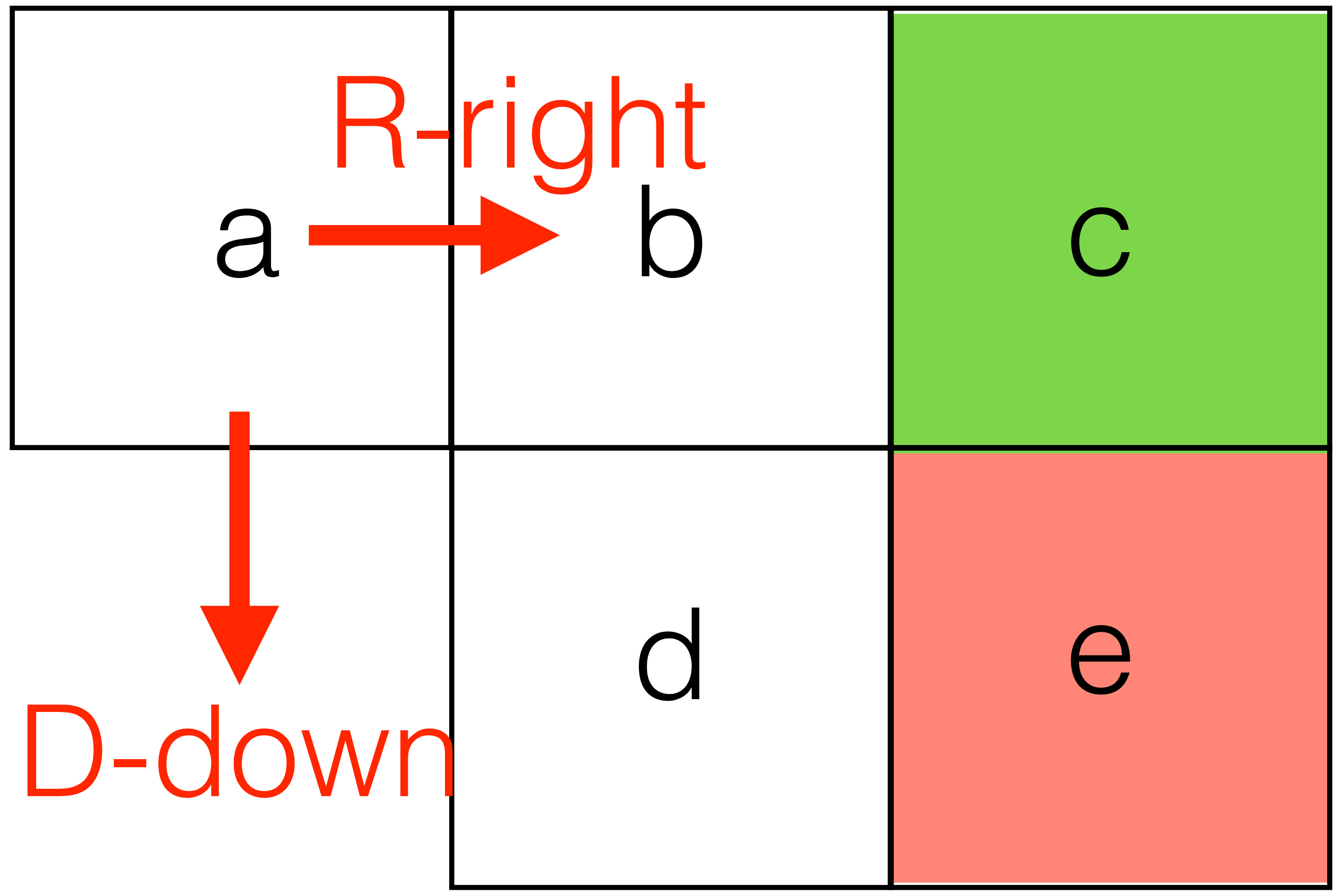


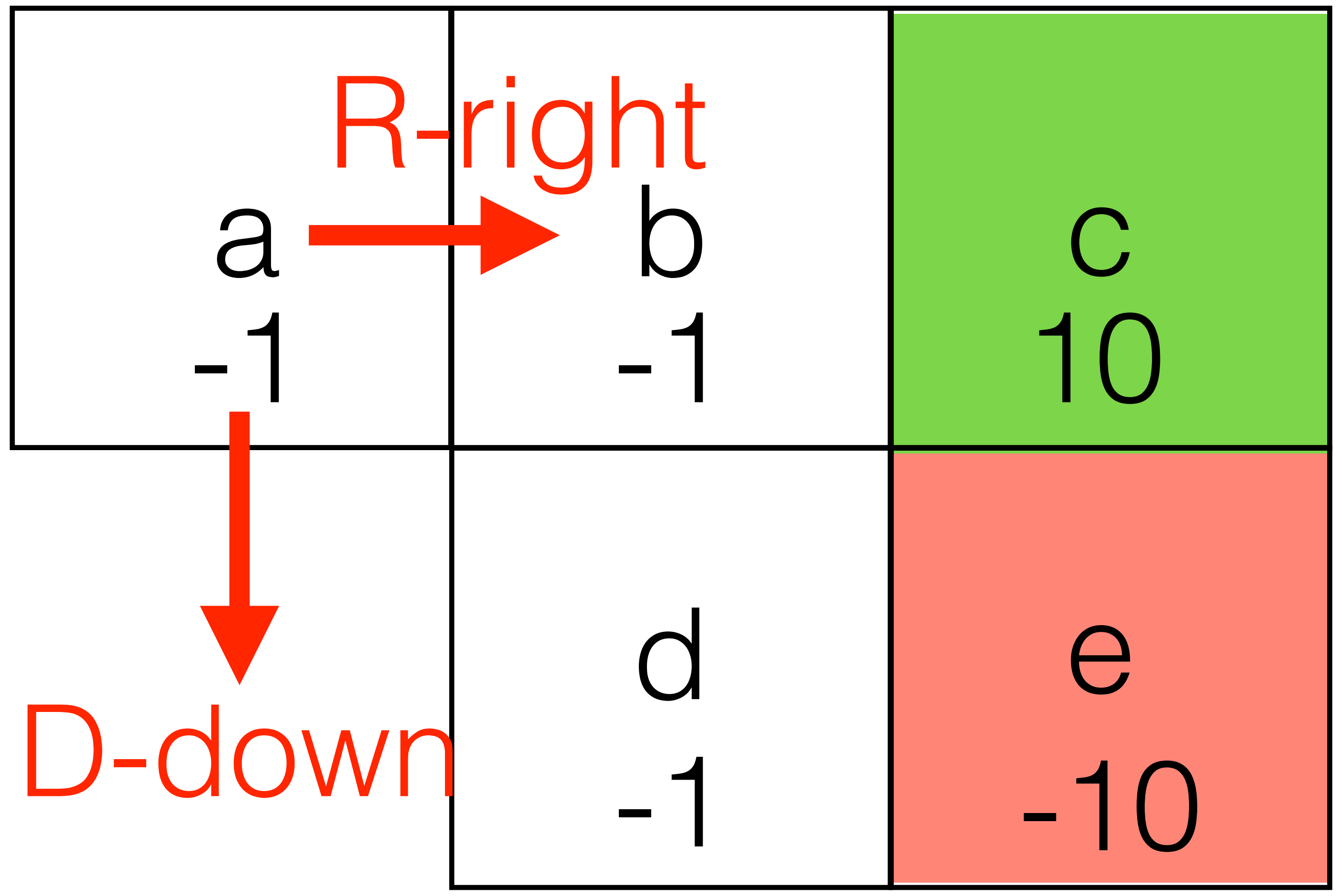
- Especially: linear system + quadratic reward function
- LQR has closed form solution => **closed loop control**

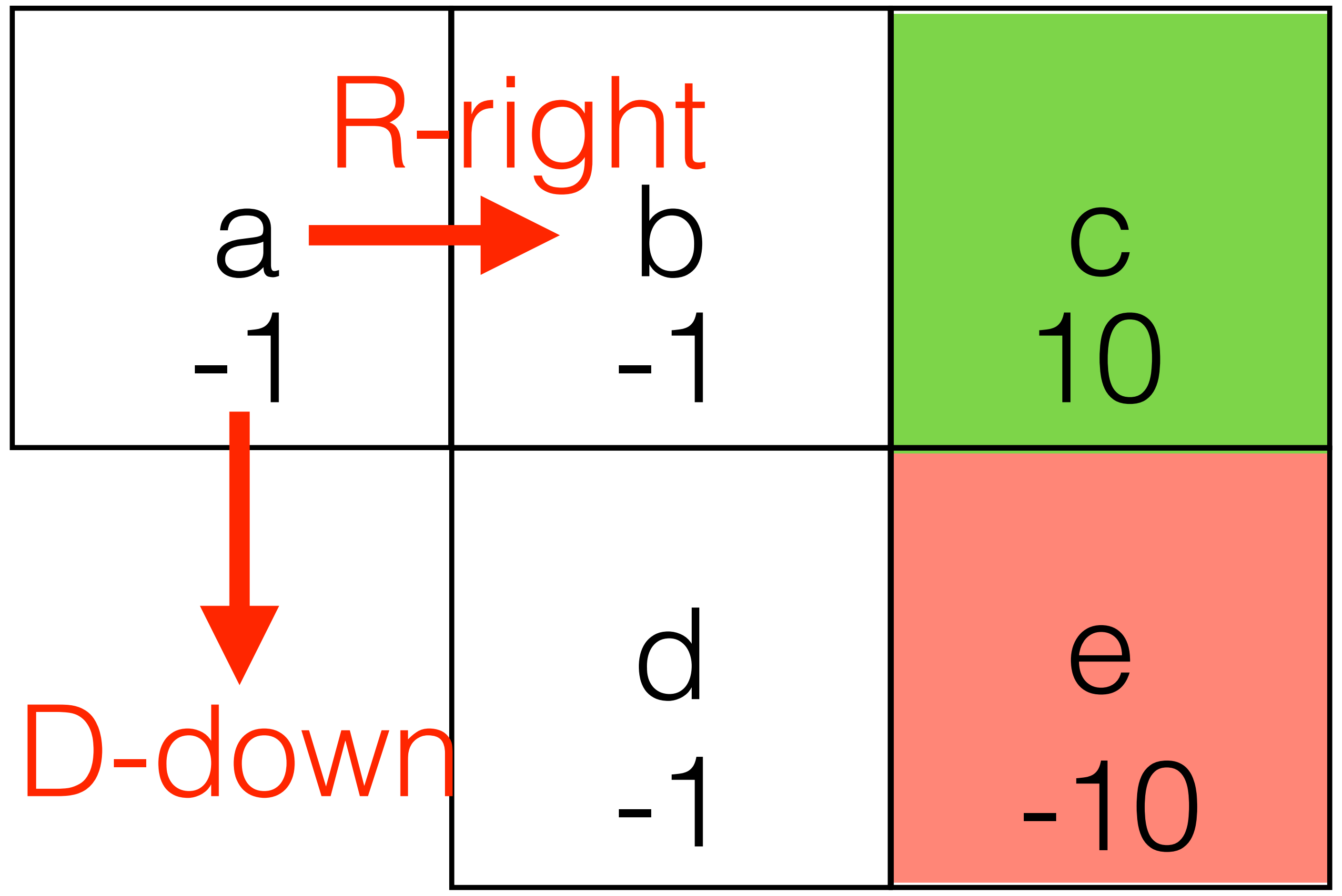
Typical problems

Input	Problem	Output
τ	**Model identification:** • given some trajectories estimate model	$p(\mathbf{x}' \mid \mathbf{x}, \mathbf{u})$
$p(\mathbf{x}' \mid \mathbf{x}, \mathbf{u})$ $r(\mathbf{x}, \mathbf{u}, \mathbf{x}')$	**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	$r(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathcal{R}$


a
b
c
d
e


R-right
a
b
c
D-down
d
e


R-right
a
-1
b
-1
c
10
D-down
d
-1
e
-10



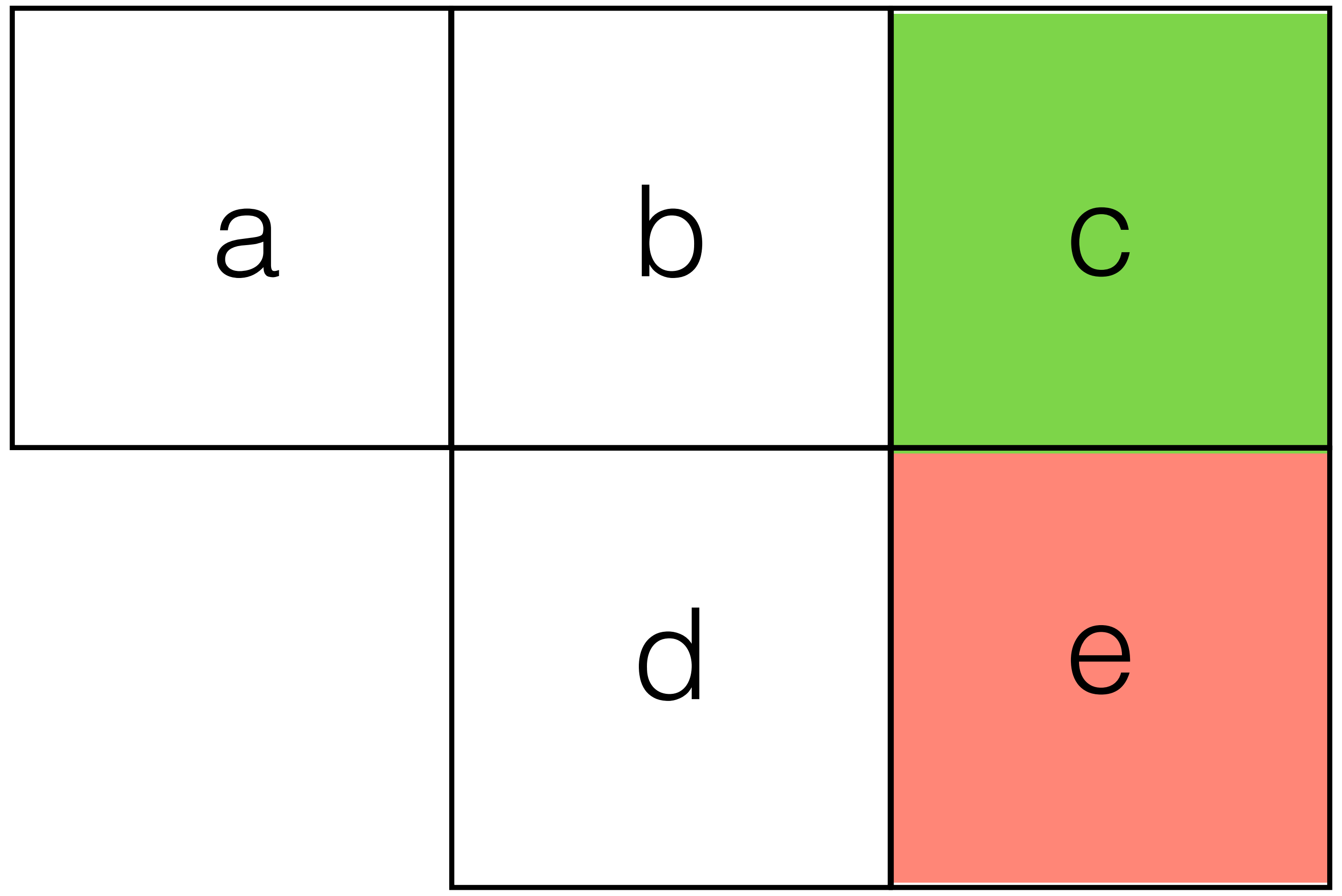
State-action value function

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

The best sum of rewards I can get, when following action u in state x and then controlling optimally

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

Q	R - right	D - down
a	8	7
b	9	-13
c	10	10
d	-11	-12
e	-10	-10



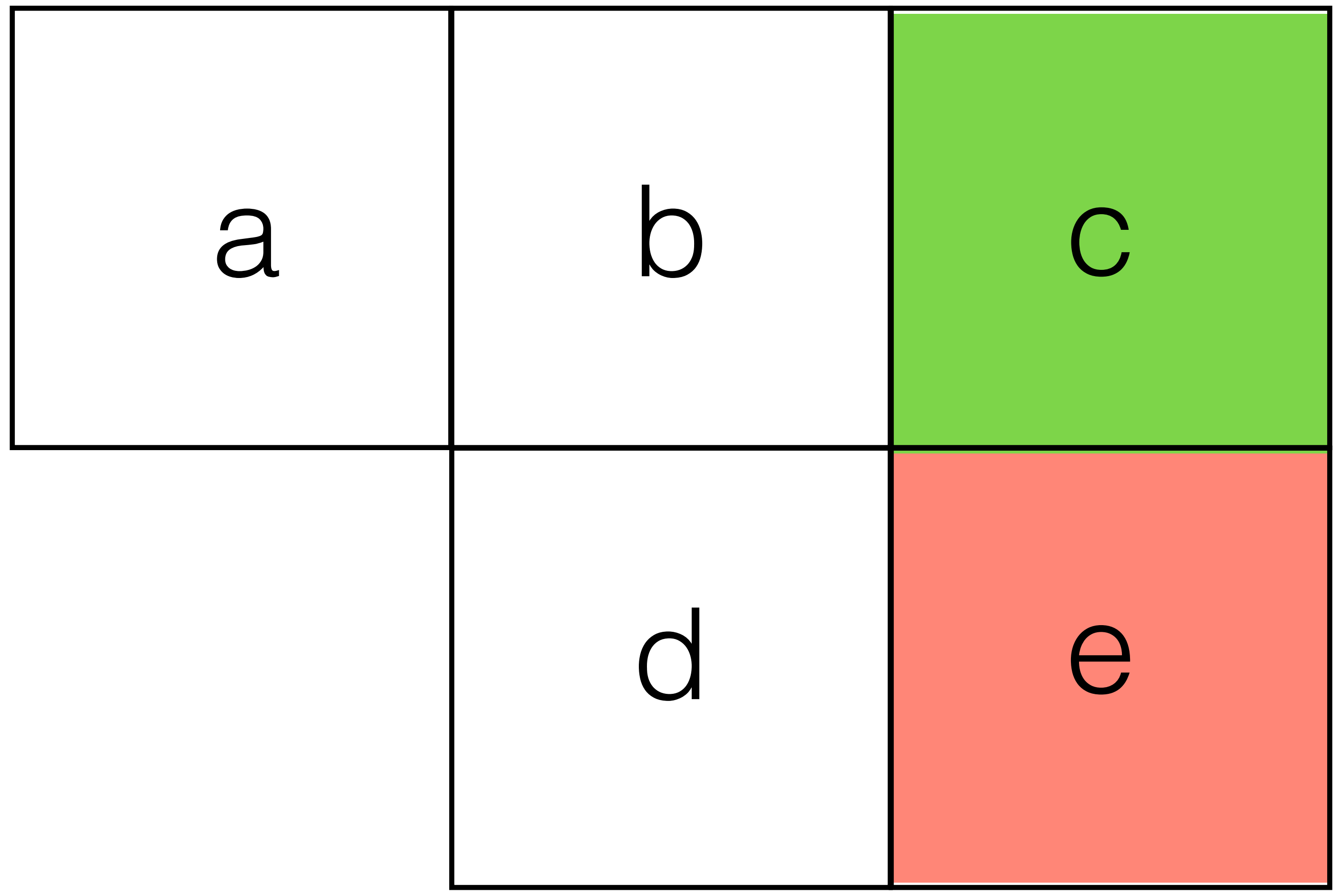
State-action value function

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$$

The best sum of rewards I can get, when following action u in state x and then controlling optimally

- Search for the Q, which satisfies Bellman equation

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

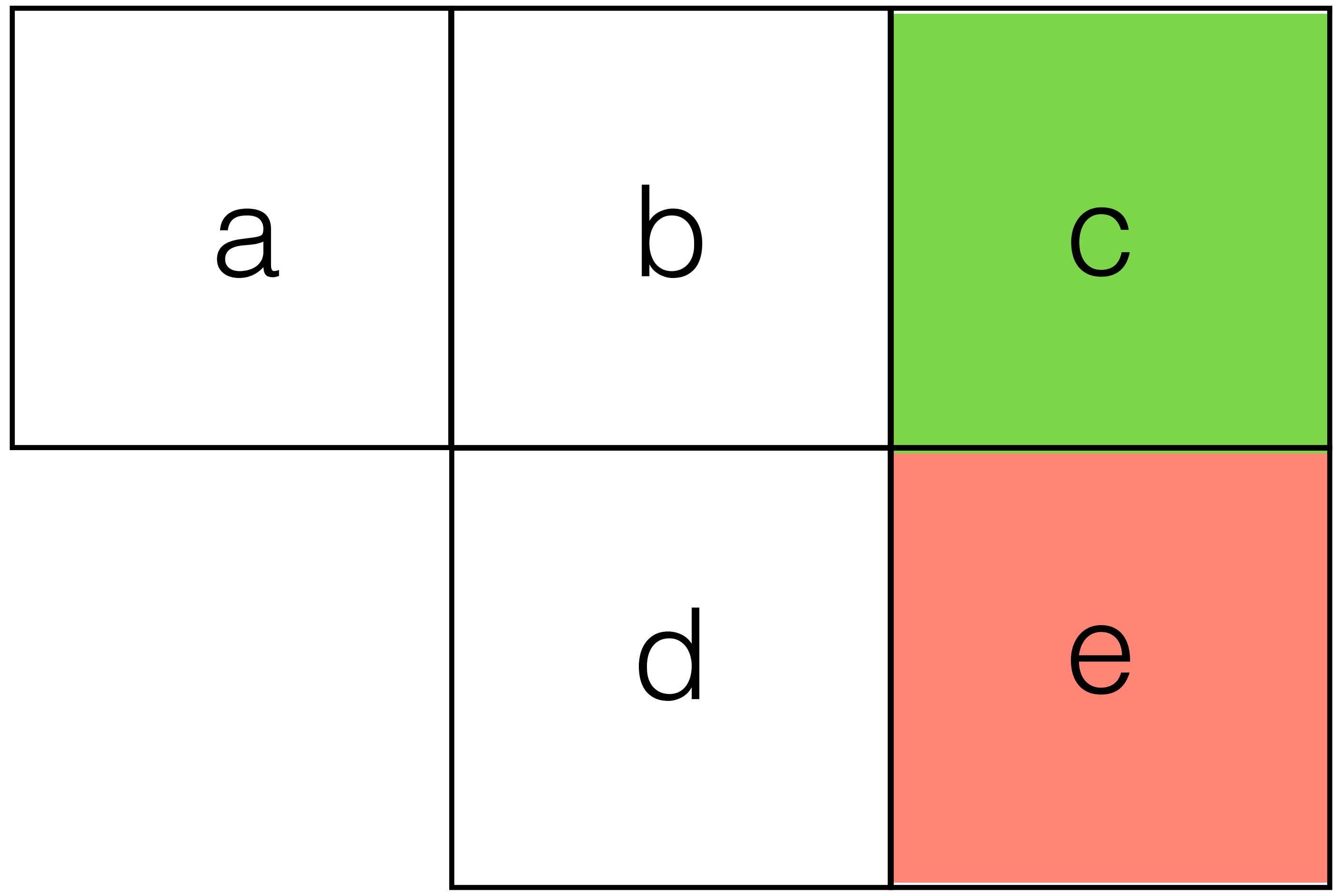


State-action value function

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

The best sum of rewards I can get, when following action u in state x and then controlling optimally

- Search for the Q, which satisfies Bellman equation
$$Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$$
- Once we find it, we can control optimally as follows:
$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \arg\max_{\pi} J_\pi$$

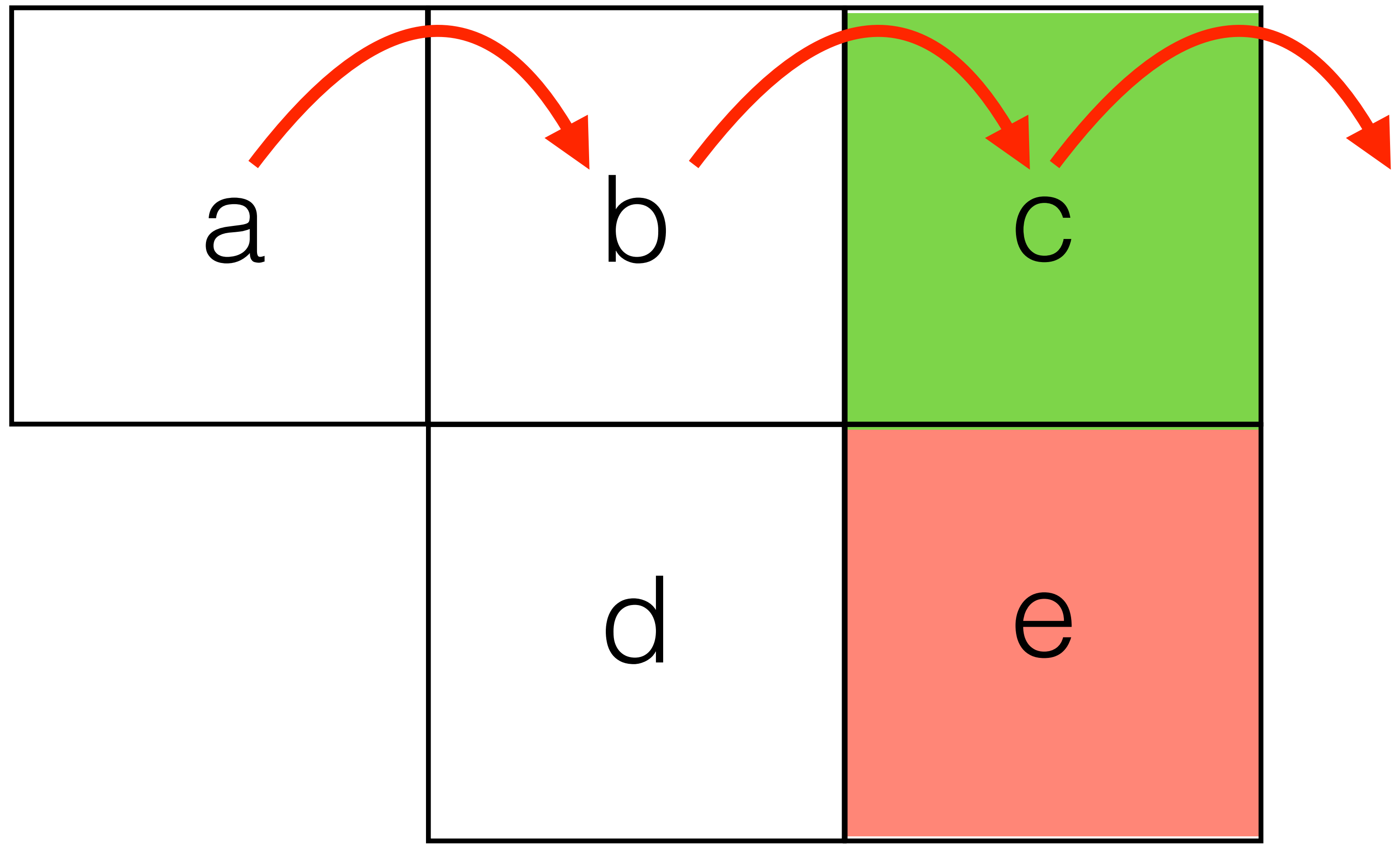


State-action value function

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$$

The best sum of rewards I can get, when following action u in state x and then controlling optimally

- Search for the Q, which satisfies Bellman equation
  $$Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$$
- Once we find it, we can control optimally as follows:
  $$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \arg\max_{\pi} J_\pi$$
- Search without model is based on collecting trajectories

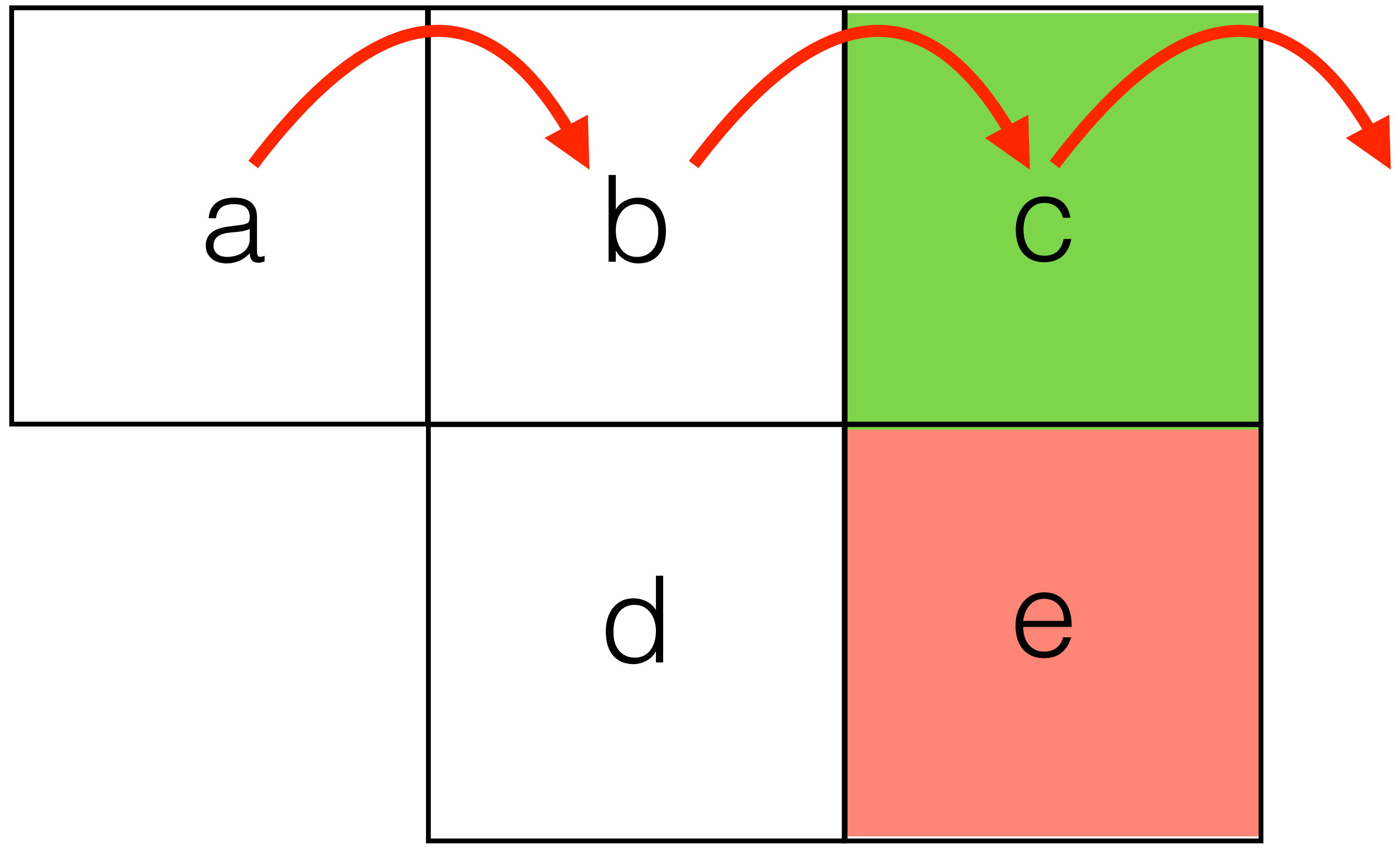


$\tau_1 : (a, R, \text{-}1),\ (b, R, \text{-}1),\ (c, R, 10)$

state action reward

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

Q	R - right	D - down
a	0	0
b	0	0
c	0	0
d	0	0
e	0	0



$$\tau_1 : (a, R, \text{-}1),\ (b, R, \text{-}1),\ (c, R, 10)$$

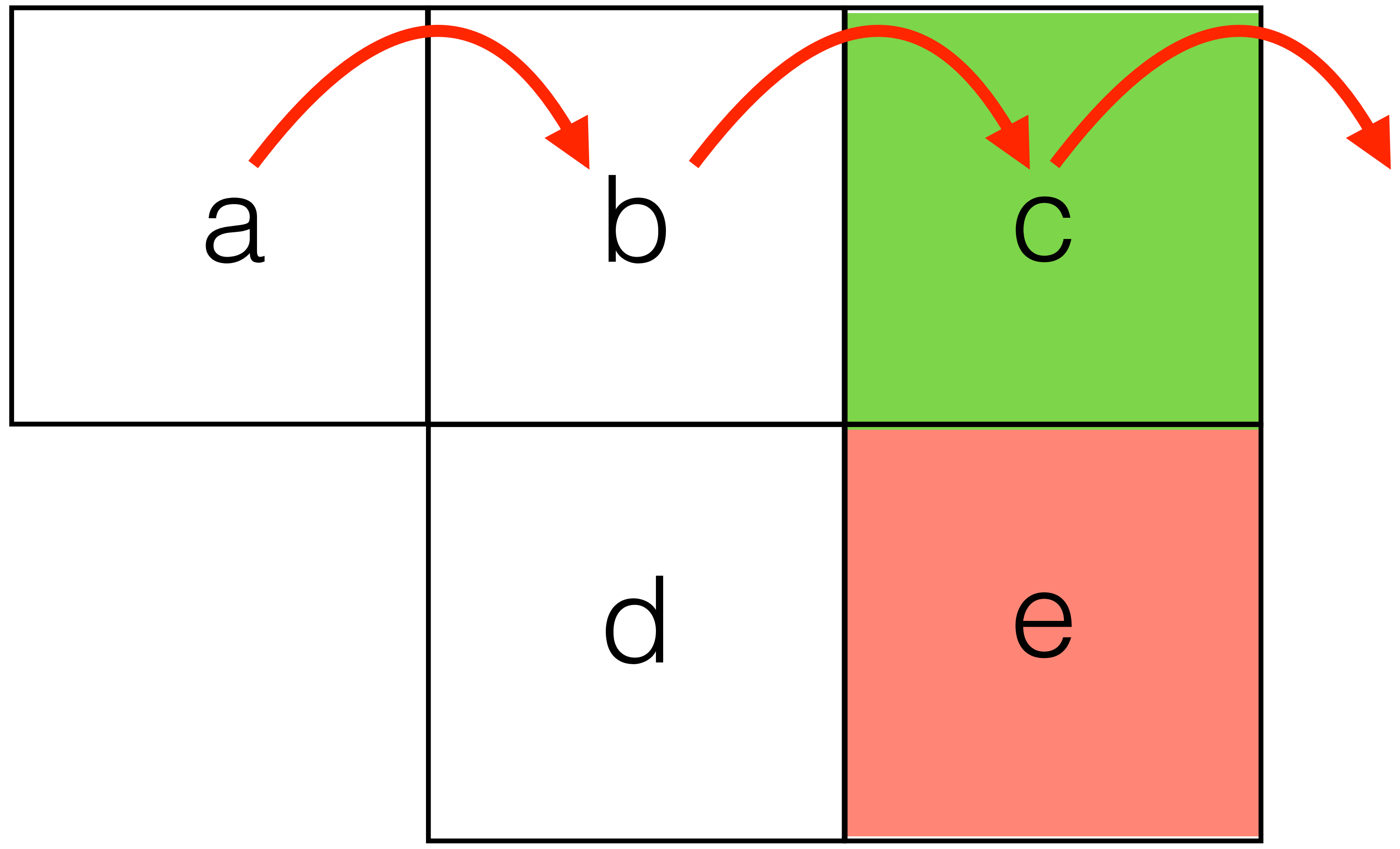
state $\mathbf{x}_1$ | action $\mathbf{u}_1$ | reward r_1 | next state $\mathbf{x}_2$

$$Q(\mathbf{x}_1, \mathbf{u}_1) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2, \mathbf{u})$$

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

Q	R - right	D - down
a	0	0
b	0	0
c	0	0
d	0	0
e	0	0

Having a trajectory, each transition gives one equation



$$\tau_1 : (a, R, \text{-}1),\ (b, R, \text{-}1),\ (c, R, 10)$$

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

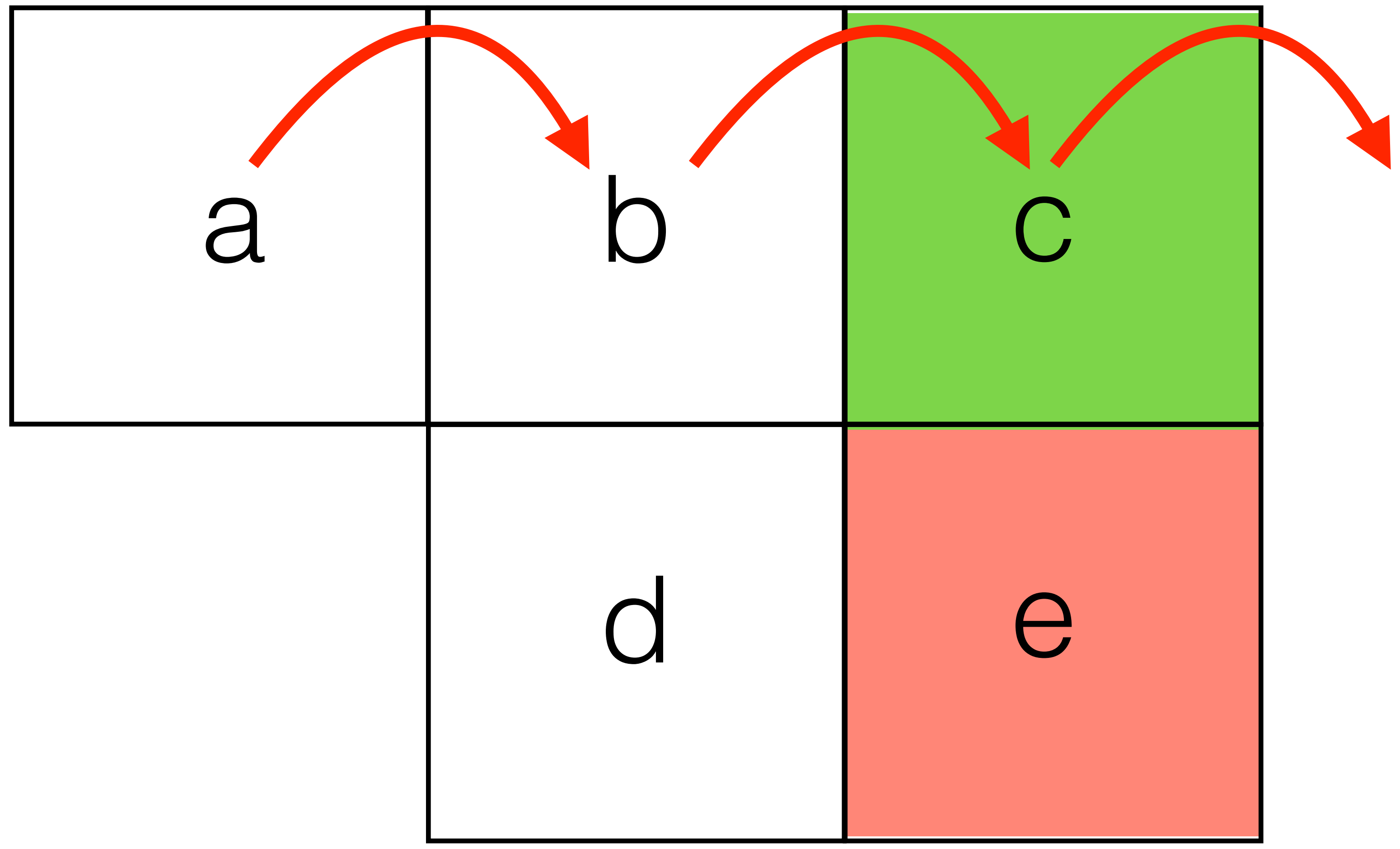
$$Q(\mathbf{x}_1, \mathbf{u}_1) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = -1 + 0.9 \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$$

Q	R - right	D - down
a	0	0
b	0	0
c	0	0
d	0	0
e	0	0

Having a trajectory, each transition gives one equation



$$\tau_1 : (a, R, \text{-}1),\ (b, R, \text{-}1),\ (c, R, 10)$$

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

$$Q(\mathbf{x}_1, \mathbf{u}_1) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = -1 + 0.9 \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$

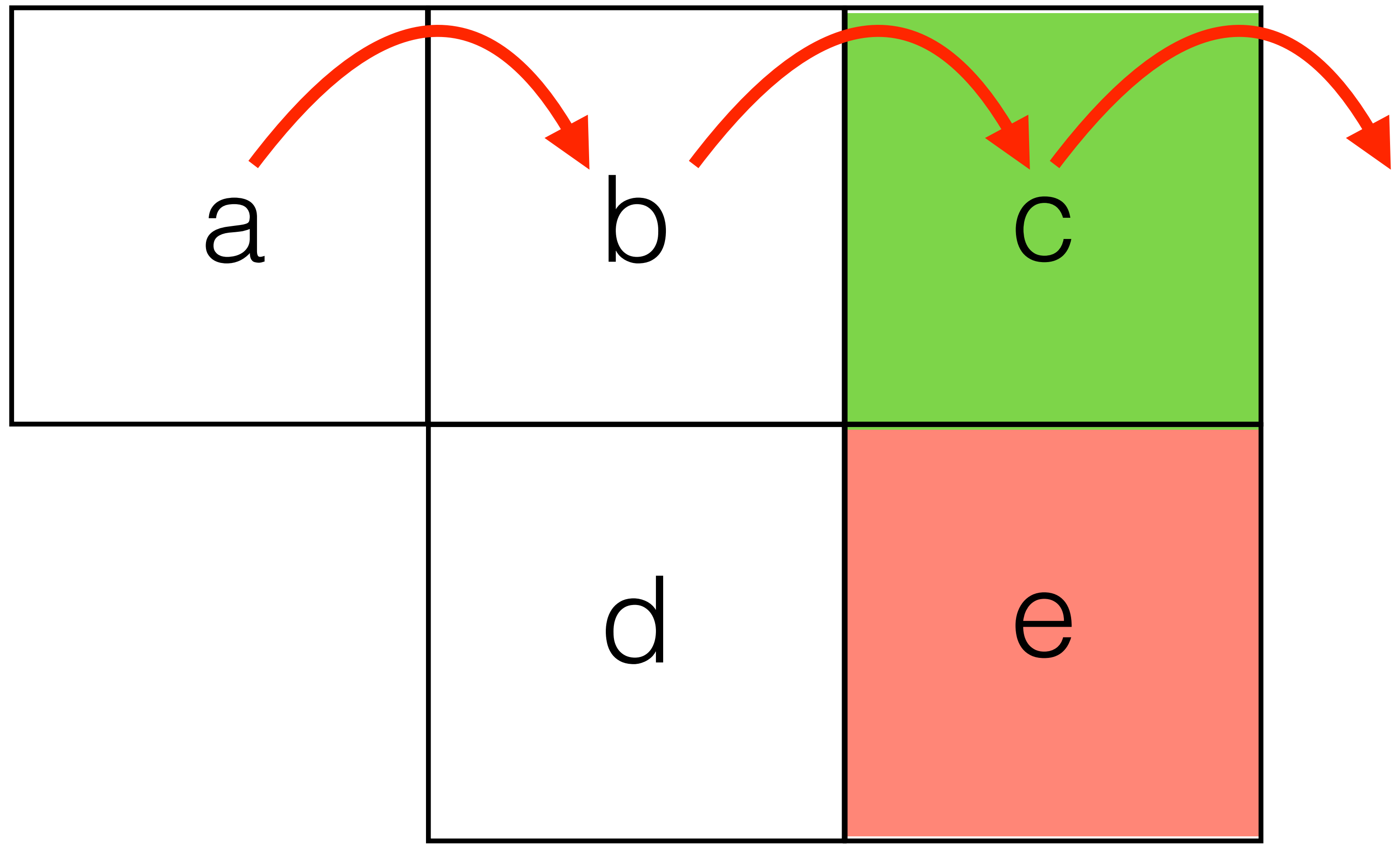
$$Q(\mathrm{b}, \mathrm{R}) = -1 + 0.9 \max_{\mathbf{u}} Q(\mathrm{c}, \mathbf{u})$$

$$Q(\mathrm{c}, \mathrm{R}) = 10$$

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$$

Q	R - right	D - down
a	0	0
b	0	0
c	0	0
d	0	0
e	0	0

Having a trajectory, each transition gives one equation



$\tau_1 : (a, R, \text{-}1),\ (b, R, \text{-}1),\ (c, R, 10)$

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

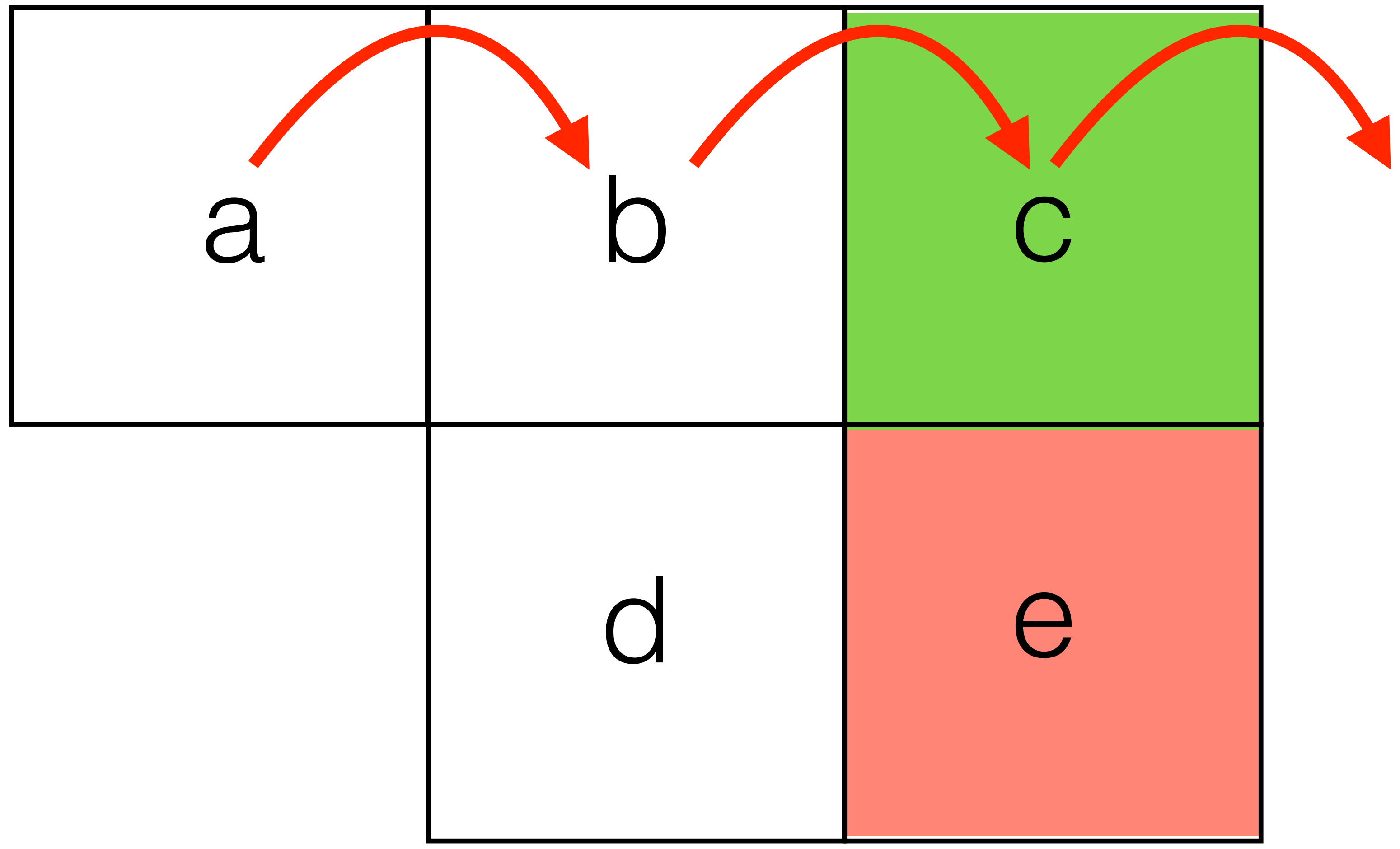
$$Q(\mathbf{x}_1, \mathbf{u}_1) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = -1 + 0.9 \max_{\mathbf{u}} Q(\mathrm{b}, \mathbf{u})$$
$$Q(\mathrm{b}, \mathrm{R}) = -1 + 0.9 \max_{\mathbf{u}} Q(\mathrm{c}, \mathbf{u})$$
$$Q(\mathrm{c}, \mathrm{R}) = 10$$

(1) Substitute RHS Q-values and recompute LHS Q-values

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

Q	R - right	D - down
a	0	0
b	0	0
c	0	0
d	0	0
e	0	0



$\tau_1 : (a, R, \text{-}1), \ (b, R, \text{-}1), \ (c, R, 10)$

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

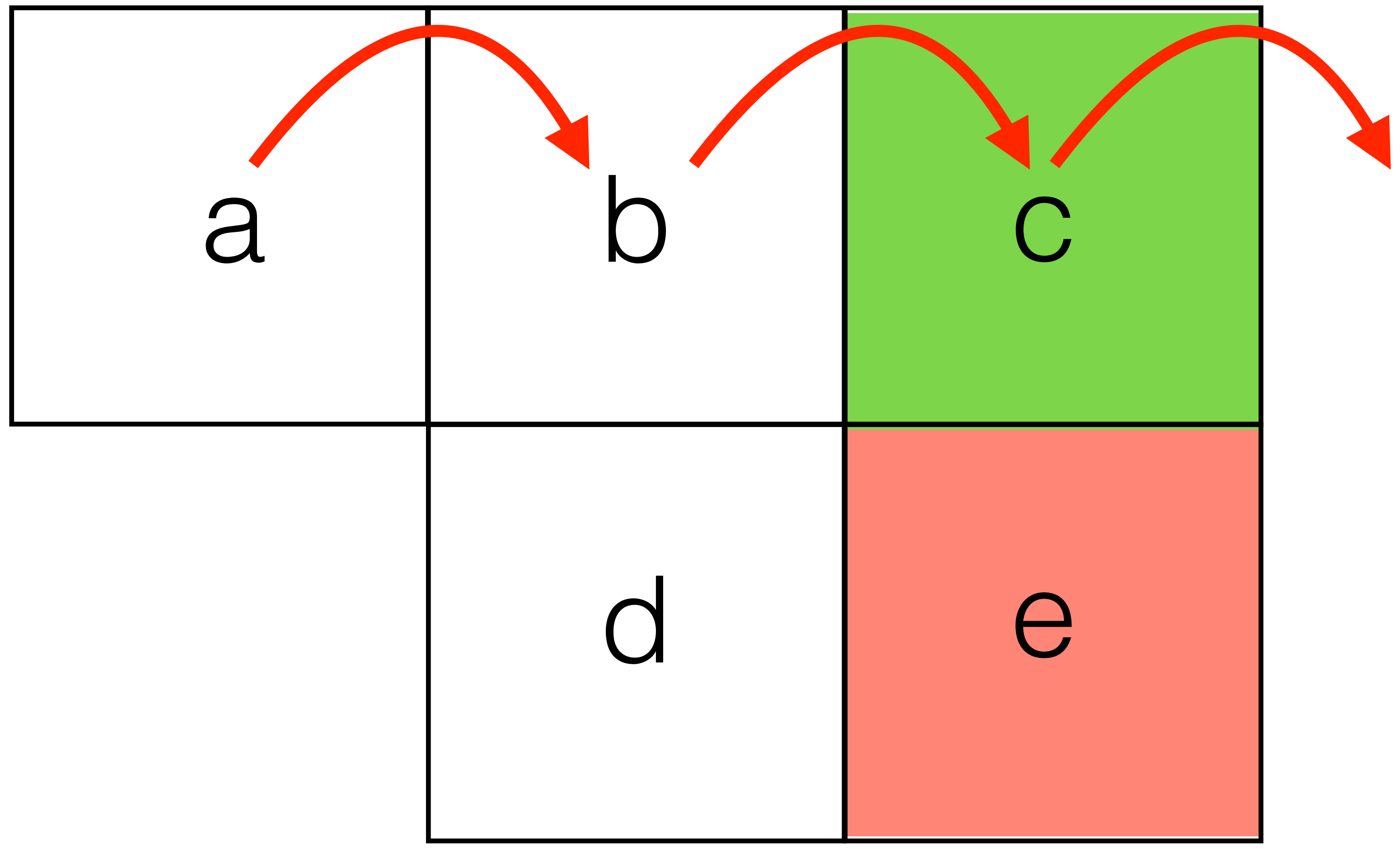
$$Q(\mathbf{x}_1, \mathbf{u}_1) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2, \mathbf{u})$$

$$Q(\mathrm{a}, \mathrm{R}) = -1 + 0$$
$$Q(\mathrm{b}, \mathrm{R}) = -1 + 0$$
$$Q(\mathrm{c}, \mathrm{R}) = 10$$

(1) Substitute RHS Q-values and recompute LHS Q-values

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$$

Q	R - right	D - down
a	-1	0
b	-1	0
c	10	0
d	0	0
e	0	0



$$\tau_1 : (a, R, \text{-}1),\ (b, R, \text{-}1),\ (c, R, 10)$$

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

$$Q(\mathbf{x}_1, \mathbf{u}_1) = r_1 + \gamma \max_{\mathbf{u}} Q(\mathbf{x}_2, \mathbf{u})$$

$$Q(\text{a}, \text{R}) = -1 + 0.9 \max_{\mathbf{u}} Q(\text{b}, \mathbf{u})$$
$$Q(\text{b}, \text{R}) = -1 + 0.9 \max_{\mathbf{u}} Q(\text{c}, \mathbf{u})$$
$$Q(\text{c}, \text{R}) = 10$$

$$Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$$

Q	R - right	D - down
a	-1	0
b	-1	0
c	10	0
d	0	0
e	0	0

(1) Substitute RHS Q-values and recompute LHS Q-values

(2) Repeat several times (search for the fixed point of the Bellman operator)

$$Q = \mathcal{B}(Q)$$

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

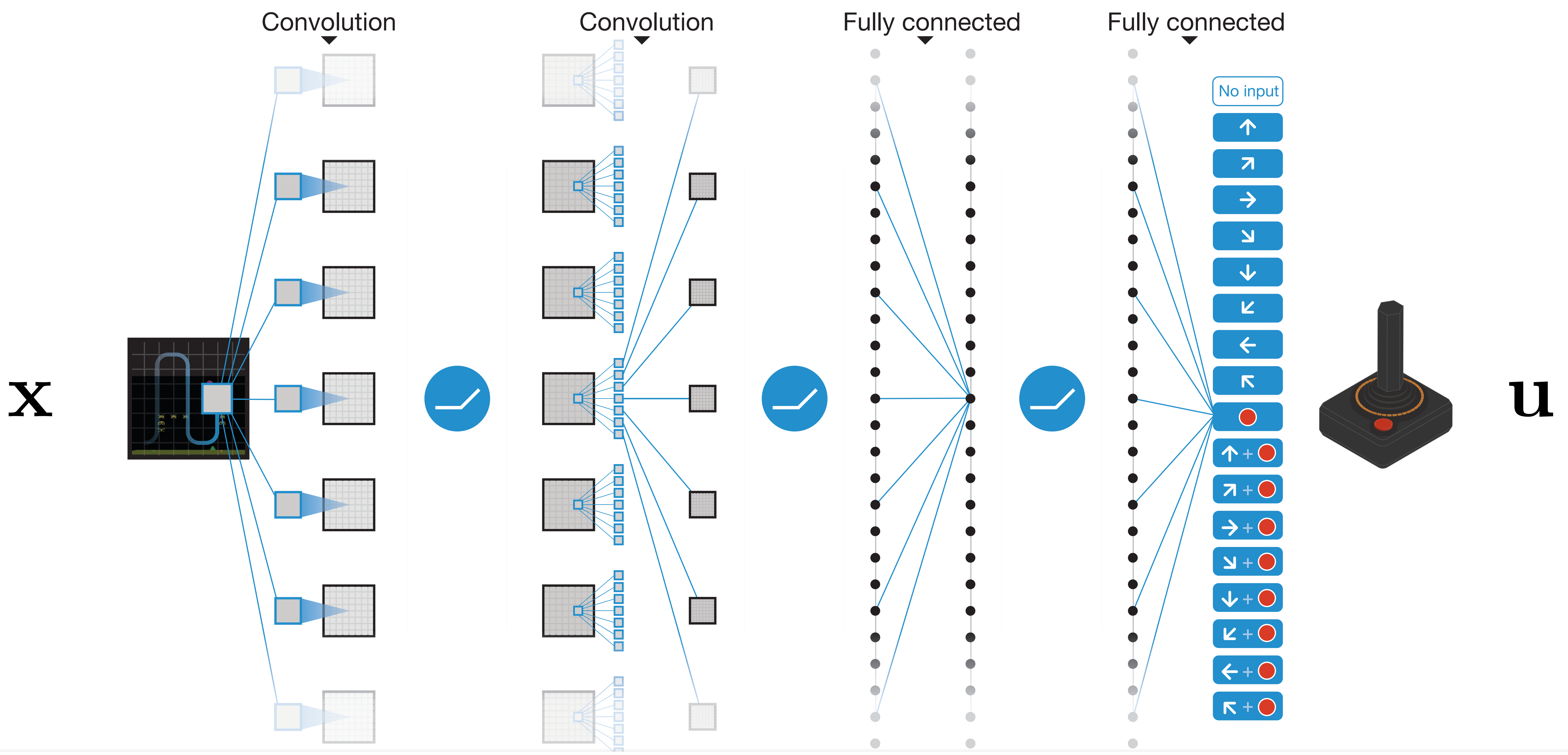
$$\arg\min_\theta \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

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)

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



Why not to use $\mathbf{u}$ in the input?

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.)

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

★ **Samples are strongly correlated !**

4. Repeat from 1

★ **Samples 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.)

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

★ **Samples are strongly correlated !**

4. 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.)

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

★ **Samples 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

★ **Samples are strongly correlated !**

★ **Approximated Q-learning does not have to converge to a fixed point.**

★ Approximated Q-learning does not have to converge to a fixed point.

Solution: Two Q-networks:

- Target net $Q_{\overline{\theta}}(\mathbf{x}, \mathbf{u})$
  (slowly updated, used for estimating targets)
- Policy net $Q_{\theta}(\mathbf{x}, \mathbf{u})$
  (regularly updated after each transition, used for exploration)

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

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

★ **Samples are strongly correlated !**

★ **Approximated Q-learning does not have to converge to a fixed point.**

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) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$
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

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) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

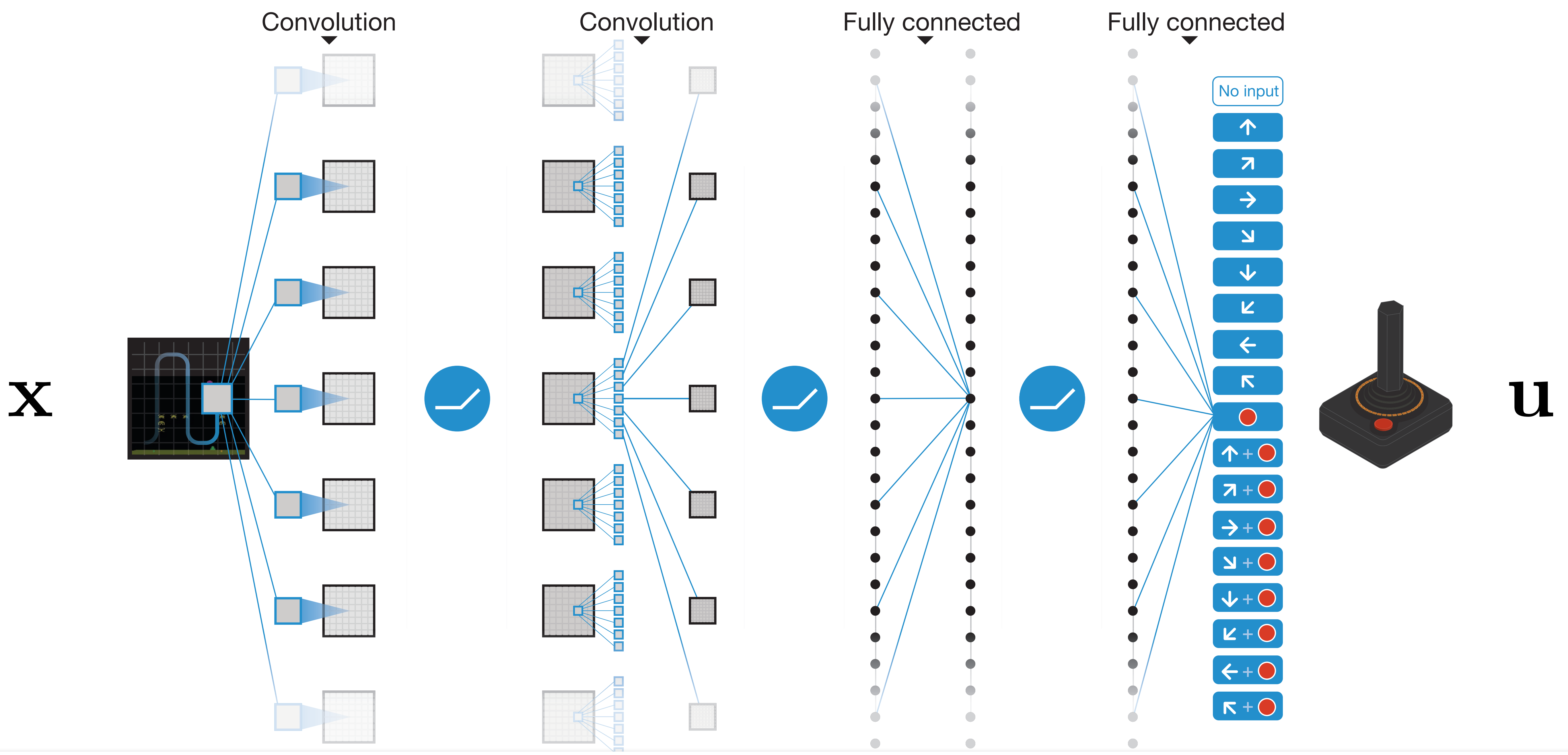
$$\arg\min_\theta \sum_{\mathbf{x},\mathbf{u},\mathbf{y}} \|Q_\theta(\mathbf{x}, \mathbf{u}) - \mathbf{y}\|$$

5. 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)

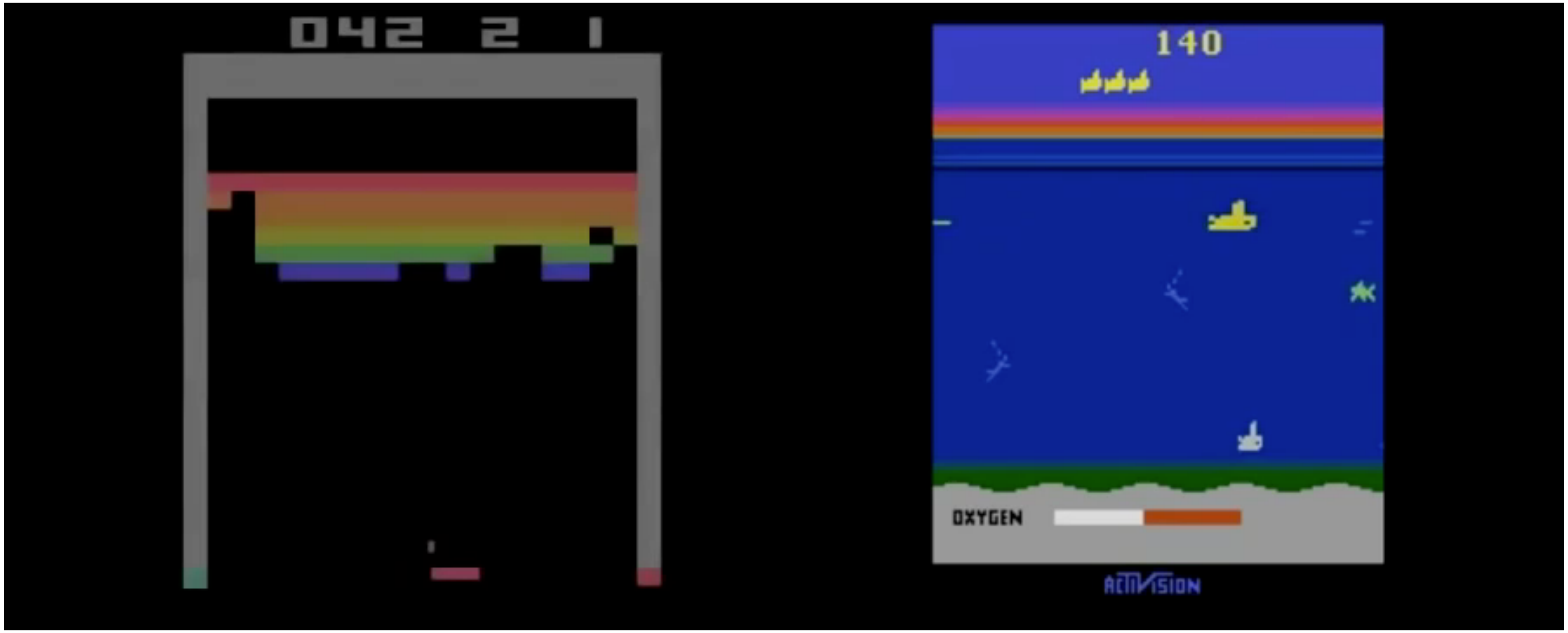
$$Q_\theta(\mathbf{x}, \mathbf{u})$$



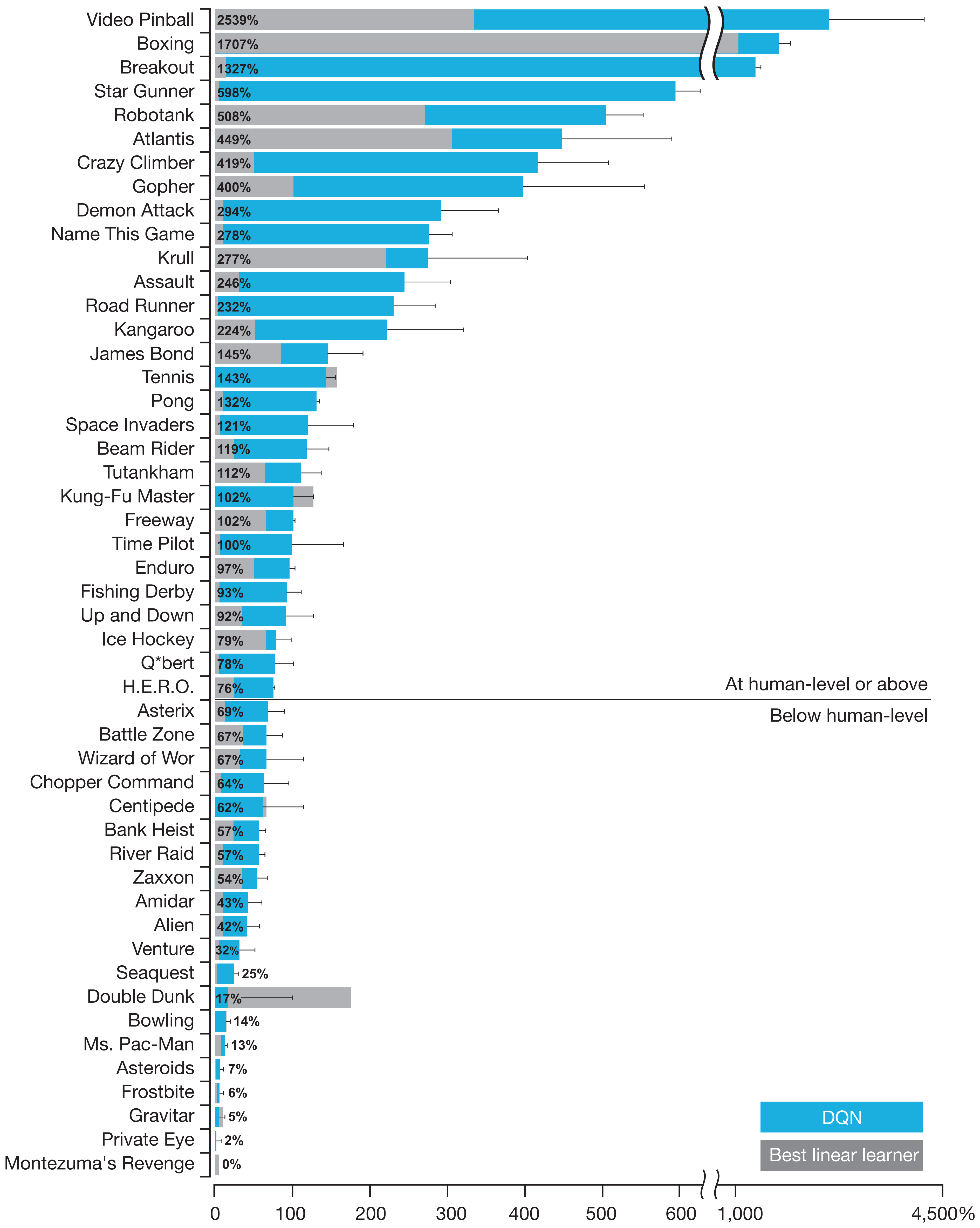
Mnih et al. Nature 2015

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

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

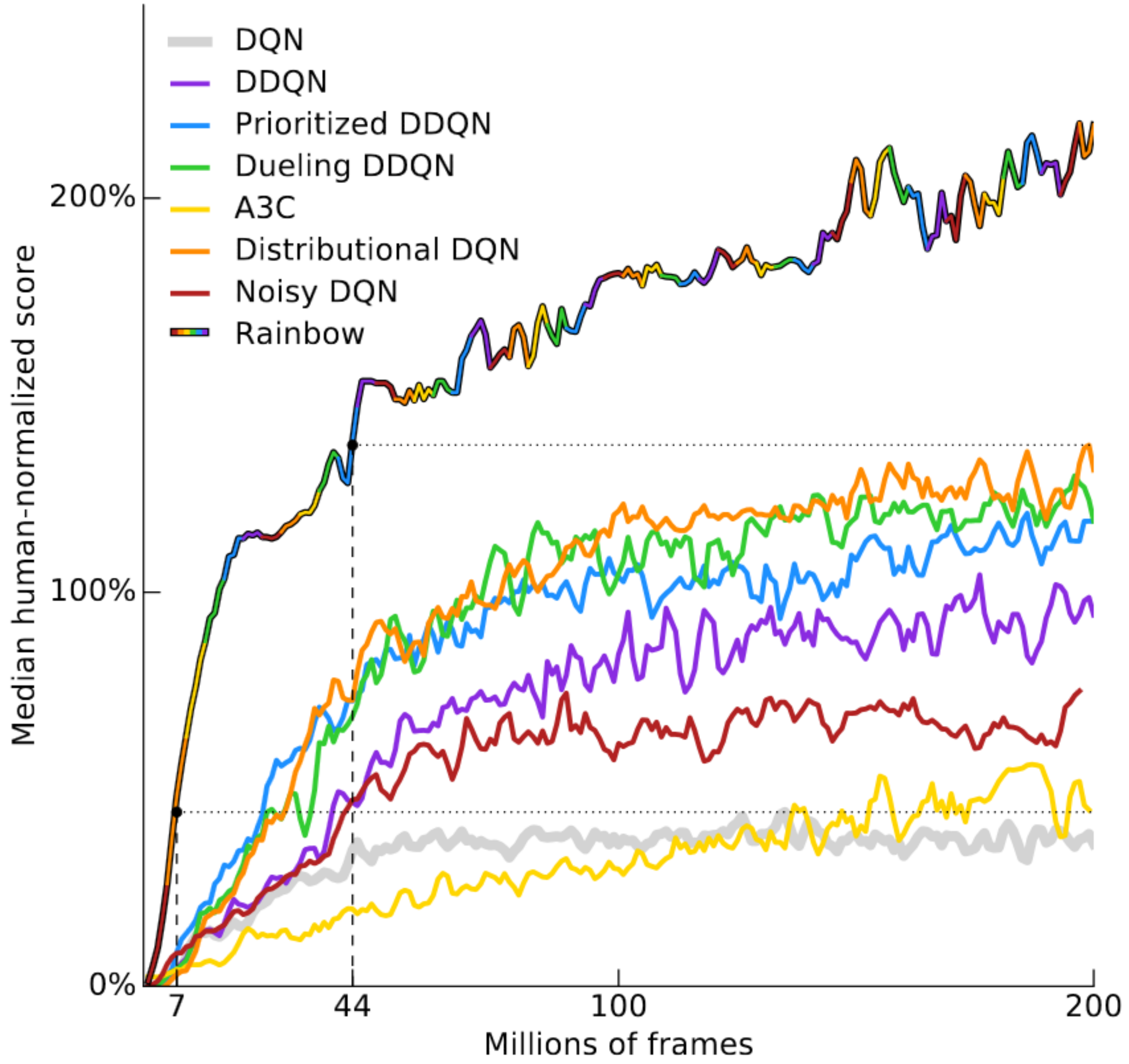


Mnih et al. Nature 2015

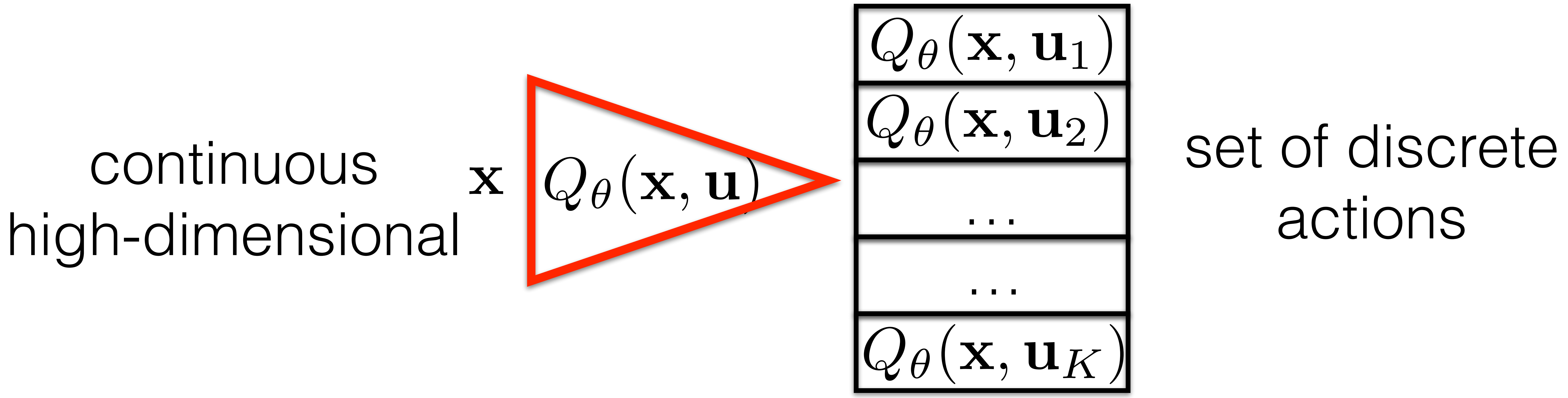


Hessel et. al Rainbow DQN, 2017

Ensemble of DQN (average of different implementations)



Approximate Q-learning (DQN)



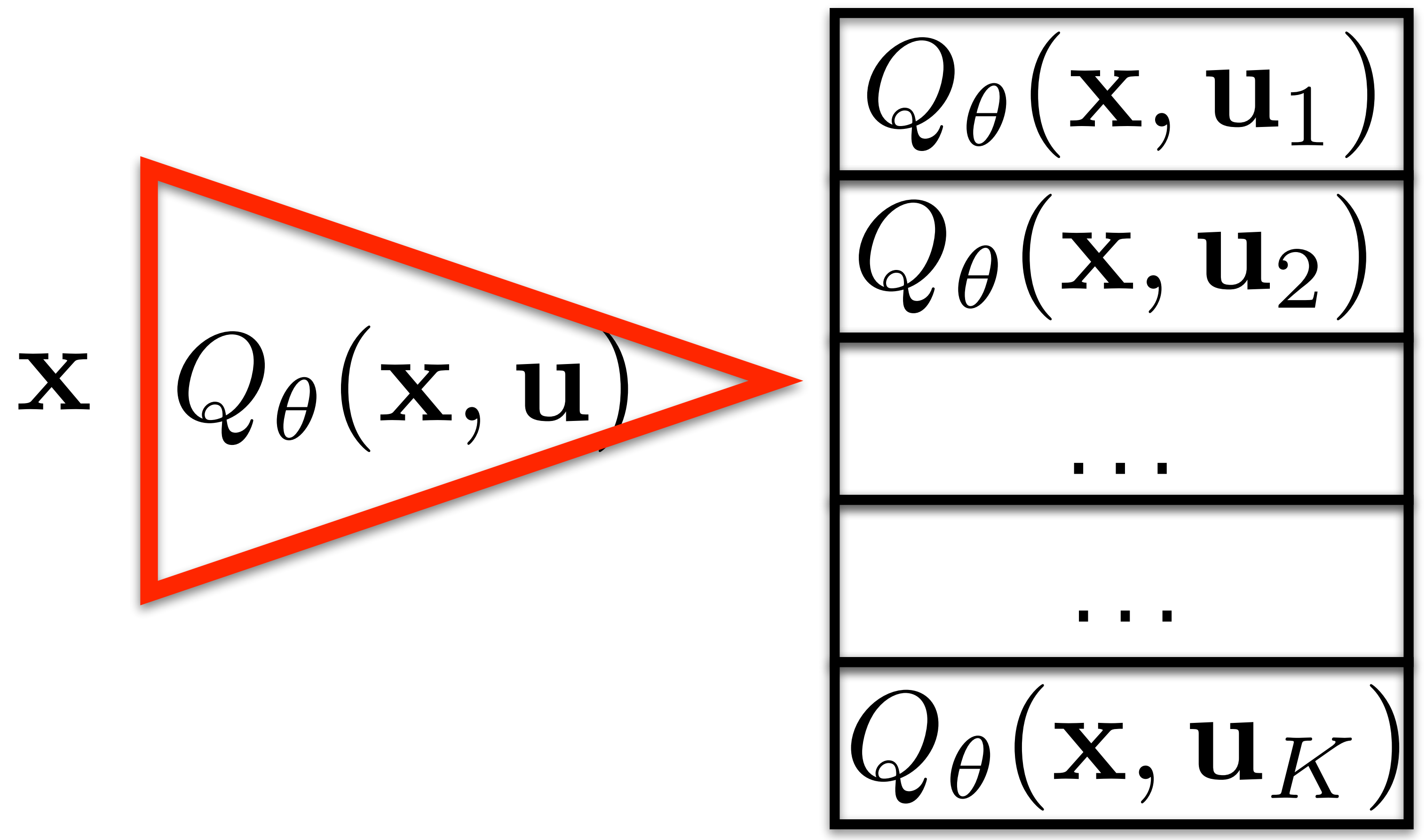
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

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

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

Approximate Q-learning (DQN)

continuous
high-dimensional



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

$\dim(\mathbf{u}) =$ **???**

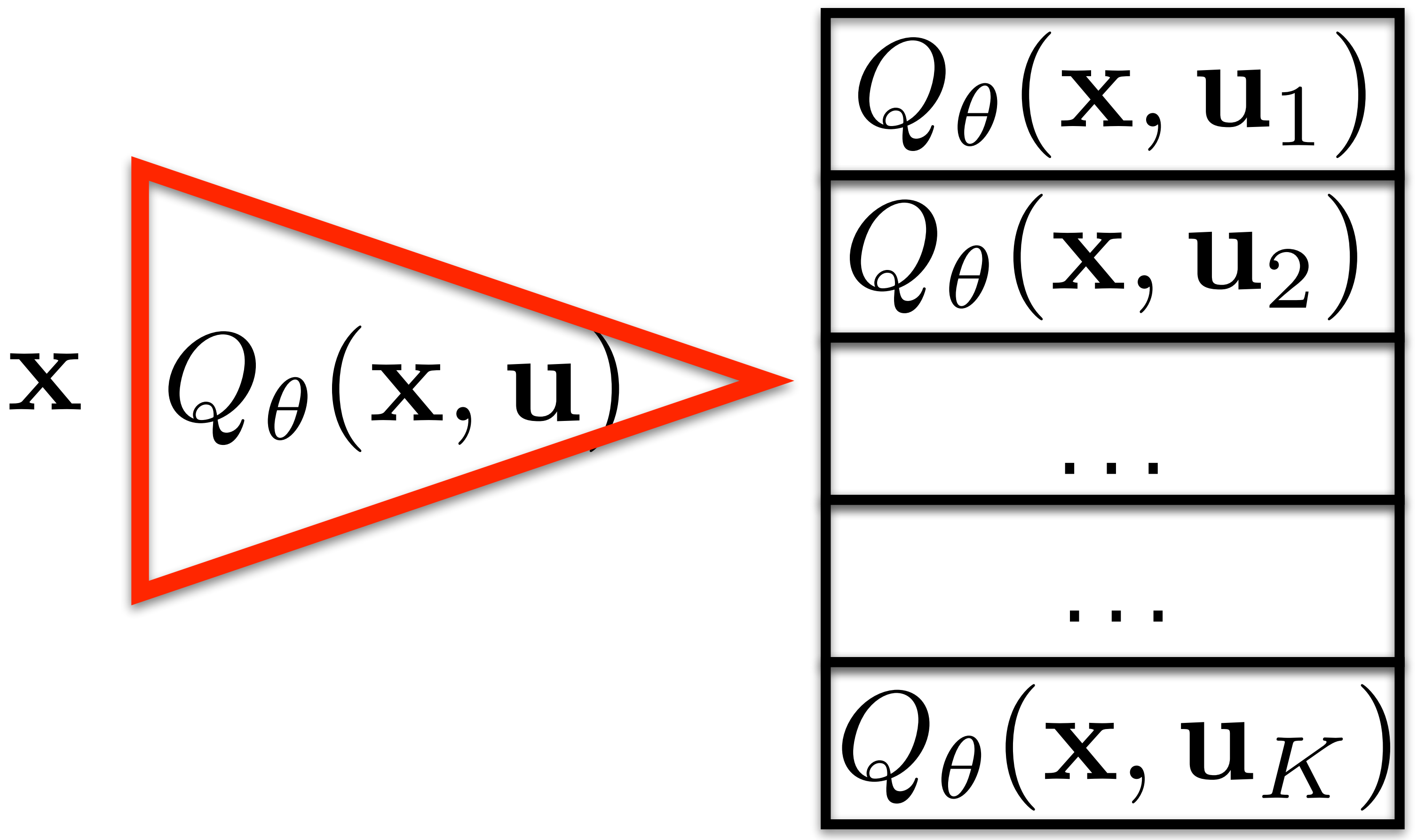
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

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

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

Approximate Q-learning (DQN)

continuous
high-dimensional



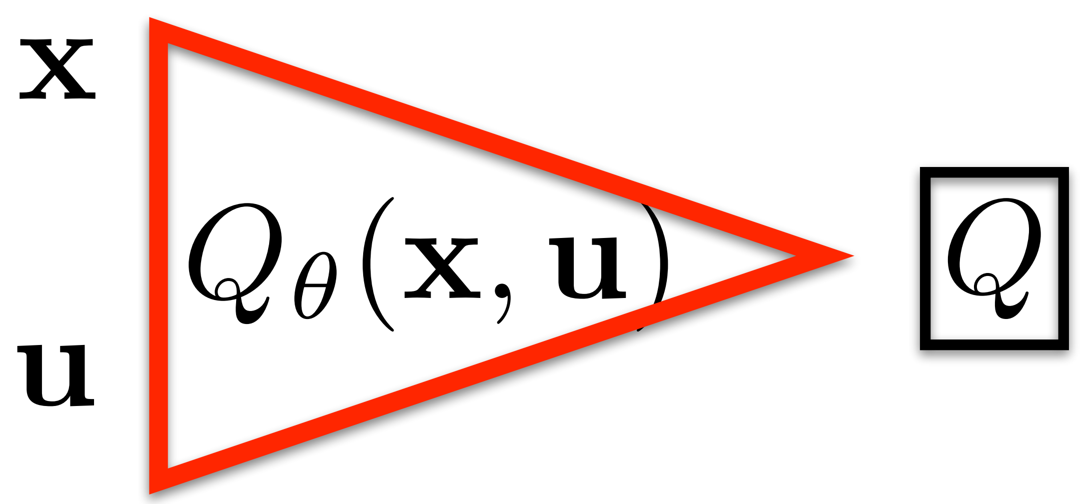
6 legs, 3 motors/leg,
10 input levels/motor

$\dim(\mathbf{u}) =$ **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 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. Update target network $\overline{\theta} := \alpha\theta + (1-\alpha)\overline{\theta}$
6. Repeat from 1

Approximate Q-learning (DQN)

continuous
high-dimensional



scalar Q-value

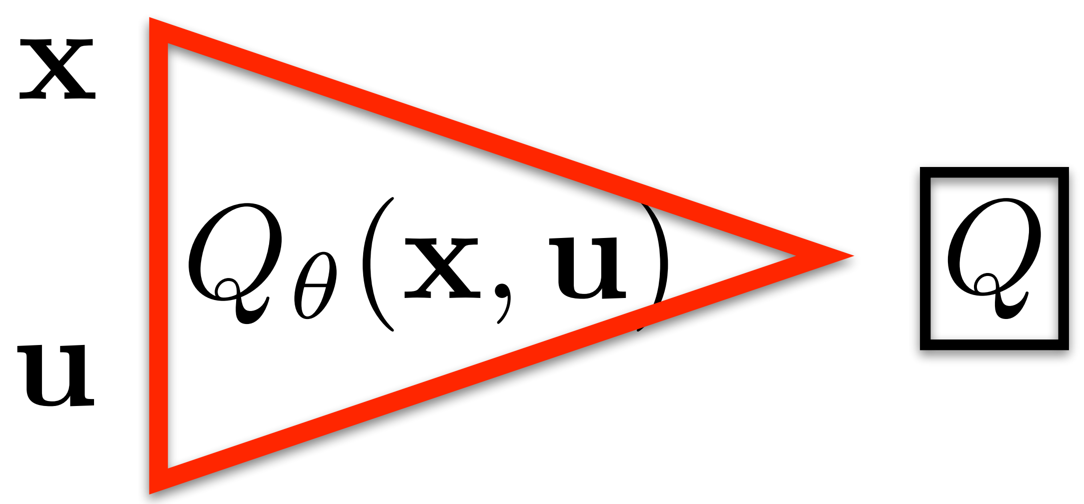
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta}}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

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

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

Approximate Q-learning (DQN)

continuous
high-dimensional



scalar Q-value

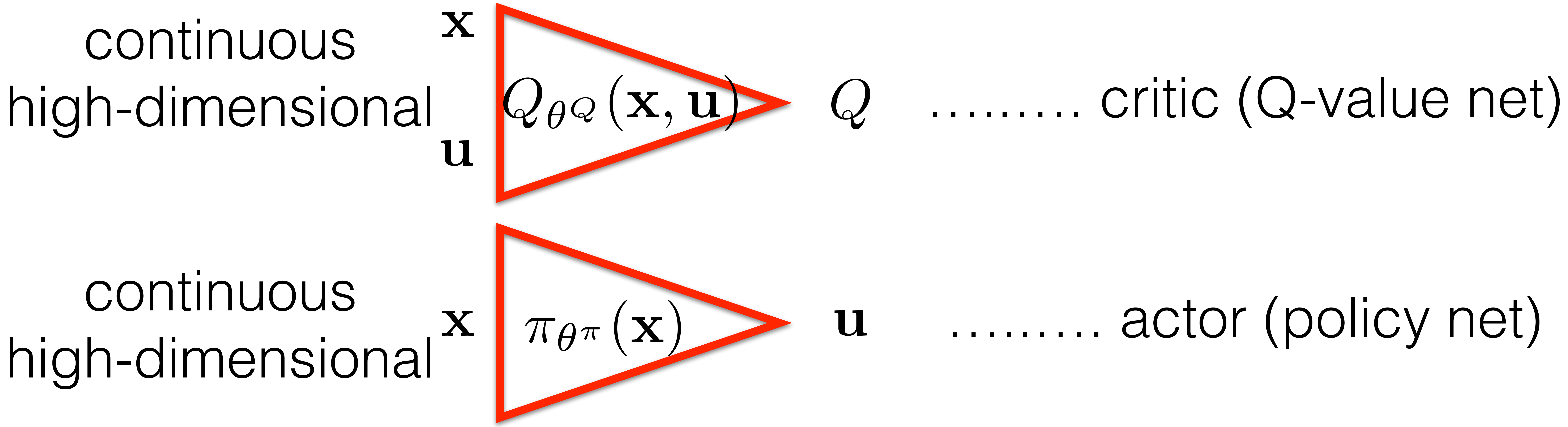
You cannot exhaustively maximize

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

Deep Deterministic Policy Gradient (DDPG)

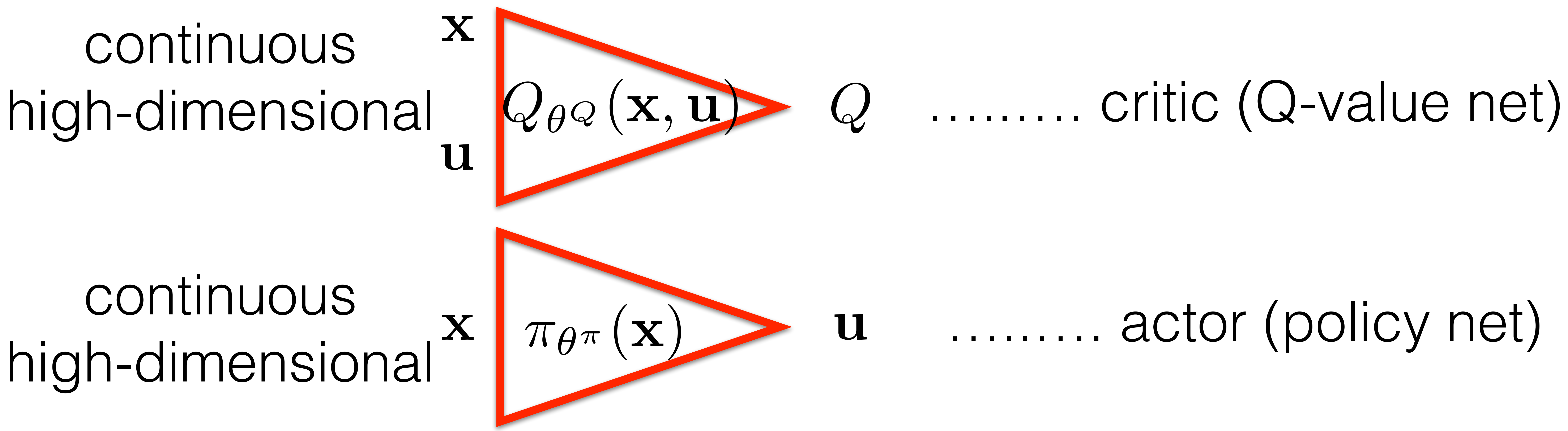


1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta^Q}}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

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

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

Deep Deterministic Policy Gradient (DDPG)



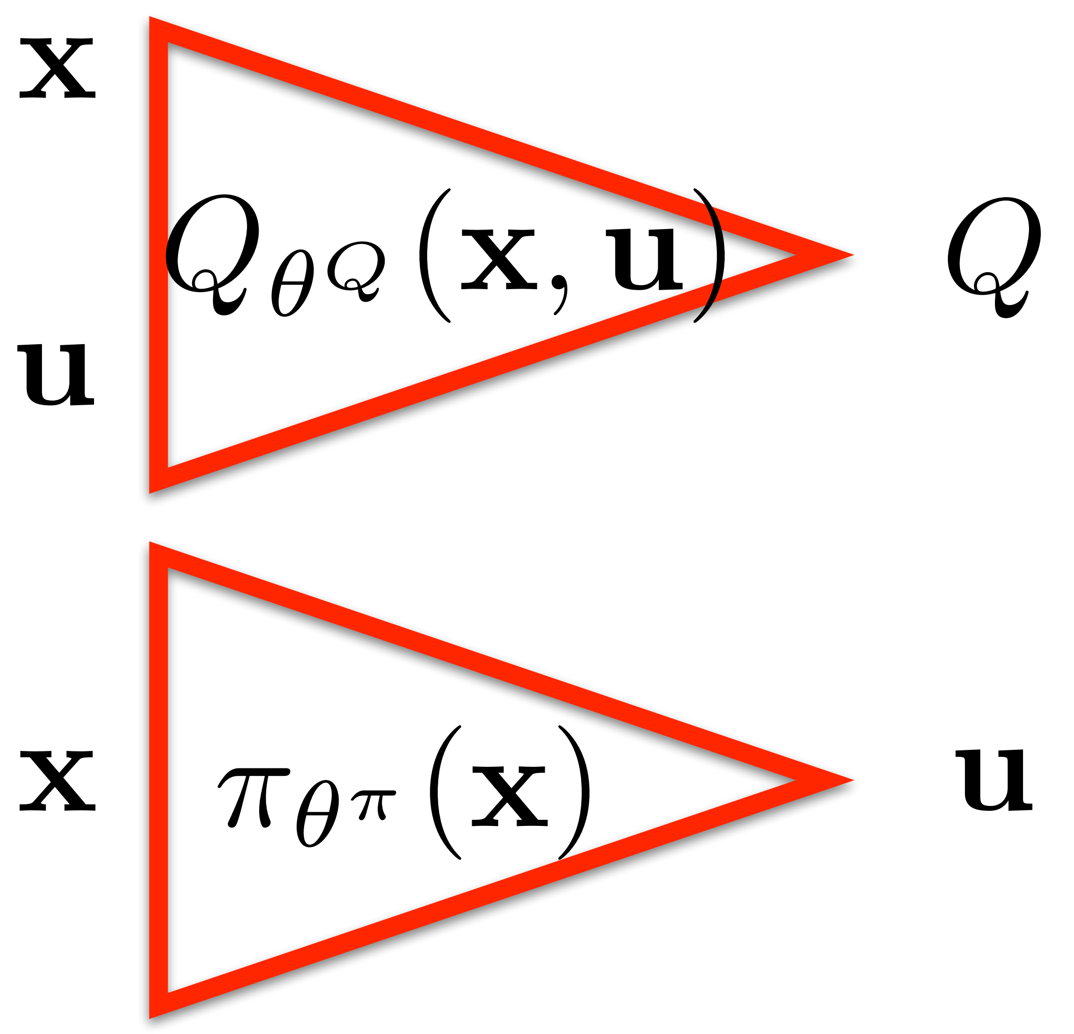
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma \max_{\mathbf{u}'} Q_{\overline{\theta^Q}}(\mathbf{x}', \mathbf{u}')$
4. Update parameters by learning (assumes i.i.d+n.n.)

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

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)

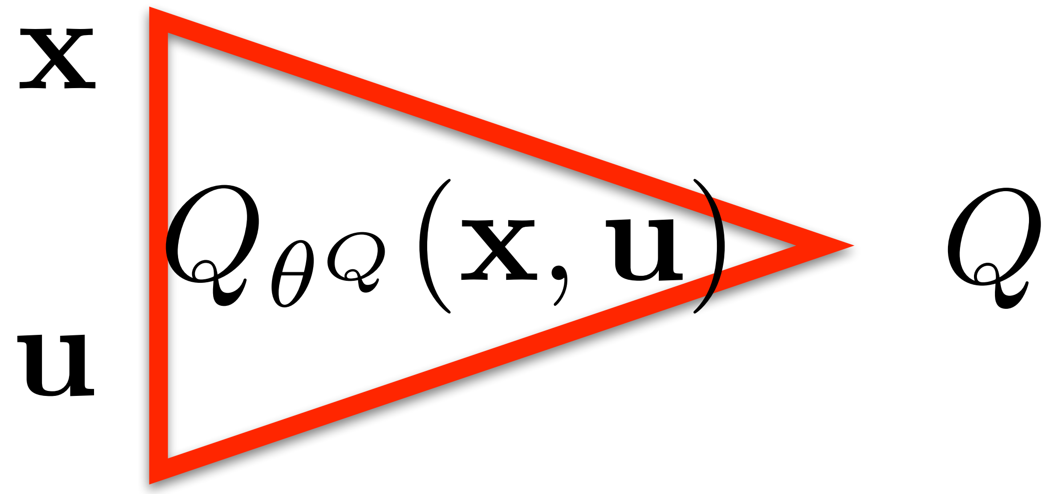
1. Collect transition $[\mathbf{x}, \mathbf{u}, r, \mathbf{x}']$=> ReplayMemory
2. Sample transition(s) at random from ReplayMemory
3. Estimate target(s) $\mathbf{y} = r + \gamma Q_{\overline{\theta^Q}}(\mathbf{x}', \pi_{\overline{\theta^\pi}}(\mathbf{x}'))$
4. Update parameters by learning (assumes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

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

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

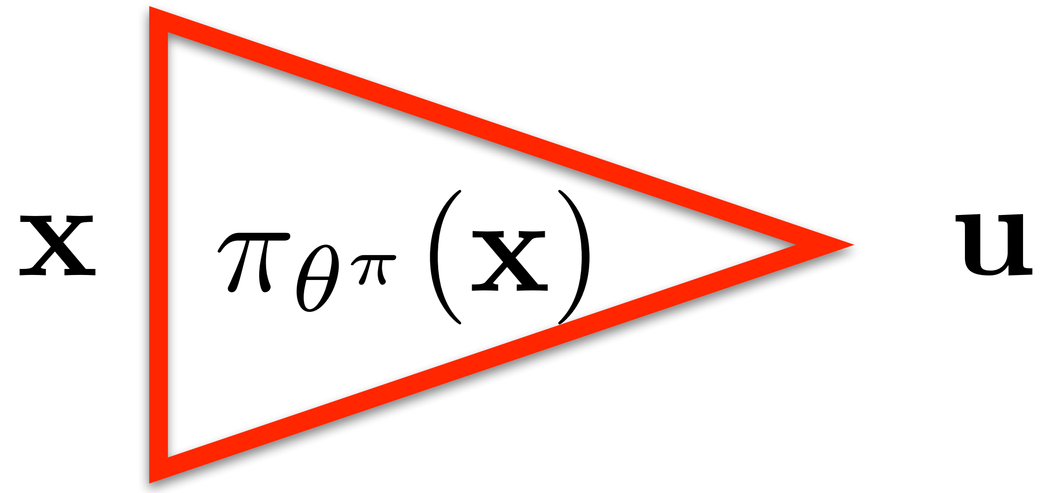
Deep Deterministic Policy Gradient (DDPG)

continuous
high-dimensional



……… critic (Q-value net)

continuous
high-dimensional

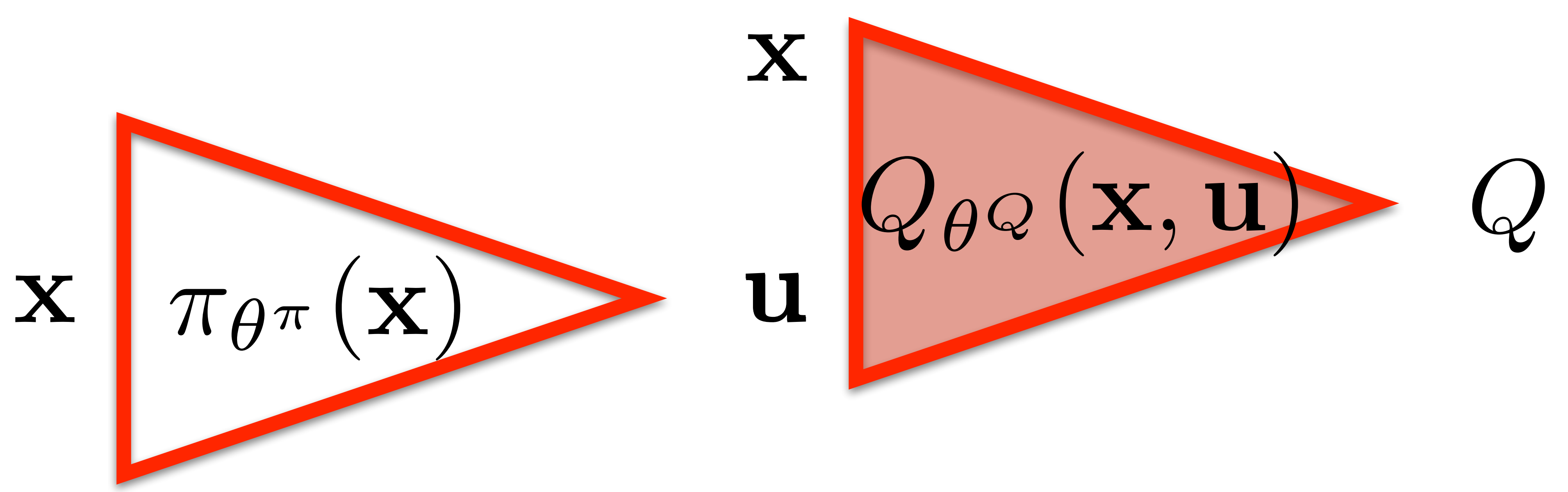


……… actor (policy net)

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 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)

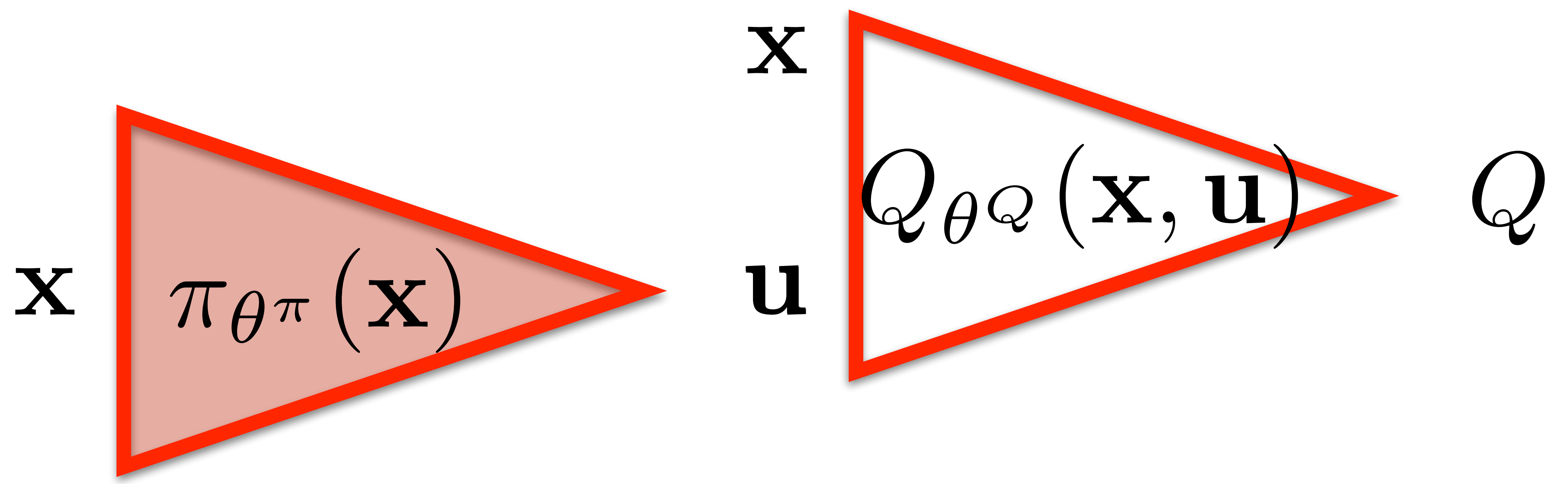
Update critic



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

Deep Deterministic Policy Gradient (DDPG) variants

Taking maximum in target equation often overestimates

$$\mathbf{y} = r + \gamma Q_{\overline{\theta^Q}}(\mathbf{x}', \pi_{\overline{\theta^\pi}}(\mathbf{x}'))$$

- [Fujimoto, 2018] Twin Delayed DDPG (TD3) https://arxiv.org/pdf/1802.09477.pdf
- Learn two Q-functions and take minimum of its outputs

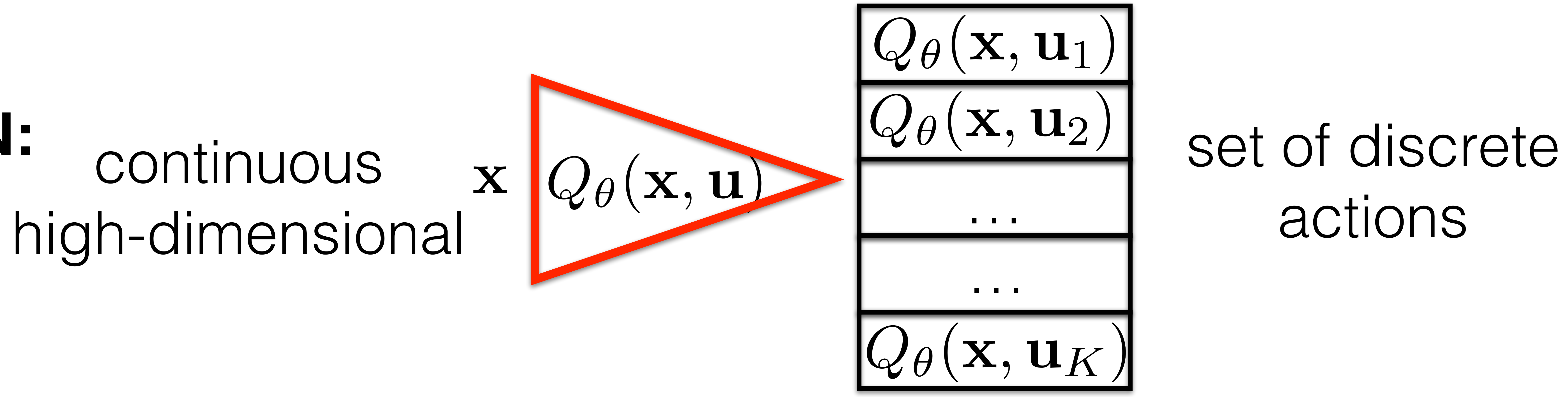
$$\mathbf{y} = r + \gamma \min_i Q_{\overline{\theta^{Q_i}}}(\mathbf{x}', \pi_{\overline{\theta^\pi}}(\mathbf{x}'))$$

- Delayed policy updates (update Q 2x more frequently)
- Add noise to policy actions

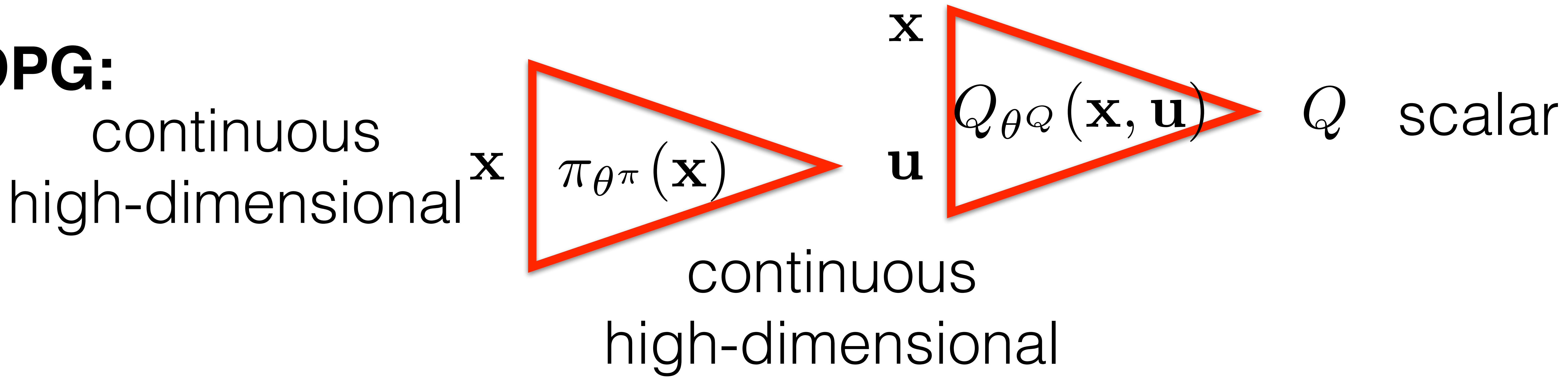
Summary

- 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)

DQN:



DDPG:



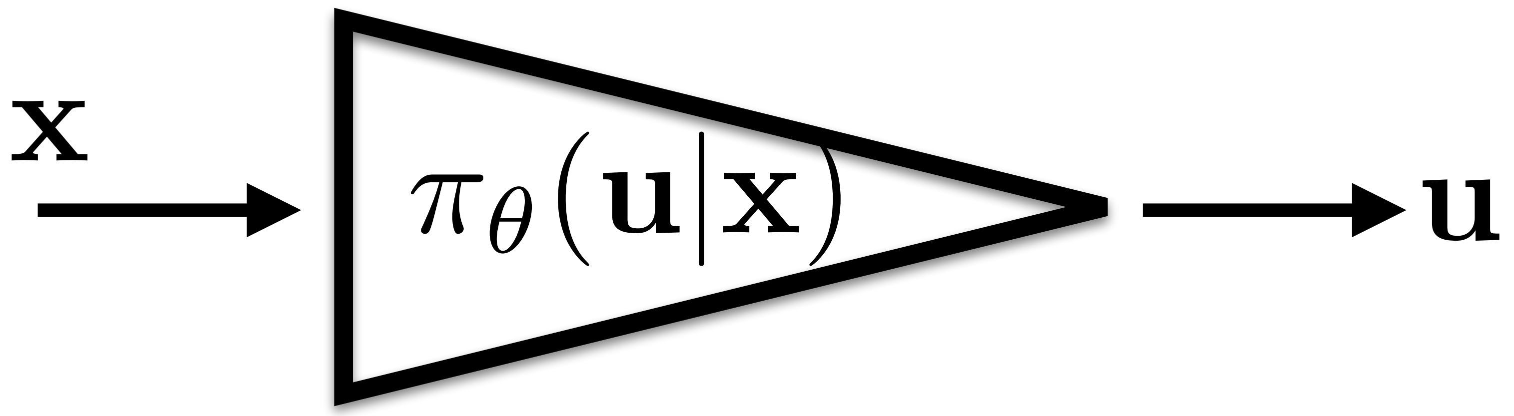
Summary

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

- Next: On-policy methods with stochastic gradient

Deterministic vs stochastic policy

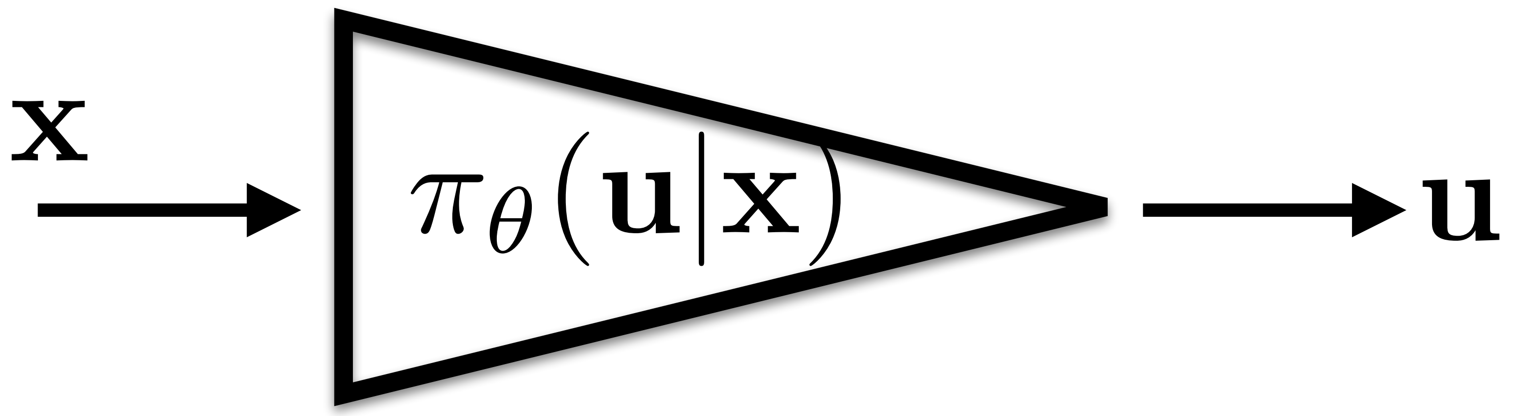
Deterministic policy for continuous control:



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

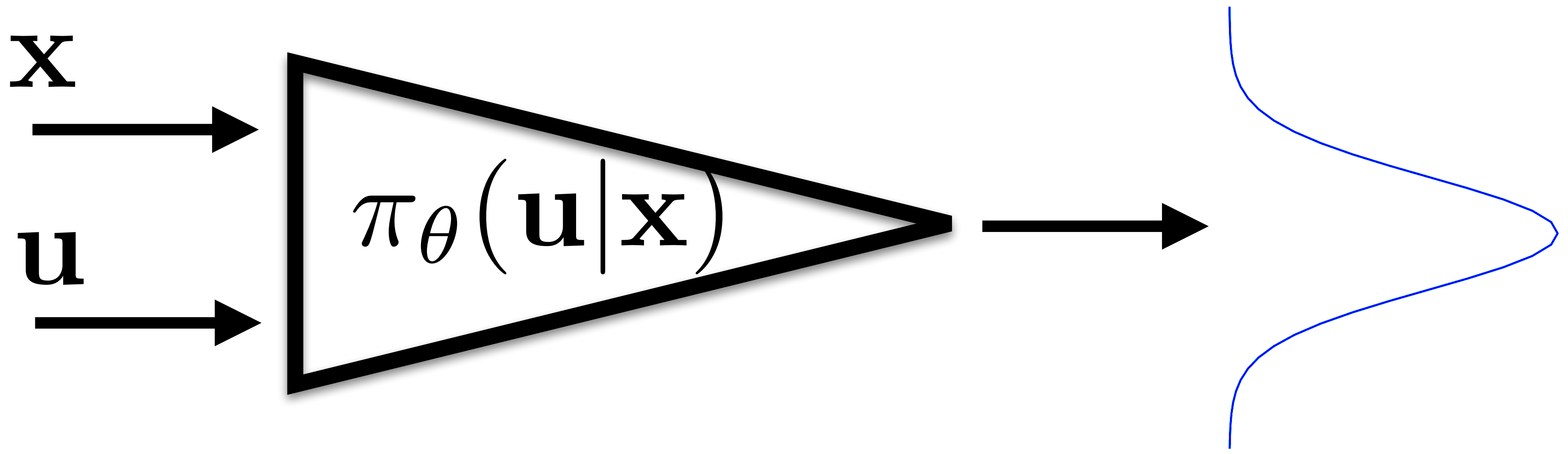
Deterministic vs stochastic policy

Deterministic policy for continuous control:



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

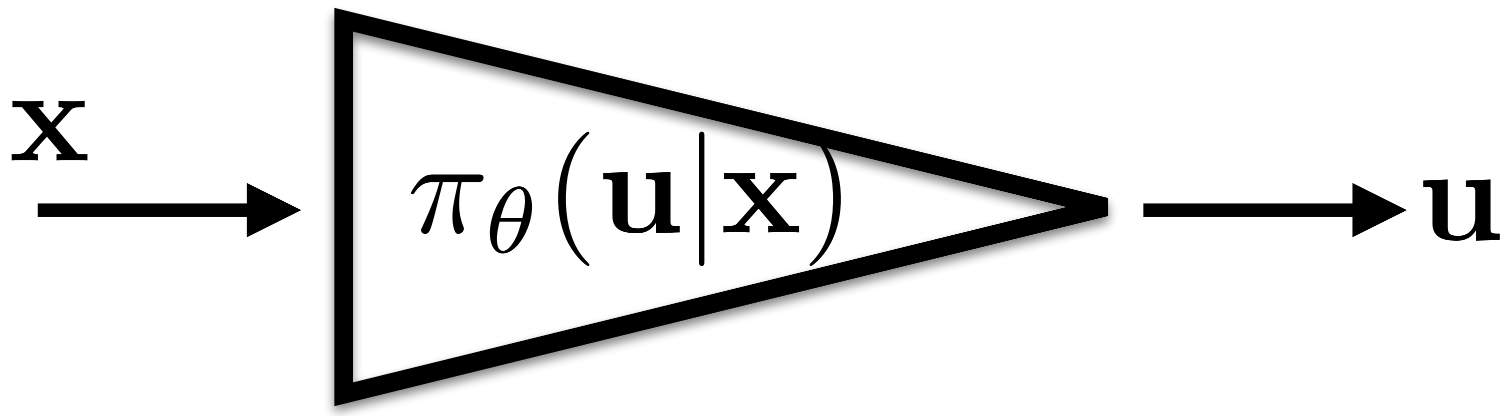
Stochastic policy for continuous control:



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

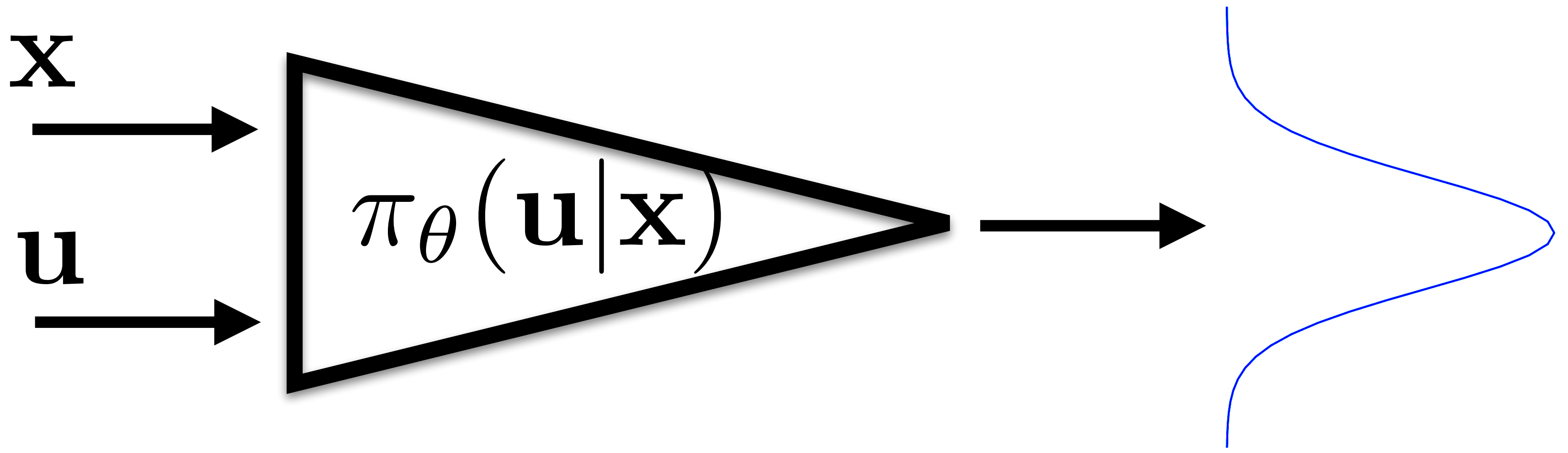
Deterministic vs stochastic policy

Deterministic policy for continuous control:



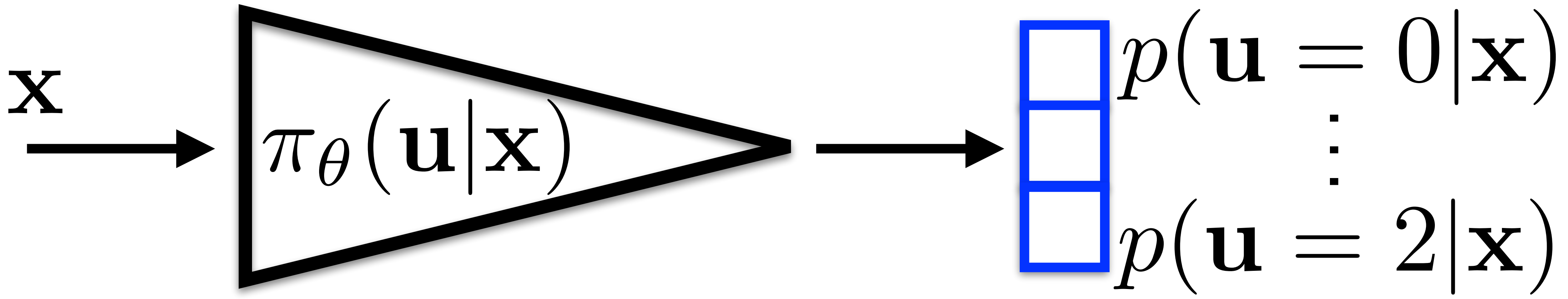
$$\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$

Stochastic policy for continuous control:



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

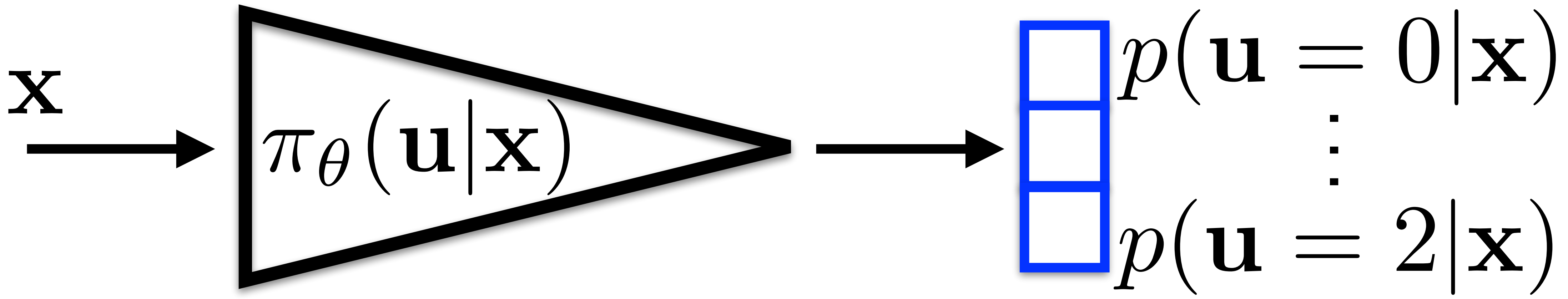
Stochastic policy for discrete control:



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

REINFORCE

Stochastic policy for discrete control:

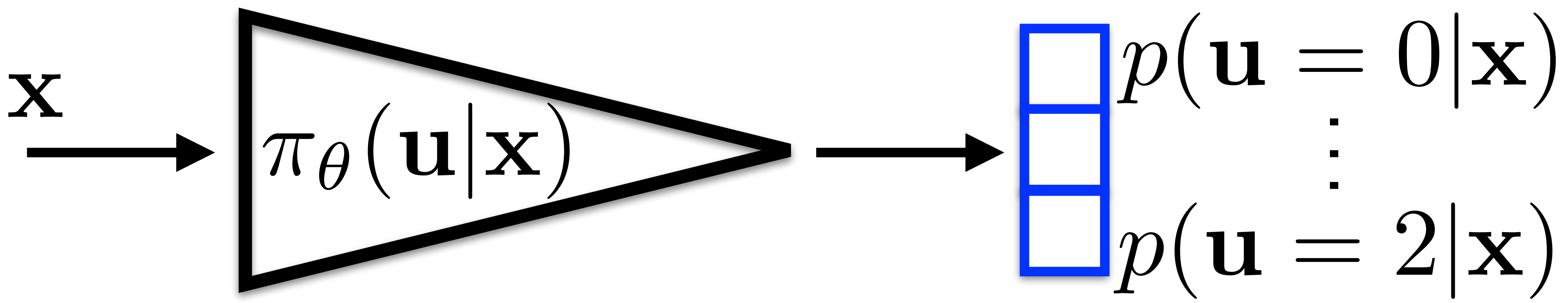


$$\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$

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
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:

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

What is the gradient???

- REINFORCE theorem:

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

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.

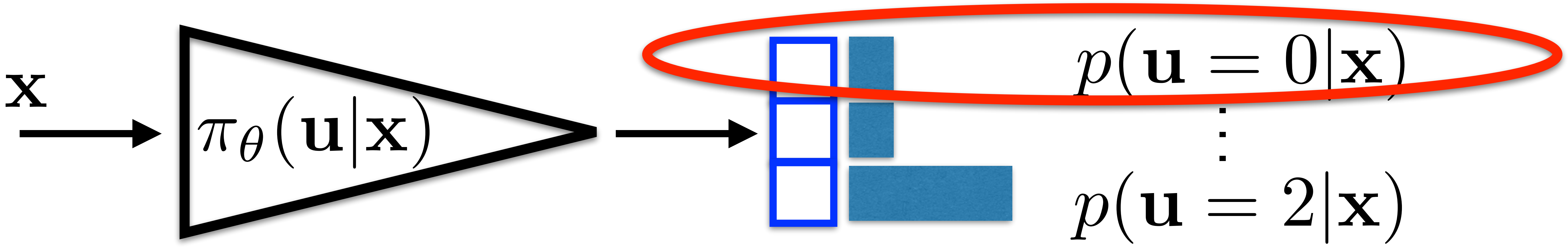
What is the gradient???

- REINFORCE theorem:

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

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 actions, that have yielded high sum of rewards.

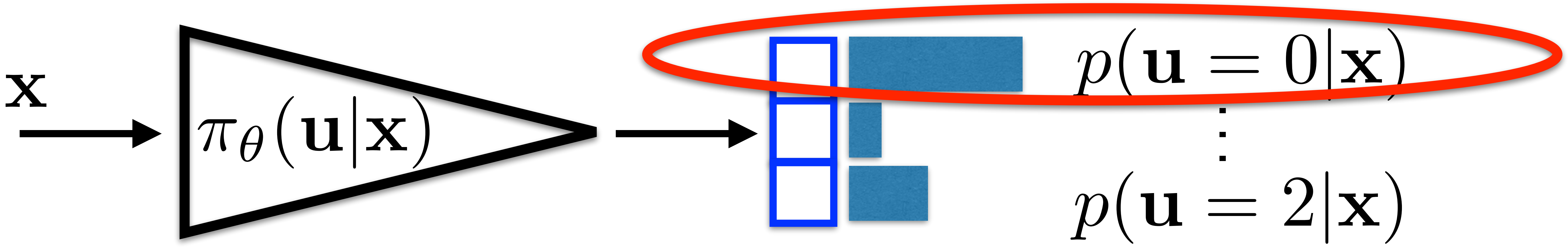
What is the gradient???

- REINFORCE theorem:

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

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 actions, that have yielded high sum of rewards.

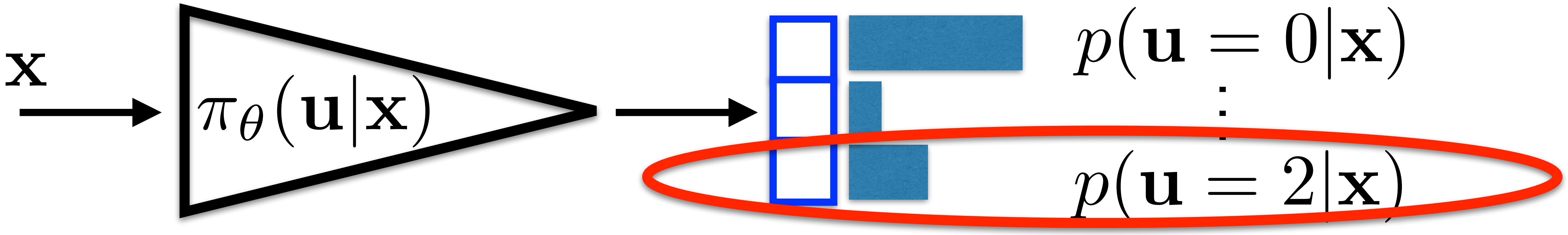
What is the gradient???

- REINFORCE theorem:

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

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 actions, that have yielded high sum of rewards.

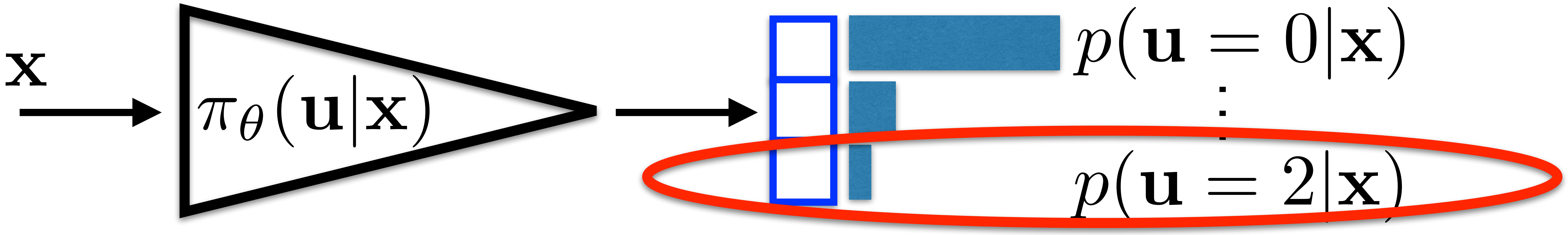
What is the gradient???

- REINFORCE theorem:

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

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.

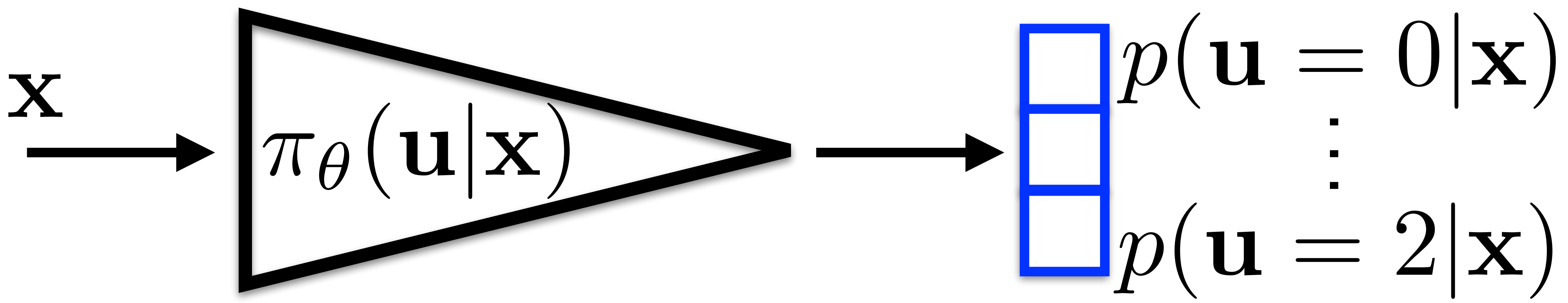


$r(\tau)$ =-100

Learning means increasing probability of predicting 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):

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

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

5. Repeat from 2

trajectory:

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

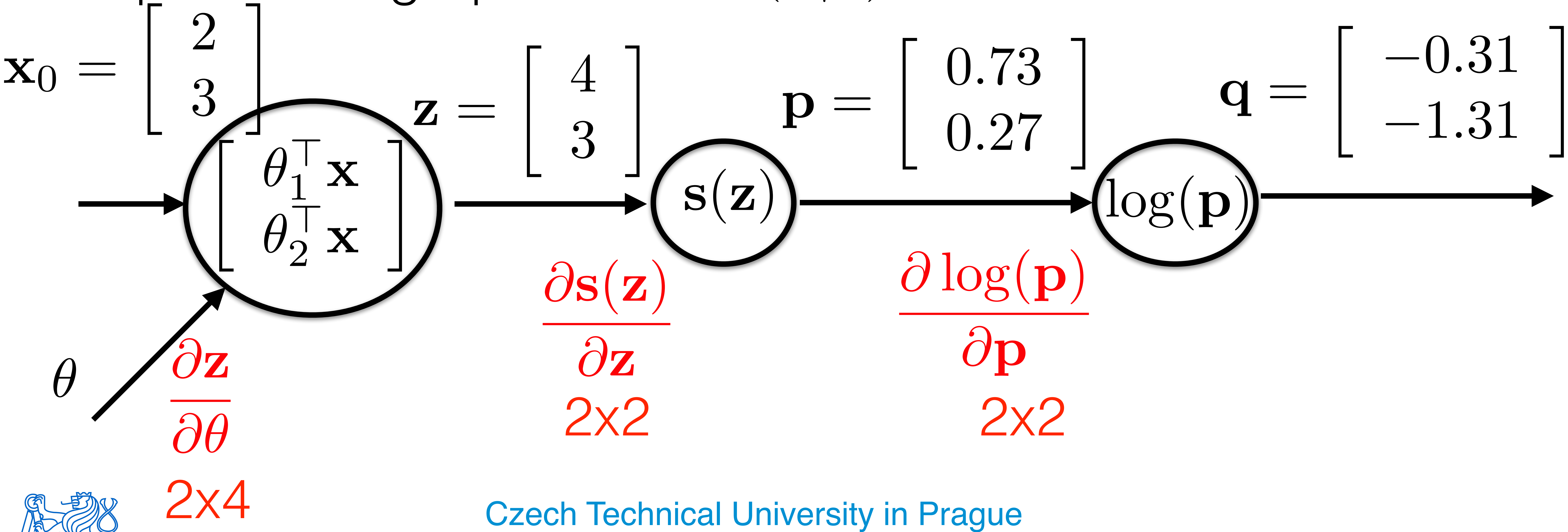
policy: $\pi_\theta(\mathbf{u}|\mathbf{x}) = \mathbf{s}\left(\begin{bmatrix} \theta_1^\top \mathbf{x} \\ \theta_2^\top \mathbf{x} \end{bmatrix}\right)$

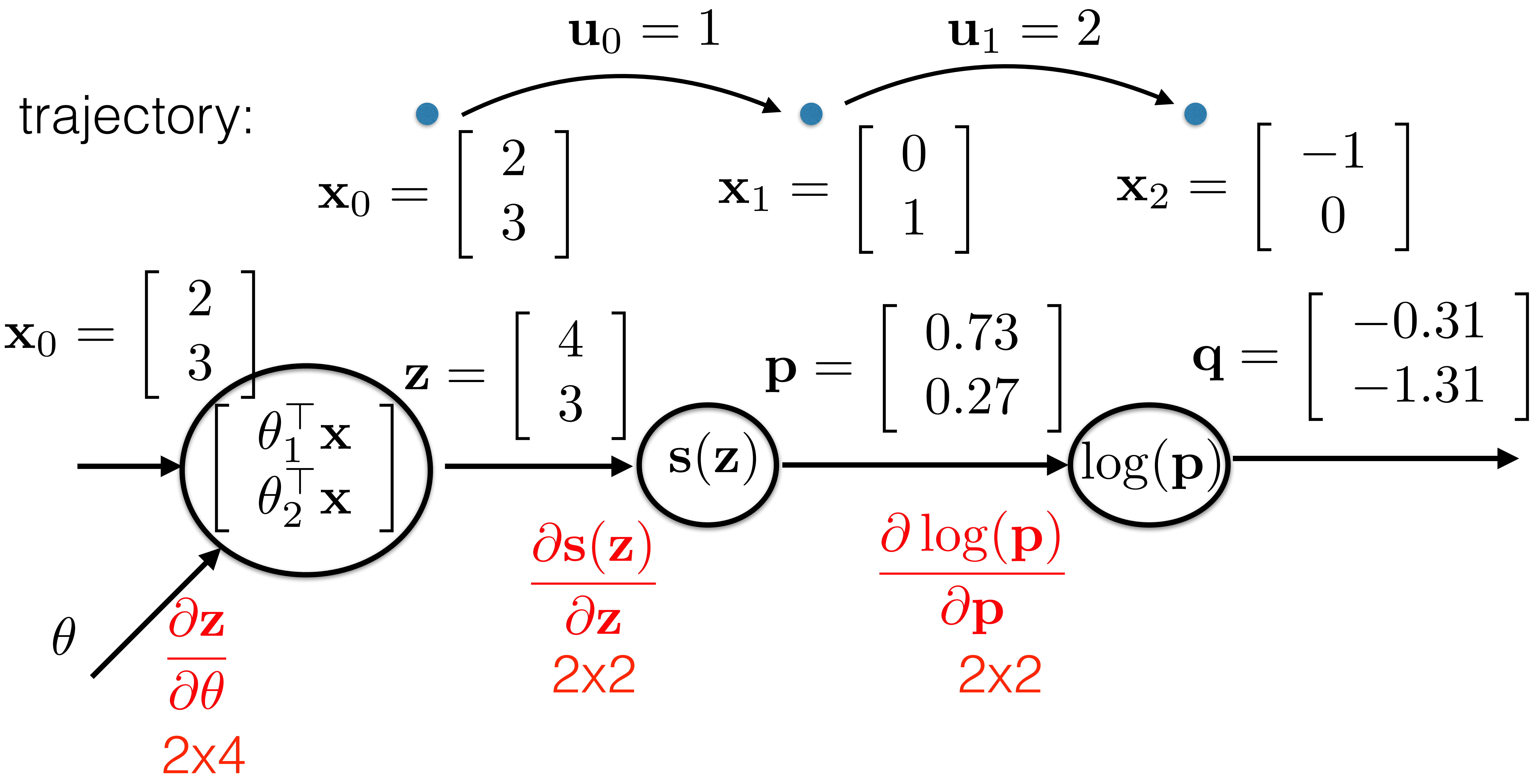
parameters: $\theta_1^\top = [2,\ 0]$, $\theta_2^\top = [0,\ 1]$

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

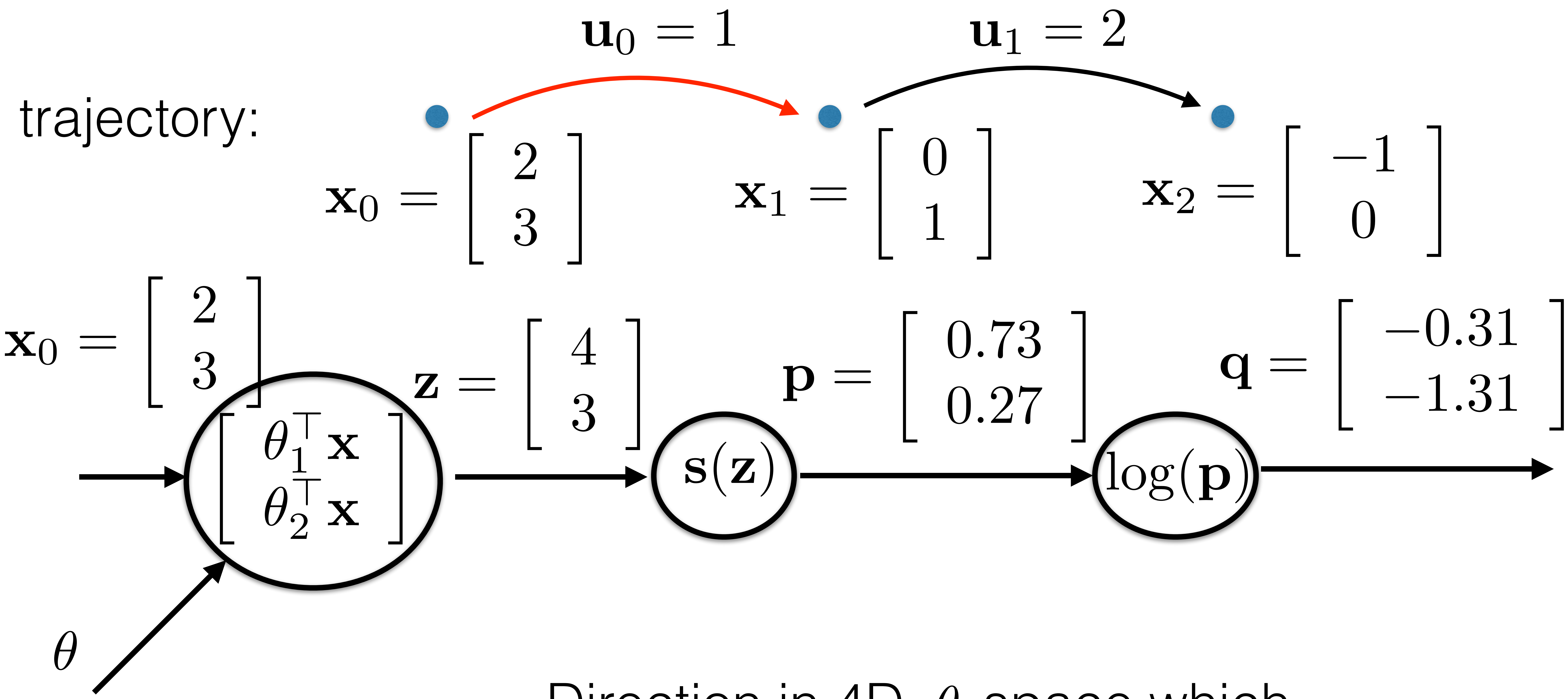
policy: $\pi_\theta(\mathbf{u}|\mathbf{x}) = \mathbf{s}\left(\begin{bmatrix} \theta_1^\top \mathbf{x} \\ \theta_2^\top \mathbf{x} \end{bmatrix}\right)$ parameters: $\theta_1^\top = [2,\ 0]$, $\theta_2^\top = [0,\ 1]$

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





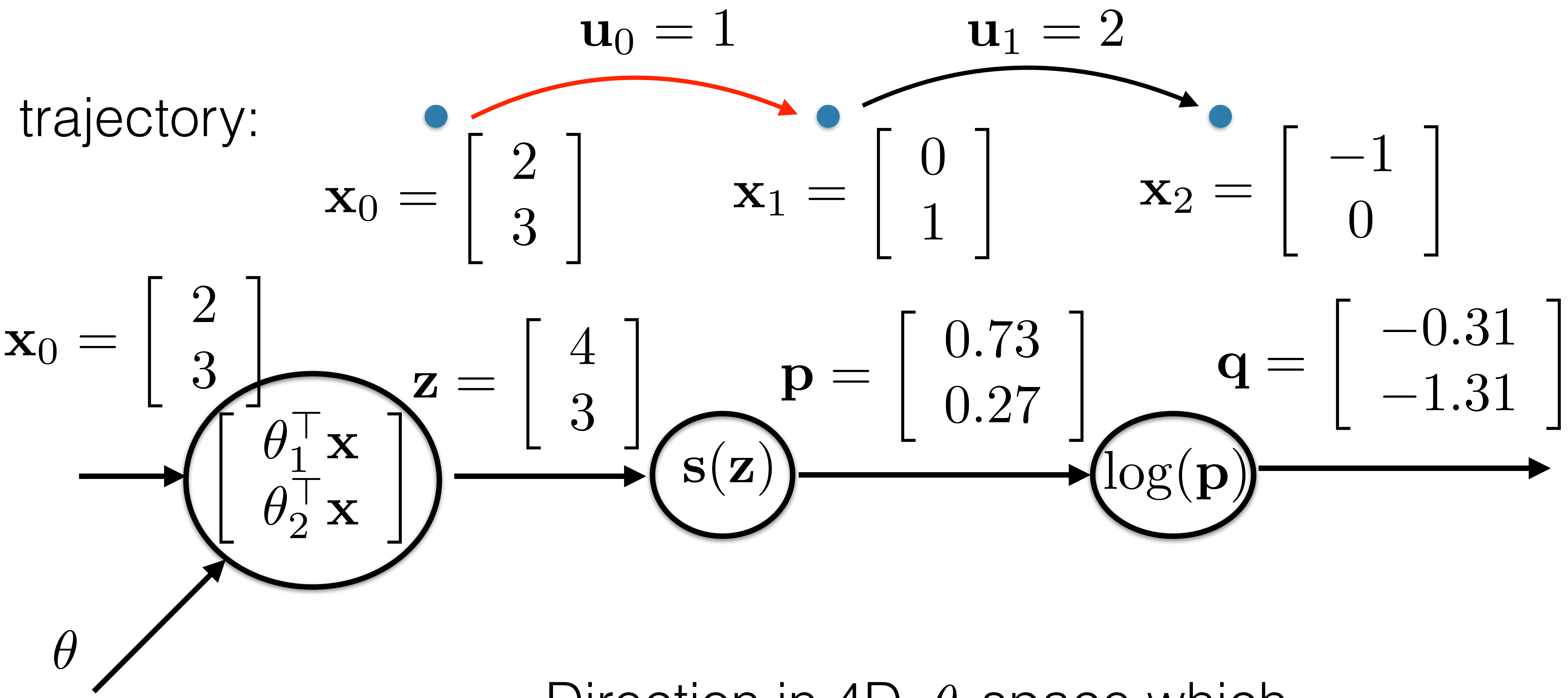
$$\frac{\partial \log \pi_\theta(\mathbf{u}|\mathbf{x})}{\partial \theta} = \textbf{???}$$



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

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

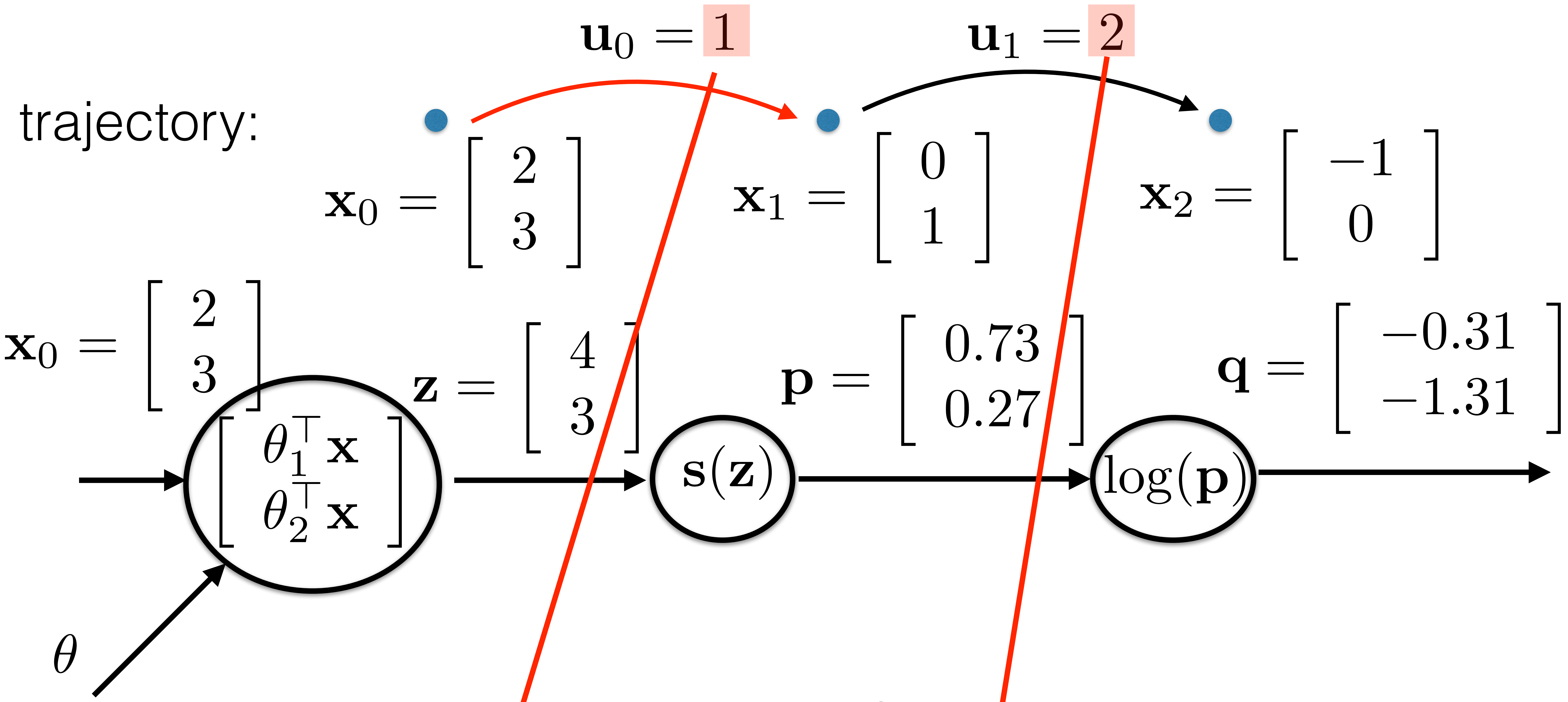
2x2 2x2 2x4 2x4



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

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

2x2 2x2 2x4 2x4



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

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

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

trajectory: $\mathbf{u}_0 = 1$, $\mathbf{u}_1 = 2$

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

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

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

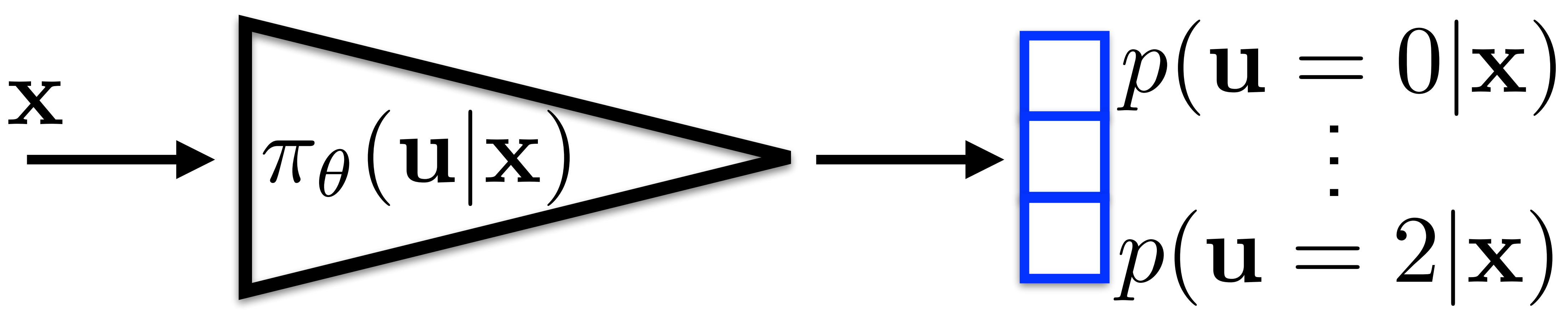
$$= \boxed{\mathbf{g}_1^\top(\mathbf{x}_0)} \cdot r(\tau) + \boxed{\mathbf{g}_2^\top(\mathbf{x}_1)} \cdot r(\tau) + \dots$$

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 is more preferred.

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

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):

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

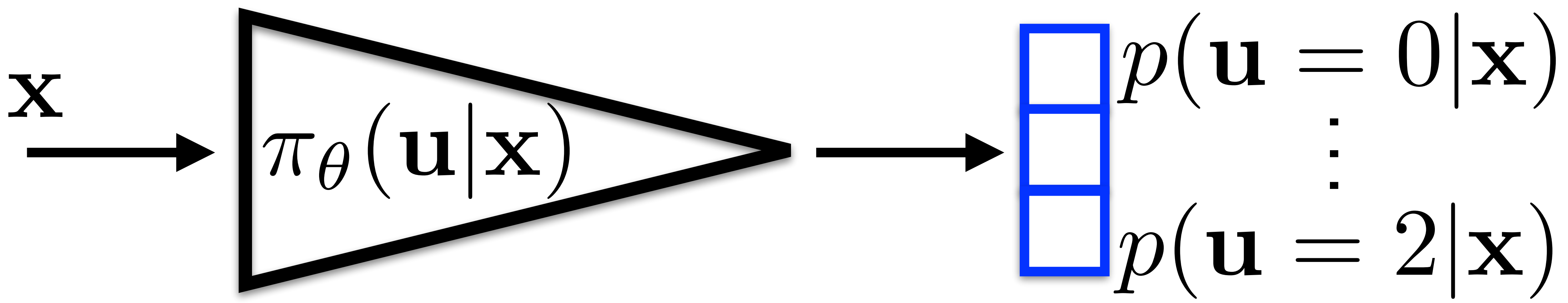
$$\theta := \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$

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

4. Update policy (actor):

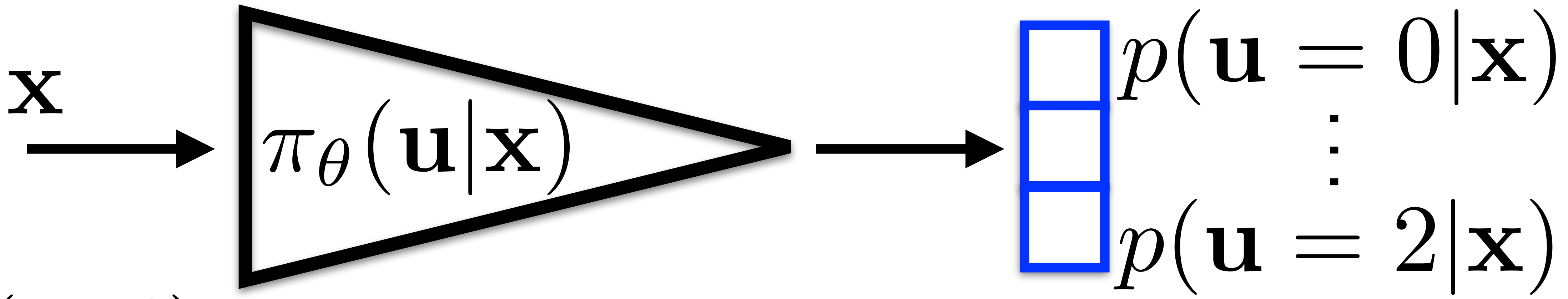
$$\frac{\partial J(\theta)}{\partial \theta} \approx \sum_{(\mathbf{u},\mathbf{x},\mathbf{x}')\in\tau} \frac{\partial \log \pi_\theta(\mathbf{u}|\mathbf{x})}{\partial \theta} \cdot \underbrace{\Big(r + \gamma V(\mathbf{x}') - V(\mathbf{x})\Big)}_{A=Q-V}$$

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

5. Repeat from 2

Advantage Actor Critic (A2C)

Stochastic policy for discrete control:



1. Initialize policy $\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$
2. Collect trajectories τ with policy $\pi_\theta(\mathbf{u}|\mathbf{x})$
3. Update value function (critic): $V_\omega(\mathbf{x}) \leftarrow r + \gamma V_\omega(\mathbf{x}')$

$$\omega := \omega - \alpha \frac{\partial \Big(r + \gamma V_\omega(\mathbf{x}') - V_\omega(\mathbf{x}) \Big)^2}{\partial \omega}$$

4. Update policy (actor):

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

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

5. Repeat from 2

Advantage Actor Critic (A2C)

Paper: https://arxiv.org/abs/1602.01783

Implementation: https://stable-baselines.readthedocs.io/en/master/modules/a2c.html

Explanation: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Known successes of RL

- Computer games controlled from pixel inputs
  - Atari 2D platformers
  - 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/
  - Starcraft II
  - AlphaGo
  - AlphaZero

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)

- Constrained Activities per Minute (APM) - Alpha Star uses significantly less APM than human players.
- Response time 350ms (approx moderate human player)
- AlphaStar does not move camera (uses zoomed out), however haze of war is used.

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

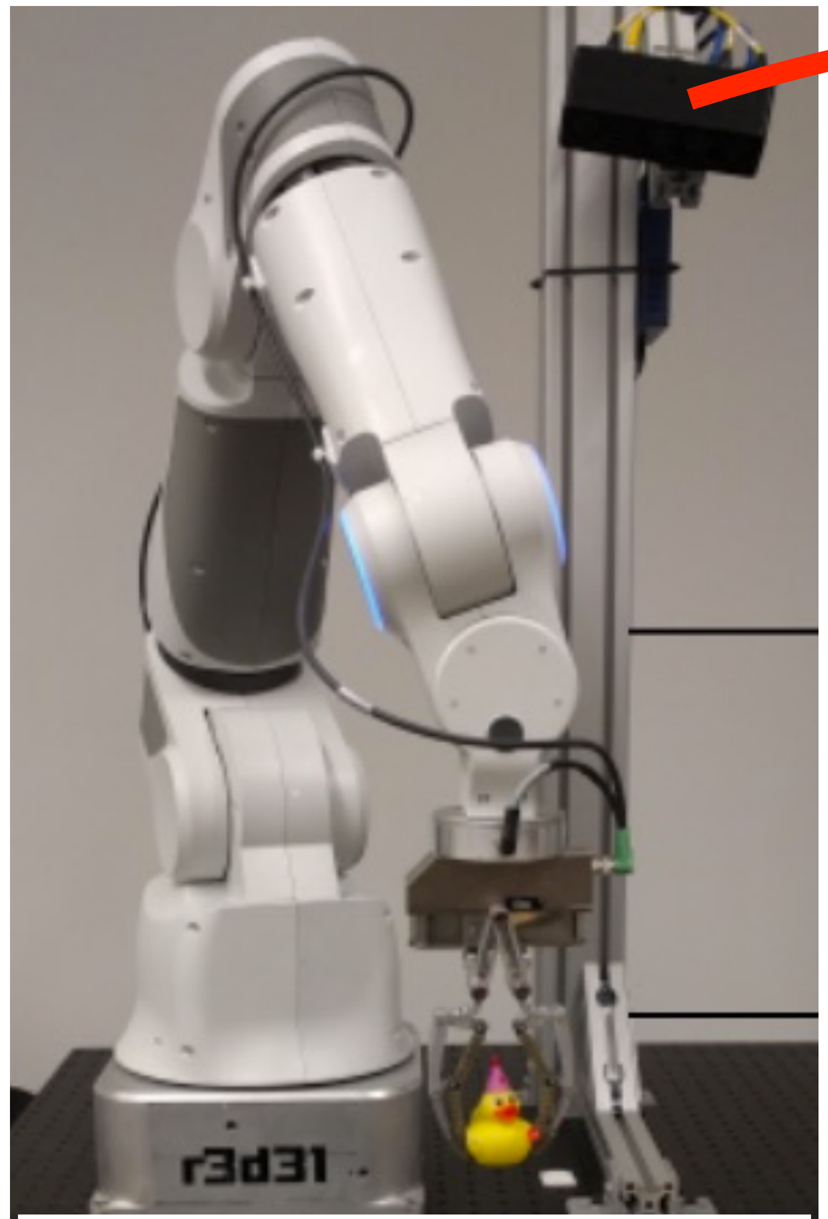
Known successes of RL

- AlphaGo/Alpha Zero https://en.wikipedia.org/wiki/AlphaZero
- SearchTrees has no chance in huge state-action spaces
  - AlphaGo:
    - beat professional Go player
    - 9 dan professional ranking
  - Alpha Zero: Top Chess Engine Championship 2017
    - 9h of self-play, no openingbooks nor endgames tables
    - 1 minute per move, 1GB RAM
    - 28 wins, 72 withdraws
- AutoML https://cloud.google.com/automl/
  - [Zoph 2016] REINFORCE learns RCNN policy which generates deep CNN architectures.

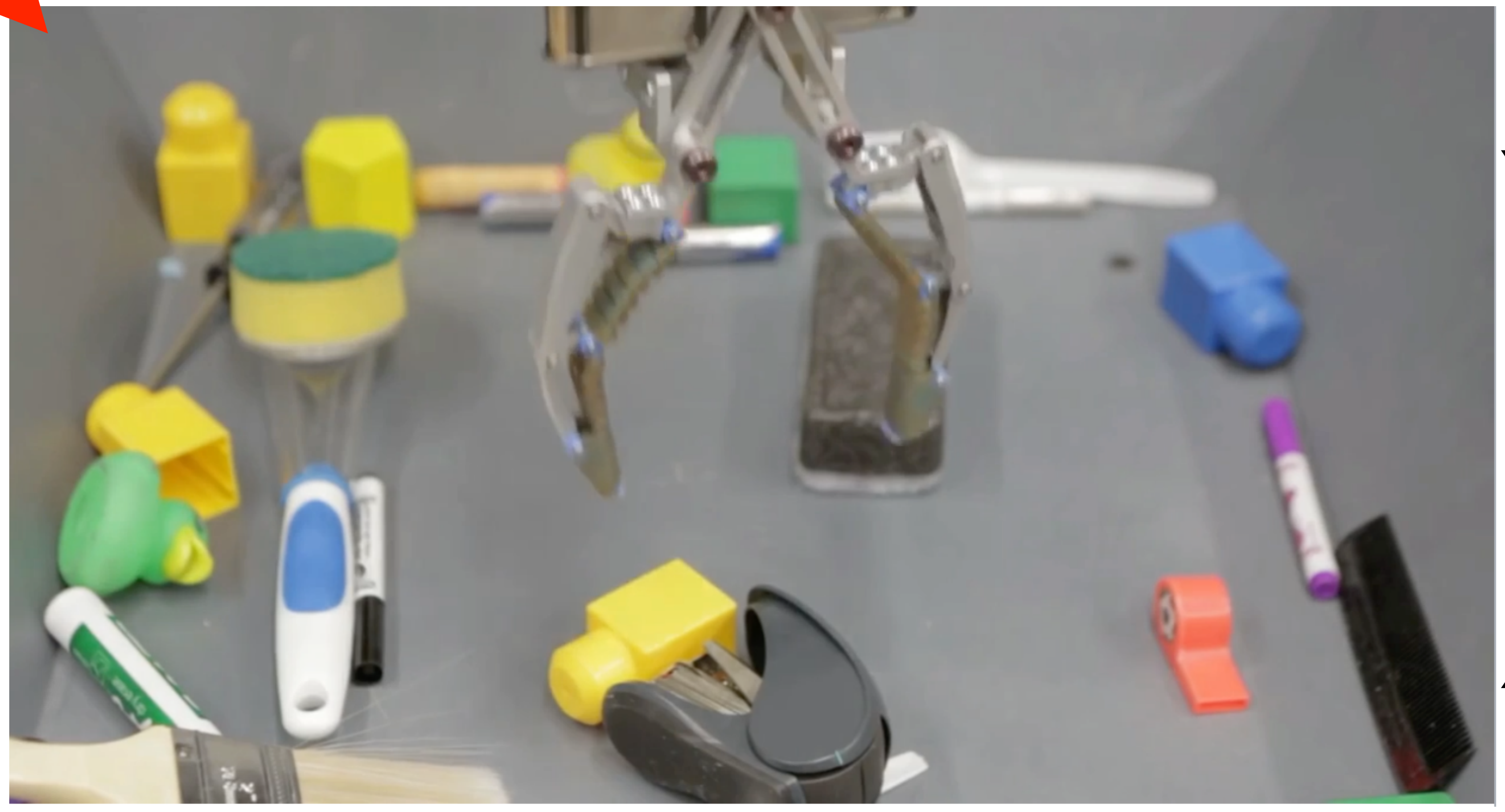
Known successes of RL - locomotion in simulation
[Heess 2017] https://arxiv.org/abs/1707.02286

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

manipulator+ RGB camera

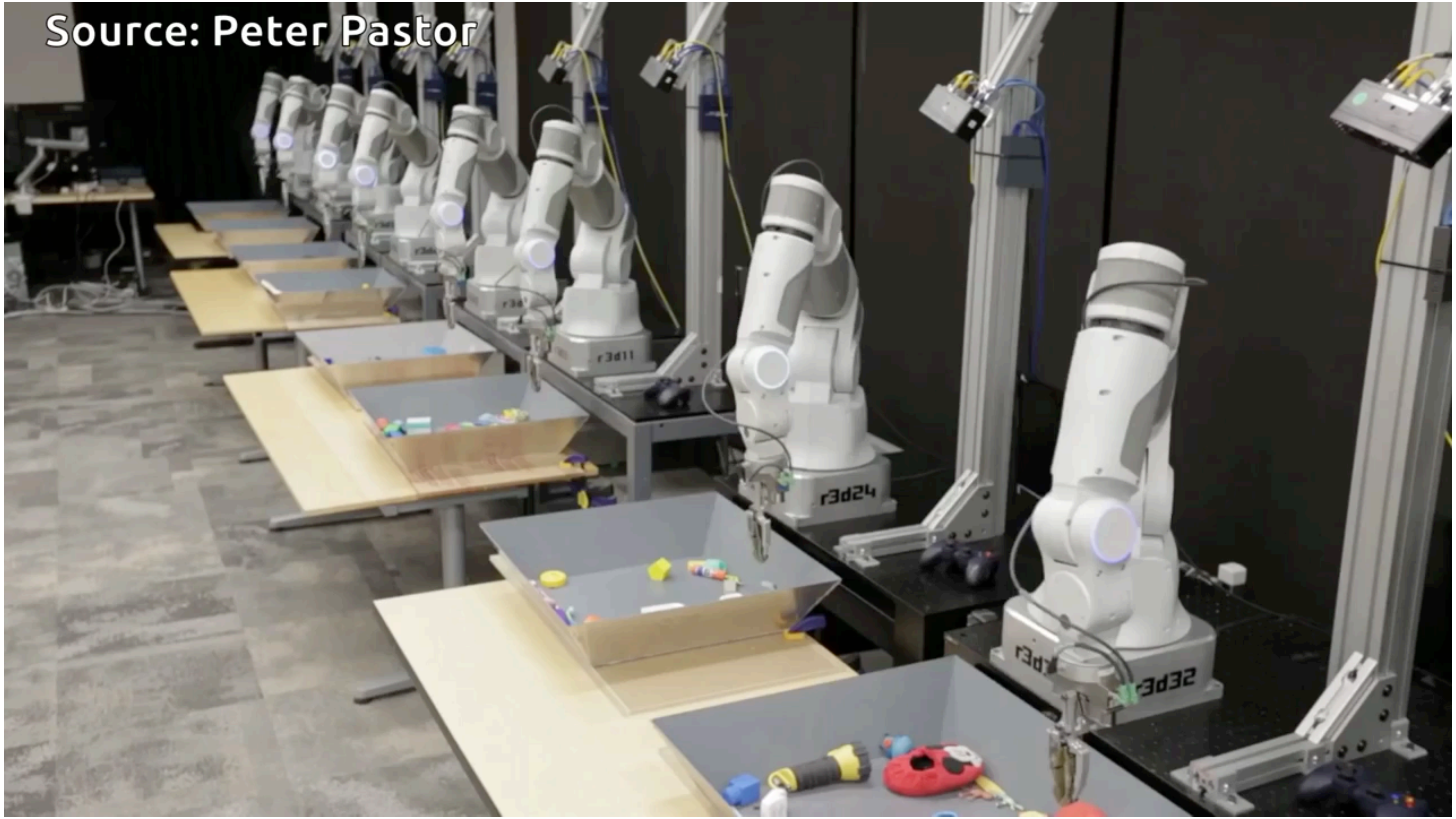


image

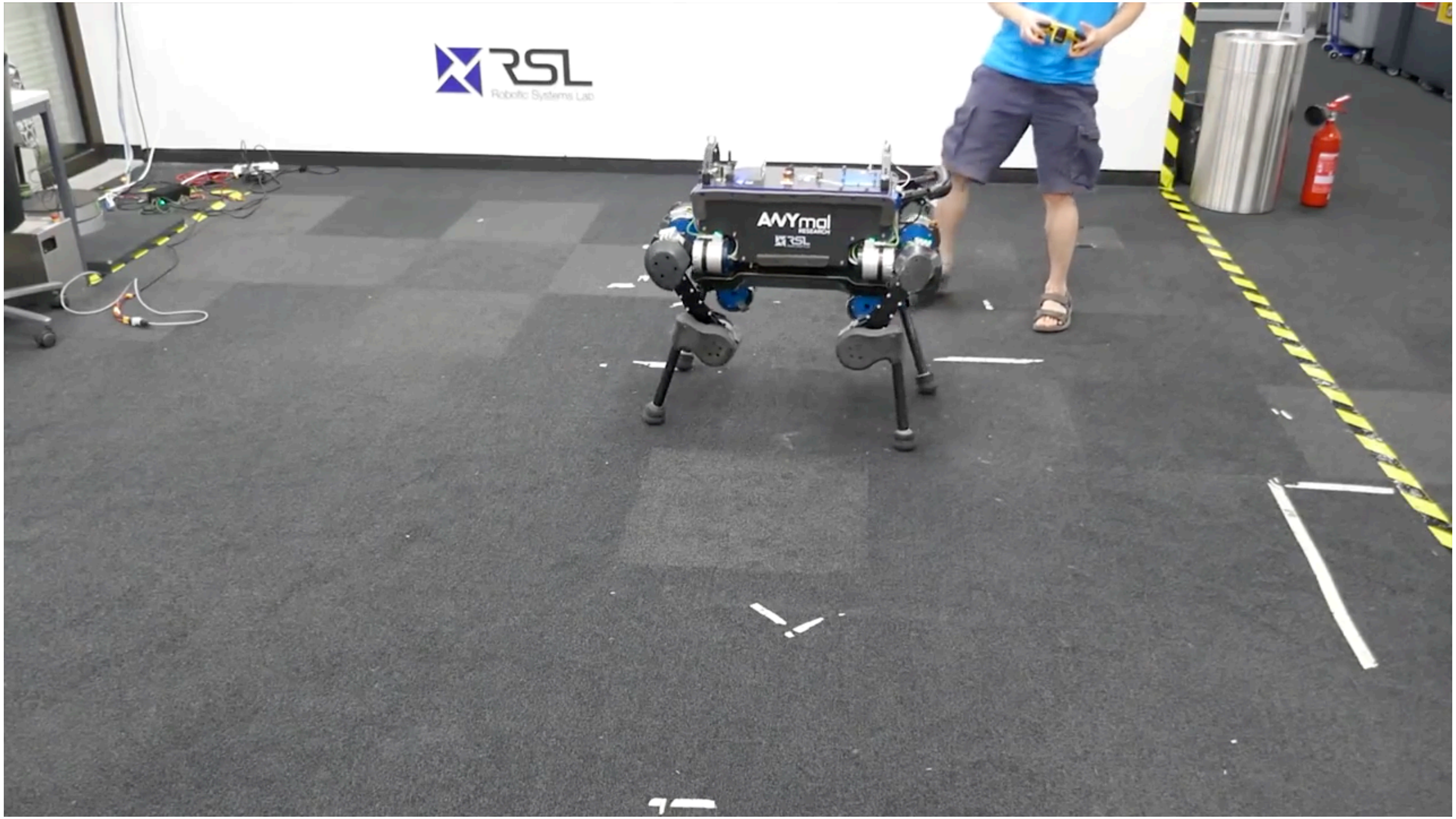
joint torques $=\pi_\theta($  $)$

Continues motion control from RGB(D)

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



No visual inputs + flat terrain => simple domain transfer

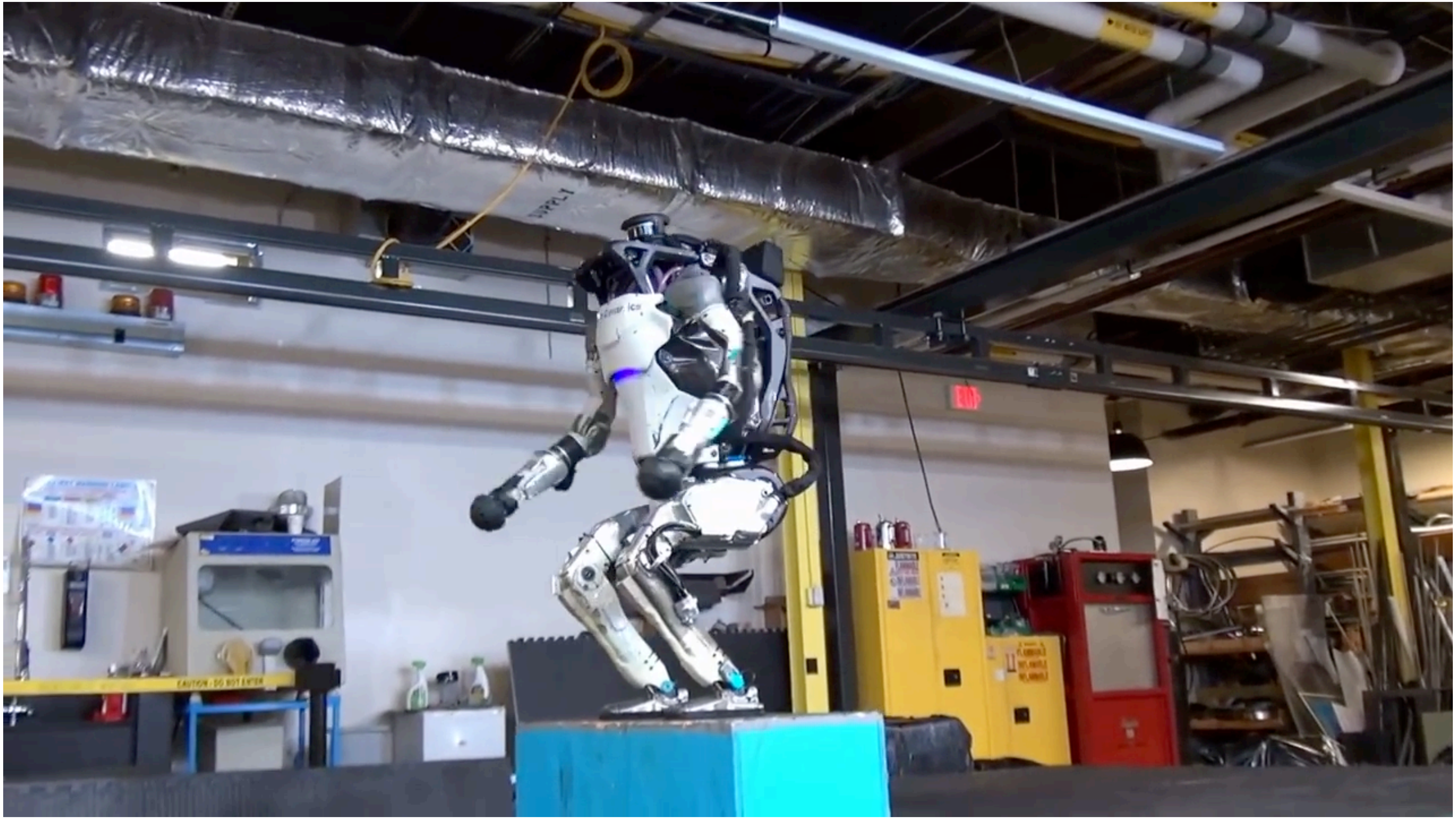


[Hwangbo, ETH Zurich, Science Robotics, 2018]

Motion and compliance control of flippers

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

Boston dynamics - Atlas - NO RL AT ALL



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

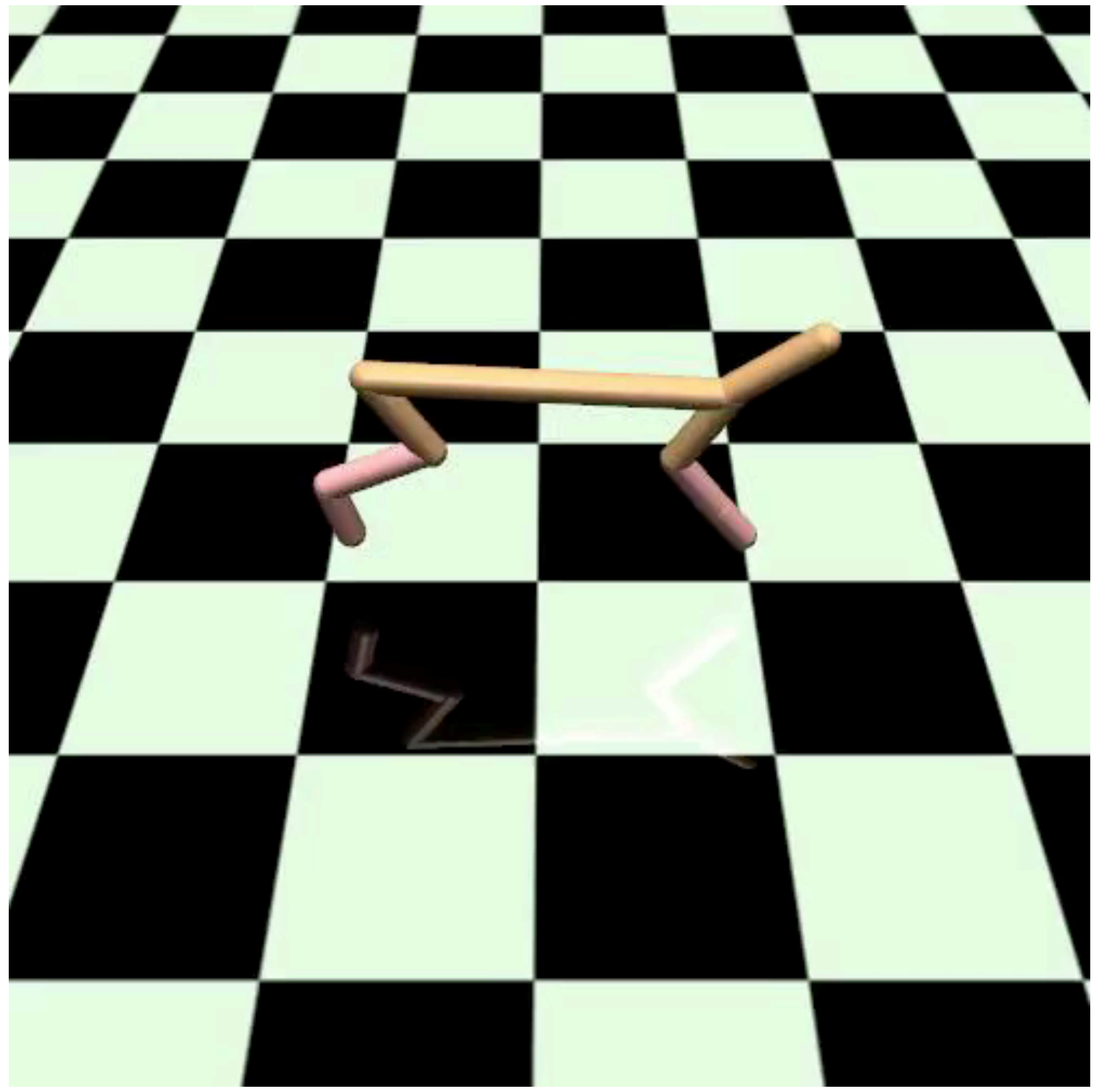
→ $\pi^* = \arg\max_\pi J_\pi$

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

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

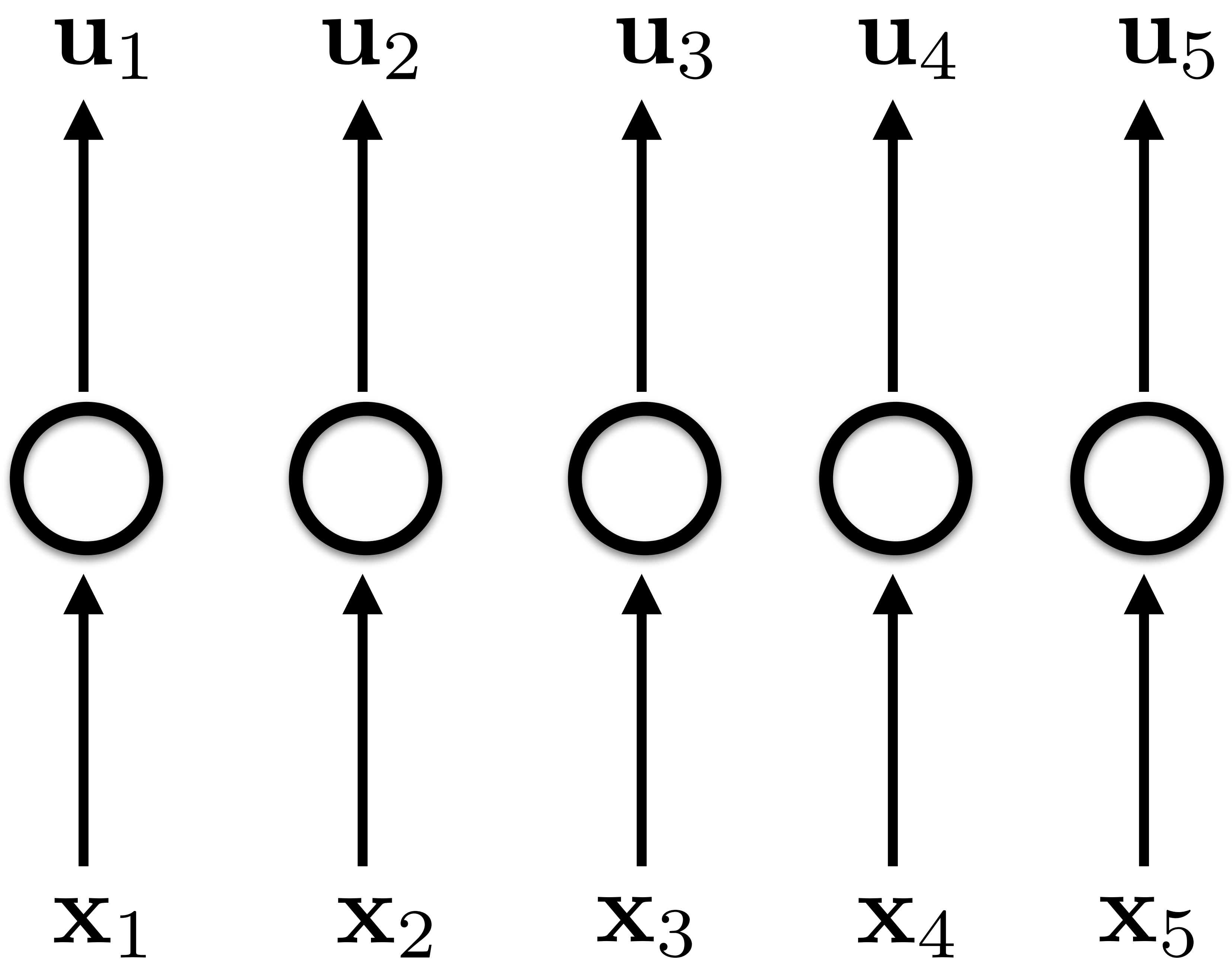
Rewards engineering

- 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

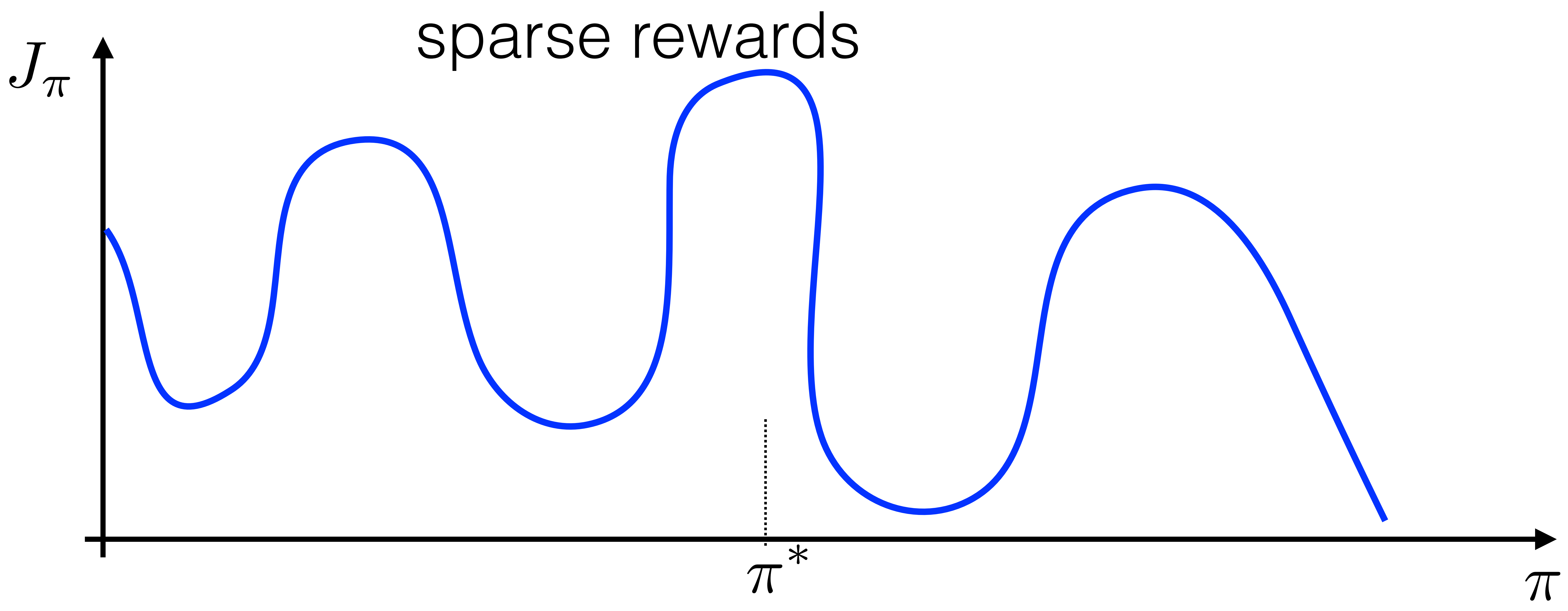
- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn

Rewards engineering

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

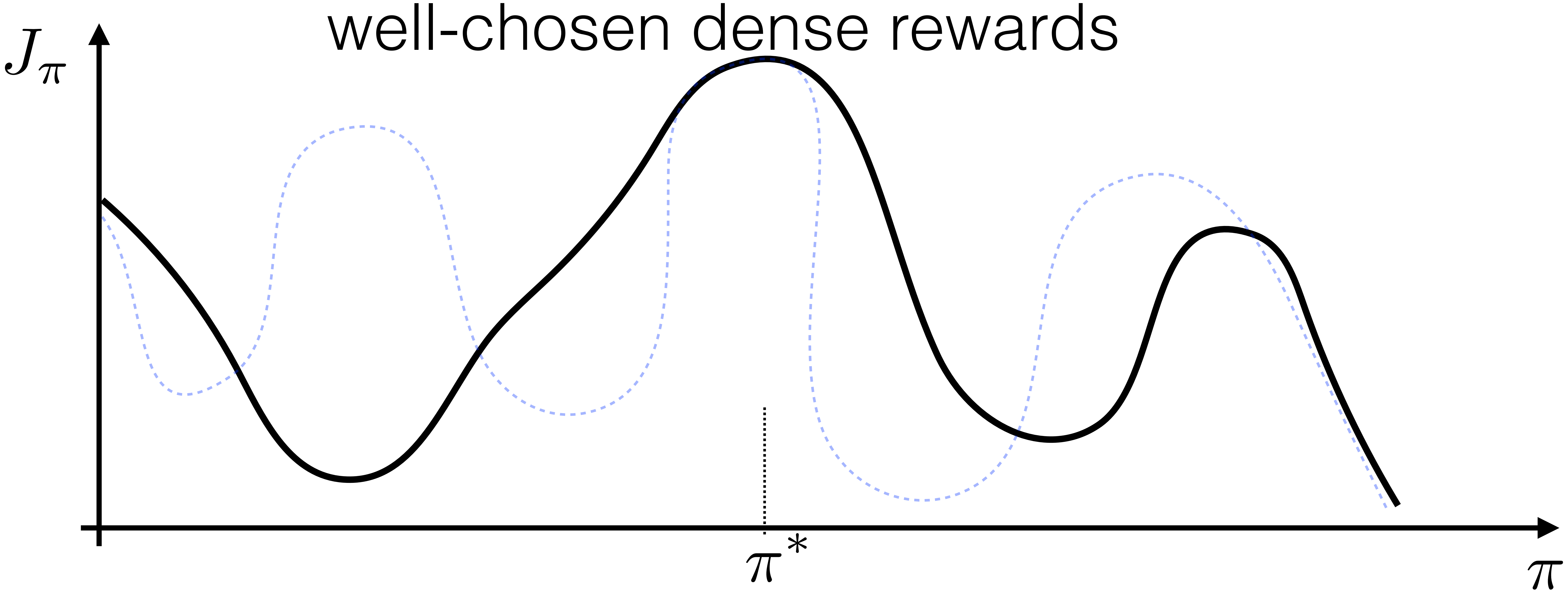
Rewards engineering

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



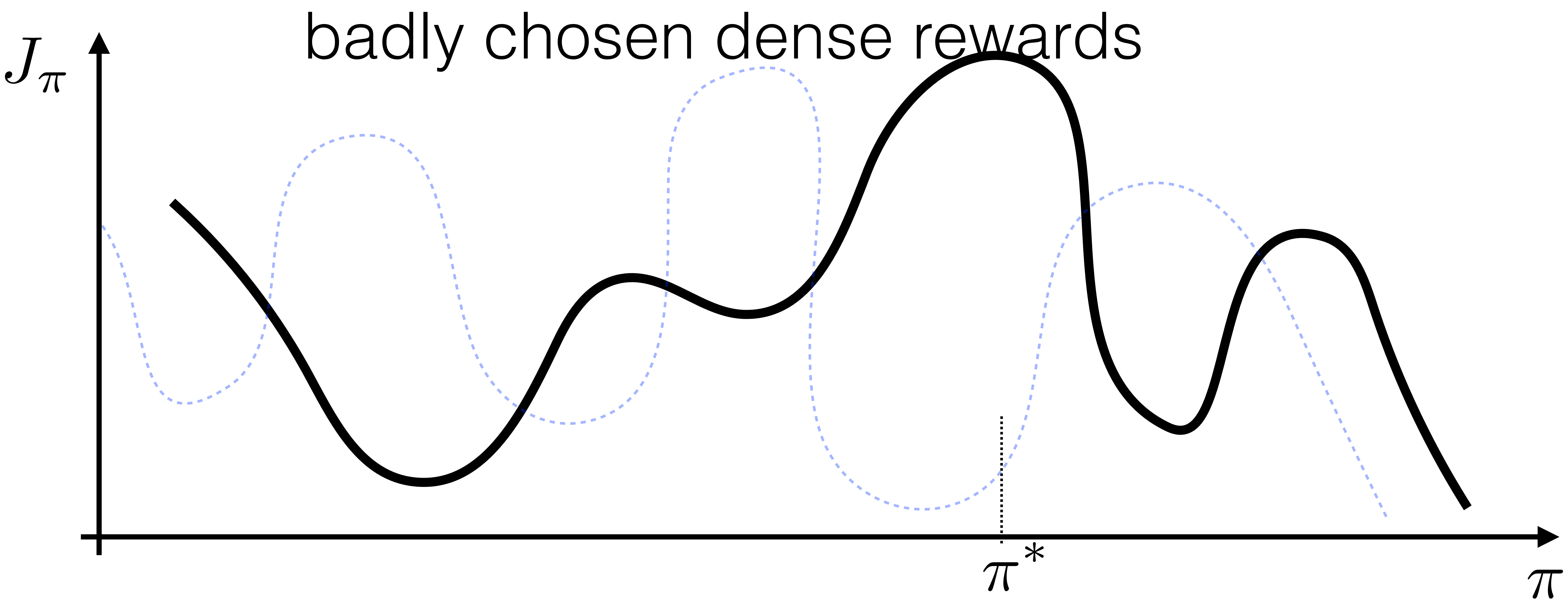
Rewards engineering

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



Rewards engineering

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



Rewards engineering

- Dense reward allows to easier find the corresponding action but they are more likely to introduce bias.
- 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 $r_\mathbf{w}$

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

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

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 $r_\mathbf{w}$

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

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

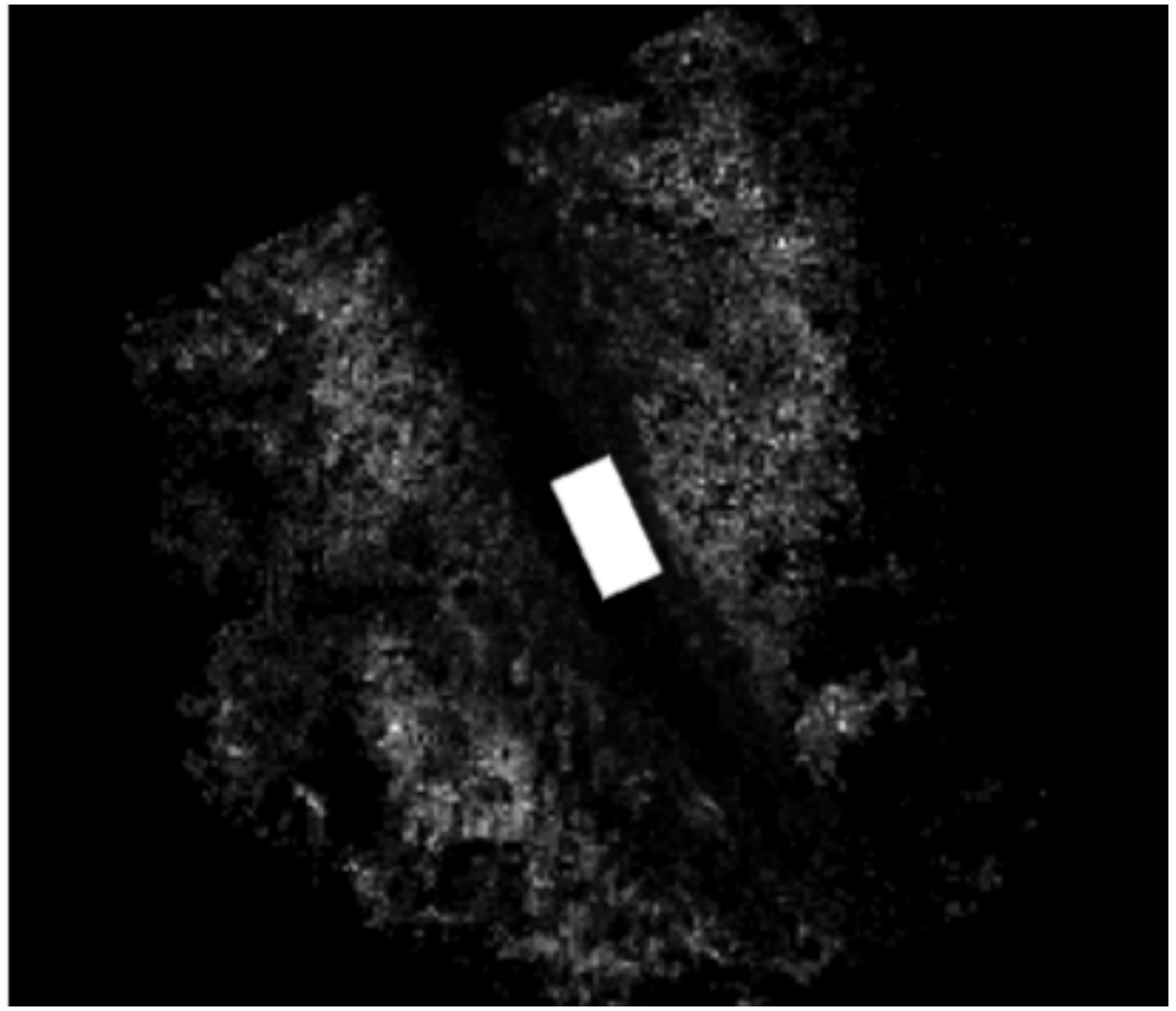
Abbeel et al. IJRR 2010

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)

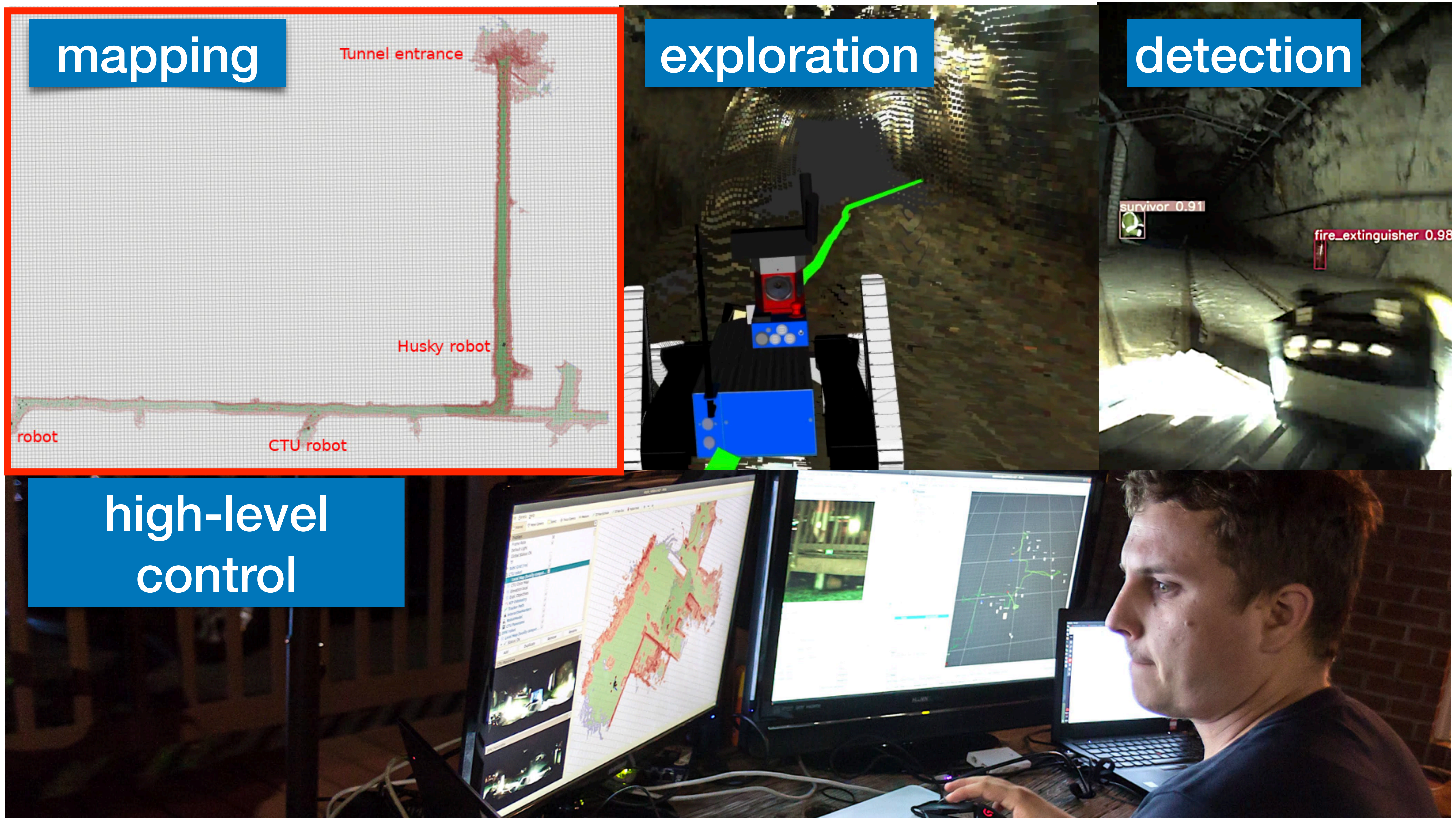


learned reward function
(traversability map)

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??

panoramic images

$\mathbf{x}$ → $\pi_\theta(\mathbf{u}|\mathbf{x})$ → motor torques + detected objects

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**

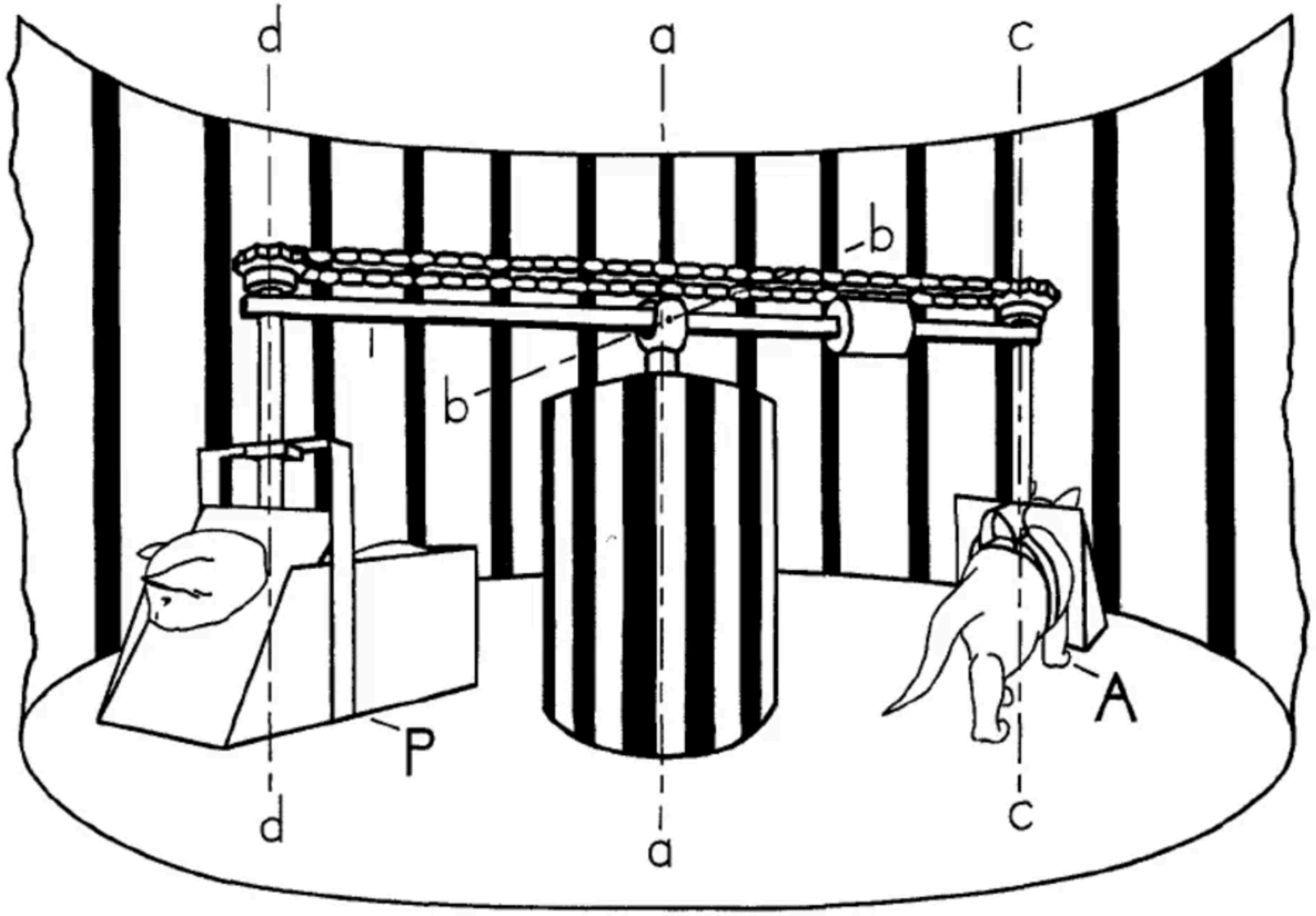


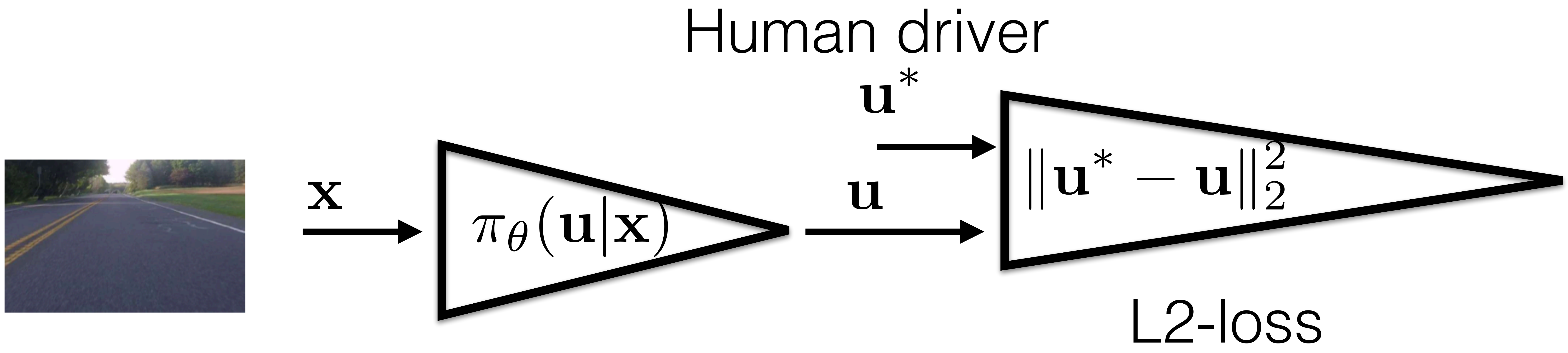
Fig. 1. Apparatus for equating motion and consequent visual feedback for an actively moving (A) and a passively moved (P) S.

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 a all-in-one network, the behaviour not interpretable.

[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?



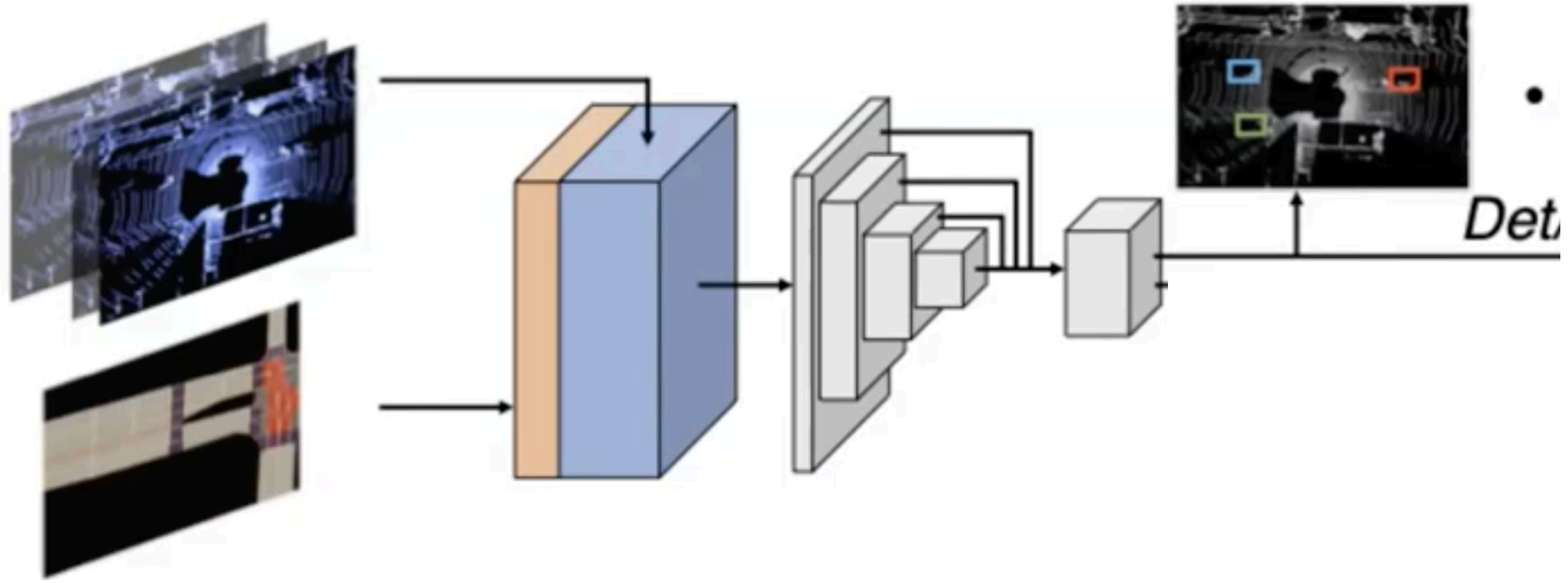
Policy:
$$\pi_\theta(\mathbf{u}|\mathbf{x}) = f(\mathbf{x}, \theta)$$

- **Reliable? Explainable? Managable?**

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

Lidar scans

t=1



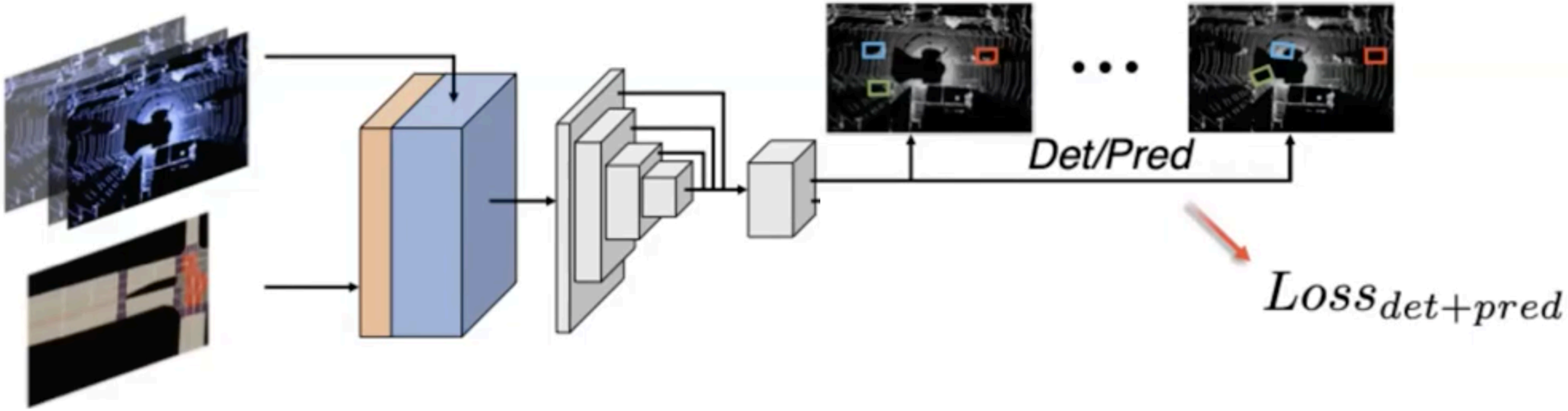
lanes
+
traffic signs
+
traffic lights

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

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

Lidar scans

t=1 ... t=T

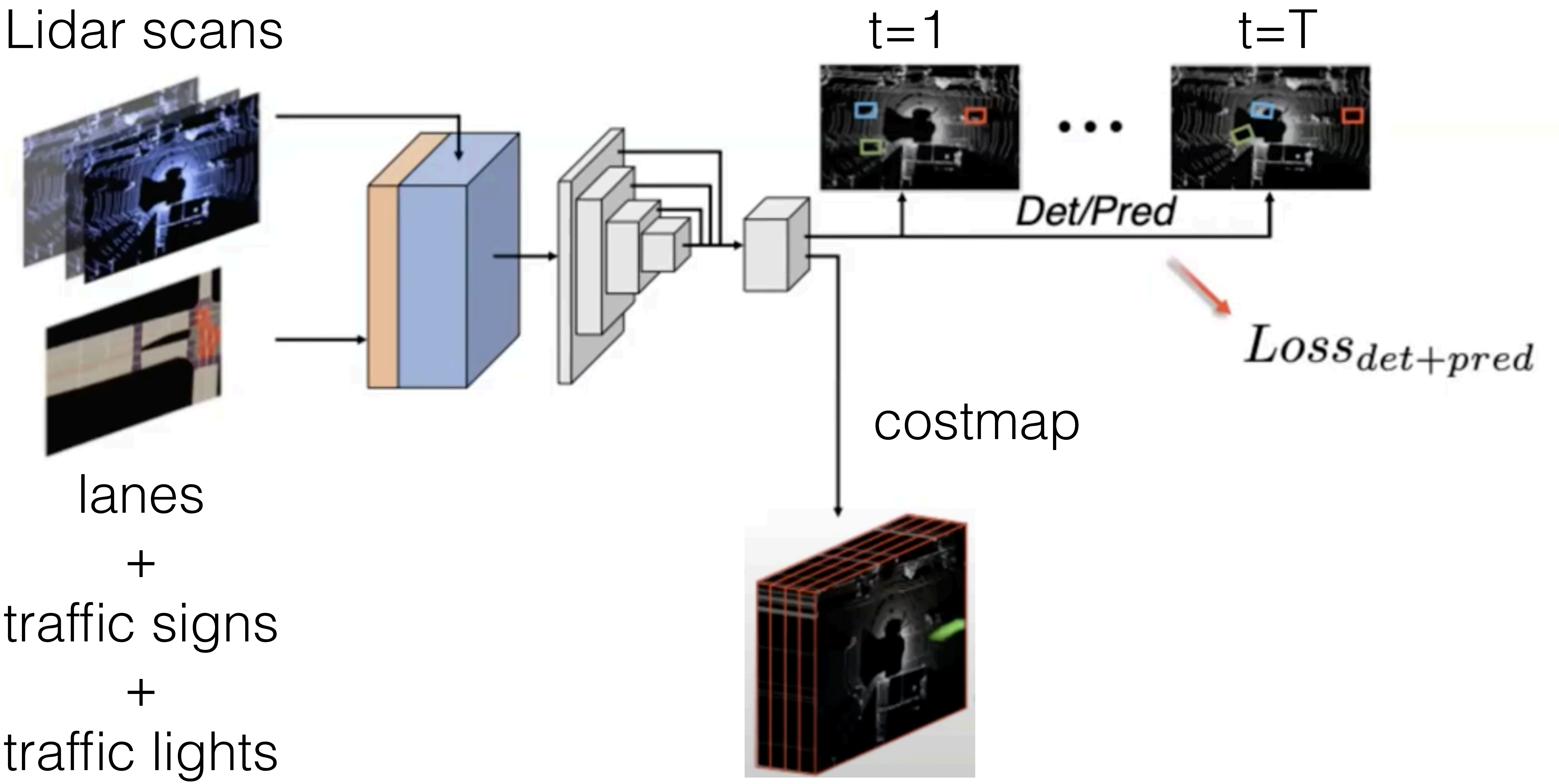


lanes
+
traffic signs
+
traffic lights

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

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

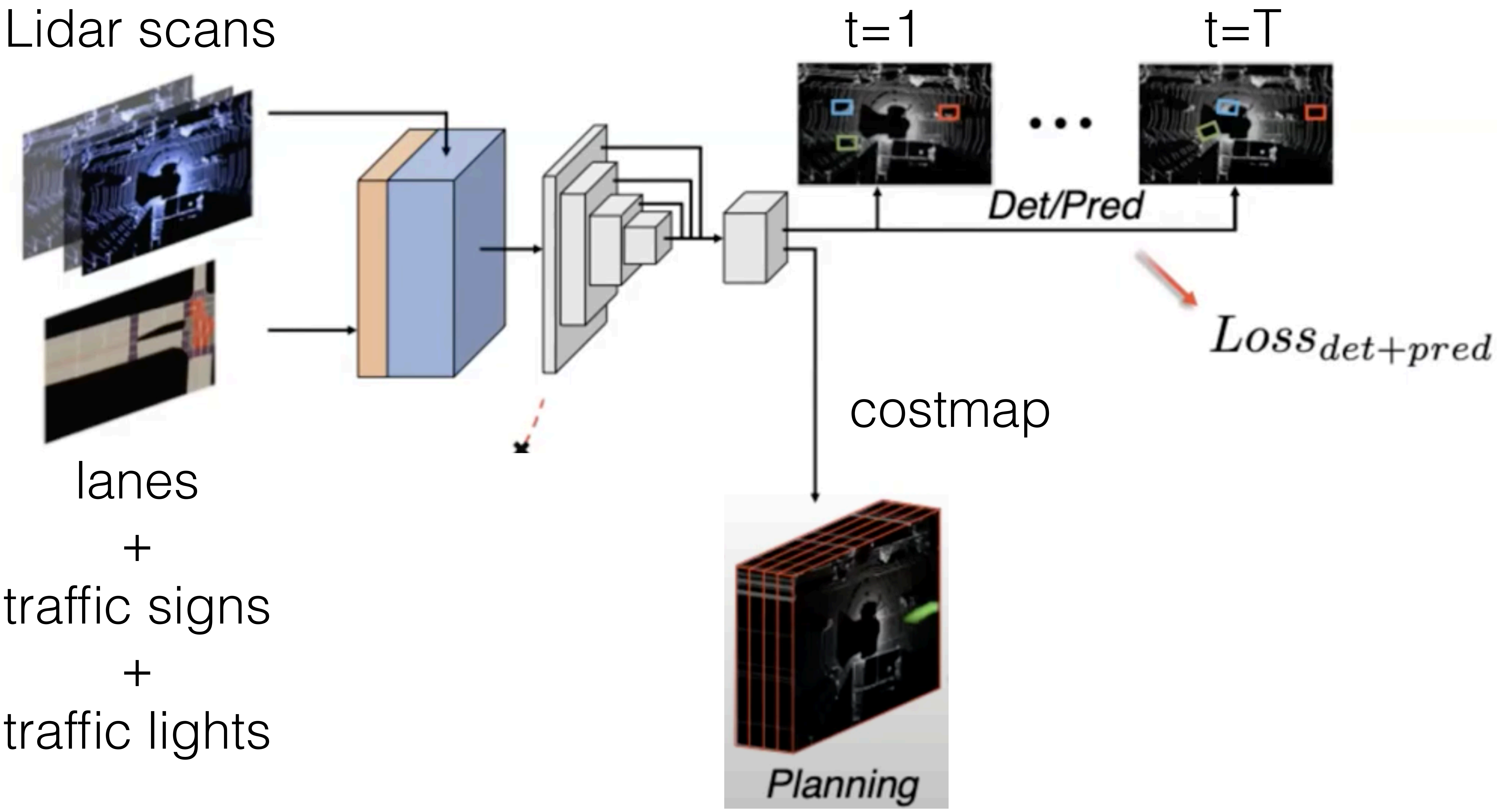
Lidar scans



lanes
+
traffic signs
+
traffic lights

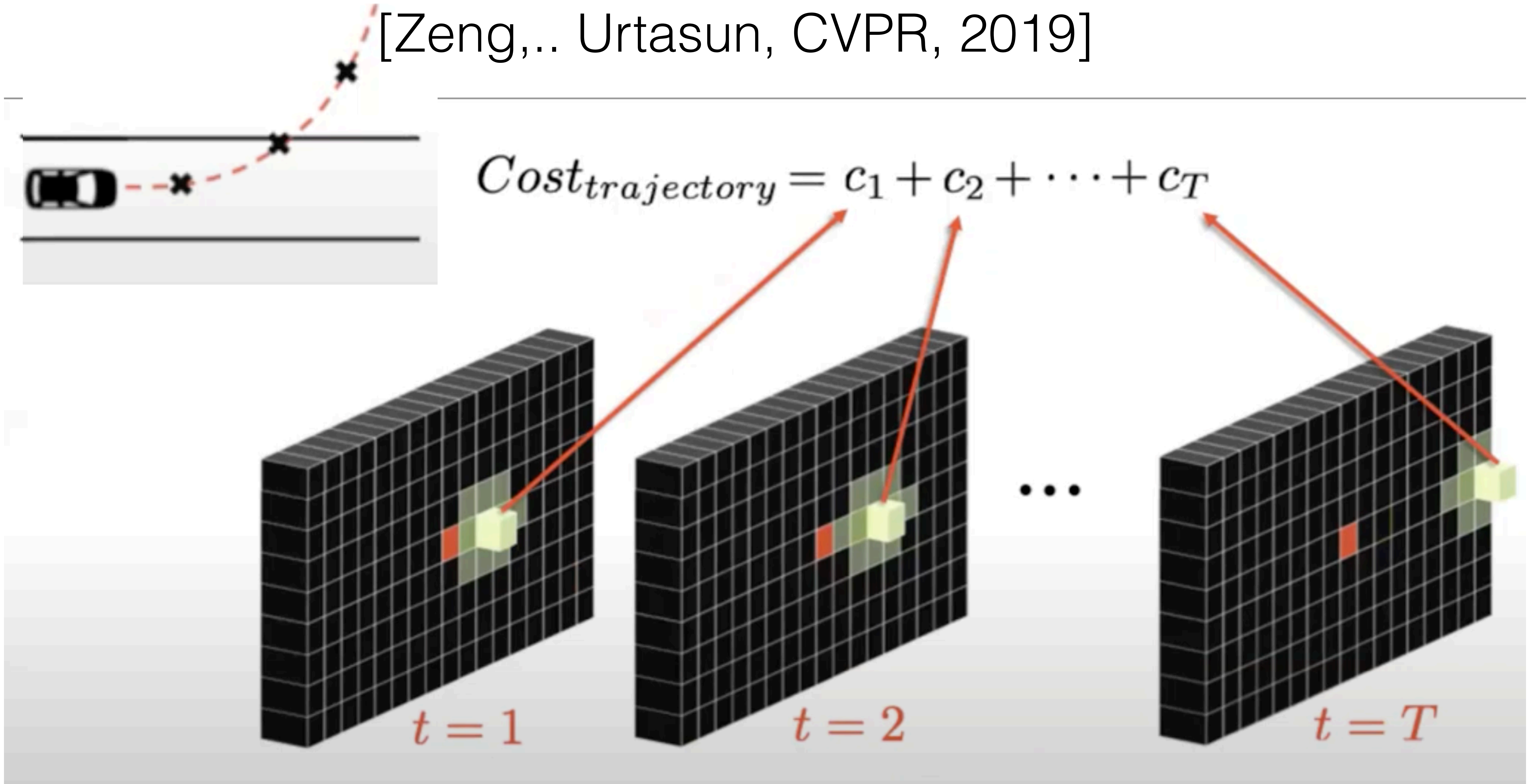
http://www.cs.toronto.edu/~wenjie/

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



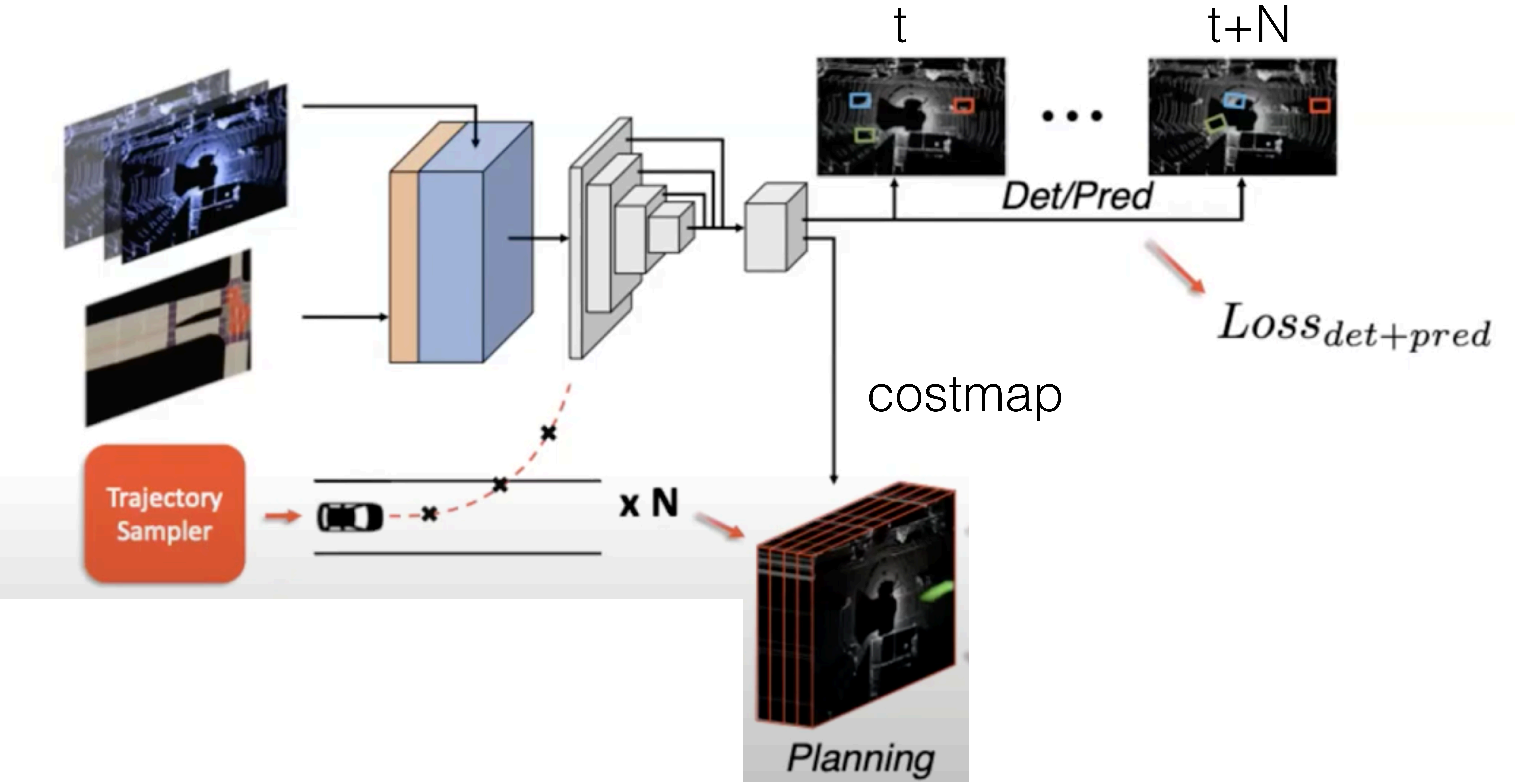
http://www.cs.toronto.edu/~wenjie/

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



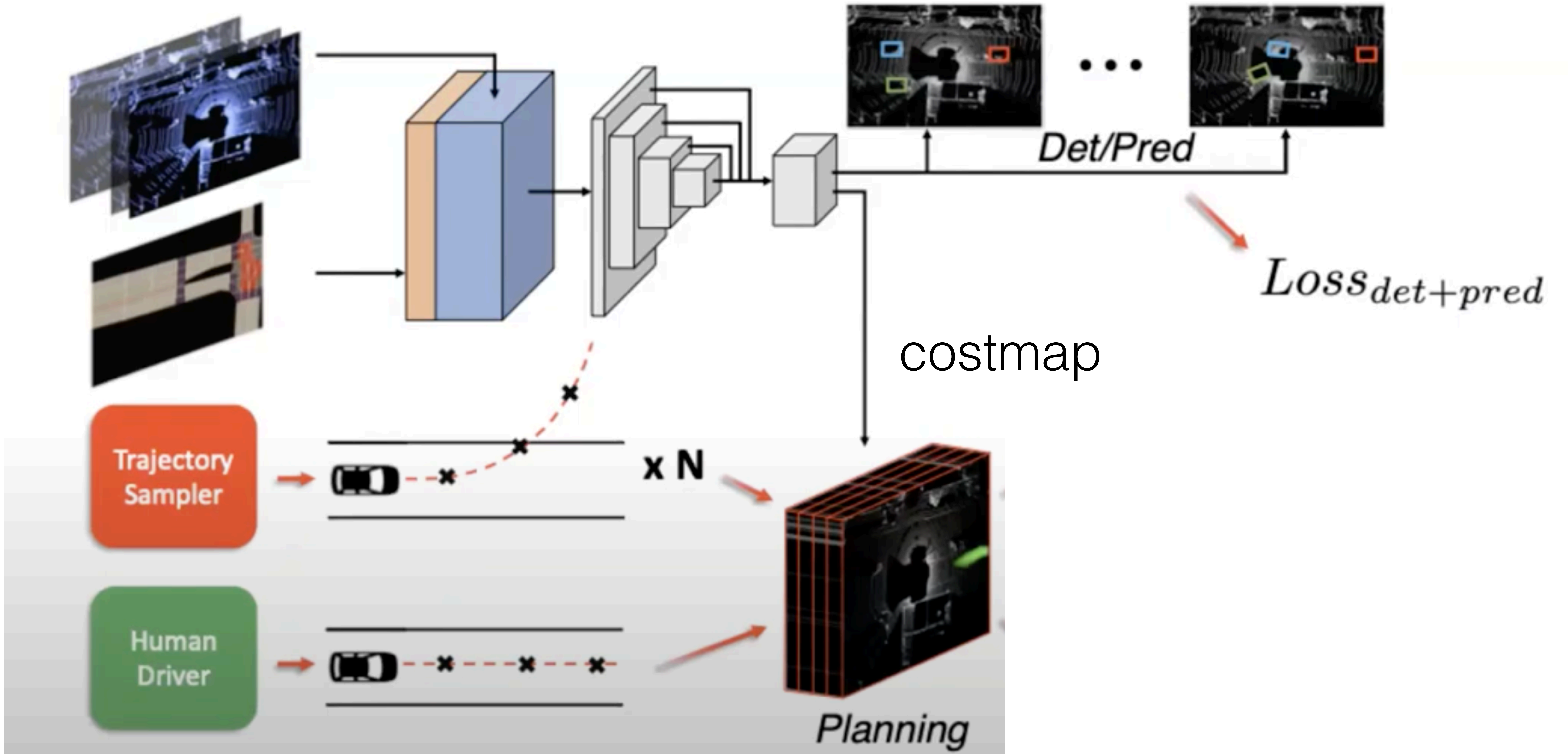
http://www.cs.toronto.edu/~wenjie/

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



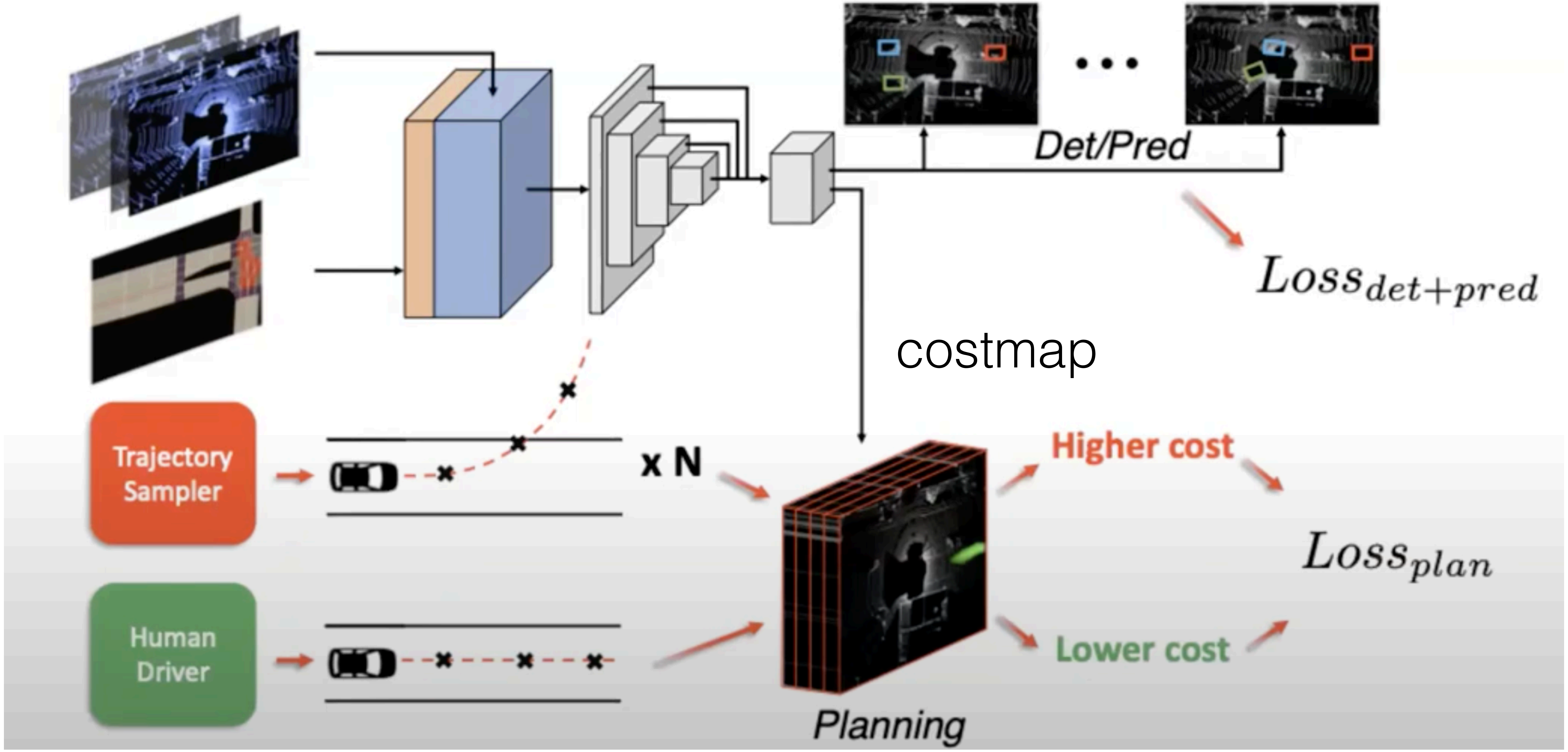
http://www.cs.toronto.edu/~wenjie/

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



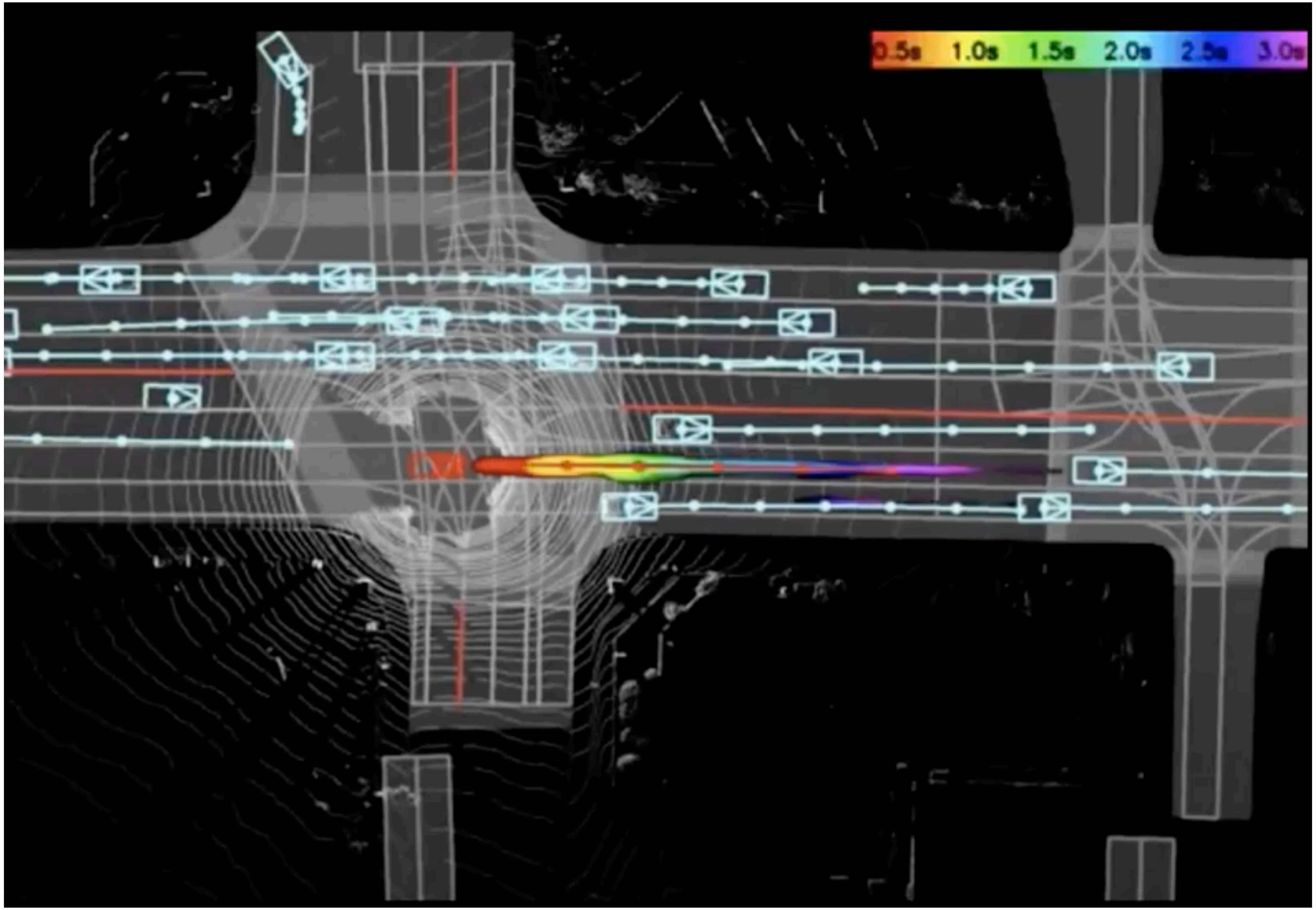
http://www.cs.toronto.edu/~wenjie/

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



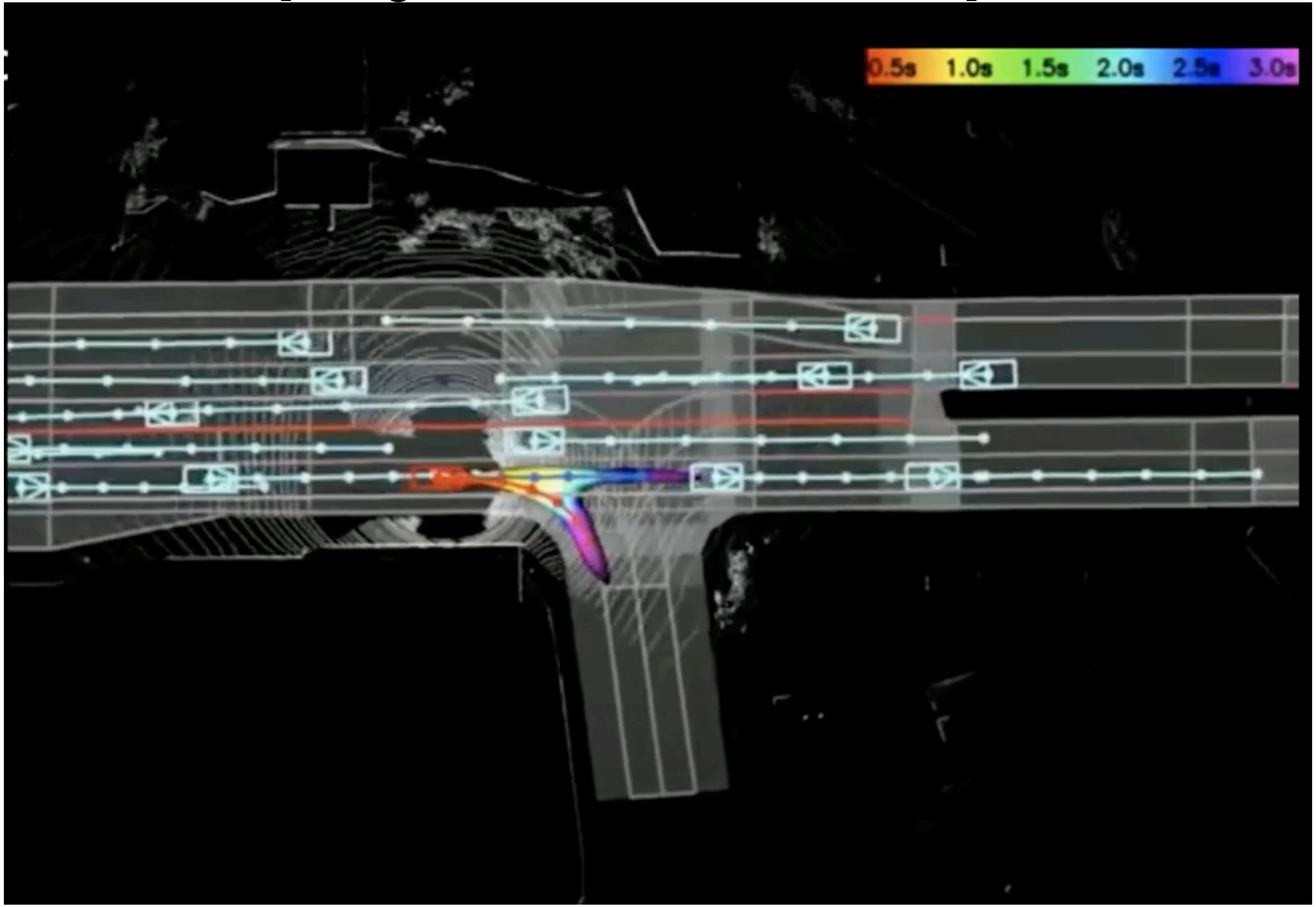
http://www.cs.toronto.edu/~wenjie/

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



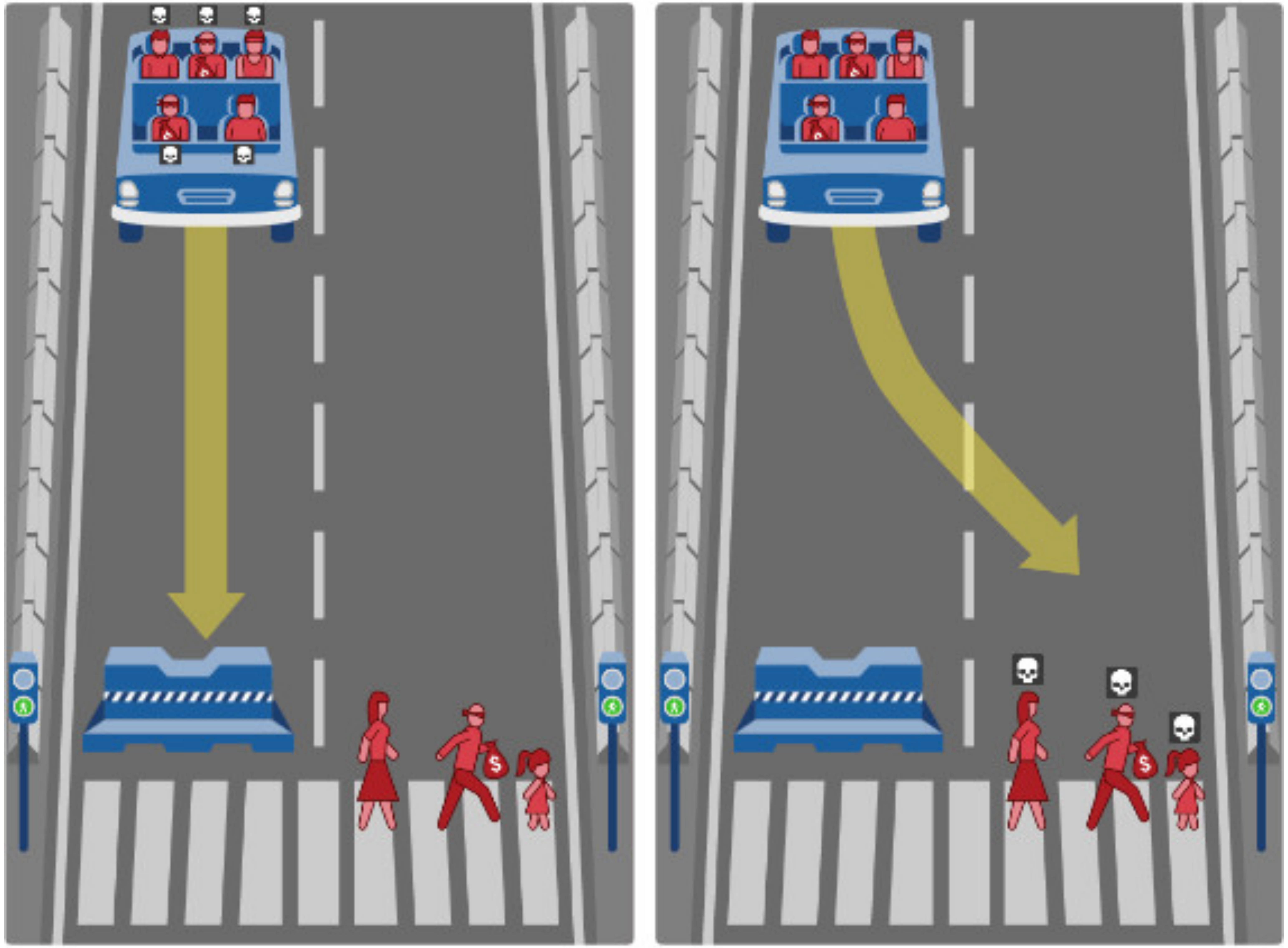
http://www.cs.toronto.edu/~wenjie/

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



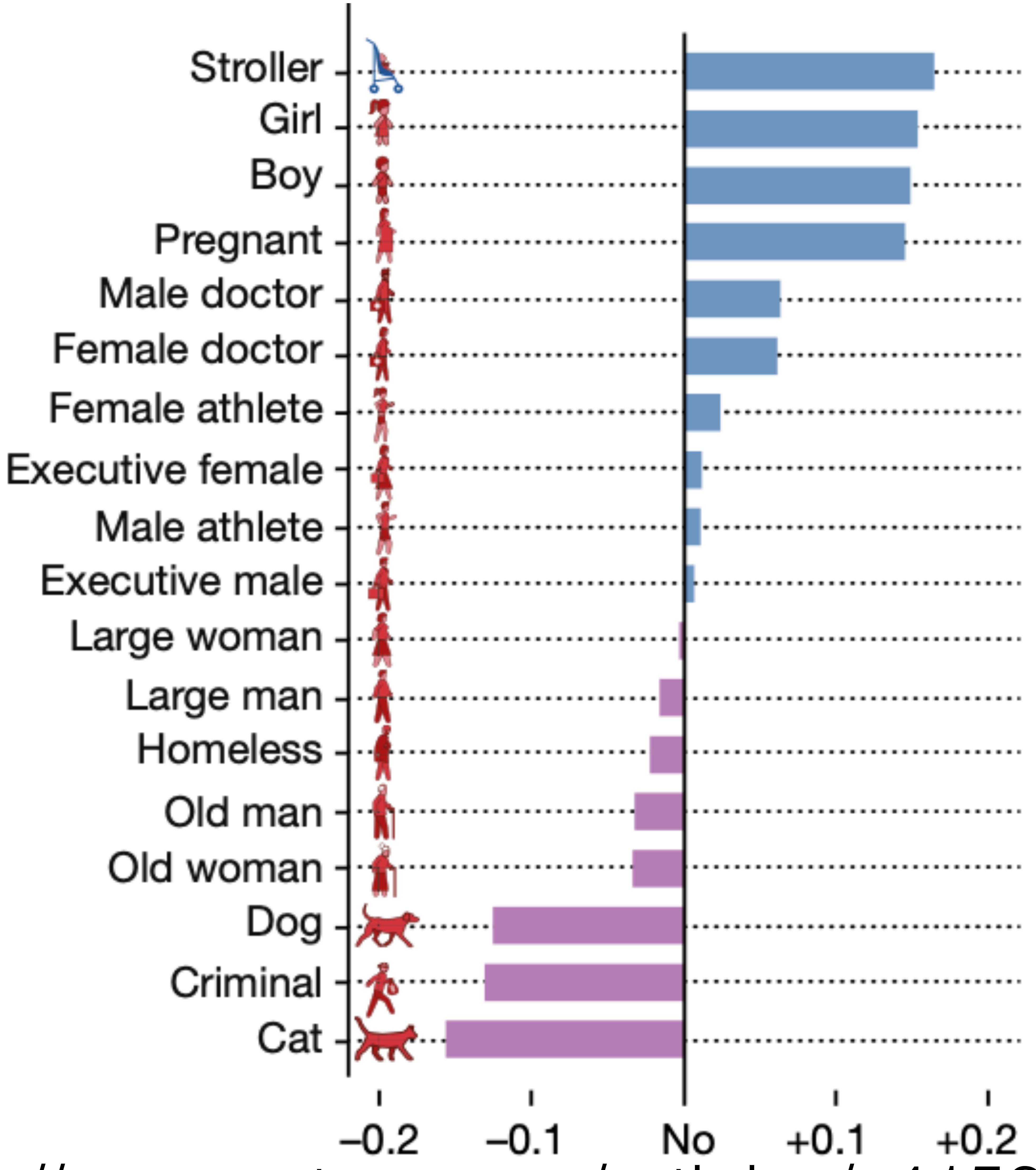
http://www.cs.toronto.edu/~wenjie/

Trolley problem



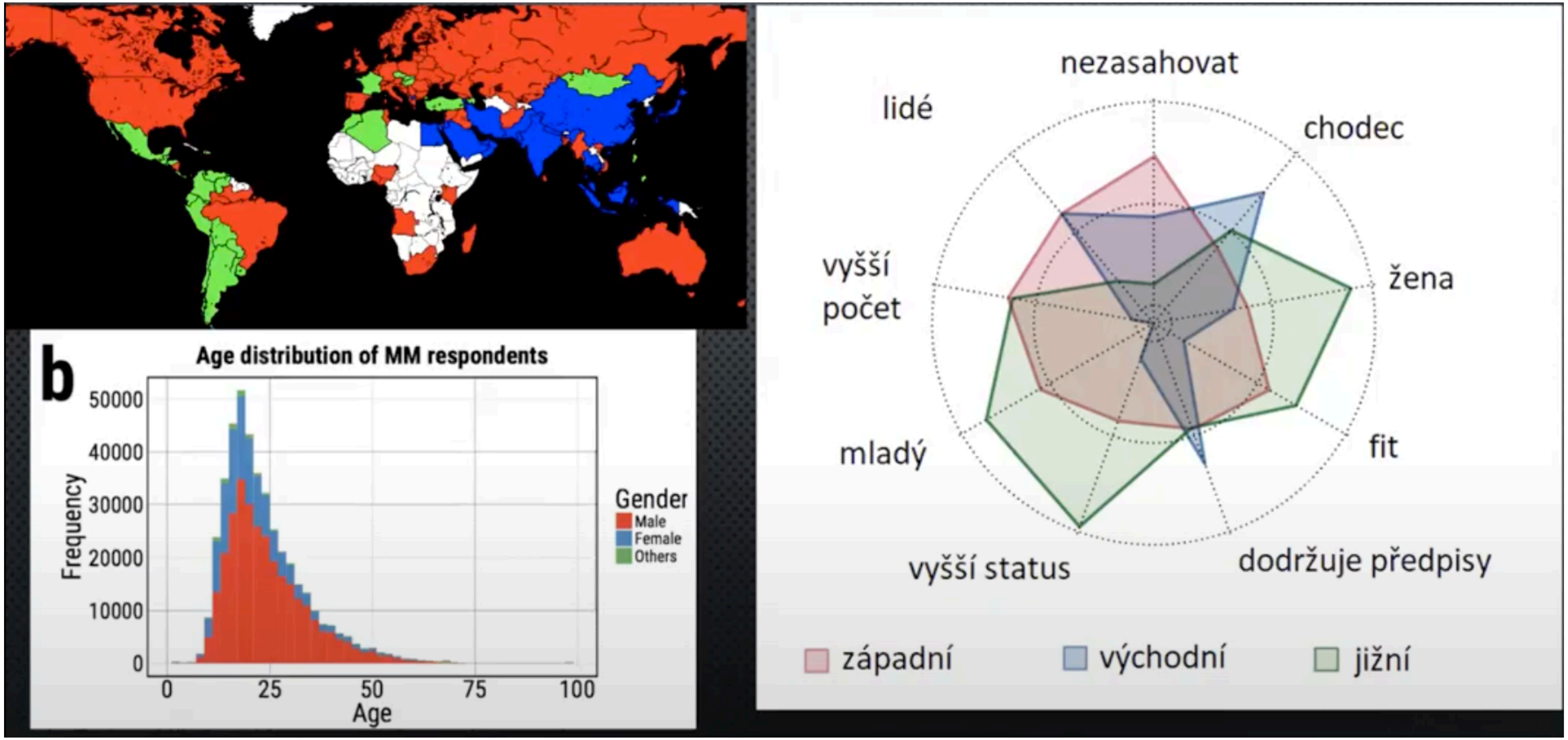
https://www.nature.com/articles/s41586-018-0637-6
[Moral Machine Experiment, Nature, 2018]

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

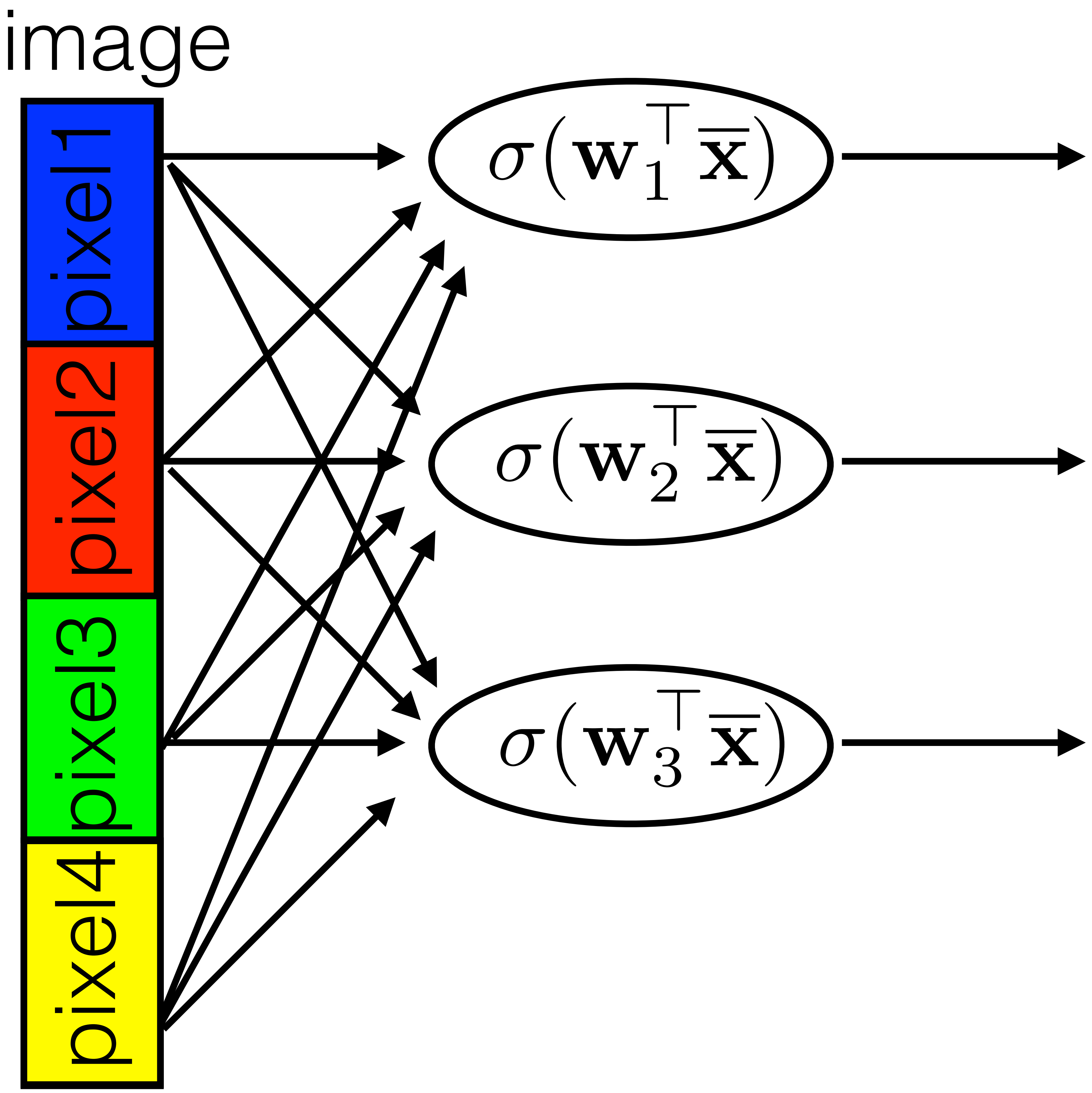
https://www.nature.com/articles/s41586-018-0637-6
[Moral Machine Experiment, Nature, 2018]

Summary

- If accurate differentiable motion model and reward functions are known, than optimal control is straightforward optimization problem (efficiently tackled by MPC)
- State-action value function is dual variable wrt policy. It serves as auxiliary function in the policy optimization:
  - actor-critic methods
  - heuristic in planning methods (LQR trees)
- **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 !!!!**

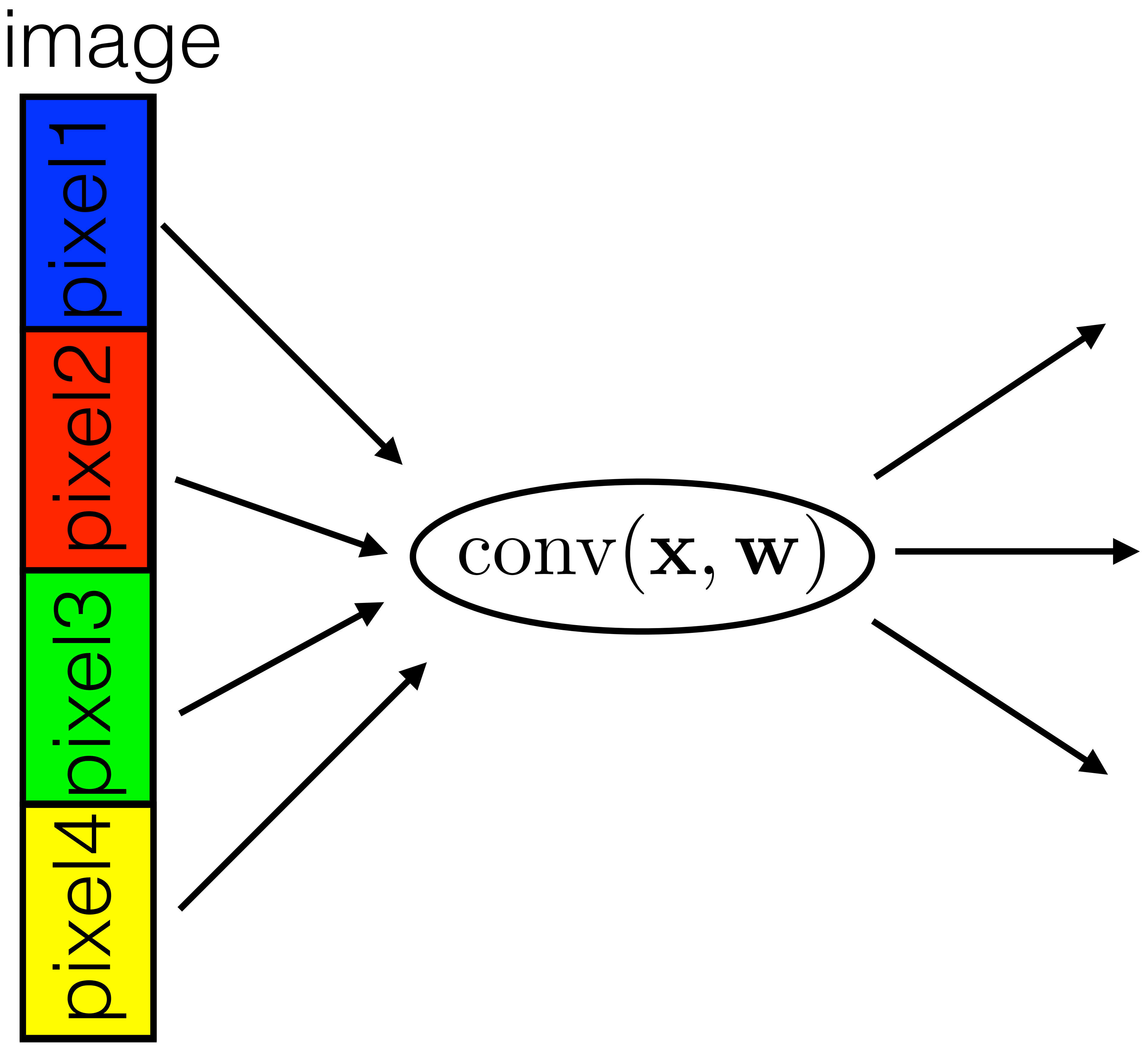
Regularization & overfitting

- Best regularization is using the right structure of the network



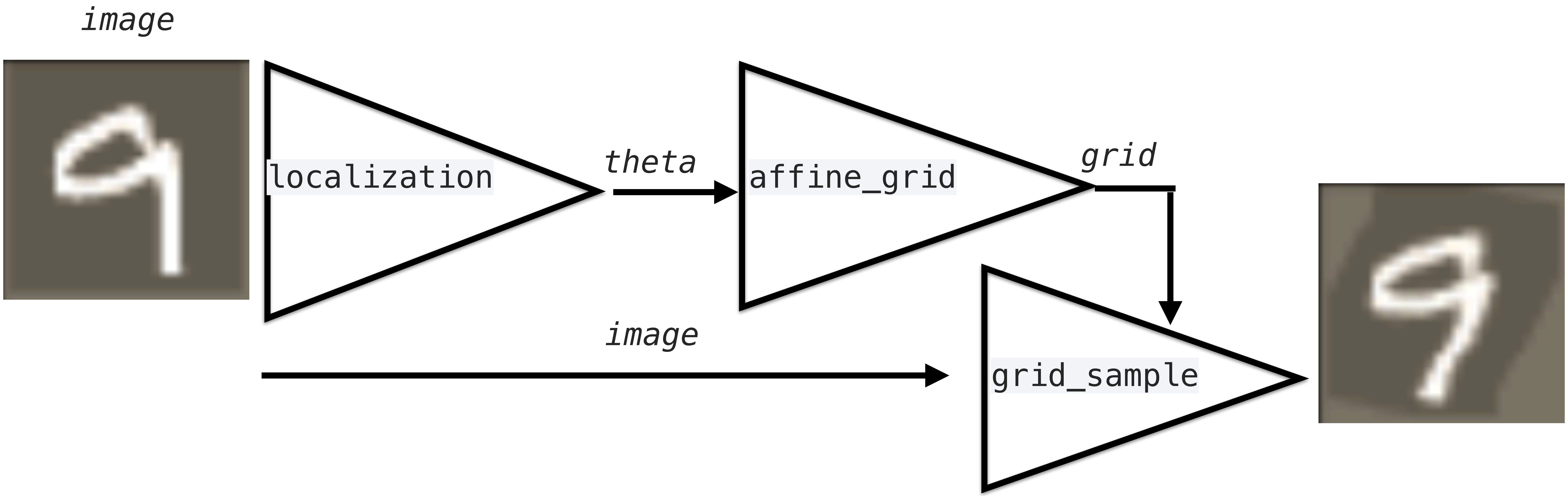
Regularization & overfitting

- Best regularization is using the right structure of the network



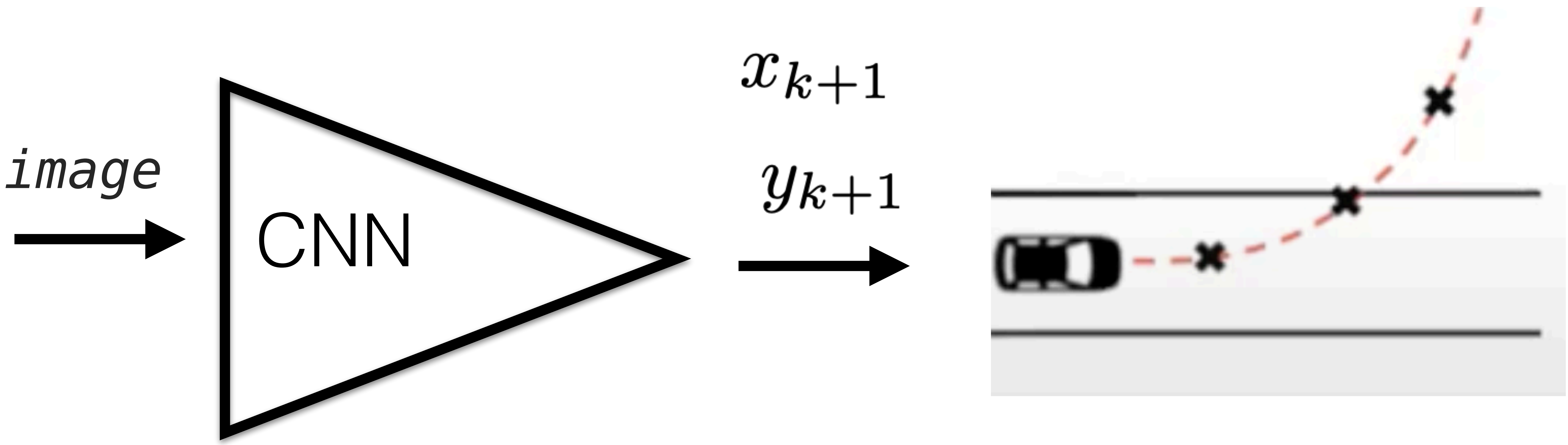
Regularization & overfitting

- Best regularization is using the right structure of the network



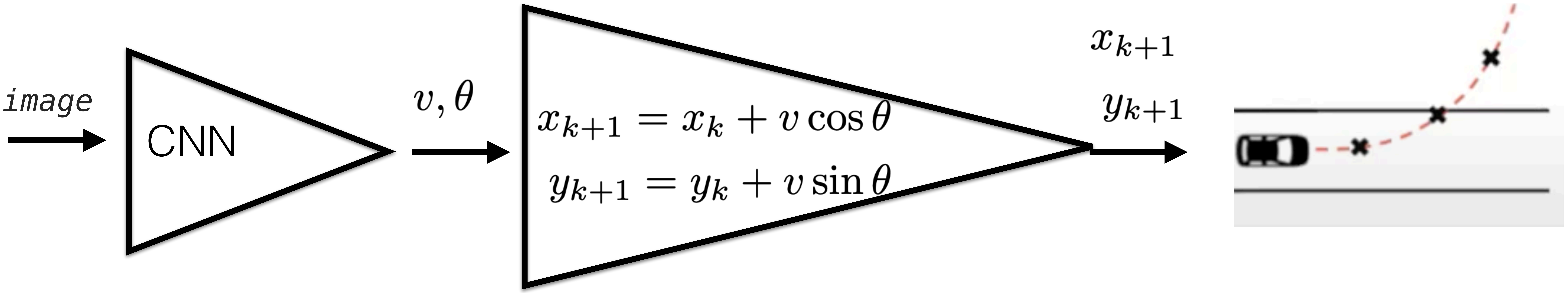
Regularization & overfitting

- Best regularization is using the right structure of the network



Regularization & overfitting

- Best regularization is using the right structure of the network



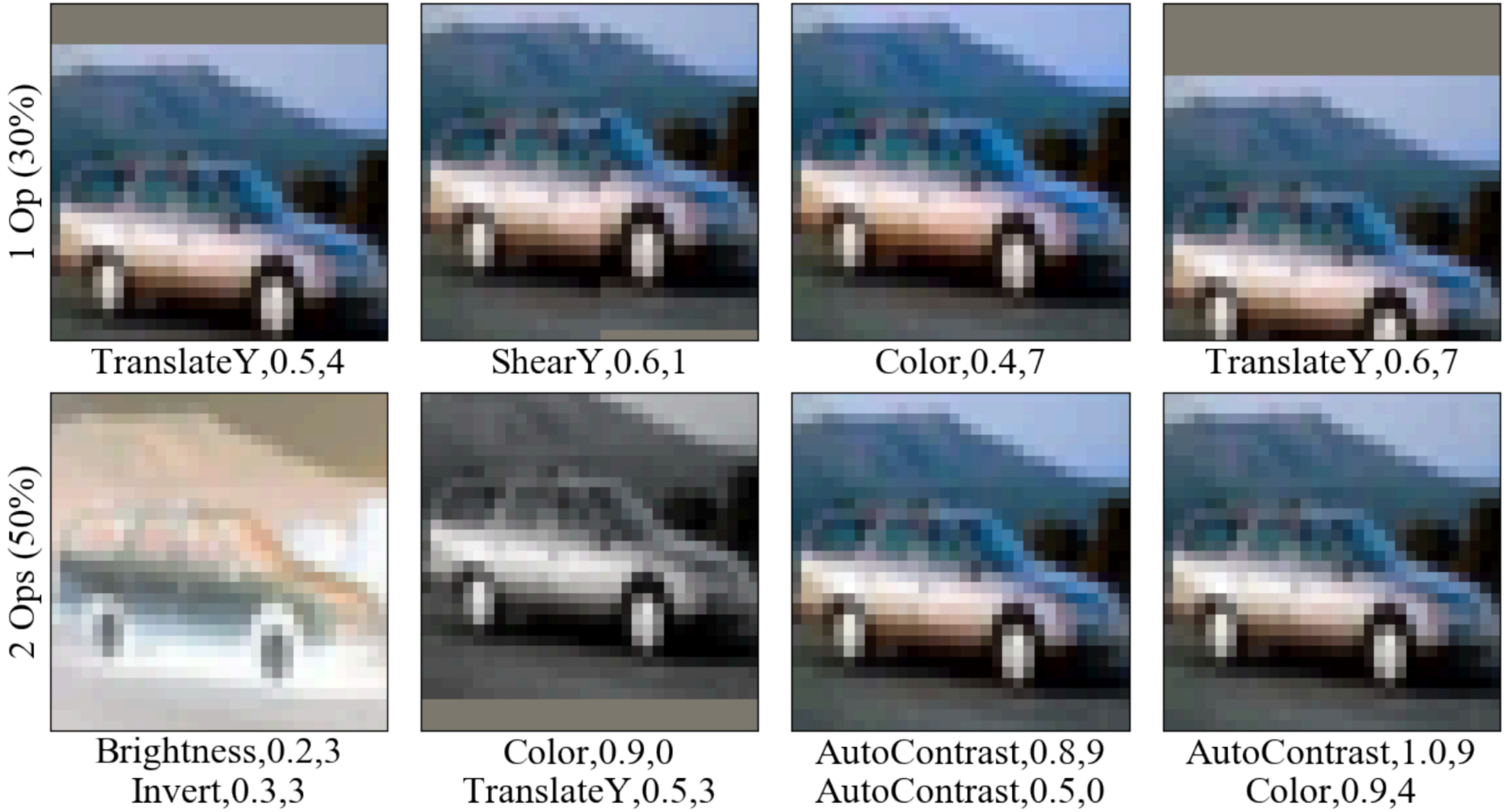
Regularization & overfitting

- Best regularization is using the right structure of the network
- L2, L1 norms on weights
  - avoids overfitting and exploding gradient
  - implemented via weight_decay parameter in PyTorch

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, 
weight_decay=1e-4)
```

Regularization & overfitting

- 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



Regularization & overfitting

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

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