

# 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



# Tasks often formalised as MDP

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

$\mathbf{x}$  ●



# Both tasks formalised as reinforcement learning problems

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



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

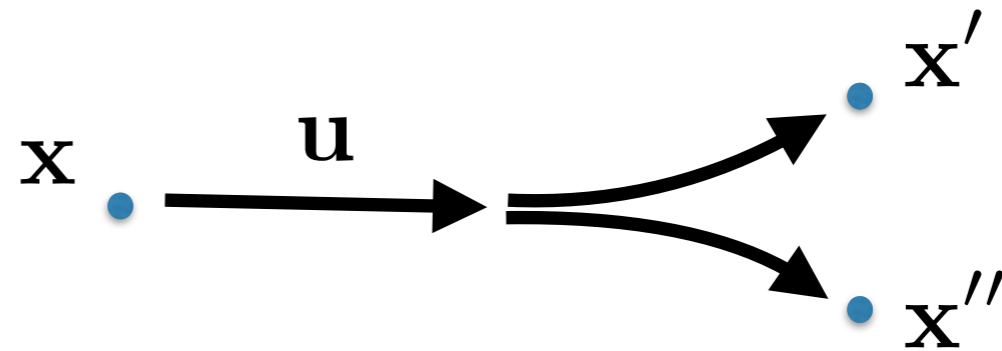


# Both tasks formalised as reinforcement learning problems

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

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

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



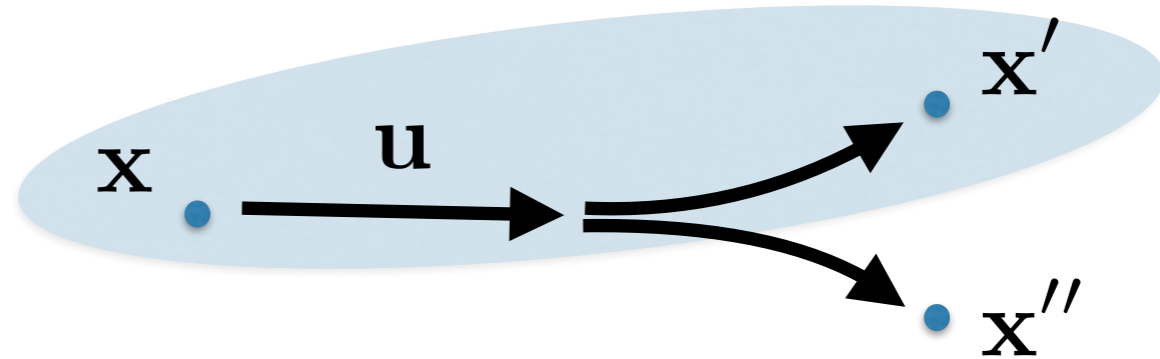
# Both tasks formalised as reinforcement learning problems

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

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

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

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



# Both tasks formalised as reinforcement learning problems

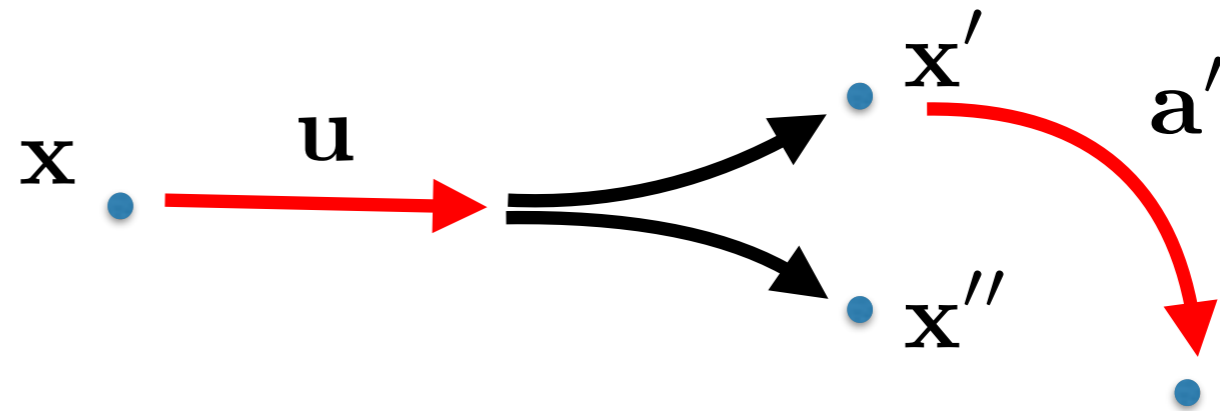
States:  $\mathbf{x} \in \mathcal{R}^n$

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



# Both tasks formalised as reinforcement learning problems

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

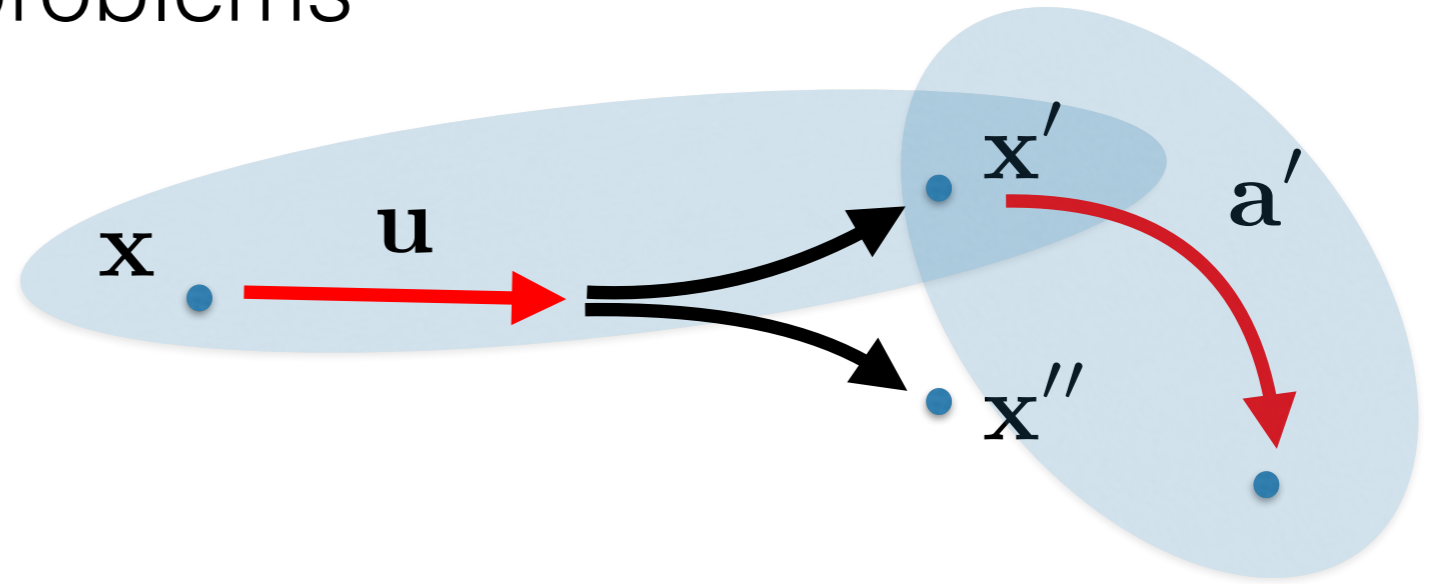
Actions:  $\mathbf{u} \in \mathcal{R}^m$

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.  $J_{\pi} = \mathbb{E} \left[ \sum_{t=0}^T r_t \right]$  )



# Challenges in real tasks

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.  $J_{\pi} = \mathbb{E} \left[ \sum_{t=0}^T r_t \right]$  )





# Challenges in real tasks

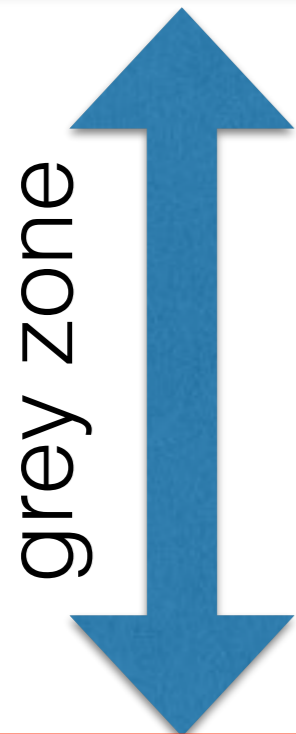
- Can I learn something without the model  $p(\mathbf{x}' | \mathbf{x}, \mathbf{u})$  just from interactions?



# Taxonomy of policy search methods

- Direct policy search (primal task)

e.g. gradient ascent for  $\pi^* = \arg \max_{\pi} J_{\pi}$



Episodic REPS [Peters, 2010]

PILCO [Deisenroth, ICML 2011]

Actor-critic (e.g. DPG [Silver, JMLR 2014])

Deep Q-learning (e.g. [Mnih, Nature 2015])

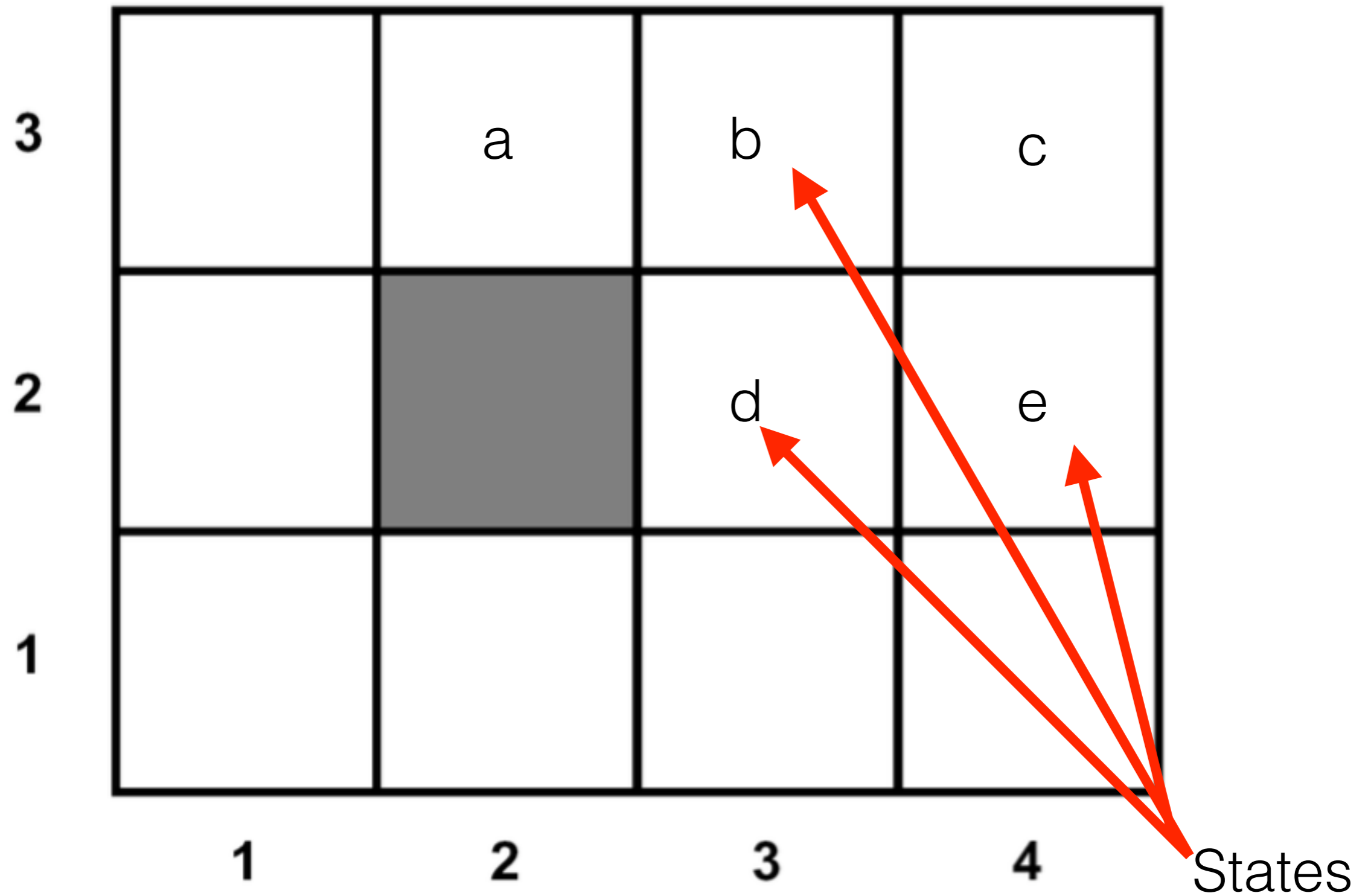
- Value-based methods (dual function [Kober, 2013])

e.g. search for  $Q(\mathbf{x}, \mathbf{a}) = r(\mathbf{x}, \mathbf{a}, \mathbf{x}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{x}', \mathbf{a}')$

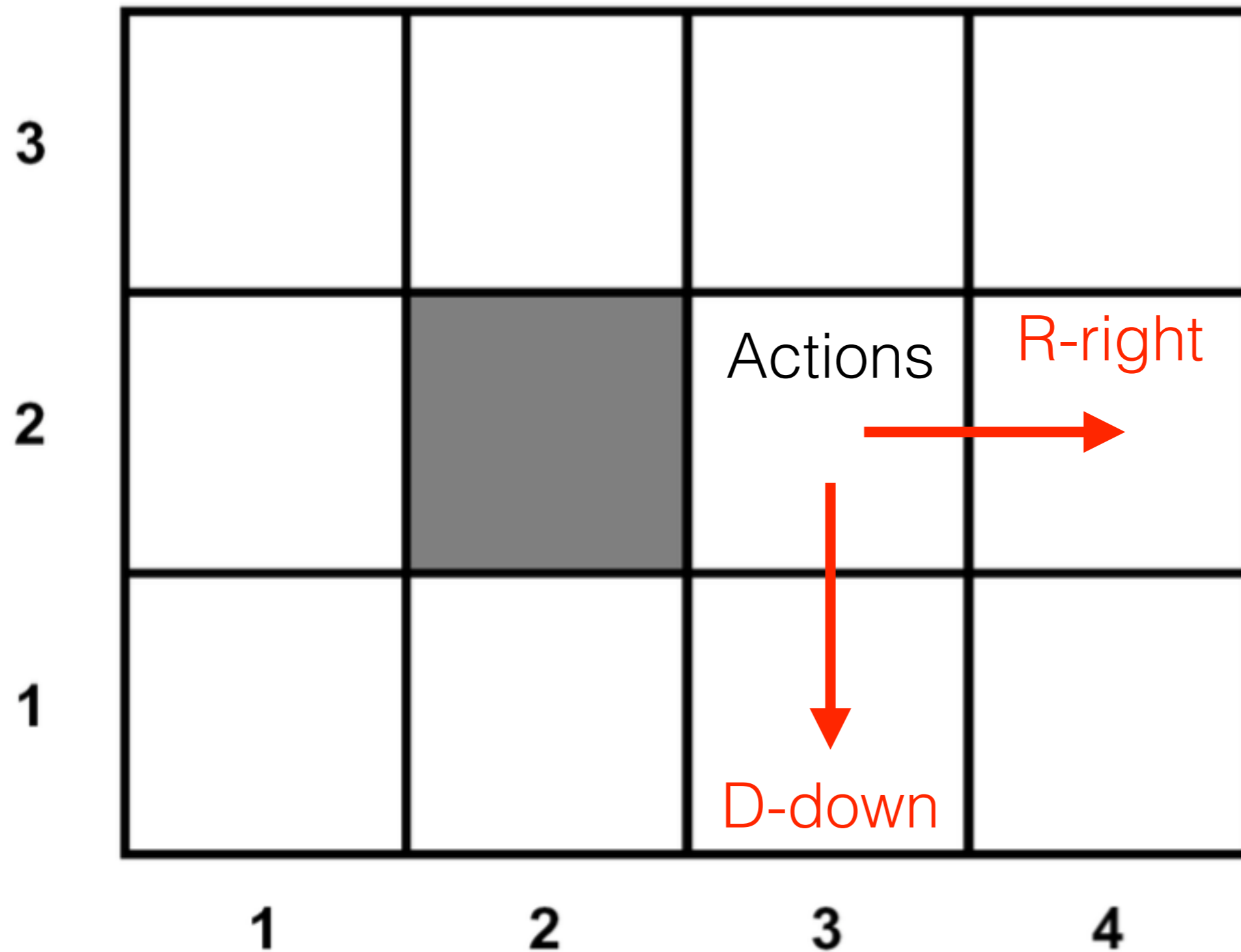
$\pi^* = \arg \max_a Q(\mathbf{x}, \mathbf{a})$



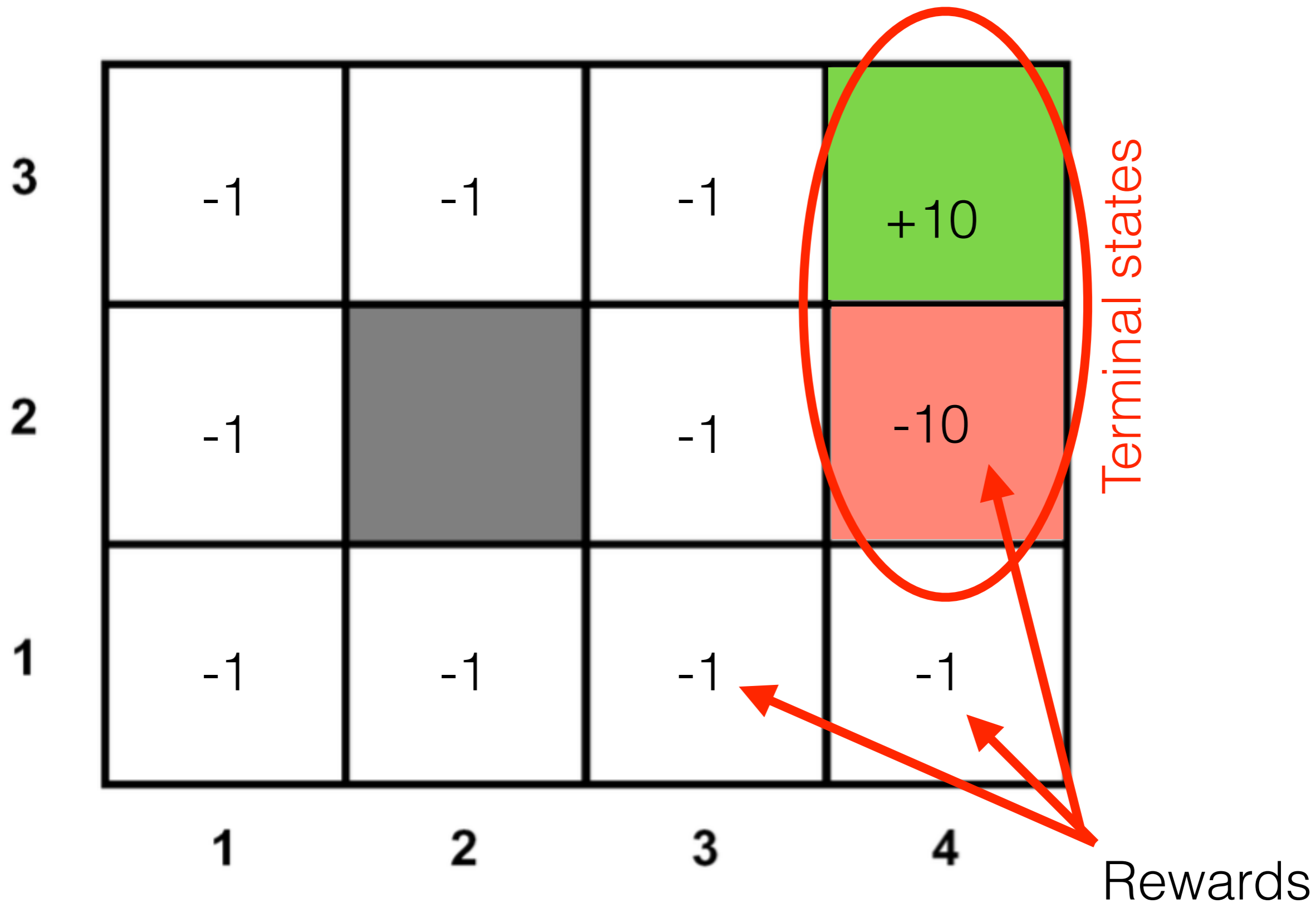
# Value-based methods: Q-learning



# Value-based methods: Q-learning



# Value-based methods: Q-learning



	a	b	c
		d	e

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



	a	b	c
		d	e

## 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
- Once we find it, we can control optimally as follows:

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

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



	a	b	c
		d	e

## State-action value function

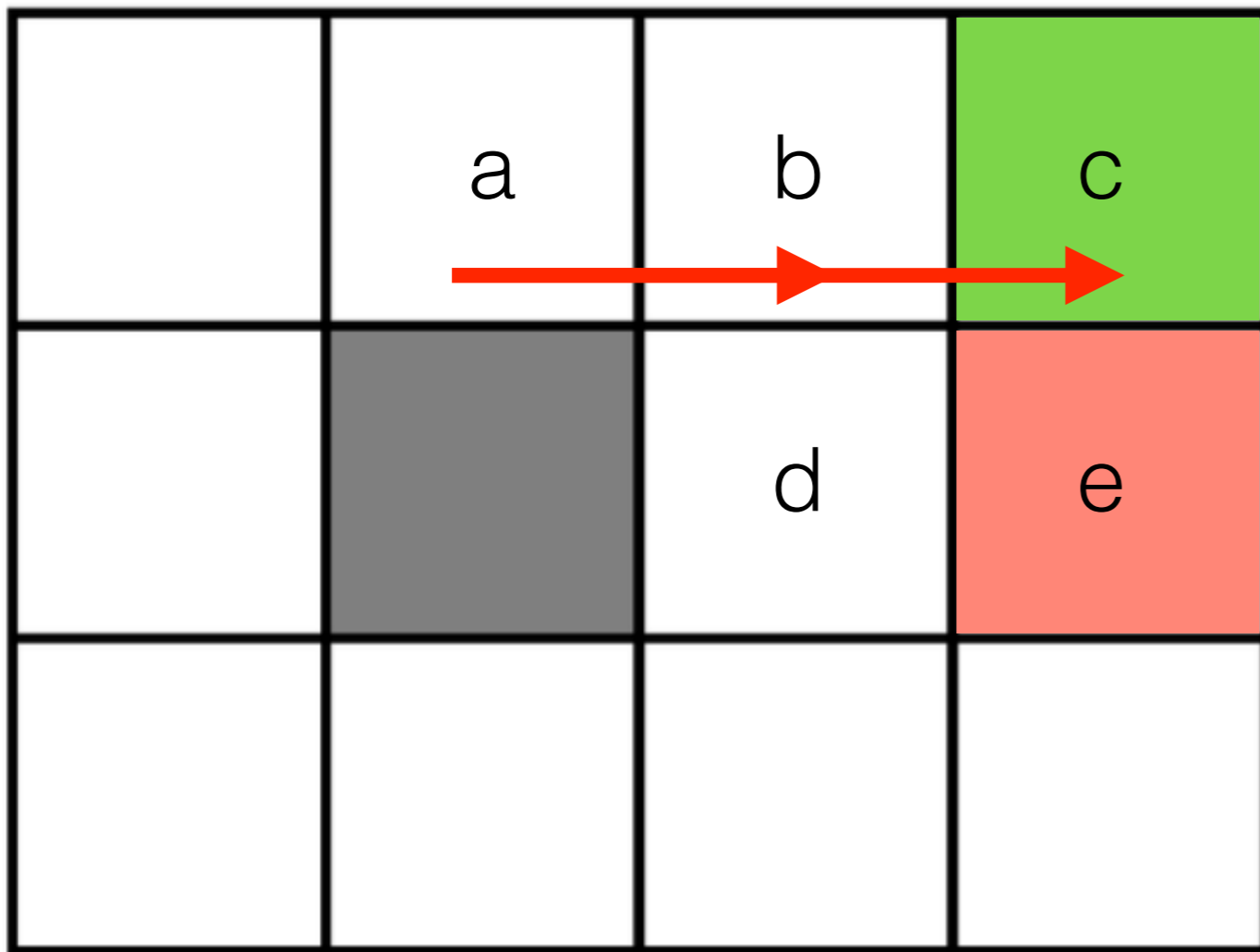
$$Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$$

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:
 
$$\pi^*(\mathbf{x}) = \arg \max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \arg \max_{\pi} J_{\pi}$$
- Search without model is based on collecting trajectories



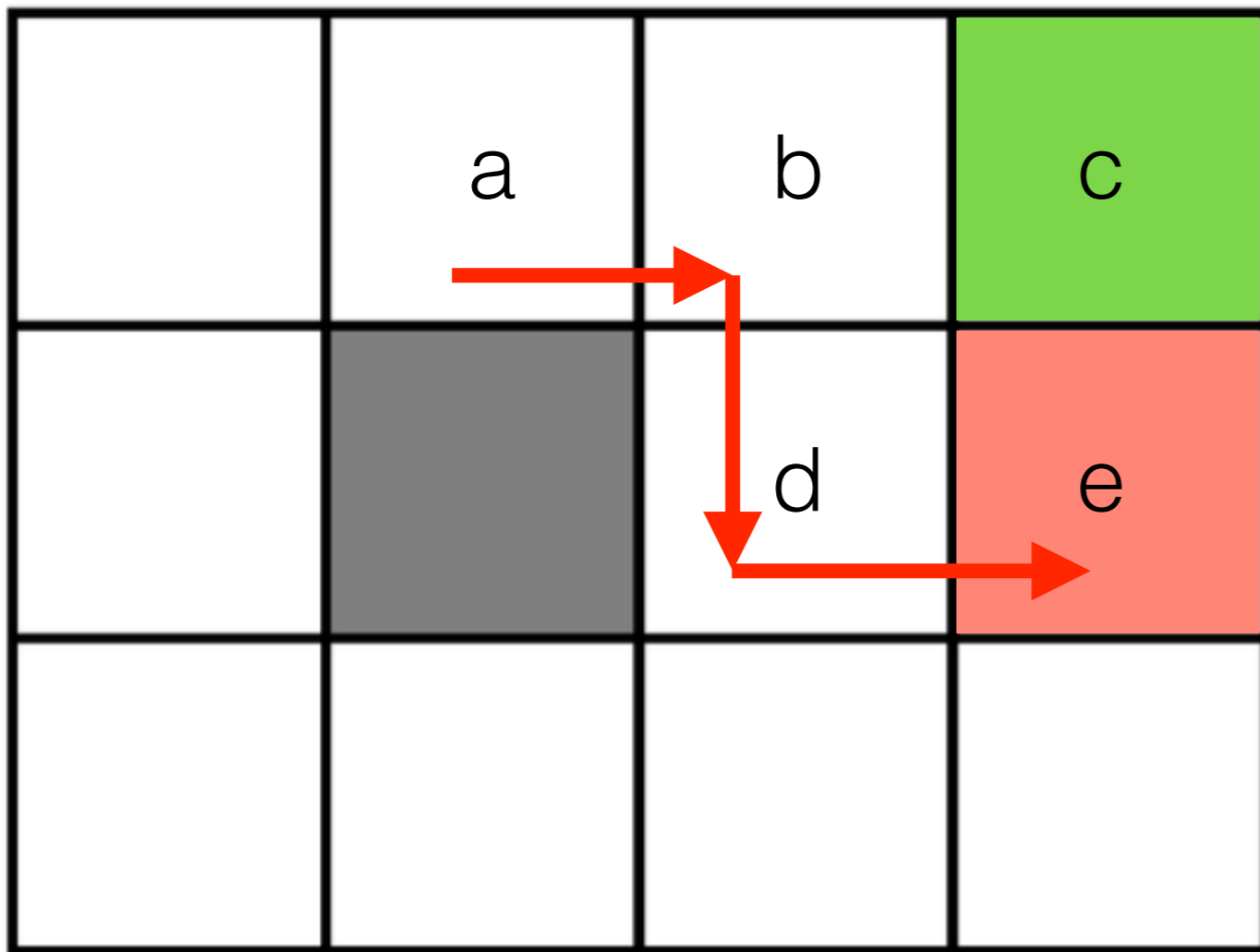




$\tau_1 :$   
 $(a, R, -1), (b, R, -1), (c, R, 10)$

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





$\tau_2 :$   
 $(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

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

Having a trajectory, each transition gives one equation



	a	b	c
		d	e

$\tau_2 :$

$(a, R, -1), (b, D, -1),$

$(d, R, -1), (e, R, -10)$



$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

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

Having a trajectory, each transition gives one equation



	a	b	c
		d	e

$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

Having a trajectory, each transition gives one equation



	a	b	c
		d	e

$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = r(e)$$

$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

Having a trajectory, each transition gives one equation



	a	b	c
		d	e

$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = r(e)$$

$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

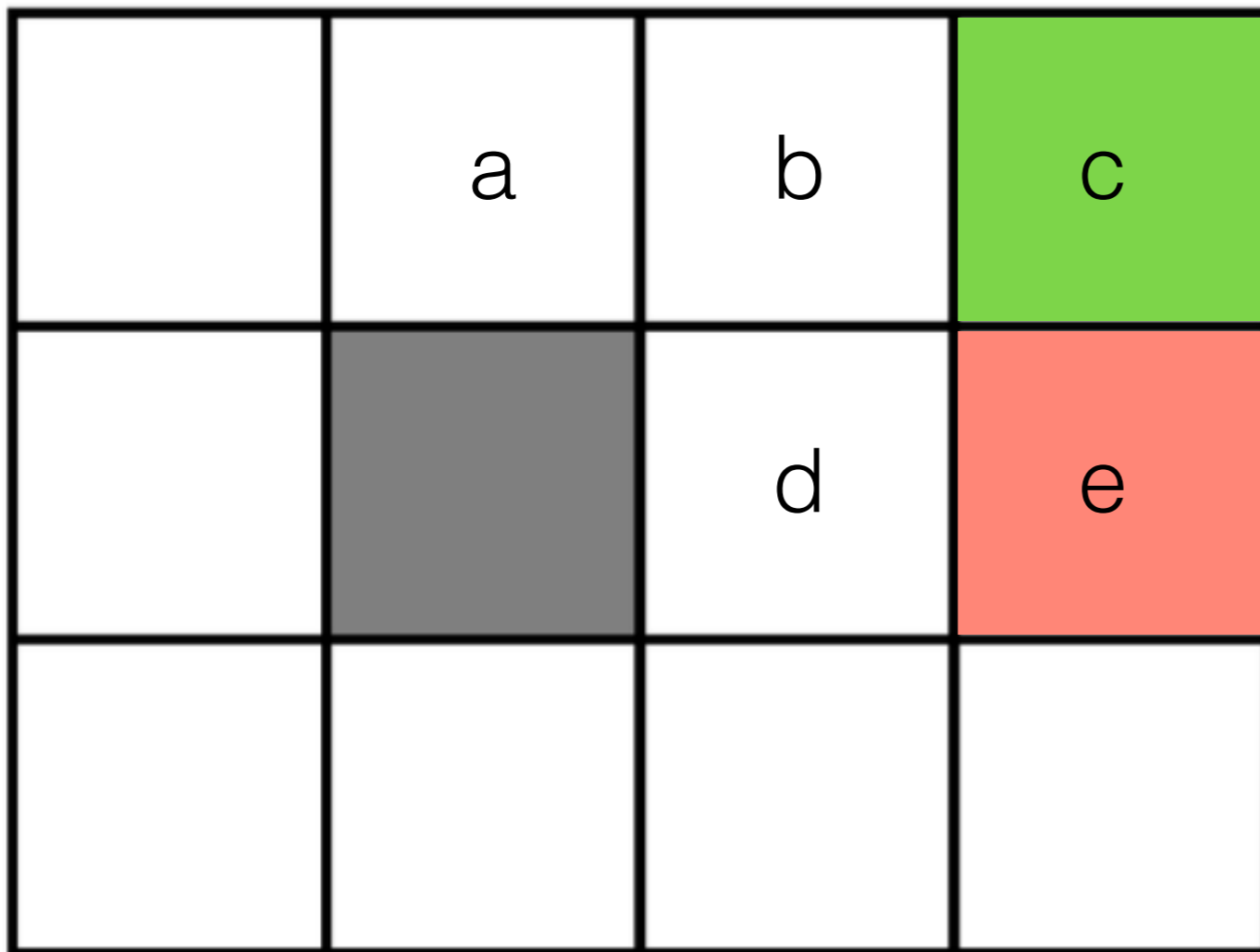
$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

Having a trajectory, each transition gives one equation





$\tau_2 :$   
 $(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = r(e)$$

$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

 unknowns

Having a trajectory, each transition gives one equation



	a	b	c
		d	e

$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = r(e)$$

$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

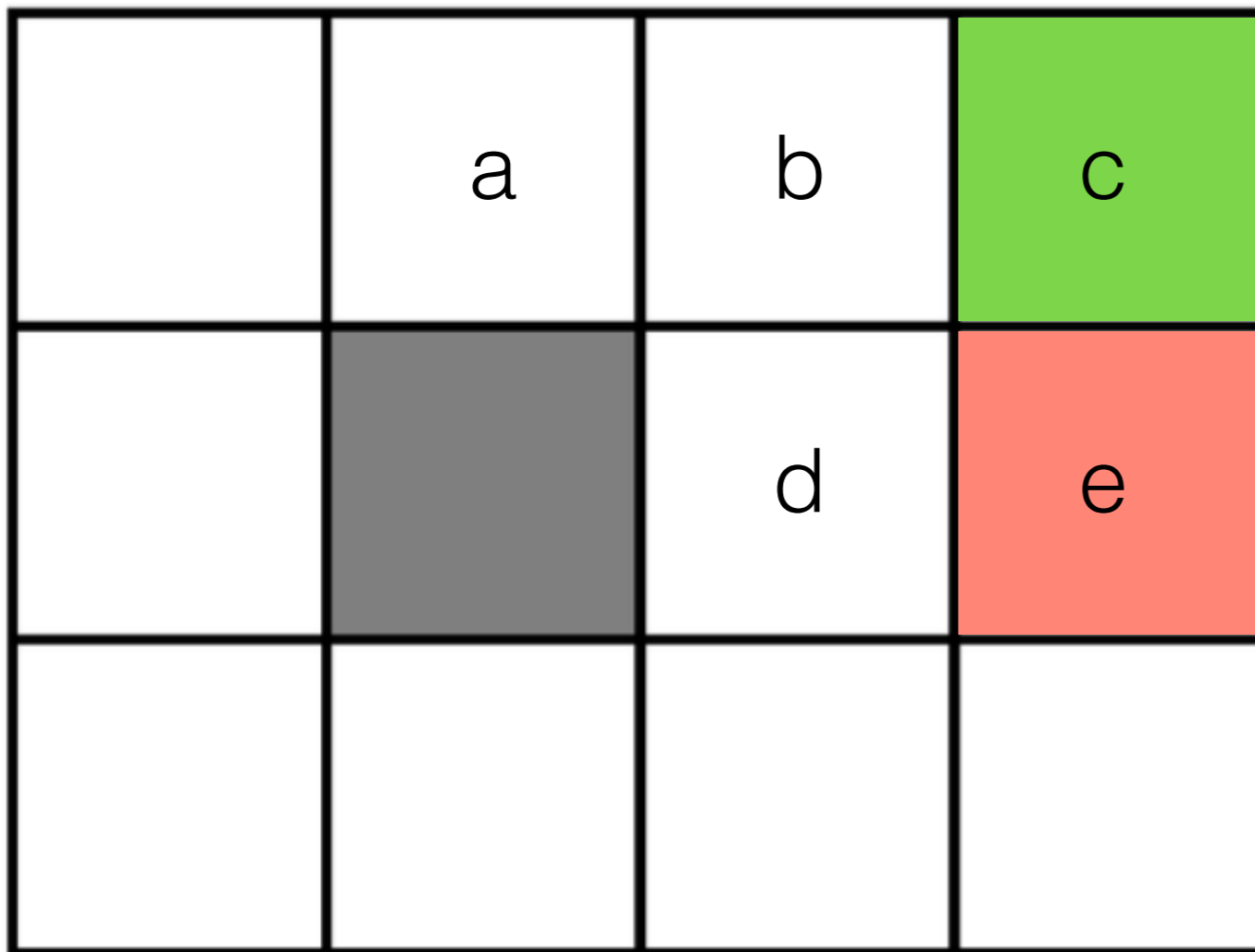
$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

(1) Substitute transitions and current Q-values to the right side and solve for left side.







$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = -10$$

$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

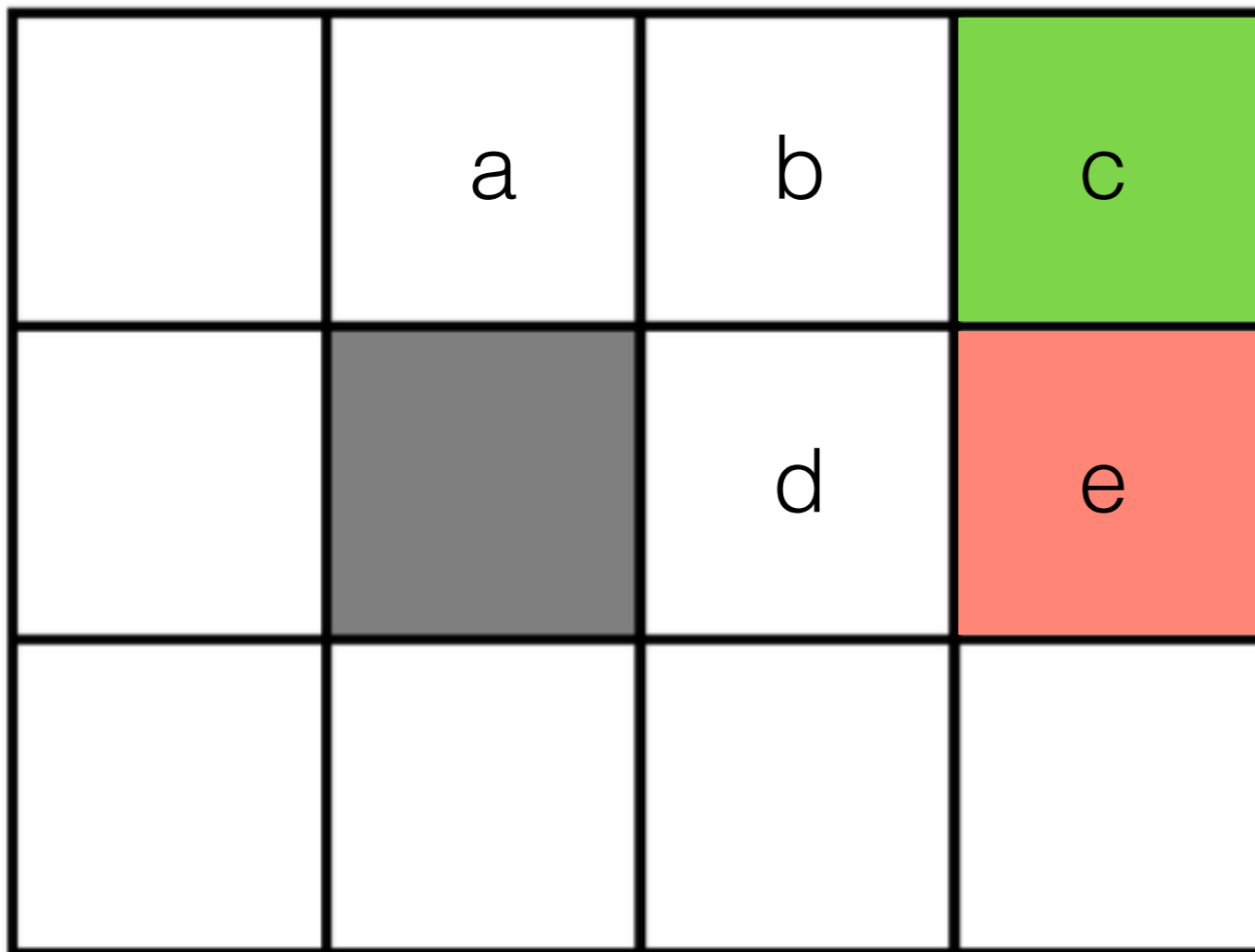
$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

(1) Substitute transitions and current Q-values to the right side and solve for left side.





$\tau_2 :$

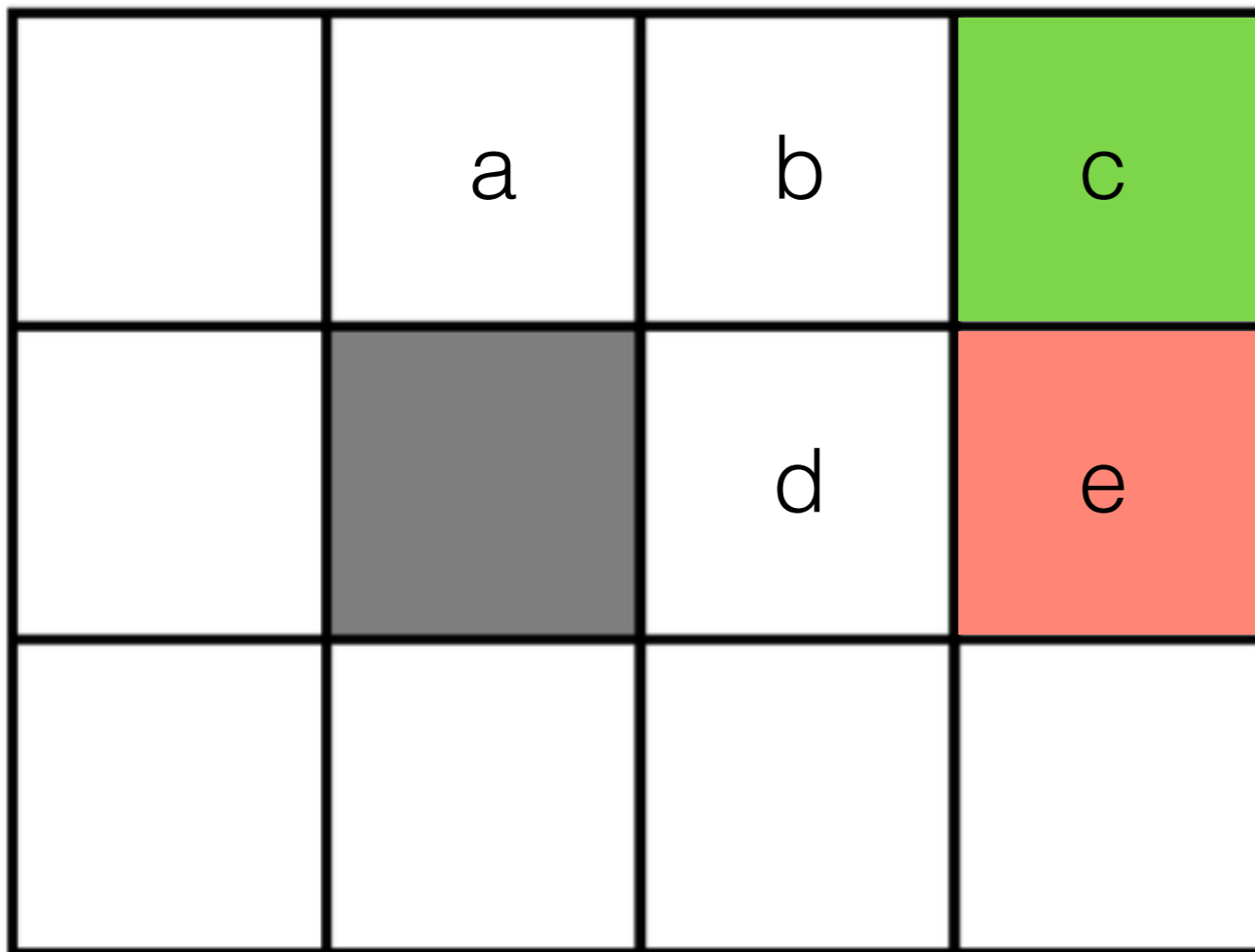
$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$Q(e, R)$	=	-10
$Q(b, D)$	=	???
$Q(d, R)$	=	$r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$
$Q(a, R)$	=	$r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$

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

(1) Substitute transitions and current Q-values to the right side and solve for left side.





$\tau_2 :$

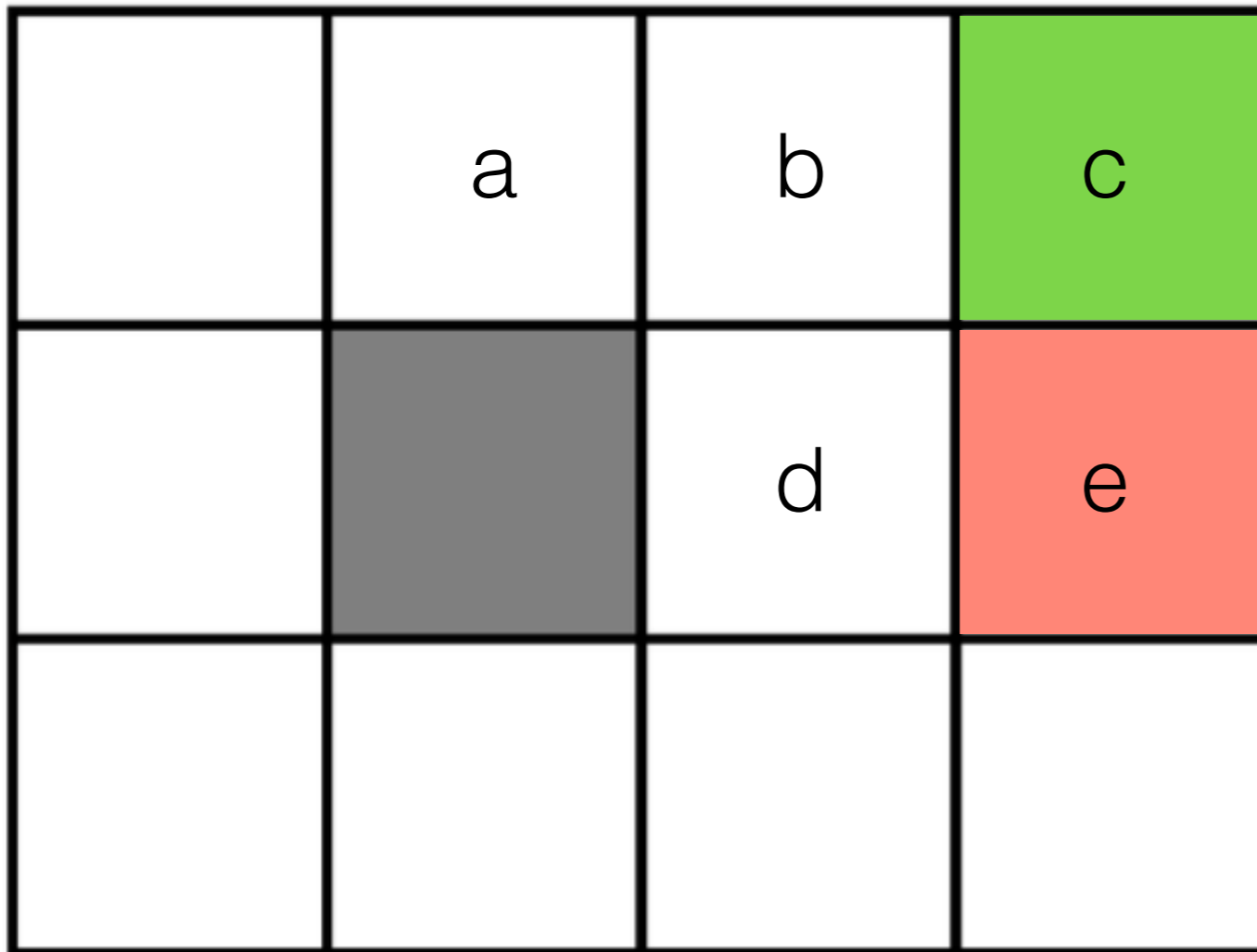
$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$Q(e, R)$	=	-10
$Q(b, D)$	=	-1
$Q(d, R)$	=	$r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$
$Q(a, R)$	=	$r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$

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

(1) Substitute transitions and current Q-values to the right side and solve for left side.





$\tau_2 :$

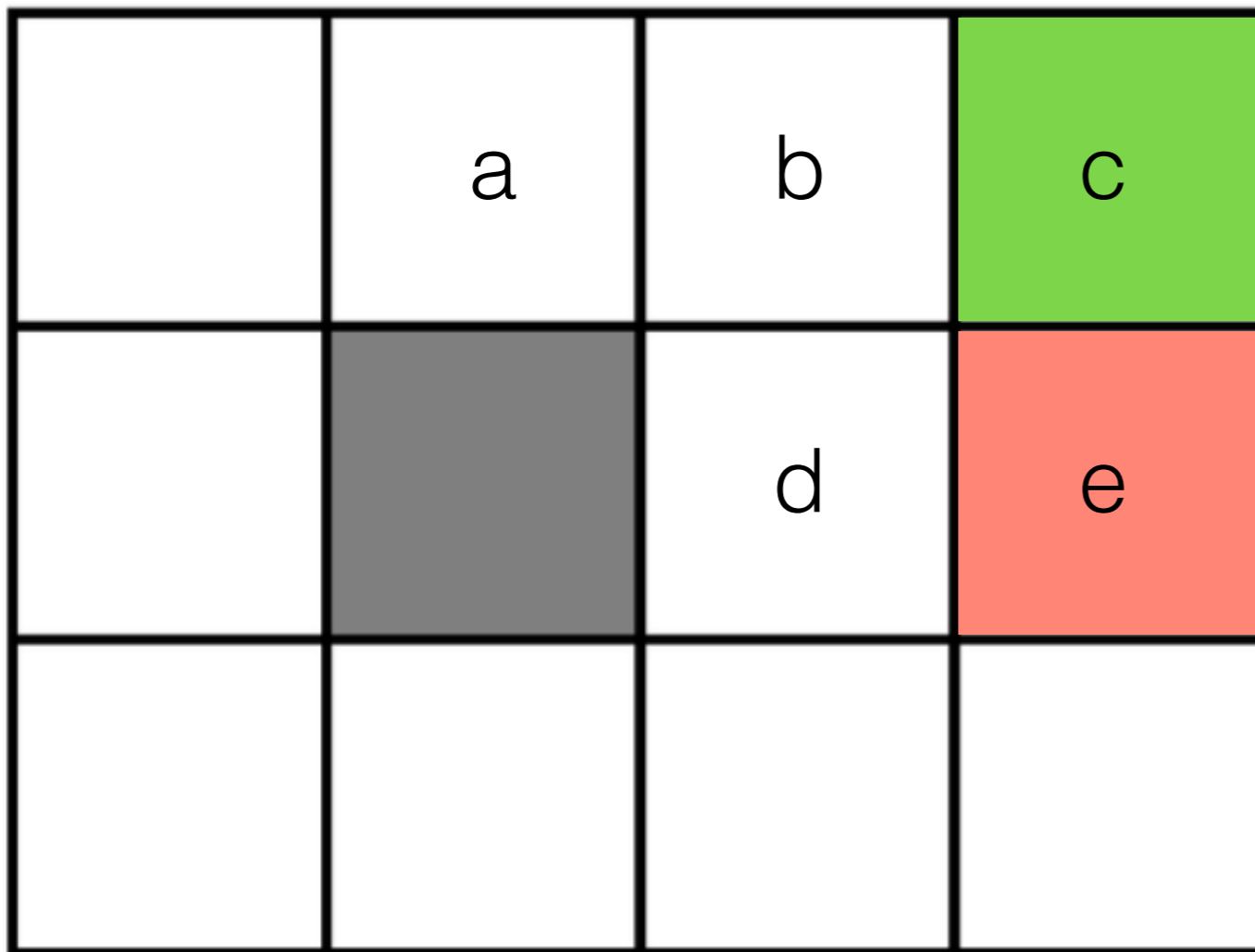
$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$Q(e, R)$	=	-10
$Q(b, D)$	=	-1
$Q(d, R)$	=	-1
$Q(a, R)$	=	$r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$

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

(1) Substitute transitions and current Q-values to the right side and solve for left side.





$\tau_2 :$

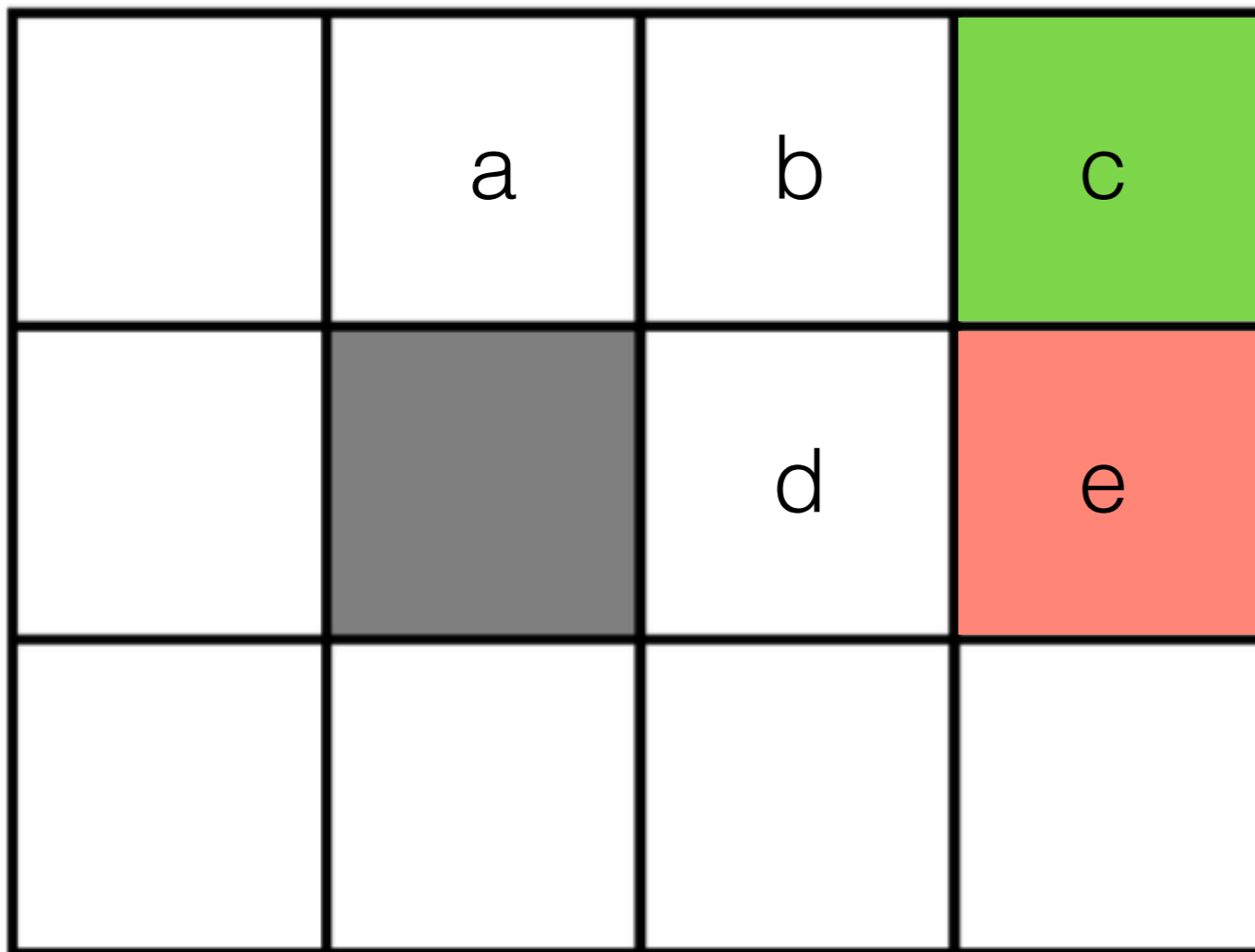
$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$Q(e, R)$	=	-10
$Q(b, D)$	=	-1
$Q(d, R)$	=	-1
$Q(a, R)$	=	-1

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

(1) Substitute transitions and current Q-values to the right side and solve for left side.





$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = r(e)$$

$$Q(b, D) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

- (1) Substitute transitions and current Q-values to the right side and solve for left side.  
 (2) Repeat several times



	a	b	c
		d	e

$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = r(e)$$

$$Q(b, R) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

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

(1) Substitute transitions and current Q-values to the right side and solve for left side.  
 (2) Repeat several times (search for the fixed point of the Bellman operator)

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



	a	b	c
		d	e

$\tau_2 :$

$(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

$$Q(e, R) = r(e)$$

$$Q(b, R) = r(b) + \max_{\mathbf{u}} Q(d, \mathbf{u})$$

$$Q(d, R) = r(d) + \max_{\mathbf{u}} Q(e, \mathbf{u})$$

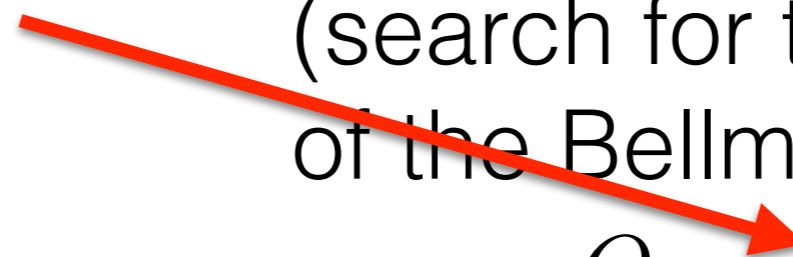
$$Q(a, R) = r(a) + \max_{\mathbf{u}} Q(b, \mathbf{u})$$

**Iterations of the Bellman operator converge to a fixed point !!!**

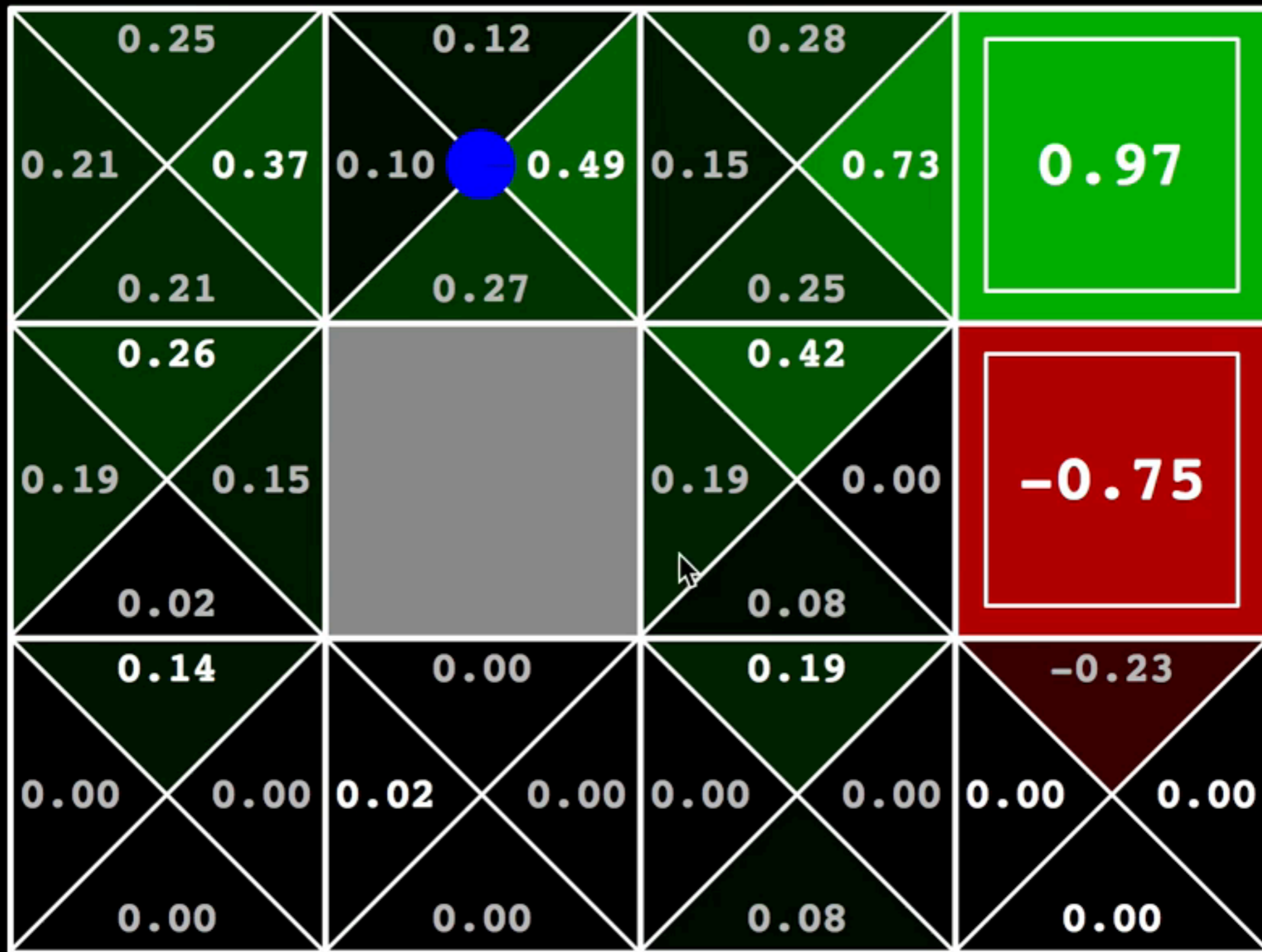
(1) Substitute transitions and current Q-values to the right side and solve for left side.

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

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







## CURRENT Q-VALUES



# Bellman equation

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

reward for transition

the best you can do from  
the following state

Which path is better?



# Bellman equation

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

reward for transition

the best you can do from  
the following state

discount factor  $\gamma \in [0; 1]$

0	0	0	0	0	0	0
0						0
+1	0	0	0	0	0	0



# Q-learning

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$
2. Solve  $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
3. Repeat from 1



# Q-learning

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$
  2. Solve  $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
  3. Repeat from 1
- Curse of dimensionality



# Q-learning

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$
  2. Solve  $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
  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 trajectories  $\tau_1, \tau_2, \tau_3, \dots$
  2. Solve  $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
  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

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$ , initialize  $\theta = \text{rand}$
2. Estimate  $\mathbf{y} = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
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 2
5. Repeat from 1



# Q-learning

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$
  2. Solve  $Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$
  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

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$ , initialize  $\theta = \text{rand}$
2. Estimate  $\mathbf{y} = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning

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

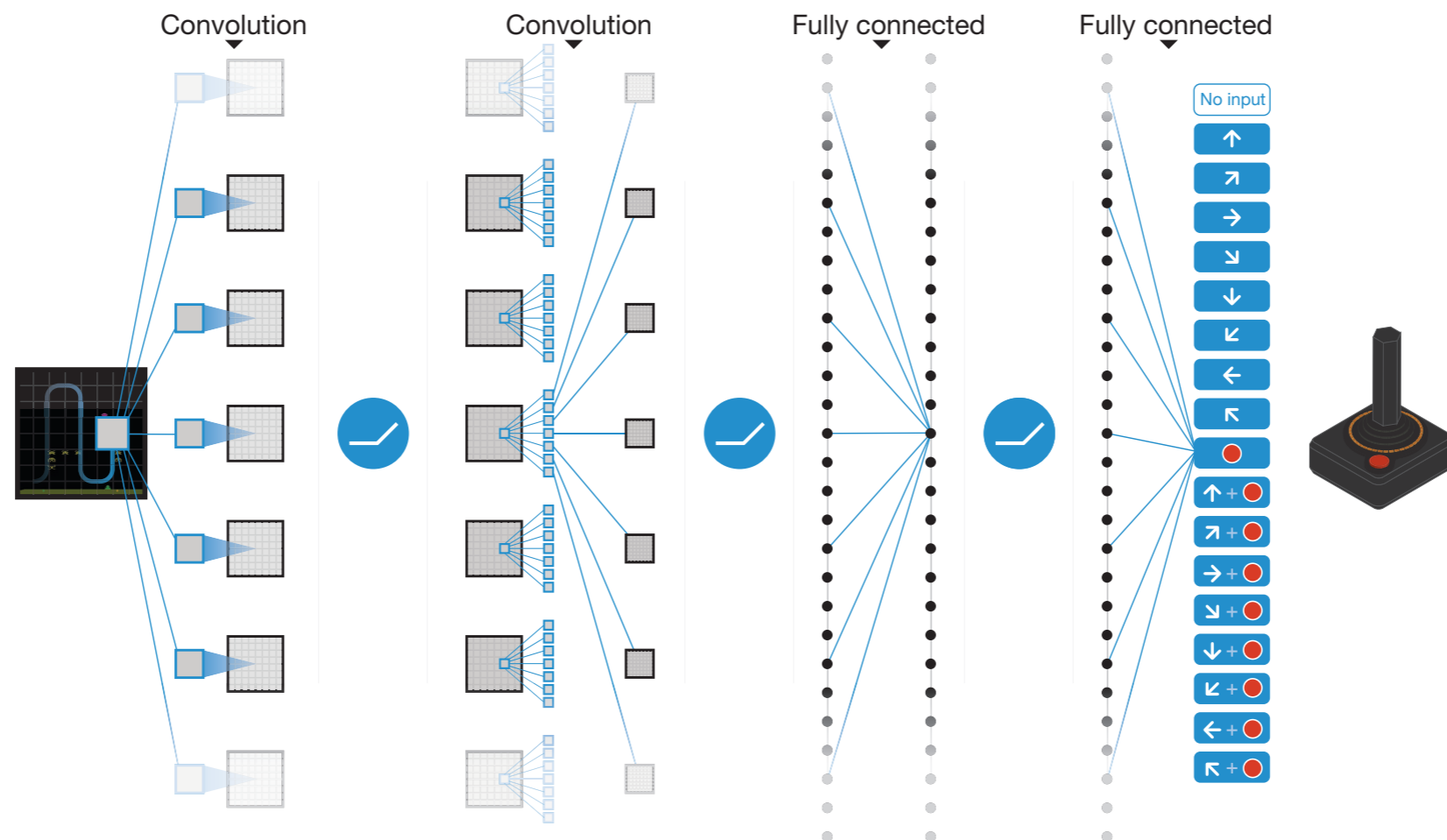
4. Repeat from 2
  5. Repeat from 1
- Approximated Q-learning does not have to converge to a fixed-point !!!**





# Mnih et al. Nature 2015

- 2600 atari games
- **state space:** pixels (e.g. VGA resolution)
- **action space:** discrete joystic actions (8 direction + 8 direction with button + neutral action)
- replay buffer (decorrelates samples to be “more i.i.d”)
- two Q-networks (suppress oscillations)

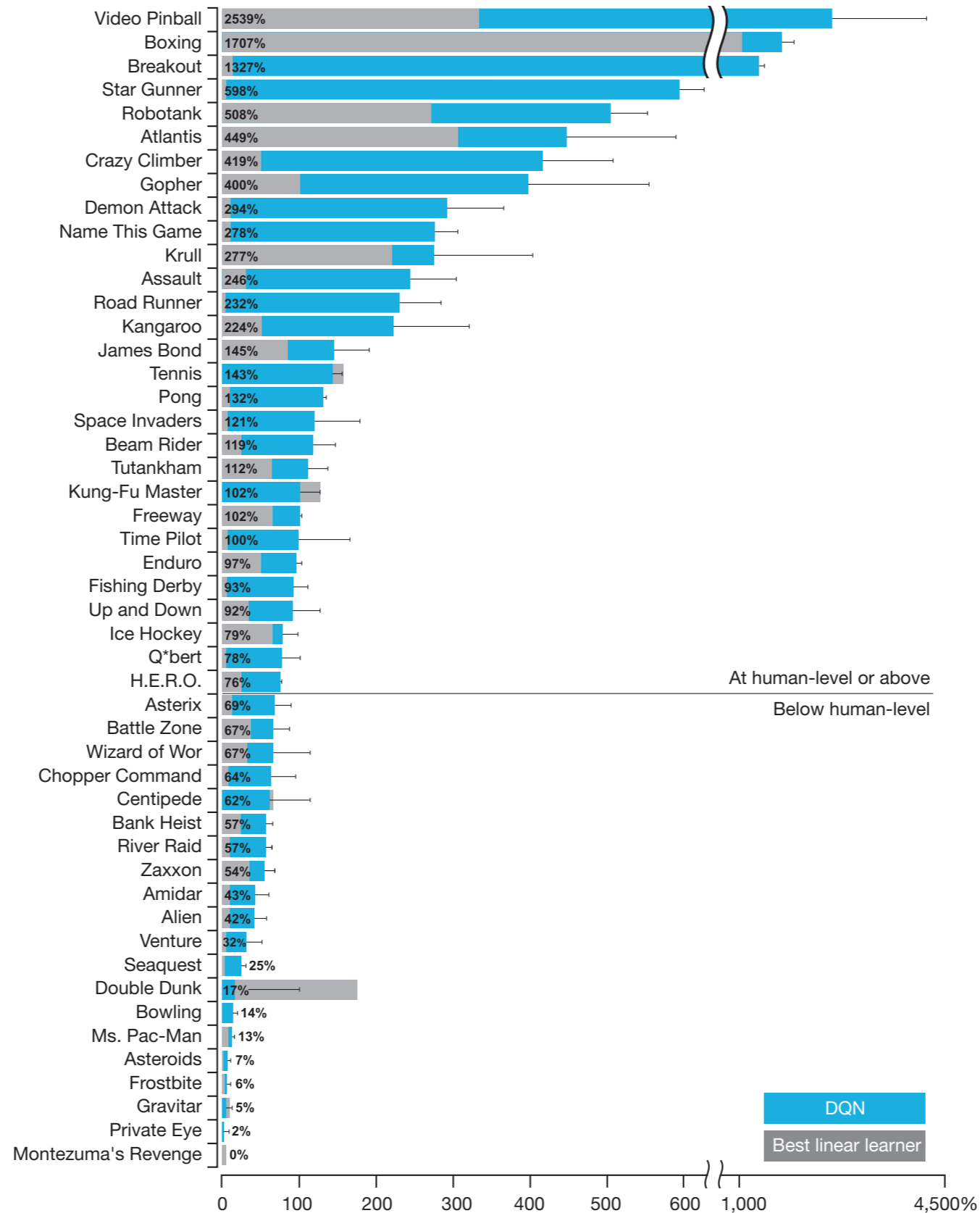


## Mnih et al. Nature 2015

- 2600 atari games
- **state space:** pixels (e.g. VGA resolution)
- **action space:** discrete joystic actions (8 directions + 8 directions with button)
- collection of control tasks: <https://gym.openai.com>

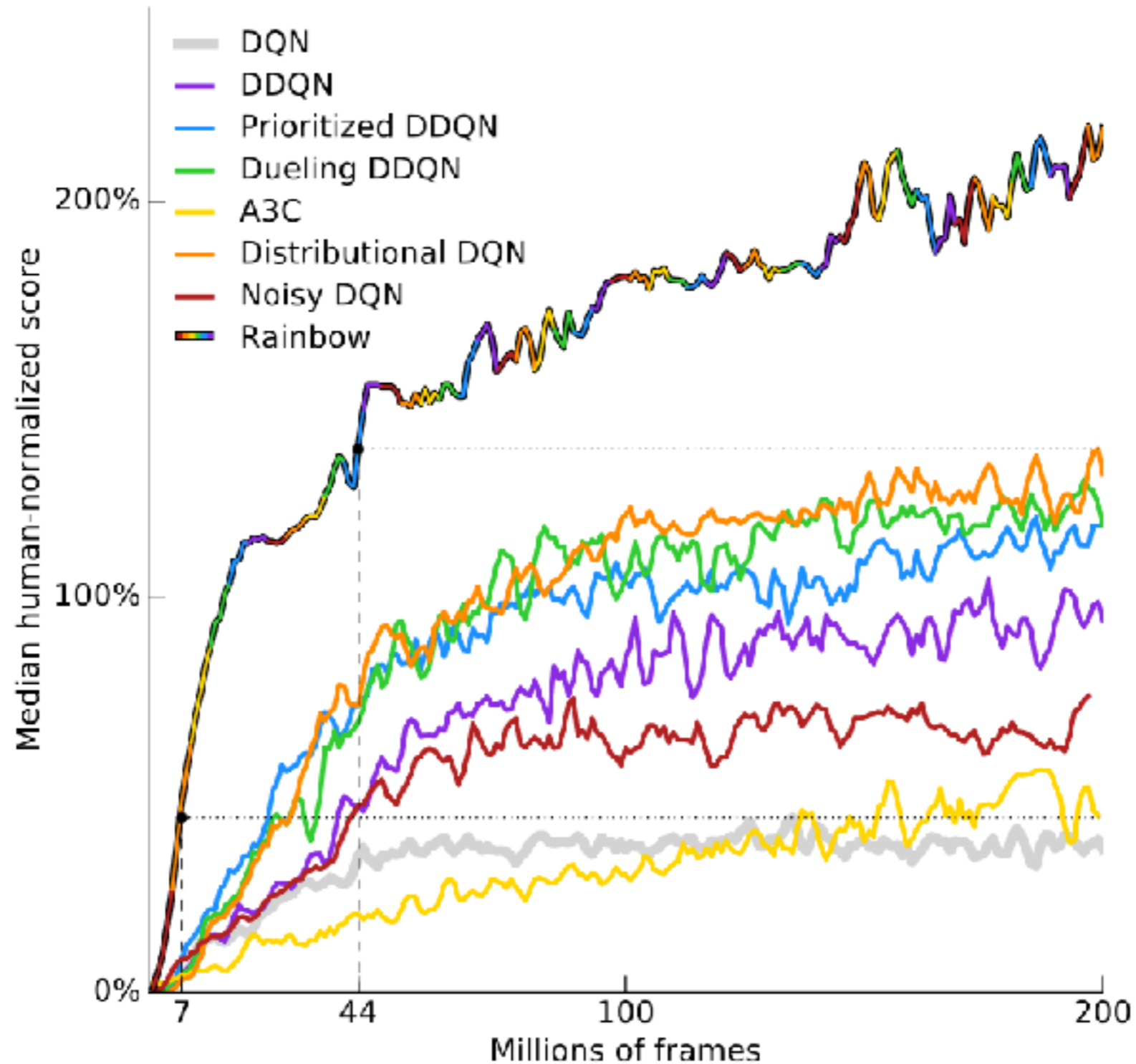


# Mnih et al. Nature 2015



# Hessel et. al Rainbow DQN, 2017

Average of different estimates helps a lot



# Value function

- If model available it is often better to train the state-value function.



	a	b	c
		d	e

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

$\tau_2 :$   
 $(a, R, -1), (b, D, -1),$   
 $(d, R, -1), (e, R, -10)$

Return of a trajectory starting from the state  $x$ :

$$G = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

$$\tau_1 : G_1 = (-1) + \gamma(-1) + \gamma^2 10$$

$$\tau_2 : G_2 = (-1) + \gamma(-1) + \gamma^2(-1) + \gamma^3(-10)$$

$$V(\mathbf{x}) = \mathbb{E}_{G \sim \pi} [G] \approx \frac{1}{N} \sum_i G_i$$



$$V(\mathbf{x}) = \mathbb{E}_{G \sim \pi}[G] \approx \frac{1}{N} \sum_i G_i$$

Such estimate has high variance =>  
re-use older estimates of  $V(\mathbf{x})$  and estimate exponentially  
weighting average



$$V(\mathbf{x}) = \mathbb{E}_{G \sim \pi}[G] \approx \frac{1}{N} \sum_i G_i$$

Such estimate has high variance =>  
re-use older estimates of  $V(\mathbf{x})$  and estimate exponentially  
weighting average

$$V(\mathbf{x}) \approx (1 - \alpha)V(\mathbf{x}) + \alpha G_i = V(\mathbf{x}) + \alpha(G_i - V(\mathbf{x}))$$





$$V(\mathbf{x}) = \mathbb{E}_{G \sim \pi}[G] \approx \frac{1}{N} \sum_i G_i$$

Such estimate has high variance =>  
re-use older estimates of  $V(\mathbf{x})$  and estimate exponentially  
weighting average

$$V(\mathbf{x}) \approx (1 - \alpha)V(\mathbf{x}) + \alpha G_i = V(\mathbf{x}) + \alpha(G_i - V(\mathbf{x}))$$

Such estimate has smaller variance but is still bad =>

$$G^{(\infty)} = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

⋮

$$G^{(2)} = r_1 + \gamma r_2 + \gamma^2 V(\mathbf{x}_3)$$

$$G^{(1)} = r_1 + \gamma V(\mathbf{x}_2)$$



$$V(\mathbf{x}) = \mathbb{E}_{G \sim \pi}[G] \approx \frac{1}{N} \sum_i G_i$$

Such estimate has high variance =>  
re-use older estimates of  $V(\mathbf{x})$  and estimate exponentially  
weighting average

$$V(\mathbf{x}) \approx (1 - \alpha)V(\mathbf{x}) + \alpha G_i = V(\mathbf{x}) + \alpha(G_i - V(\mathbf{x}))$$

Such estimate has smaller variance but is still bad =>

$$G^{(\infty)} = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

MC estimate:  
high variance, no bias

⋮

$$G^{(2)} = r_1 + \gamma r_2 + \gamma^2 V(\mathbf{x}_3)$$

$$G^{(1)} = r_1 + \gamma V(\mathbf{x}_2)$$



TD estimate:  
small variance,  
strong bias



$$G^{(\infty)} = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

$\vdots$

$$G^{(2)} = r_1 + \gamma r_2 + \gamma^2 V(\mathbf{x}_3)$$

$$G^{(1)} = r_1 + \gamma V(\mathbf{x}_2)$$

Convex combination of all possible return estimates

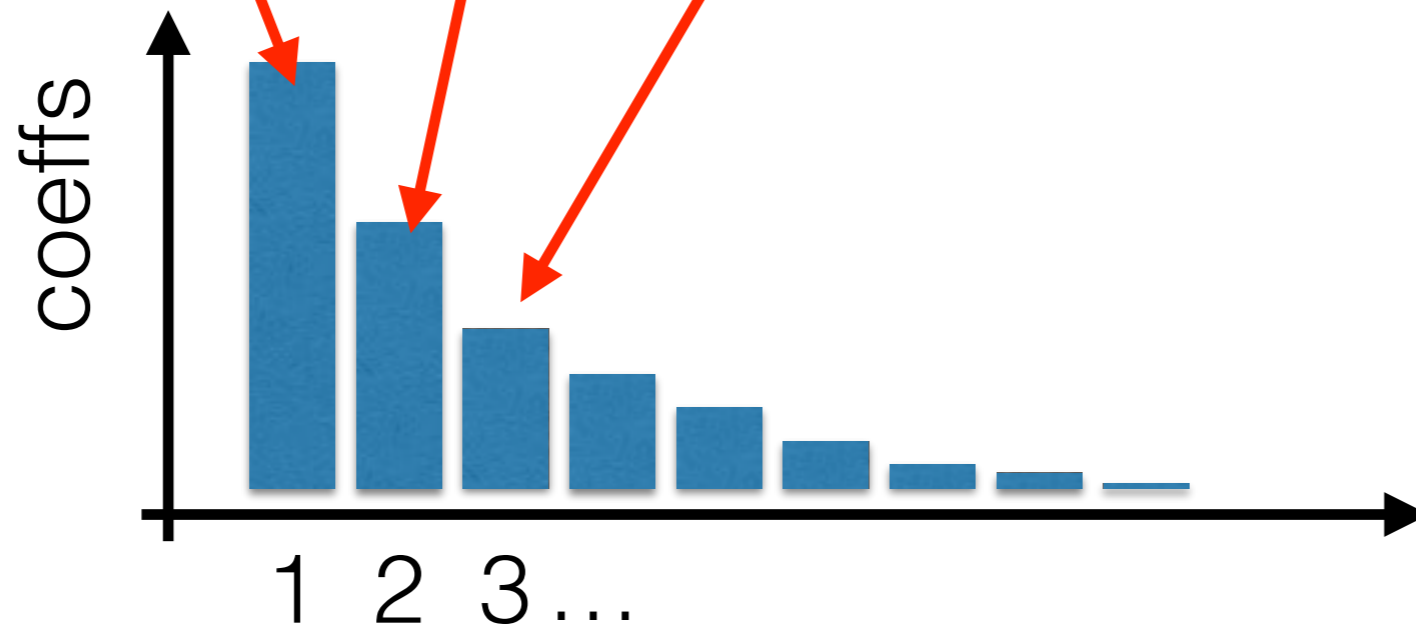
$$\begin{aligned} G^\lambda &= (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G^{(n)} = \\ &= (1 - \lambda) G^{(1)} + (1 - \lambda) \lambda G^{(2)} + (1 - \lambda) \lambda^2 G^{(3)} + \dots \end{aligned}$$

coeffs sums to 1



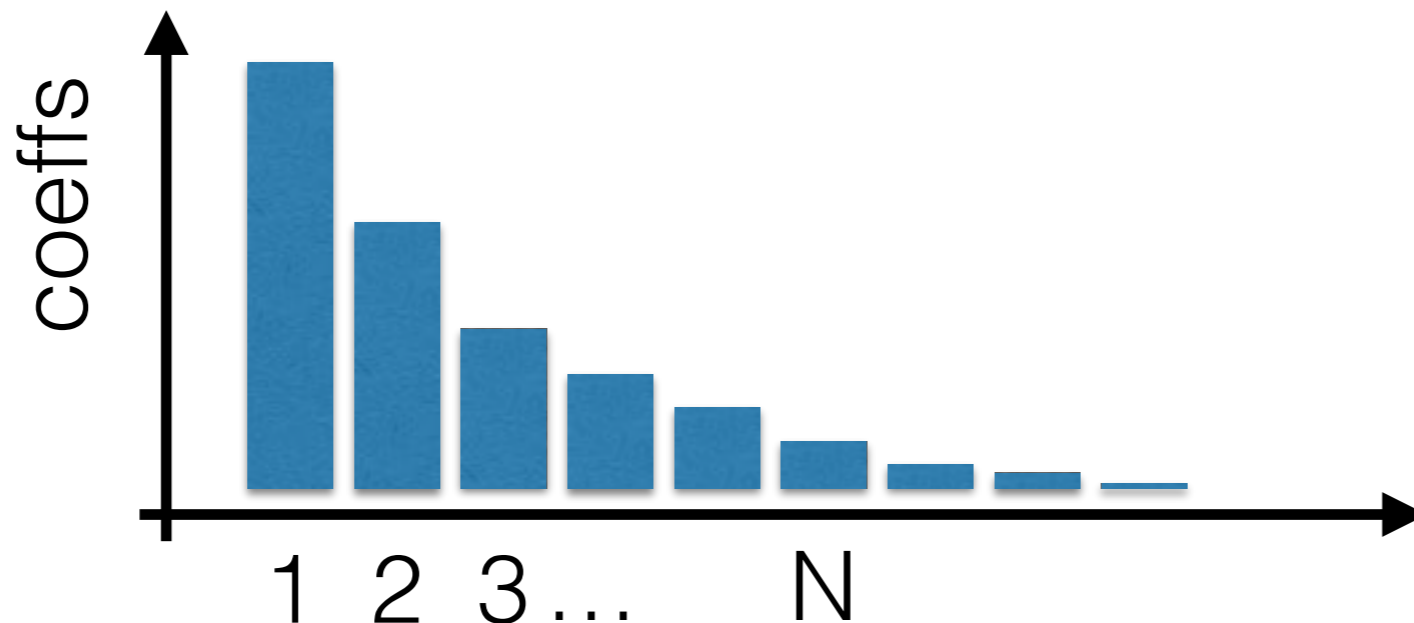
$$G^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G^{(n)} =$$

$$= (1 - \lambda) G^{(1)} + (1 - \lambda) \lambda G^{(2)} + (1 - \lambda) \lambda^2 G^{(3)} + \dots$$



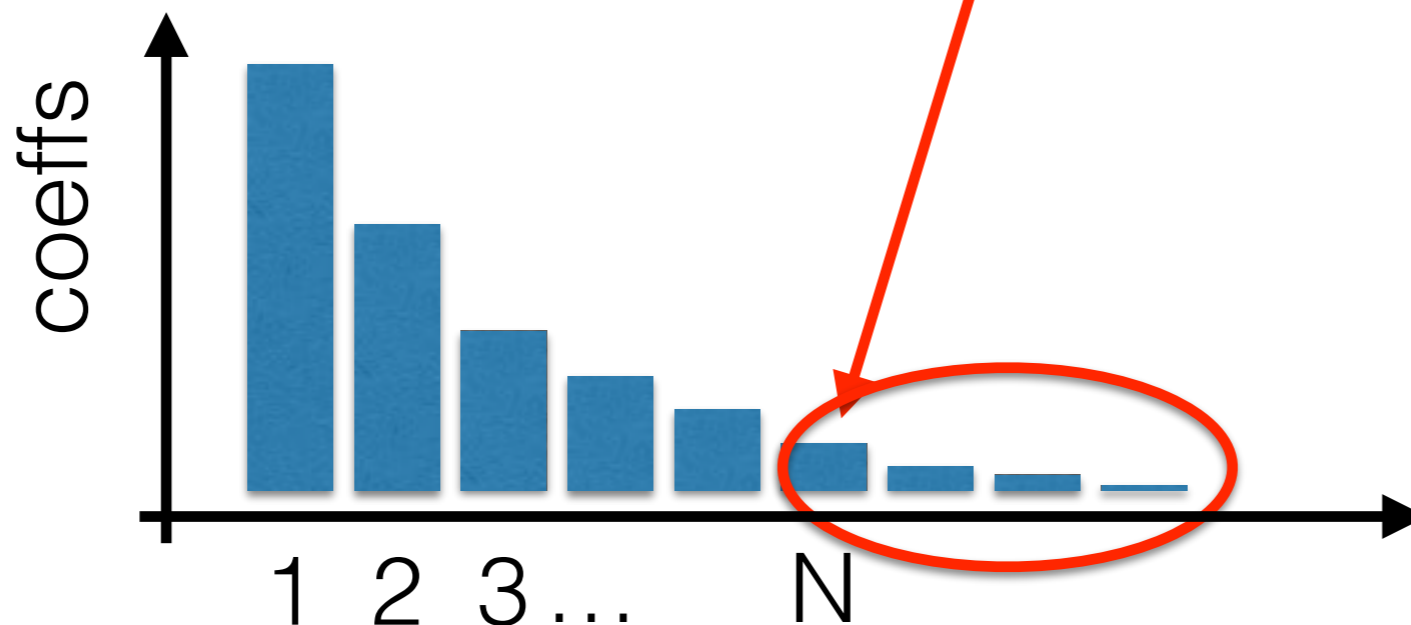
$$\begin{aligned}
G^\lambda &= (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G^{(n)} = \\
&= (1 - \lambda)G^{(1)} + (1 - \lambda)\lambda G^{(2)} + (1 - \lambda)\lambda^2 G^{(3)} + \dots \\
&\quad \dots (1 - \lambda)\lambda^{(N-2)} G^{(N-1)}
\end{aligned}$$

- In reality, sequences have finite length



$$\begin{aligned}
G^\lambda &= (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G^{(n)} = \\
&= (1 - \lambda)G^{(1)} + (1 - \lambda)\lambda G^{(2)} + (1 - \lambda)\lambda^2 G^{(3)} + \dots \\
&\quad \dots (1 - \lambda)\lambda^{(N-2)} G^{(N-1)} + \lambda^{(N-1)} G^{(N)}
\end{aligned}$$

- In reality, sequences have finite length
- Last coeff sums up all coeffs from N to infinity.



$$\begin{aligned}
G^\lambda &= (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G^{(n)} = \\
&= (1 - \lambda)G^{(1)} + (1 - \lambda)\lambda G^{(2)} + (1 - \lambda)\lambda^2 G^{(3)} + \dots \\
&\quad \dots (1 - \lambda)\lambda^{(N-2)} G^{(N-1)} + \lambda^{(N-1)} G^{(N)}
\end{aligned}$$

$TD(\lambda)$  learning algorithm

1. collect trajectories
2. for each state  $\mathbf{x}$  estimate  $G^\lambda$
3. Update state-value function:  
 $V(\mathbf{x}) = V(\mathbf{x}) + \alpha(G^\lambda - V(\mathbf{x}))$
4. repeat from 1



State value function  $V(\mathbf{x}_1)$  is approximated from traj. which

- started in  $\mathbf{x}_1$

$$G^{(\infty)} = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

⋮

$$G^{(2)} = r_1 + \gamma r_2 + \gamma^2 V(\mathbf{x}_3)$$

$$G^{(1)} = r_1 + \gamma V(\mathbf{x}_2)$$

Similarly state-action function  $Q(\mathbf{x}_1, \mathbf{u}_1)$  can be approximated but only from trajectories which

- started in  $\mathbf{x}_1$
- followed action  $\mathbf{u}_1$

$$\hat{Q}^{(\infty)} = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

⋮

$$\hat{Q}^{(1)} = r_1 + \gamma V(\mathbf{x}_2)$$

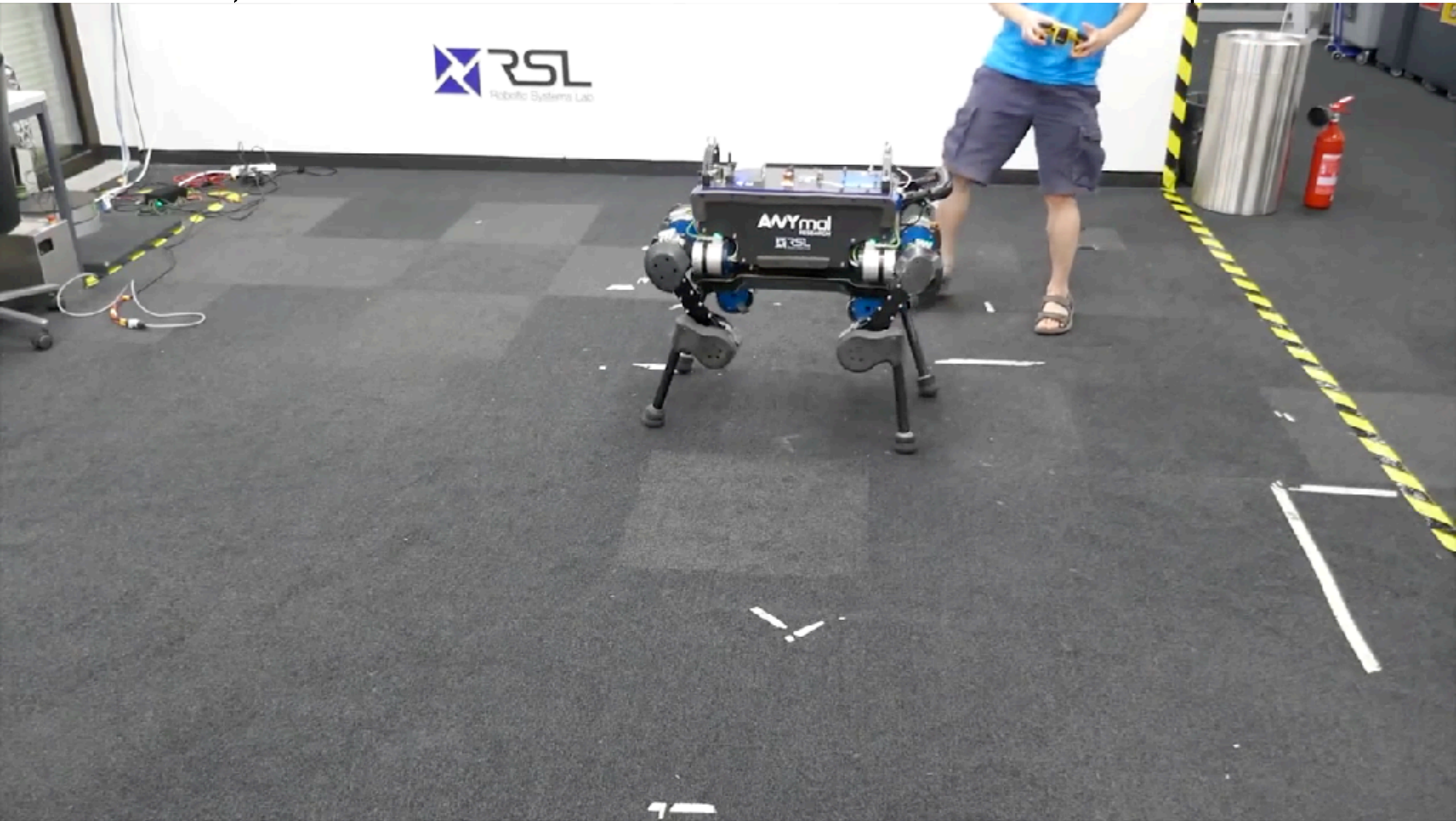




- Learning has been shown to be possible in simulation
- but can I use it on a real robot??
- !!! millions (or billions) of real-world trials are needed



If exteroceptive sensors are not used and terrain is trivial, then transfer from accurate simulation is possible



[Hwangbo, ETH Zurich, Science Robotics, 2018]

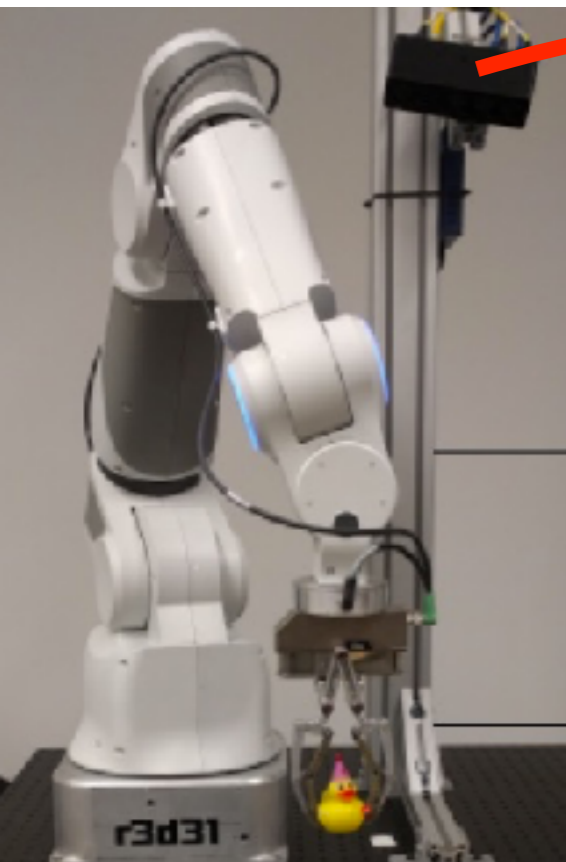


Czech Technical University in Prague  
Faculty of Electrical Engineering, Department of Cybernetics

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

Another option is to avoid simulation completely !!!

manipulator+ RGB camera

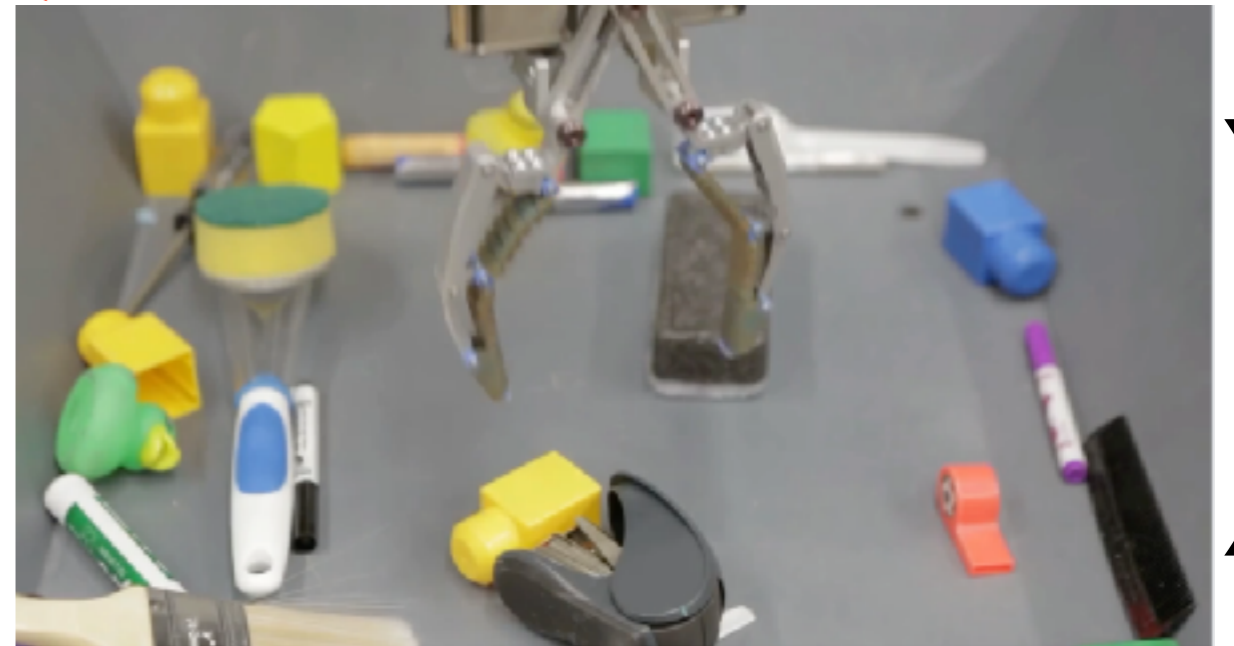


joint torques



*image*

$$= \pi_{\theta} \left( \right)$$

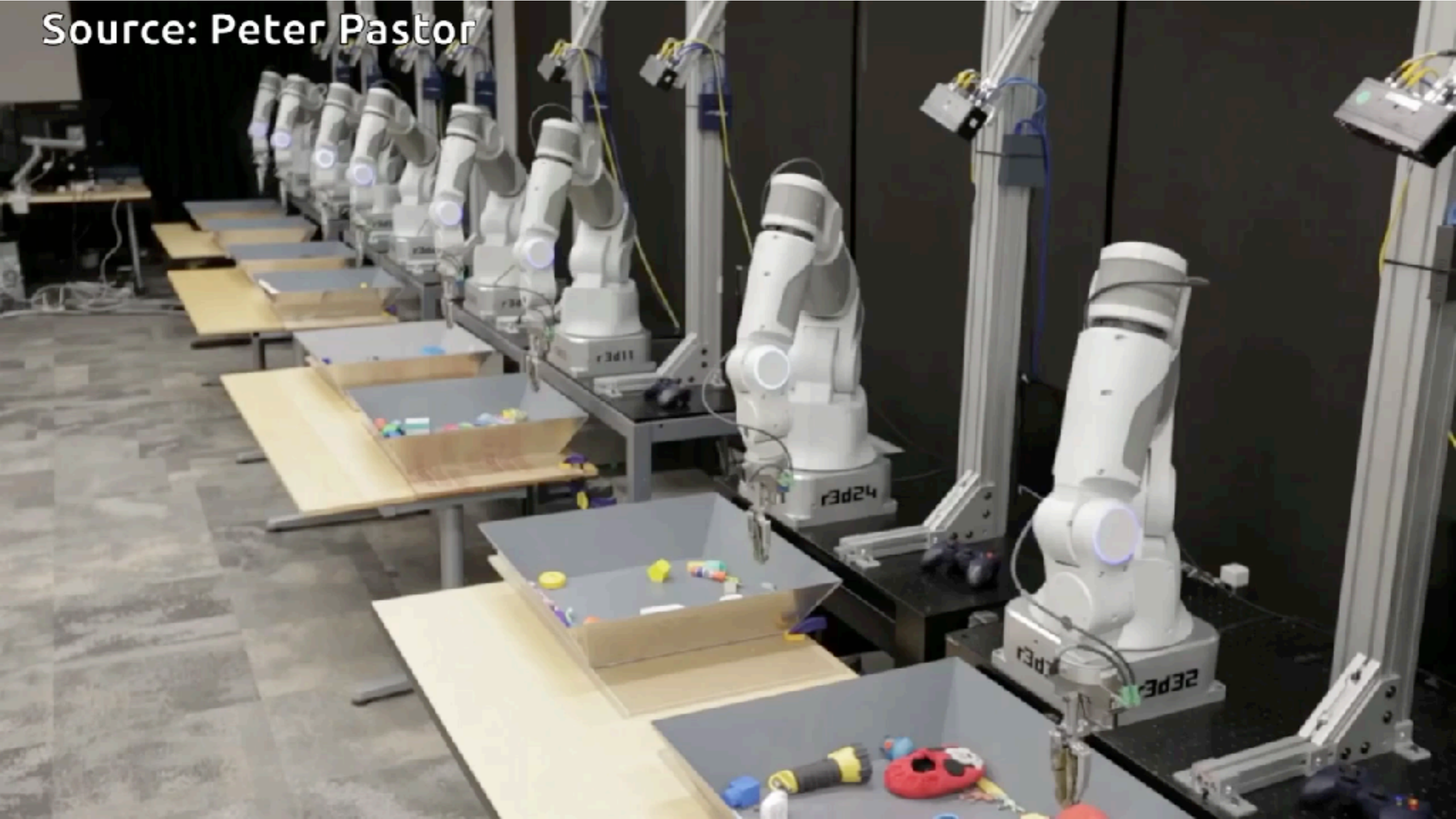


Continues motion control from RGB(D)



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

Source: Peter Pastor



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

# Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.



# Learning from expert demonstrations

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



# Learning from expert demonstrations

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



## Learning from expert demonstrations

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





## Learning from expert demonstrations

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

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

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



## Learning from expert demonstrations

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

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

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

3. Solve underlying RL 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



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

Czech Technical University in Prague

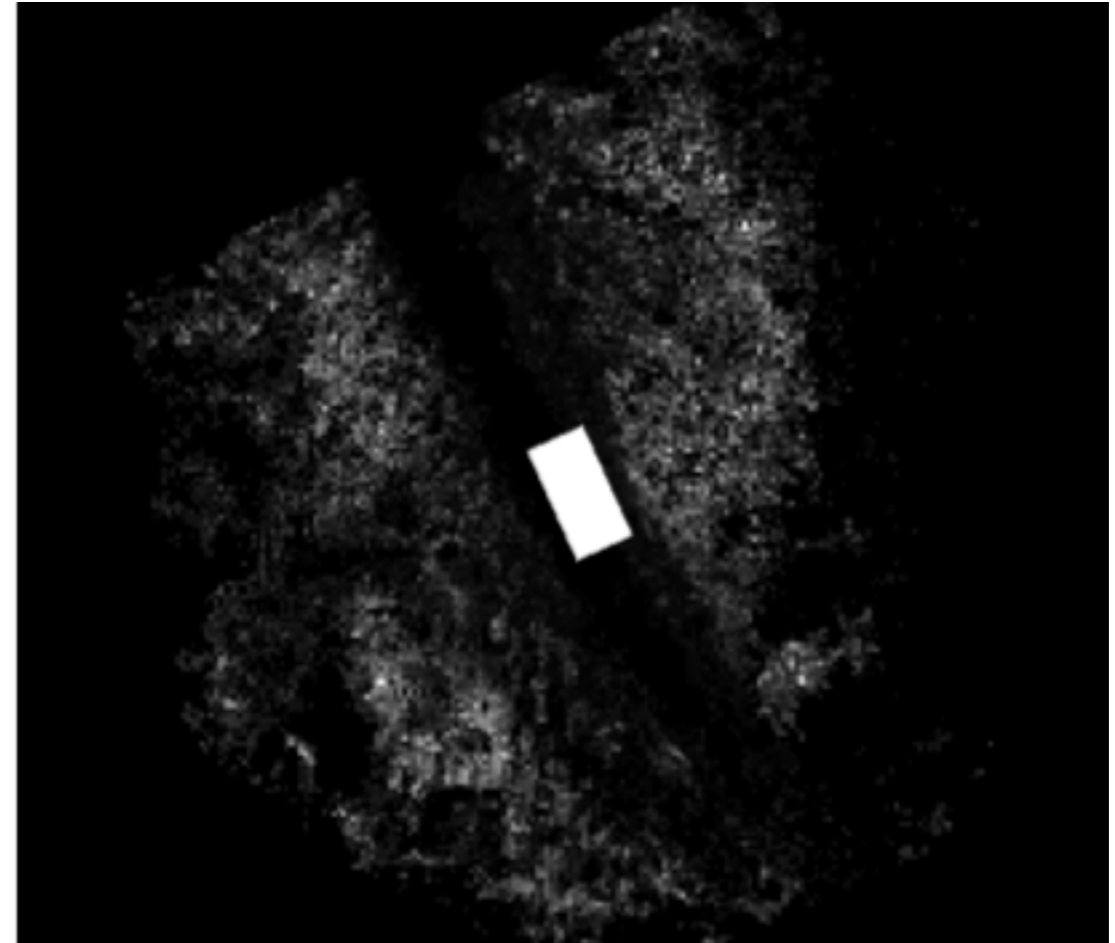
Faculty of Electrical Engineering, Department of Cybernetics



# Silver et al. IJRR 2010



input image (state)



learned reward function  
(traversability map)

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

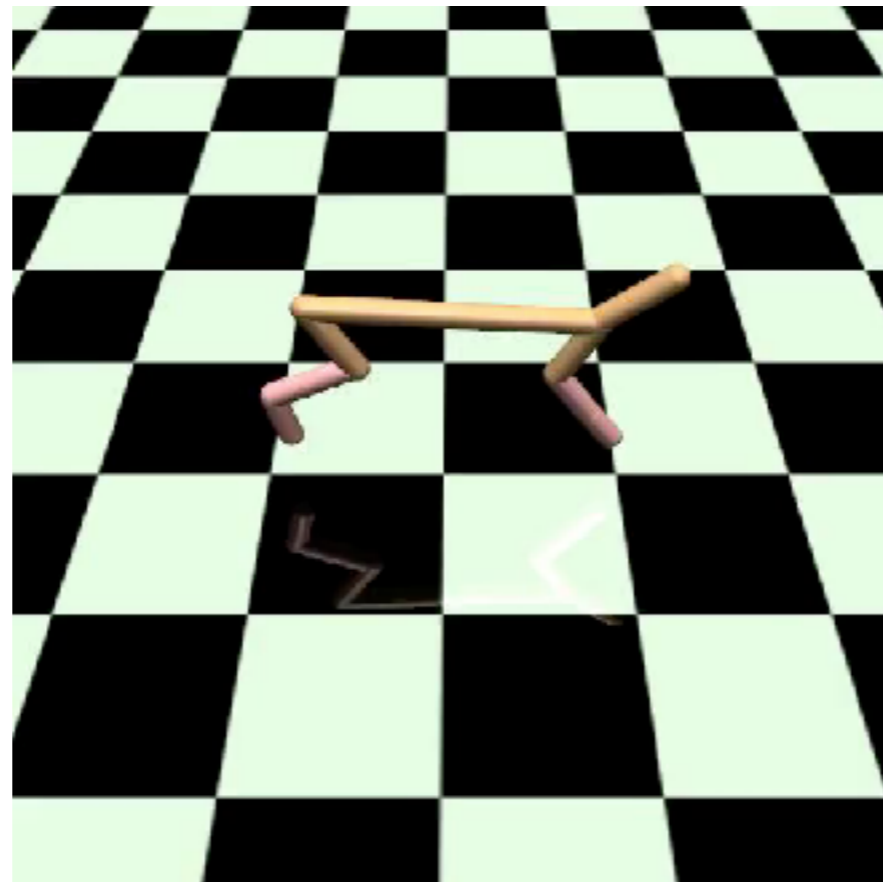
Czech Technical University in Prague

Faculty of Electrical Engineering, Department of Cybernetics



# Reward shaping

- 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)



# Reward shaping

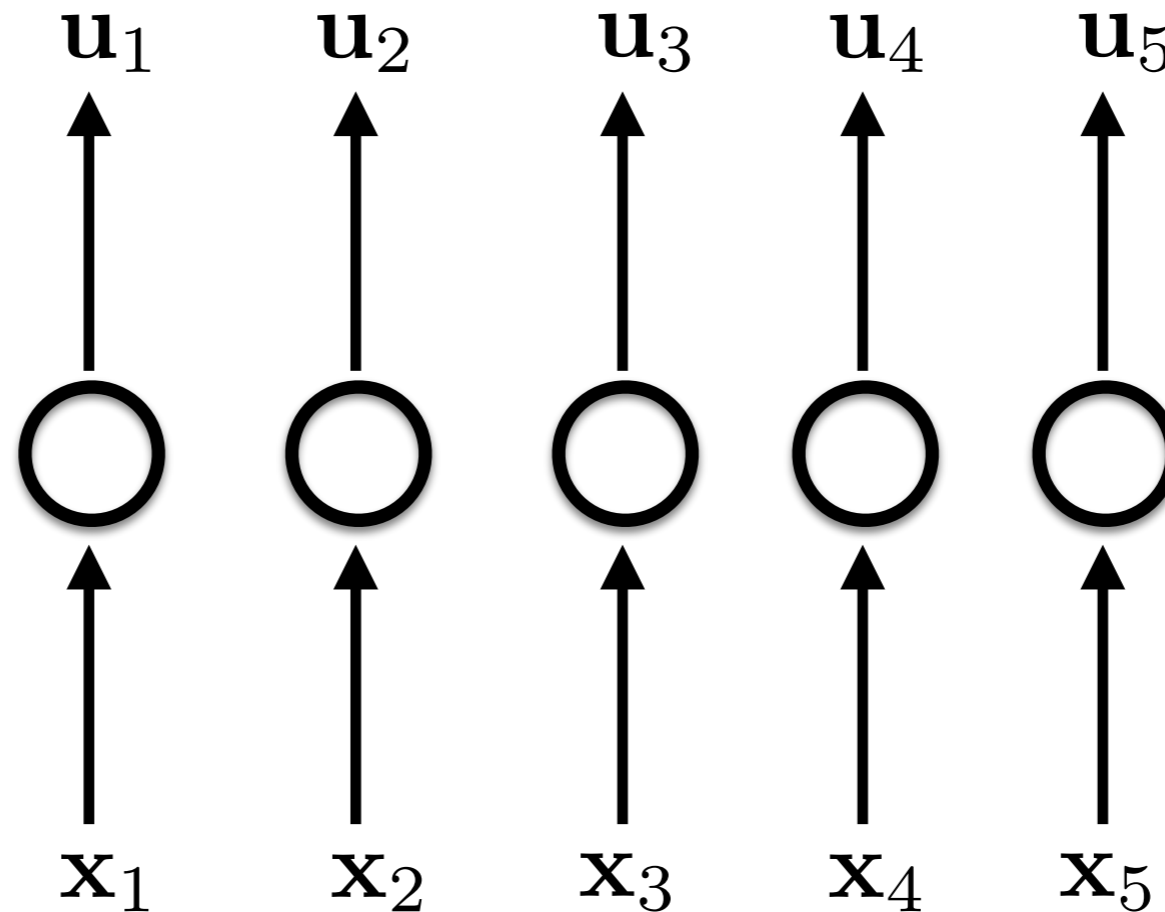
- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn





# Reward shaping

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



Sparse rewards	0	0	0	0	10
Dense rewards	2	2	2	2	2



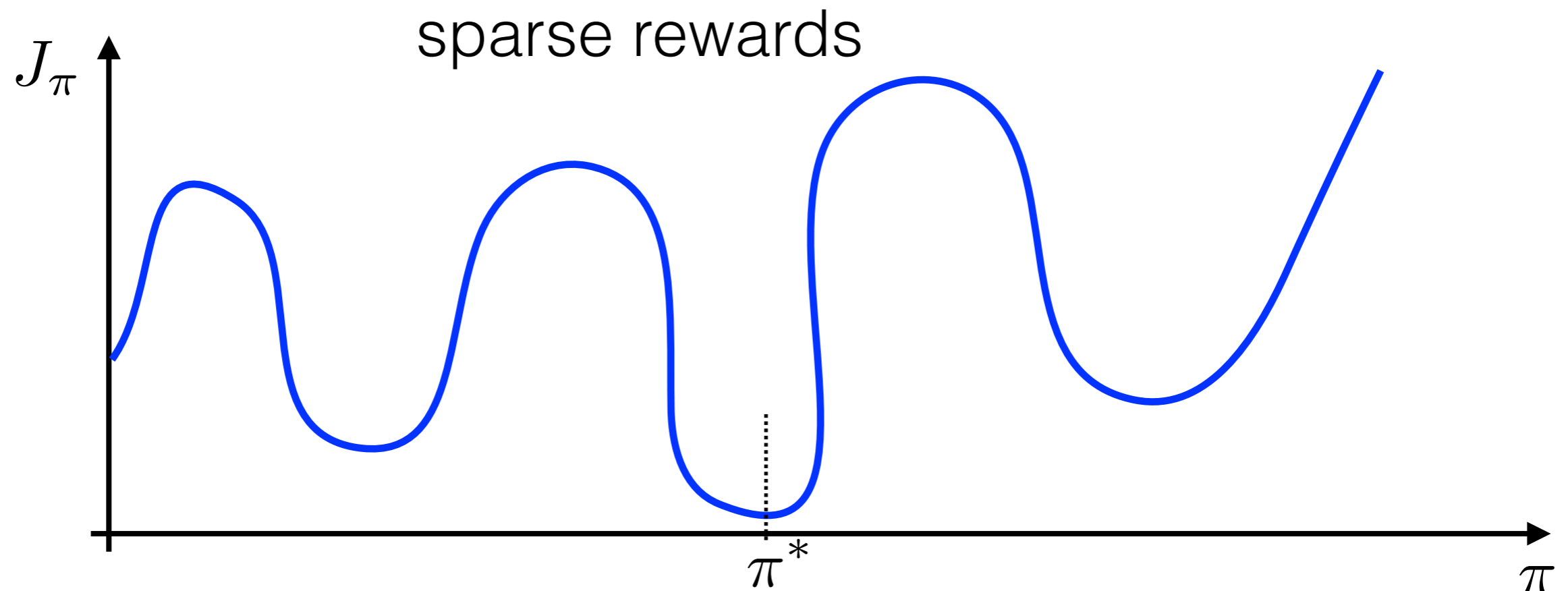
# Reward shaping

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



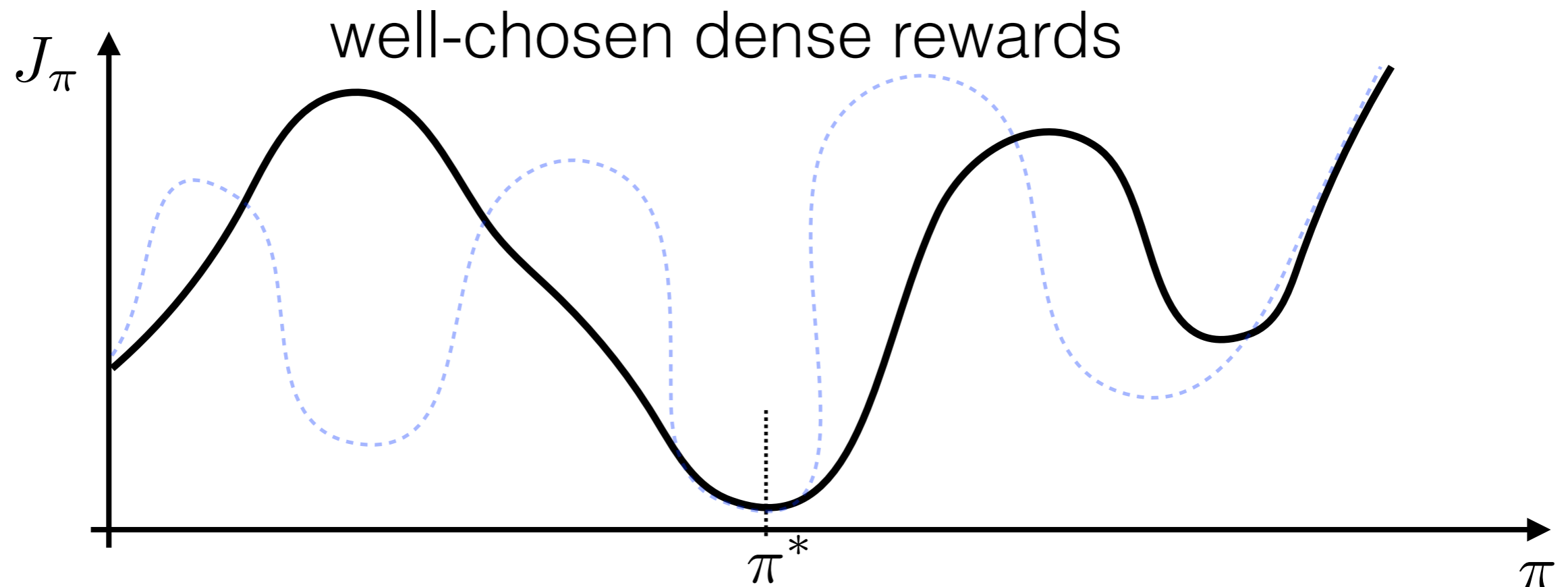
# Reward shaping

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



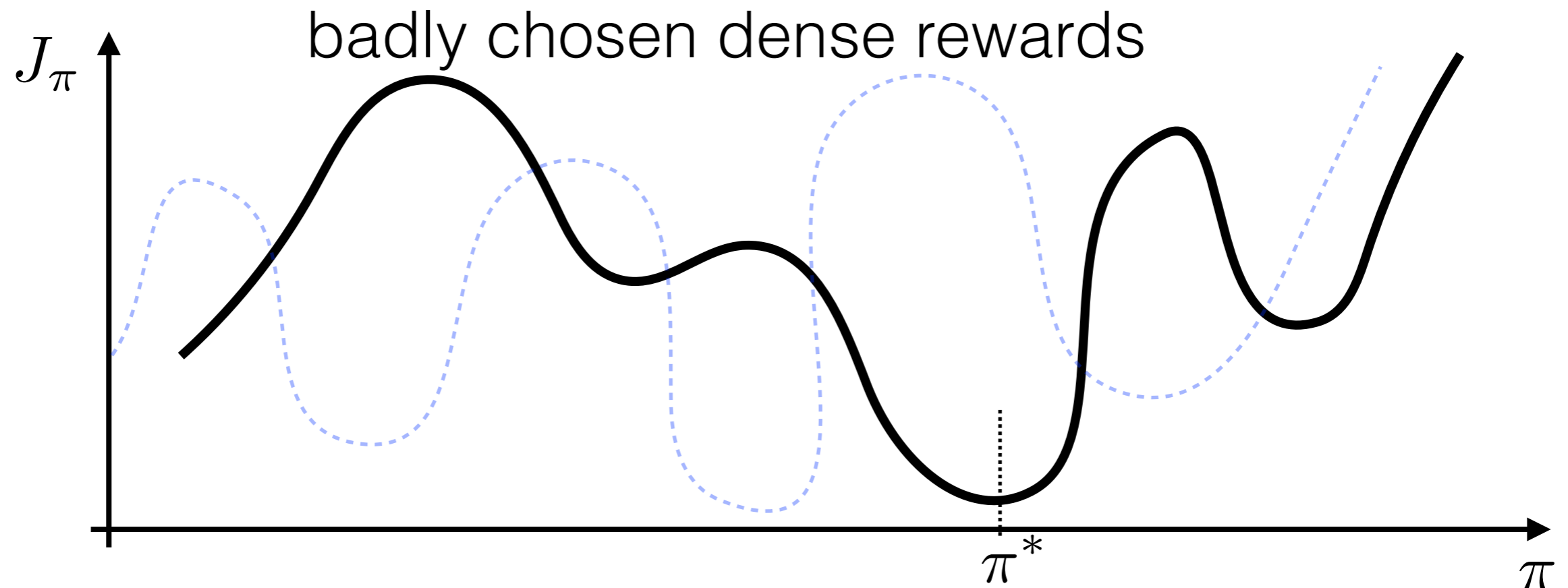
# Reward shaping

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



# Reward shaping

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



# Reward shaping

- 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 ...)



# Disadvantages of value-based methods

- Resulting policy is deterministic => exploration unclear=> eps-greedy exploration is often inefficient
- Handling continuous action-space is complicated (requires online optimization during inference)
- Learning of value based methods minimize estimation error of Q-function (does not directly maximize policy rewards).



# Taxonomy of policy search methods

- Direct policy search (primal task)

e.g. gradient ascent for  $\pi^* = \arg \max_{\pi} J_{\pi}$



Episodic REPS [Peters, 2010]

PILCO [Deisenroth, ICML 2011]

Actor-critic (e.g. DPG [Silver, JMLR 2014])

Deep Q-learning (e.g. [Mnih, Nature 2015])

- Value-based methods (dual function [Kober, 2013])

e.g. search for  $Q(\mathbf{x}, \mathbf{a}) = r(\mathbf{x}, \mathbf{a}, \mathbf{x}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{x}', \mathbf{a}')$

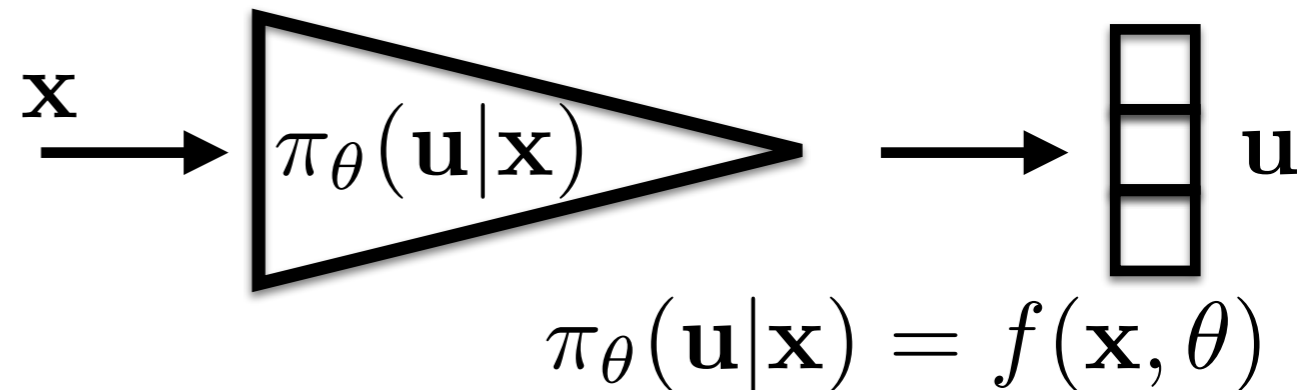
$\pi^* = \arg \max_a Q(\mathbf{x}, \mathbf{a})$





## Primal task

Stochastic policy for discrete control:



- Let us consider episodic setting:
  - Initialize in some start state
  - Run the policy in the environment
  - Generate trajectory  $\tau$
  - Obtain reward for the generated trajectory  $r(\tau)$
  - Update policy parameters

Example:

- Throwing a ball into a basket (what is suitable reward?)



# Primal task - episodic setting

1. Randomly initialize policy  $\pi_\theta$



# Primal task - episodic setting

1. Randomly initialize policy  $\pi_\theta$
2. Collect trajectories  $\tau$  with policy  $\pi_\theta$



# Primal task - episodic setting

1. Randomly initialize policy  $\pi_\theta$
2. Collect trajectories  $\tau$  with policy  $\pi_\theta$
3. Denote  $p(\tau|\pi_\theta)$  probability of  $\tau$  occurs when following  $\pi_\theta$



## Primal task - episodic setting

1. Randomly initialize policy  $\pi_\theta$
2. Collect trajectories  $\tau$  with policy  $\pi_\theta$
3. Denote  $p(\tau|\pi_\theta)$  probability of  $\tau$  occurs when following  $\pi_\theta$
4. Define criterion

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau) d\tau \approx \frac{1}{N} \sum_{i=1}^N r(\tau_i)$$



## Primal task - episodic setting

1. Randomly initialize policy  $\pi_\theta$
2. Collect trajectories  $\tau$  with policy  $\pi_\theta$
3. Denote  $p(\tau|\pi_\theta)$  probability of  $\tau$  occurs when following  $\pi_\theta$
4. Define criterion

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau) d\tau \approx \frac{1}{N} \sum_{i=1}^N r(\tau_i)$$

5. Optimize criterion (e.g. gradient descent)

$$\theta^* = \arg \min_{\theta} J(\theta)$$

6. Repeat from 2



## Primal task - episodic setting

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau) d\tau \approx \frac{1}{N} \sum_{i=1}^N r(\tau_i)$$

$$\theta^* = \arg \min_{\theta} J(\theta)$$

- What do I need for gradient descent optimization?  $\frac{\partial J(\theta)}{\partial \theta}^\top$
- Perturb parameters by  $\Delta\theta_i$  and estimate  $J(\theta + \Delta\theta_i)$

$$J(\theta + \Delta\theta_i) = J(\theta) + \frac{\partial J(\theta)}{\partial \theta}^\top \Delta\theta_i$$

$$\Delta\theta_i^\top \frac{\partial J(\theta)}{\partial \theta} = J(\theta) - J(\theta + \Delta\theta_i)$$



## Primal task - episodic setting

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau) d\tau \approx \frac{1}{N} \sum_{i=1}^N r(\tau_i)$$

$$\theta^* = \arg \min_{\theta} J(\theta)$$

- What do I need for gradient descent optimization?  $\frac{\partial J(\theta)}{\partial \theta}^\top$
- Perturb parameters by  $\Delta\theta_i$  and estimate  $J(\theta + \Delta\theta_i)$

$$J(\theta + \Delta\theta_i) = J(\theta) + \frac{\partial J(\theta)}{\partial \theta}^\top \Delta\theta_i$$

$$\Delta\theta_i^\top \frac{\partial J(\theta)}{\partial \theta} = J(\theta + \Delta\theta_i) - J(\theta)$$

$$\underbrace{\begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}}_{\text{matrix A}} \frac{\partial J(\theta)}{\partial \theta} = \underbrace{\begin{bmatrix} J(\theta + \Delta\theta_1) - J(\theta) \\ \vdots \\ J(\theta + \Delta\theta_n) - J(\theta) \end{bmatrix}}_{\text{vector b}}$$

matrix A

vector b





# Primal task - episodic setting

$$\underbrace{\begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}}_{\text{matrix A}} \frac{\partial J(\theta)}{\partial \theta} = \underbrace{\begin{bmatrix} J(\theta + \Delta\theta_1) - J(\theta) \\ \vdots \\ J(\theta + \Delta\theta_n) - J(\theta) \end{bmatrix}}_{\text{vector b}}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta + \Delta\theta_1) - J(\theta) \\ \vdots \\ J(\theta + \Delta\theta_n) - J(\theta) \end{bmatrix}$$



## Primal task - episodic setting

1. Randomly initialize  $\theta$
2. Collect trajectories randomly perturbed policy  $\pi_{\theta + \Delta\theta_i}$
3. Compute gradient  $\frac{\partial J(\theta)}{\partial \theta}^\top$  using pseudo-inverse

$$\frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n) \end{bmatrix}$$

4. Update parameters

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



## Primal task - episodic setting

REINFORCE: better gradient approximation

- stochastic policy  $\pi_{\theta}(\mathbf{u}|\mathbf{x}) : X \times U \rightarrow [0; 1]$

- criterion

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

- gradient of the criterion

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

- likelihood ratio trick expresses gradient of the prob distr.



## Primal task - episodic setting

REINFORCE: better gradient approximation

- stochastic policy  $\pi_{\theta}(\mathbf{u}|\mathbf{x}) : X \times U \rightarrow [0; 1]$

- criterion

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

- gradient of the criterion

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

- likelihood ratio trick expresses gradient of the prob distr.

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



## Primal task - episodic setting

- after substitution

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= \int_T p(\tau|\pi_\theta) \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) d\tau = \\ &= \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \left[ \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) \right] \approx \frac{1}{N} \sum_{i=1}^N \frac{\partial \log p(\tau_i|\pi_\theta)}{\partial \theta} r(\tau_i) \end{aligned}$$



## Primal task - episodic setting

- after substitution

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \int_T p(\tau|\pi_\theta) \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) d\tau = \\ &= \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \left[ \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) \right] \approx \frac{1}{N} \sum_{i=1}^N \frac{\partial \log p(\tau_i|\pi_\theta)}{\partial \theta} r(\tau_i)\end{aligned}$$

- where prob distribution simplified using MDP assumption

$$p(\tau|\pi_\theta) = p(\mathbf{x}_0) \prod_k p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) \pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$$

$$\begin{aligned}\frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} [\log p(\mathbf{x}_0) + \sum_k \log(p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)) + \\ &+ \sum_k \log(\pi_\theta(\mathbf{u}_k|\mathbf{x}_k))] = \sum_k \frac{\partial \log \pi_\theta(\mathbf{u}_k|\mathbf{x}_k)}{\partial \theta}\end{aligned}$$



## Primal task - episodic setting

- after substitution

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \int_T p(\tau|\pi_\theta) \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) d\tau = \\ &= \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \left[ \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) \right] \approx \frac{1}{N} \sum_{i=1}^N \frac{\partial \log p(\tau_i|\pi_\theta)}{\partial \theta} r(\tau_i)\end{aligned}$$

- where prob distribution simplified using MDP assumption

$$p(\tau|\pi_\theta) = p(\mathbf{x}_0) \prod_k p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) \pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$$

$$\begin{aligned}\frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} [\log p(\mathbf{x}_0) + \sum_k \log(p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)) + \\ &+ \sum_k \log(\pi_\theta(\mathbf{u}_k|\mathbf{x}_k))] = \sum_k \frac{\partial \log \pi_\theta(\mathbf{u}_k|\mathbf{x}_k)}{\partial \theta}\end{aligned}$$



## Primal task - episodic setting

- after substitution

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \int_T p(\tau|\pi_\theta) \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) d\tau = \\ &= \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \left[ \frac{\partial \log p(\tau|\pi_\theta)}{\partial \theta} r(\tau) \right] \approx \frac{1}{N} \sum_{i=1}^N \frac{\partial \log p(\tau_i|\pi_\theta)}{\partial \theta} r(\tau_i)\end{aligned}$$

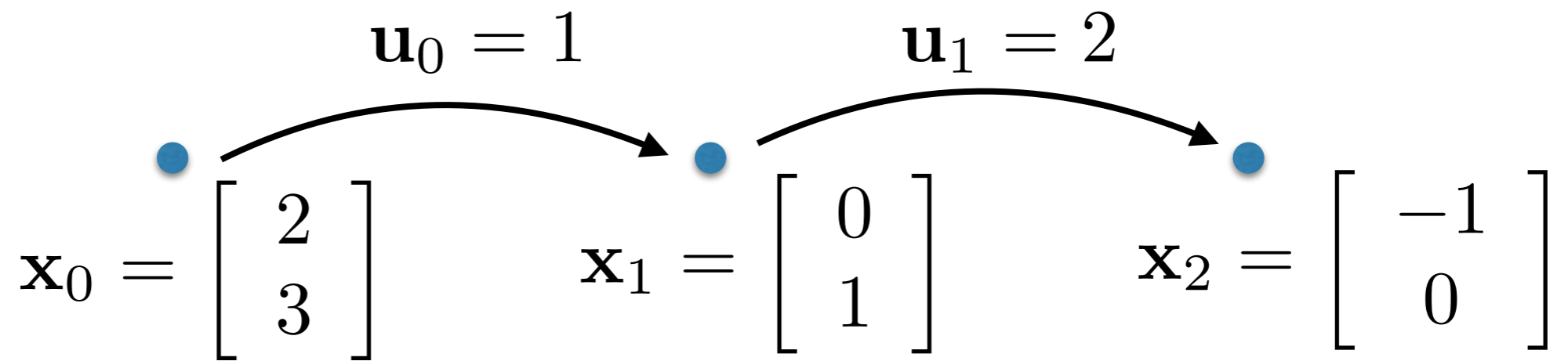
Let us demonstrate meaning of policy gradient on a trajectory  $\tau$  on the following example

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_k \frac{\partial \log \pi_\theta(\mathbf{u}_k | \mathbf{x}_k)}{\partial \theta} \cdot \underbrace{\sum_j r(\mathbf{u}_j, \mathbf{x}_j)}_{r(\tau)}$$





trajectory:

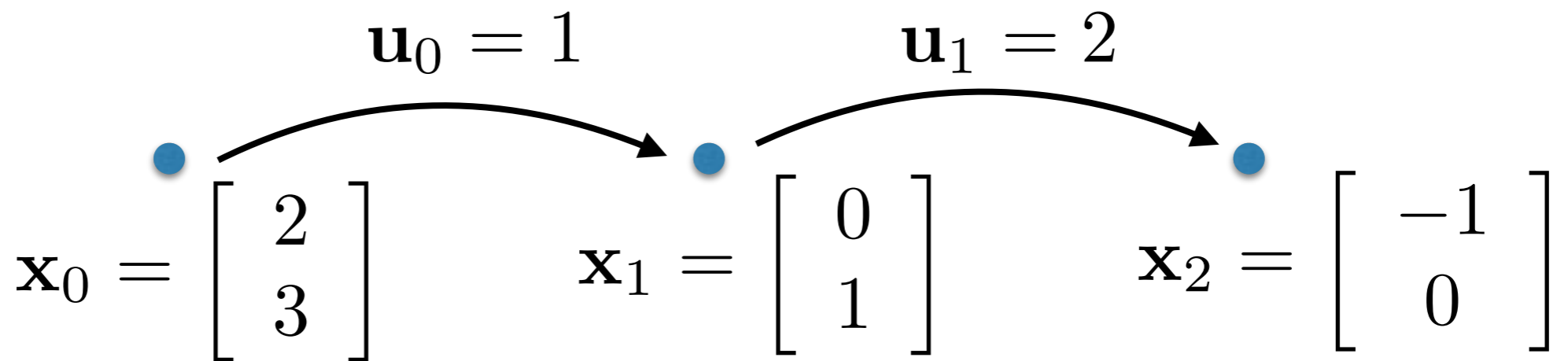


policy:  $\pi_{\theta}(\mathbf{u}|\mathbf{x}) = \mathbf{s} \left( \begin{bmatrix} \theta_1^{\top} \mathbf{x} \\ \theta_2^{\top} \mathbf{x} \end{bmatrix} \right)$       parameters:  $\theta_1^{\top} = [2, 0]$   
 $\theta_2^{\top} = [0, 1]$



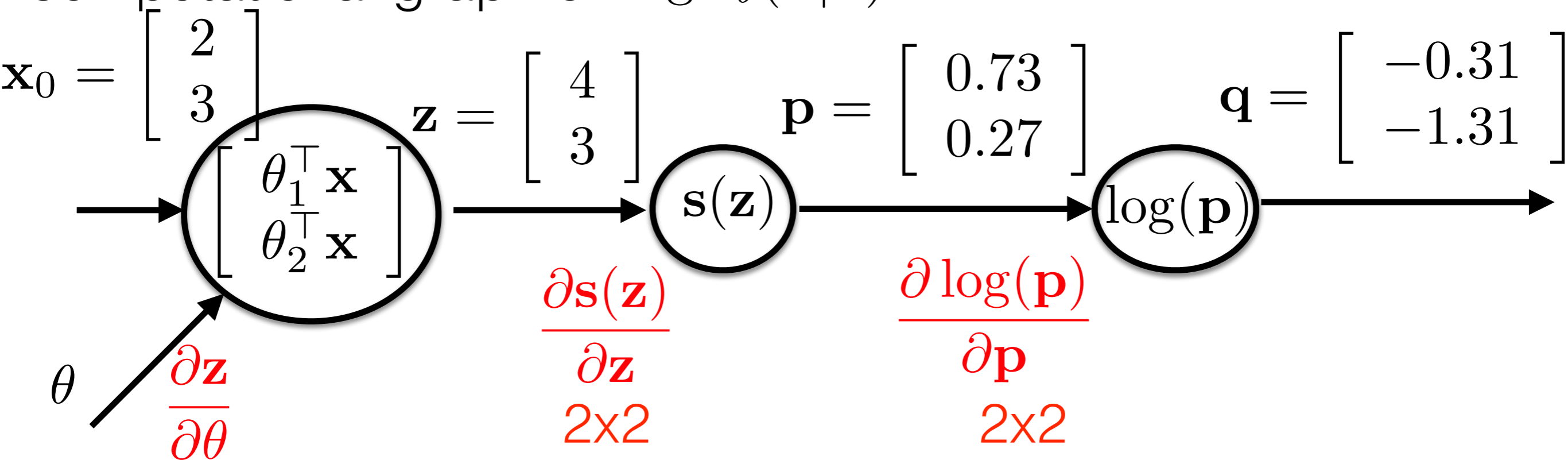
2x4

trajectory:

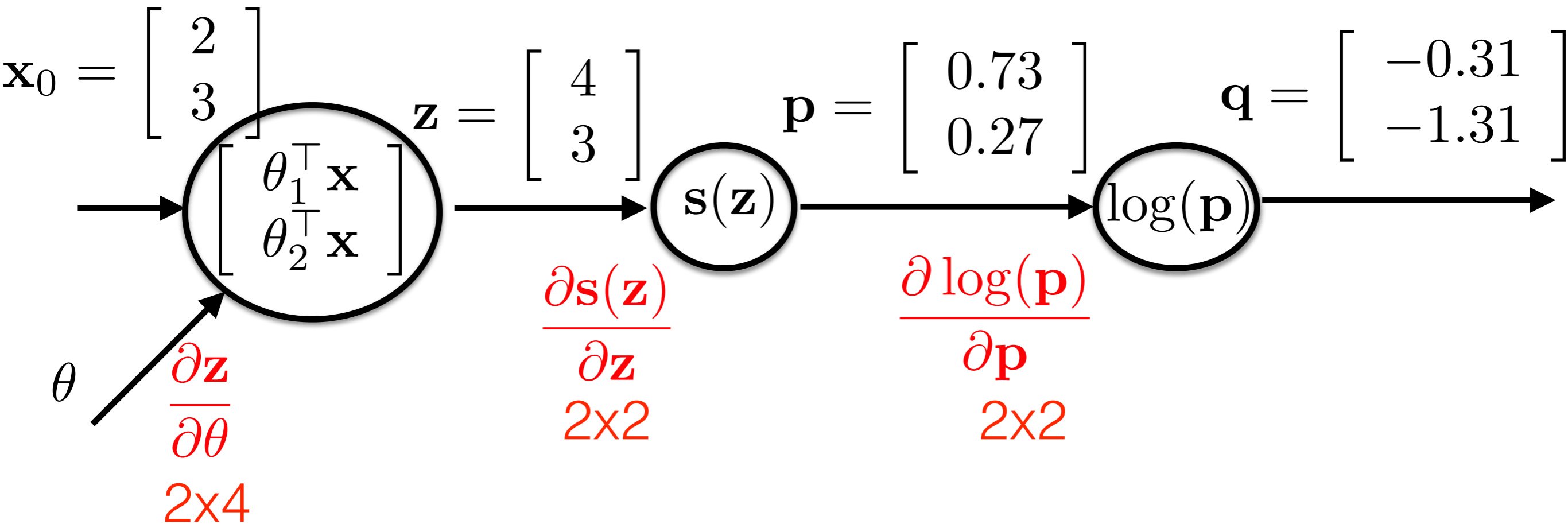
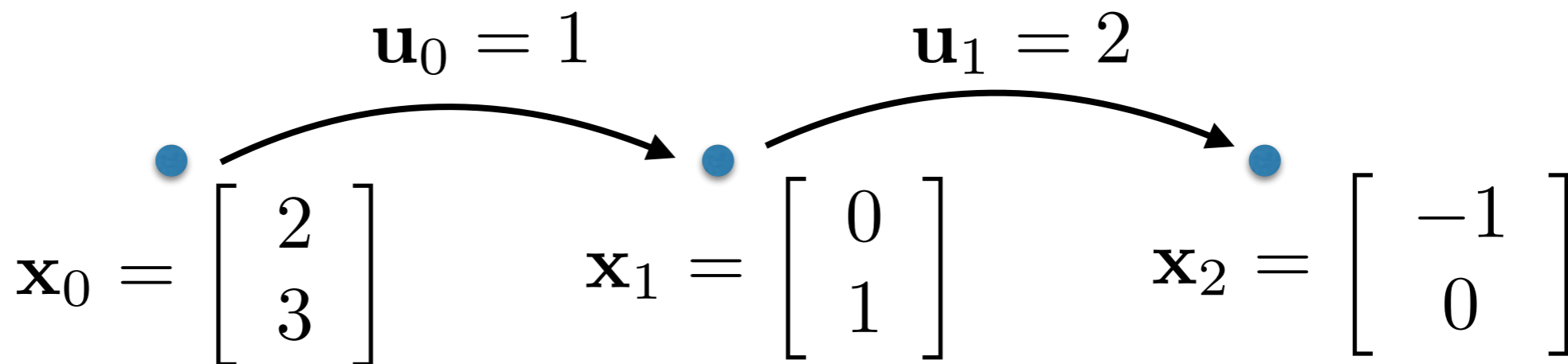


policy:  $\pi_{\theta}(\mathbf{u}|\mathbf{x}) = \mathbf{s} \left( \begin{bmatrix} \theta_1^{\top} \mathbf{x} \\ \theta_2^{\top} \mathbf{x} \end{bmatrix} \right)$  parameters:  $\theta_1^{\top} = [2, 0]$   
 $\theta_2^{\top} = [0, 1]$

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



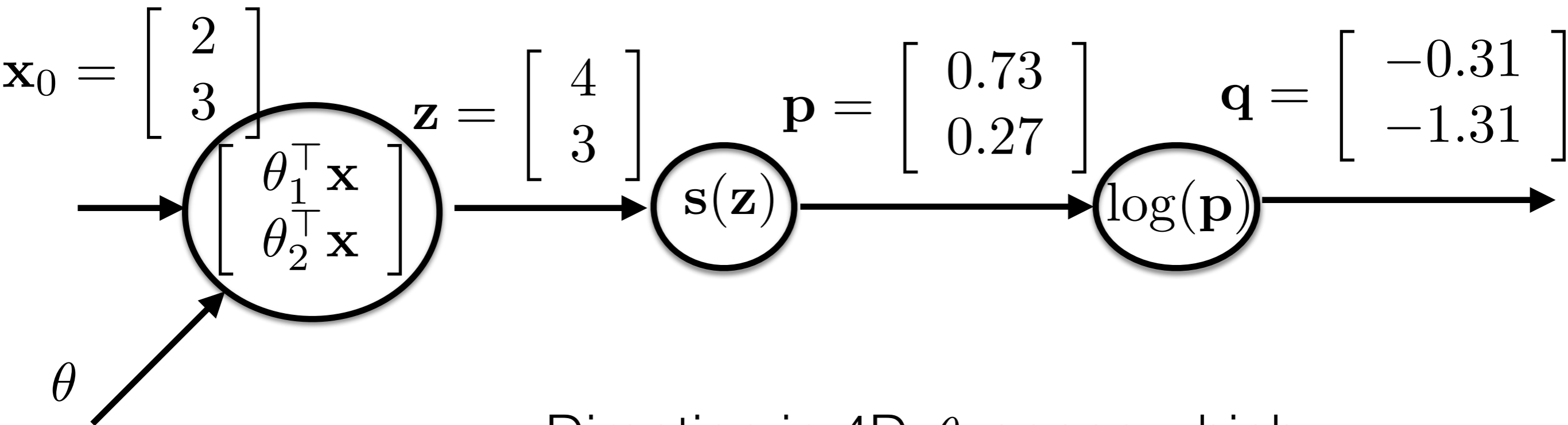
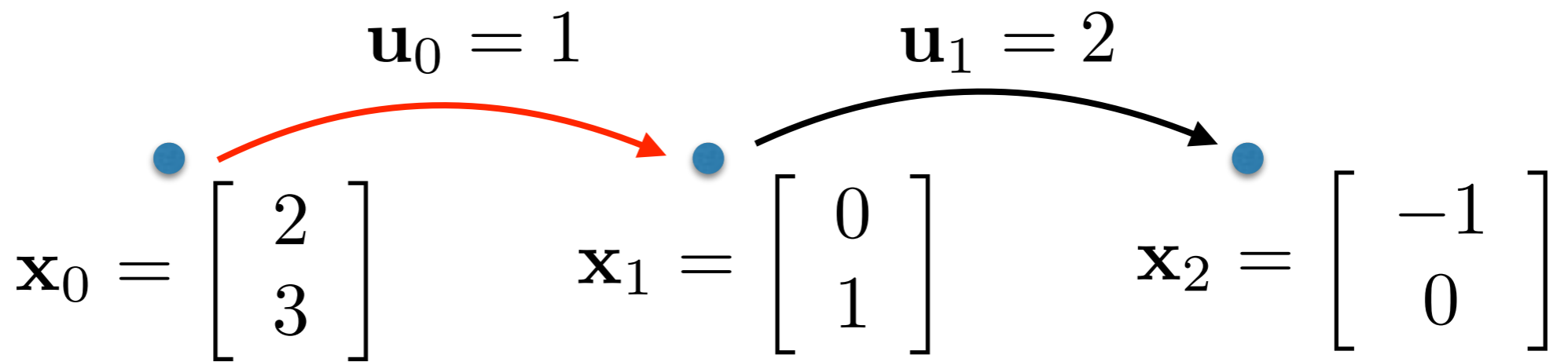
trajectory:



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



trajectory:



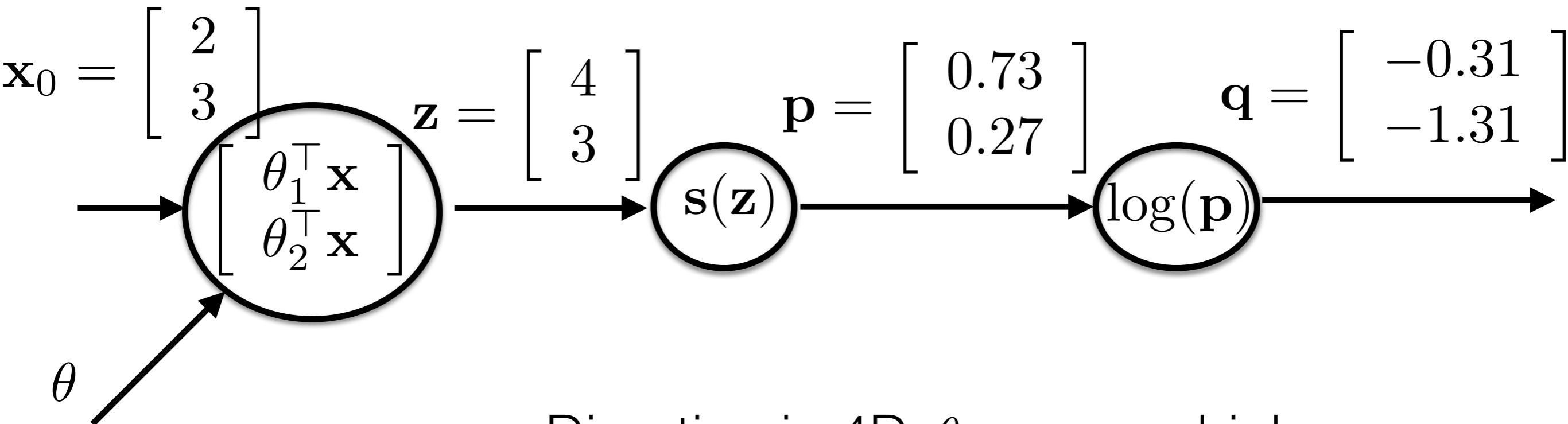
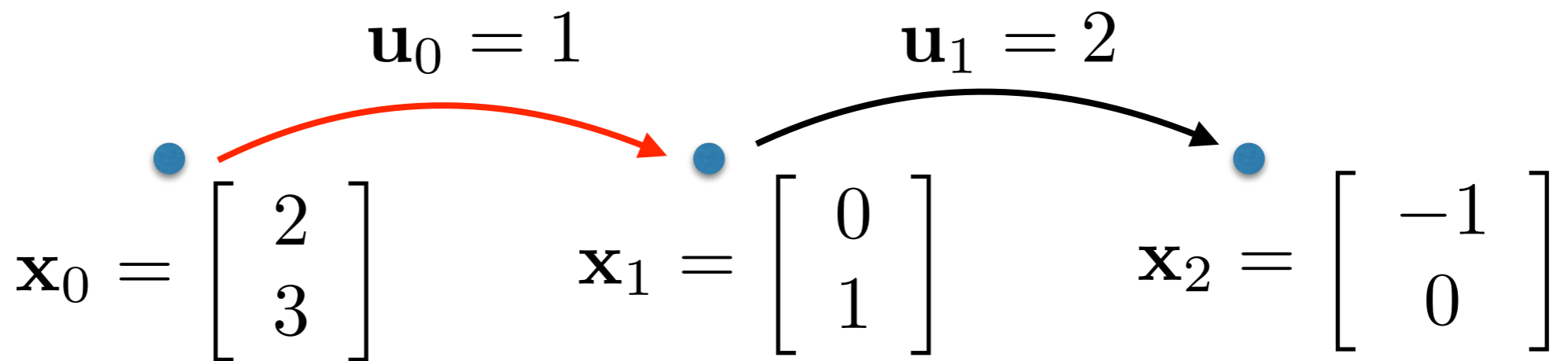
Direction in 4D  $\theta$ -space which increases prob. of choosing control  $\mathbf{u} = 1$

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

$2 \times 2$      $2 \times 2$      $2 \times 4$                        $2 \times 4$



trajectory:

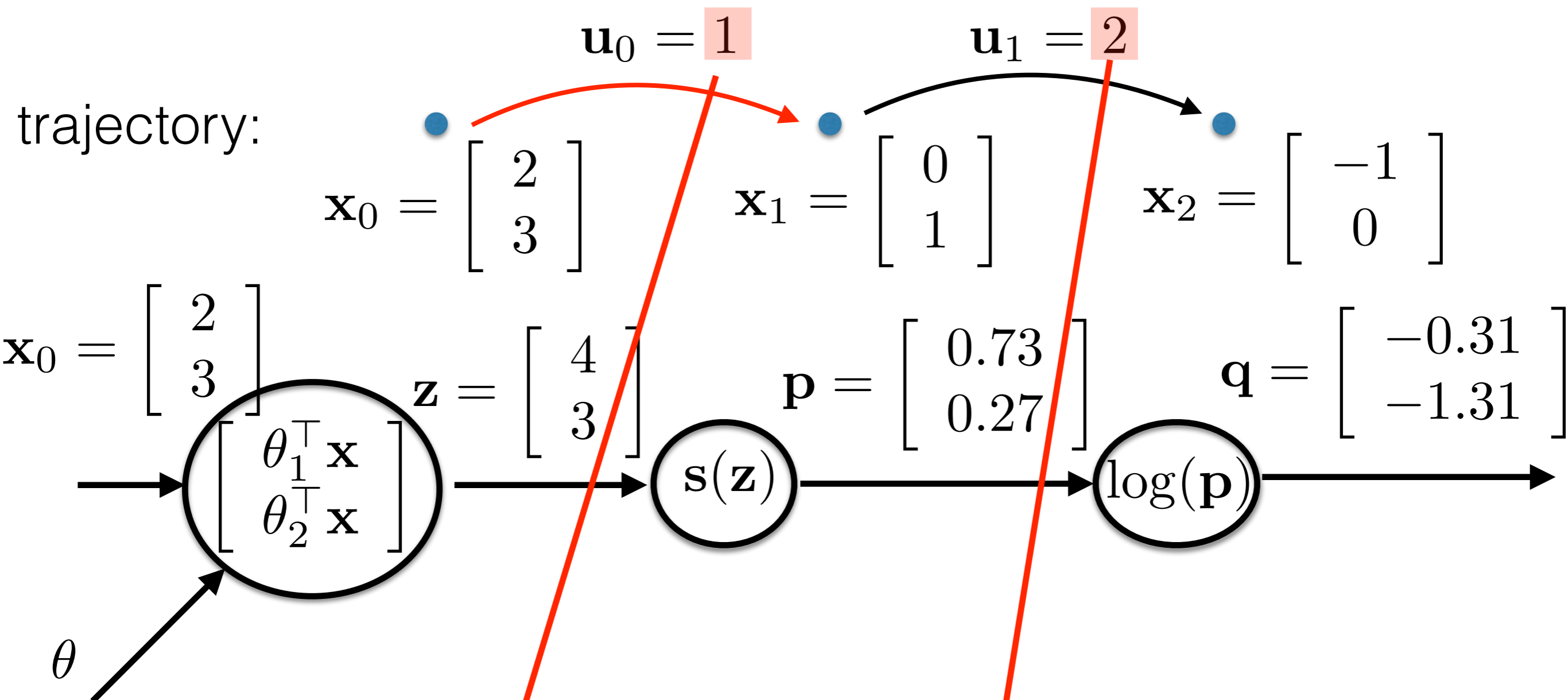


Direction in 4D  $\theta$ -space which increases prob. of choosing control  $\mathbf{u} = 2$

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

$2 \times 2$      $2 \times 2$      $2 \times 4$                        $2 \times 4$



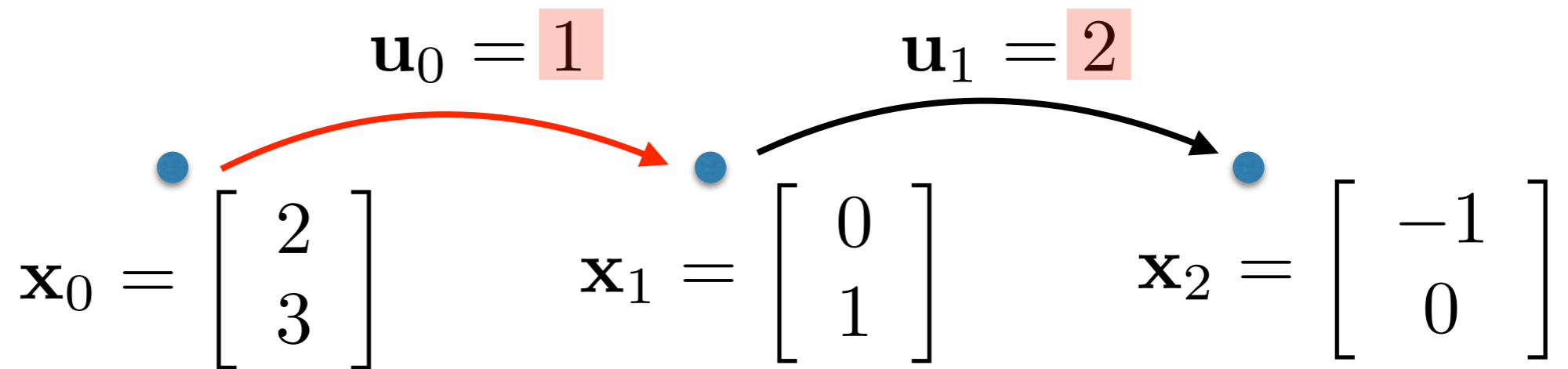


By substituting actions and states from the trajectory into the policy gradient

$$\begin{aligned}
 \frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial \log \pi_{\theta}(\mathbf{u}_0 | \mathbf{x}_0)}{\partial \theta} \cdot r(\tau) + \frac{\partial \log \pi_{\theta}(\mathbf{u}_1 | \mathbf{x}_1)}{\partial \theta} \cdot r(\tau) + \dots \\
 &= \mathbf{g}_1^{\top}(\mathbf{x}_0) \cdot r(\tau) + \mathbf{g}_2^{\top}(\mathbf{x}_1) \cdot r(\tau) + \dots
 \end{aligned}$$



trajectory:



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

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial \log \pi_{\theta}(\mathbf{u}_0 | \mathbf{x}_0)}{\partial \theta} \cdot r(\tau) + \frac{\partial \log \pi_{\theta}(\mathbf{u}_1 | \mathbf{x}_1)}{\partial \theta} \cdot r(\tau) + \dots \\ &= \boxed{\mathbf{g}_1^{\top}(\mathbf{x}_0)} \cdot r(\tau) + \boxed{\mathbf{g}_2^{\top}(\mathbf{x}_1)} \cdot r(\tau) + \dots \end{aligned}$$

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

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

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



# Primal task - discrete control in episodic settings

- policy (random ini)

- collect N trajectories

$$\tau_1 = \{(\mathbf{x}_{11}, \mathbf{u}_{11}), (\mathbf{x}_{21}, \mathbf{u}_{21}), \dots, (\mathbf{x}_{M1}, \mathbf{u}_{M1})\}$$

⋮

$$\tau_N = \{(\mathbf{x}_{1N}, \mathbf{u}_{1N}), (\mathbf{x}_{2N}, \mathbf{u}_{2N}), \dots, (\mathbf{x}_{MN}, \mathbf{u}_{MN})\}$$

- compute gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^M \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \sum_{j=1}^M r(\mathbf{u}_{ji}, \mathbf{x}_{ji})$$

- update parameters

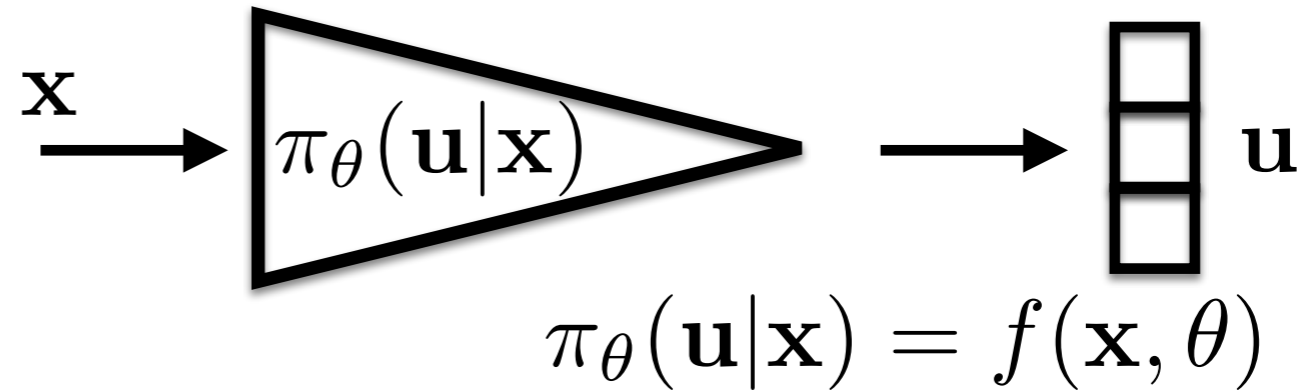
$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$





# Primal task - discrete control in episodic settings

- policy (random ini)



- collect N trajectories

$$\tau_1 = \{(\mathbf{x}_{11}, \mathbf{u}_{11}), (\mathbf{x}_{21}, \mathbf{u}_{21}), \dots, (\mathbf{x}_{M1}, \mathbf{u}_{M1})\}$$

⋮

$$\tau_N = \{(\mathbf{x}_{1N}, \mathbf{u}_{1N}), (\mathbf{x}_{2N}, \mathbf{u}_{2N}), \dots, (\mathbf{x}_{MN}, \mathbf{u}_{MN})\}$$

- compute gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^M \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \sum_{j=1}^M r(\mathbf{u}_{ji}, \mathbf{x}_{ji})$$

- update parameters

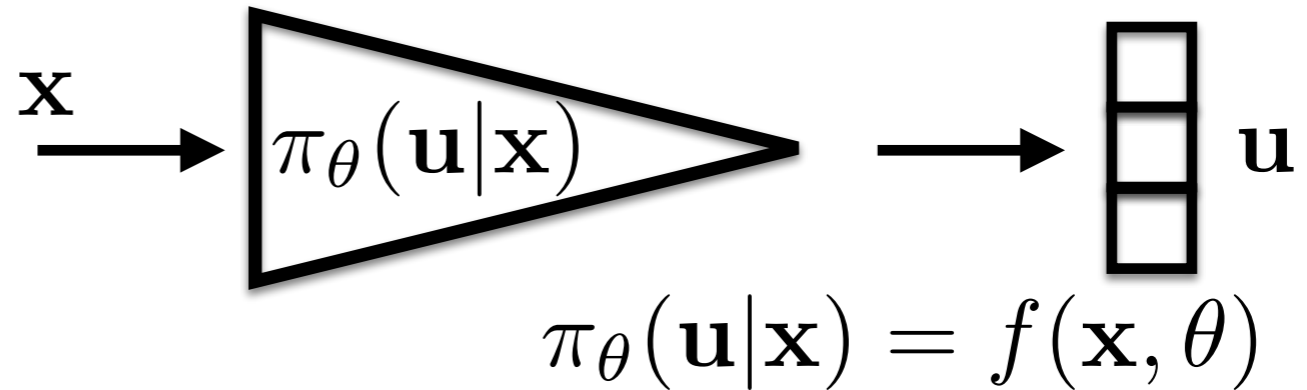
$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$

How to compute it ???



# Primal task - discrete control in episodic settings

- policy (random ini)



- collect N trajectories

$$\tau_1 = \{(\mathbf{x}_{11}, \mathbf{u}_{11}), (\mathbf{x}_{21}, \mathbf{u}_{21}), \dots, (\mathbf{x}_{M1}, \mathbf{u}_{M1})\}$$

⋮

$$\tau_N = \{(\mathbf{x}_{1N}, \mathbf{u}_{1N}), (\mathbf{x}_{2N}, \mathbf{u}_{2N}), \dots, (\mathbf{x}_{MN}, \mathbf{u}_{MN})\}$$

- compute gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^M \frac{\partial(-\mathcal{L}\{f(\mathbf{x}, \theta), \mathbf{u}_{ki}\})}{\partial \theta} \cdot \sum_{j=1}^M r(\mathbf{u}_{ji}, \mathbf{x}_{ji})$$

Maximize minus

- update parameters

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

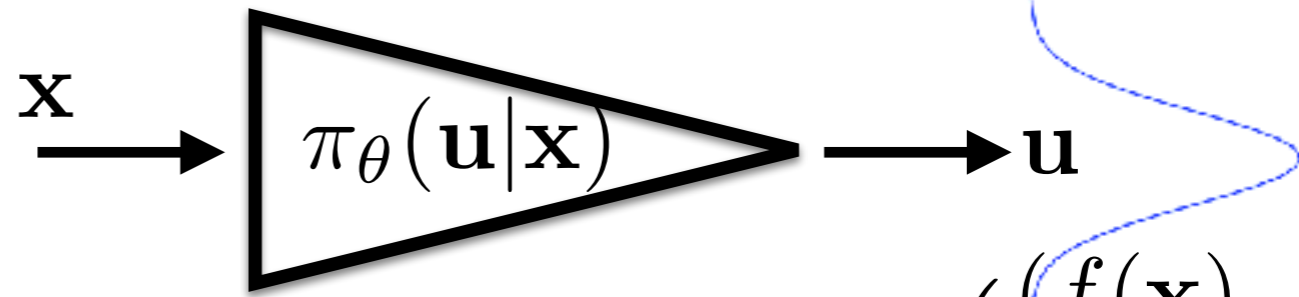
cross-entropy loss weighted

by sum of rewards



# Primal task - continuous control in episodic settings

- policy (random ini)



- collect N trajectories

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

$$\tau_1 = \{(\mathbf{x}_{11}, \mathbf{u}_{11}), (\mathbf{x}_{21}, \mathbf{u}_{21}), \dots, (\mathbf{x}_{M1}, \mathbf{u}_{M1})\}$$

⋮

$$\tau_N = \{(\mathbf{x}_{1N}, \mathbf{u}_{1N}), (\mathbf{x}_{2N}, \mathbf{u}_{2N}), \dots, (\mathbf{x}_{MN}, \mathbf{u}_{MN})\}$$

- compute gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^M \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \sum_{j=1}^M r(\mathbf{u}_{ji}, \mathbf{x}_{ji})$$

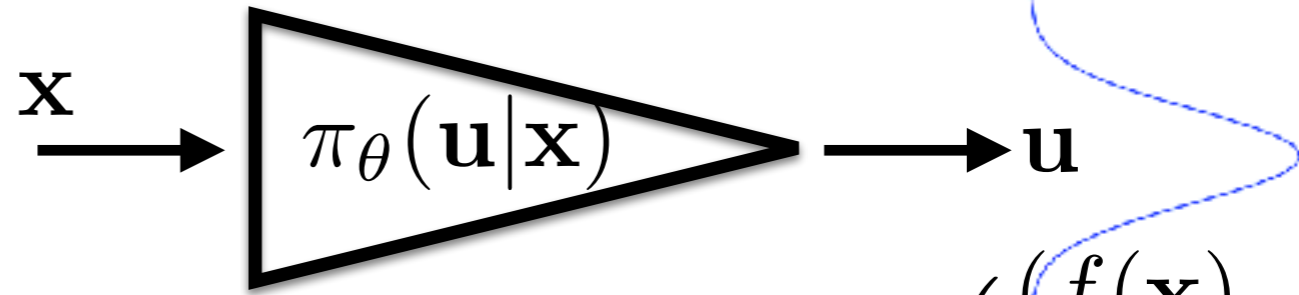
- update parameters

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



# Primal task - continuous control in episodic settings

- policy (random ini)



- collect N trajectories

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

$$\tau_1 = \{(\mathbf{x}_{11}, \mathbf{u}_{11}), (\mathbf{x}_{21}, \mathbf{u}_{21}), \dots, (\mathbf{x}_{M1}, \mathbf{u}_{M1})\}$$

⋮

$$\tau_N = \{(\mathbf{x}_{1N}, \mathbf{u}_{1N}), (\mathbf{x}_{2N}, \mathbf{u}_{2N}), \dots, (\mathbf{x}_{MN}, \mathbf{u}_{MN})\}$$

- compute gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^M \frac{\partial \|\| f(\mathbf{x}, \theta) - \mathbf{u}_{ki} \|\|_2^2}{\partial \theta} \cdot \sum_{j=1}^M r(\mathbf{u}_{ji}, \mathbf{x}_{ji})$$

- update parameters

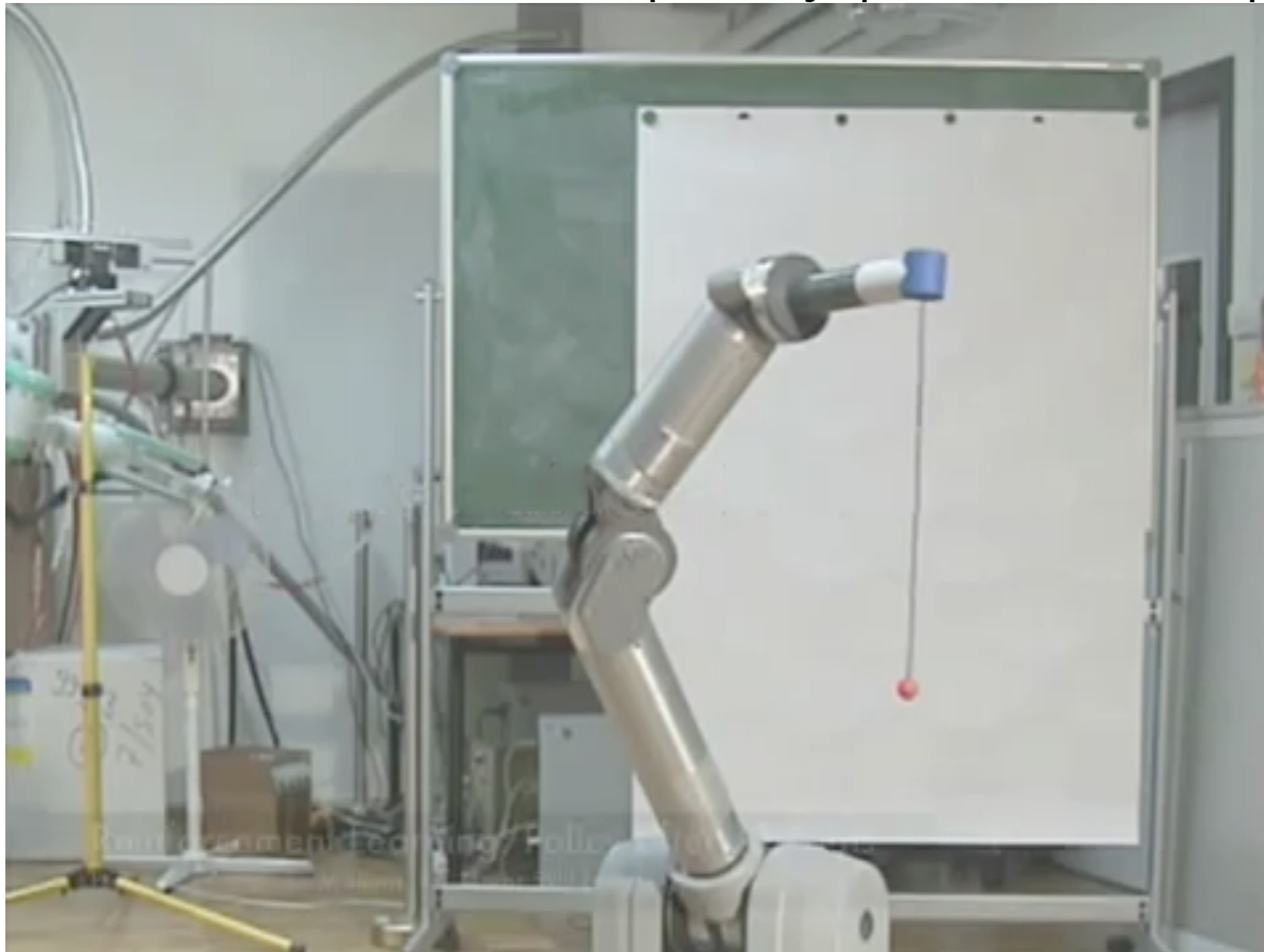
$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$

L2 loss



## Peters et al. NOW 2013

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



Czech Technical University in Prague

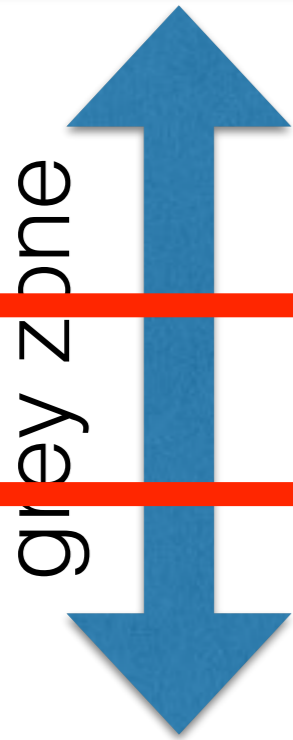
Faculty of Electrical Engineering, Department of Cybernetics



# Taxonomy of policy search methods

- Direct policy search (primal task)

e.g. gradient ascent for  $\pi^* = \arg \max_{\pi} J_{\pi}$



Episodic REPS [Peters, 2010]

Generalized Advantage function [Schulman 2016]

Actor-critic (e.g. DDPG [Lilicrap 2015])

Deep Q-learning (e.g. [Mnih, Nature 2015])

- Value-based methods (dual function [Kober, 2013])

e.g. search for  $Q(\mathbf{x}, \mathbf{a}) = r(\mathbf{x}, \mathbf{a}, \mathbf{x}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{x}', \mathbf{a}')$

$\pi^* = \arg \max_a Q(\mathbf{x}, \mathbf{a})$



# Primal task - policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^{\infty} \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \left( \sum_{j=1}^{\infty} r(\mathbf{u}_{ji}, \mathbf{x}_{ji}) \right)$$



# Primal task - policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^{\infty} \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \left( \sum_{j=k}^{\infty} r(\mathbf{u}_{ji}, \mathbf{x}_{ji}) \right)$$





# Primal task - policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^{\infty} \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \left( \sum_{j=k}^{\infty} r(\mathbf{u}_{ji}, \mathbf{x}_{ji}) \right)$$

- state-action function (policy gradient theorem):

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^{\infty} \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot Q(\mathbf{u}_{ki}, \mathbf{x}_{ki})$$



# Primal task - policy gradients for stochastic policy [Schulman et al 2016]

- temporal coherence

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^{\infty} \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \left( \sum_{j=k}^{\infty} r(\mathbf{u}_{ji}, \mathbf{x}_{ji}) \right)$$

- state-action function (policy gradient theorem):

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^{\infty} \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot Q(\mathbf{u}_{ki}, \mathbf{x}_{ki})$$

- baseline

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^{\infty} \frac{\partial \log \pi_{\theta}(\mathbf{u}_{ki} | \mathbf{x}_{ki})}{\partial \theta} \cdot \underbrace{\left( Q(\mathbf{u}_{ki}, \mathbf{x}_{ki}) - V(\mathbf{x}_{ki}) \right)}_{A(\mathbf{u}_{ki}, \mathbf{x}_{ki})}$$



## Advantage function

- When deciding optimally it is enough to decide which of the actions yields higher Q-values.
- Estimation of exact Q-values is not necessary
- When  $Q(x, a_1) = 99$  and  $Q(x, a_2) = 101$ , it is enough to estimate that Q-value of  $a_2$  is bigger than Q-value of  $a_1$ .
- Predicting such values by a deep neural network causes that most of the weights will be sacrificed to unimportant information that Q-values are around 100.
- Consequently advantage function is introduced.

$$A(\mathbf{x}, \mathbf{u}) = Q(\mathbf{x}, \mathbf{u}) - V(\mathbf{x})$$

- Generalized Advantage Estimation yields lower variance and faster learning

<https://arxiv.org/pdf/1506.02438.pdf>

Czech Technical University in Prague

Faculty of Electrical Engineering, Department of Cybernetics



# Generalized Advantage function Estimation [ICLR 2016]

<https://arxiv.org/pdf/1506.02438.pdf>

$$A(\mathbf{x}, \mathbf{u}) = Q(\mathbf{x}, \mathbf{u}) - V(\mathbf{x})$$



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

$$A^{(2)}(\mathbf{x}_1, \mathbf{u}) = -V(\mathbf{x}_1) + r_1 + \gamma r_2 + \gamma^2 V(\mathbf{x}_3)$$

⋮

$$A^{(N)}(\mathbf{x}_1, \mathbf{u}) = -V(\mathbf{x}_1) + r_1 + \gamma r_2 + \dots + \gamma^N V(\mathbf{x}_N)$$

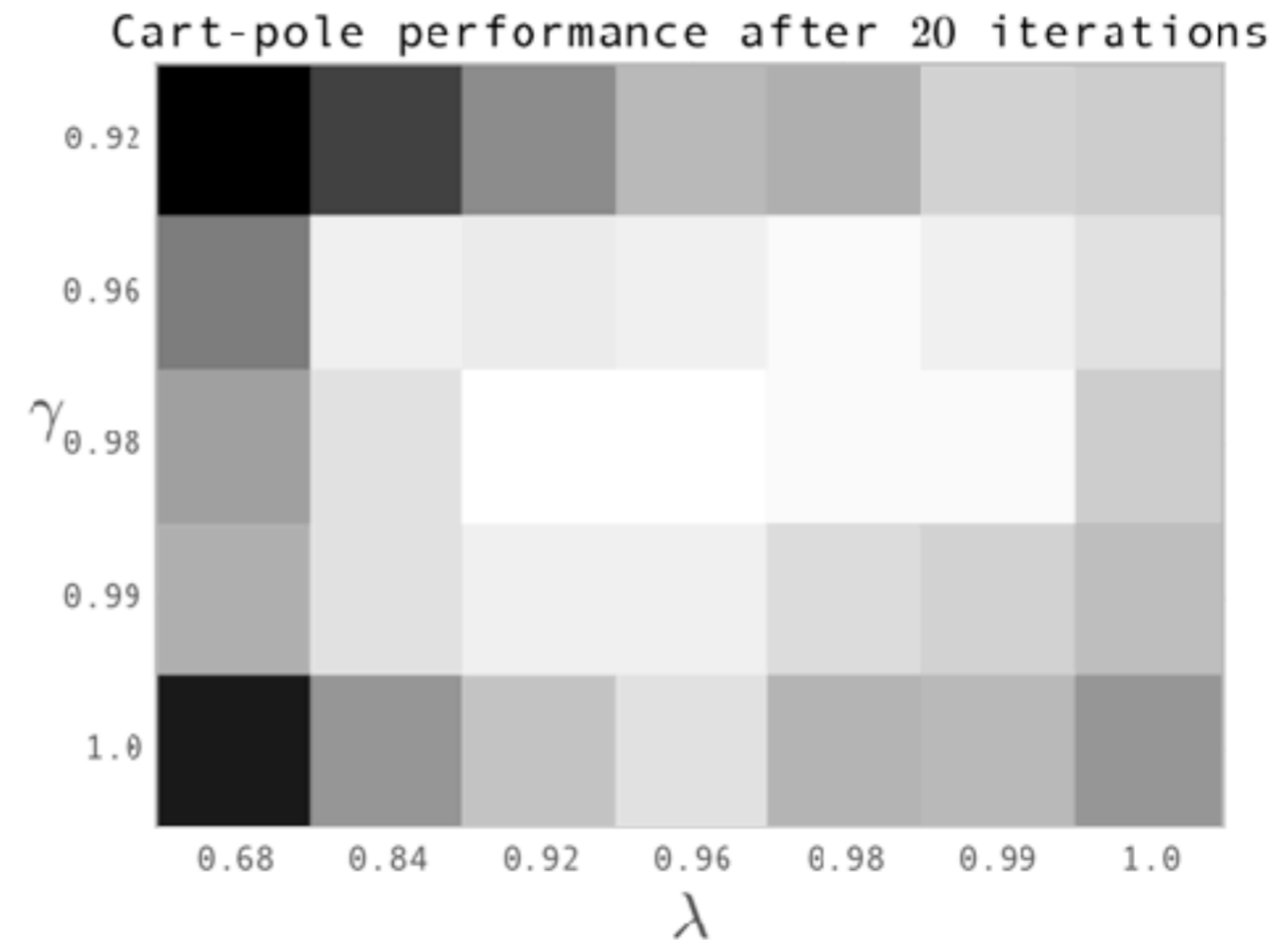
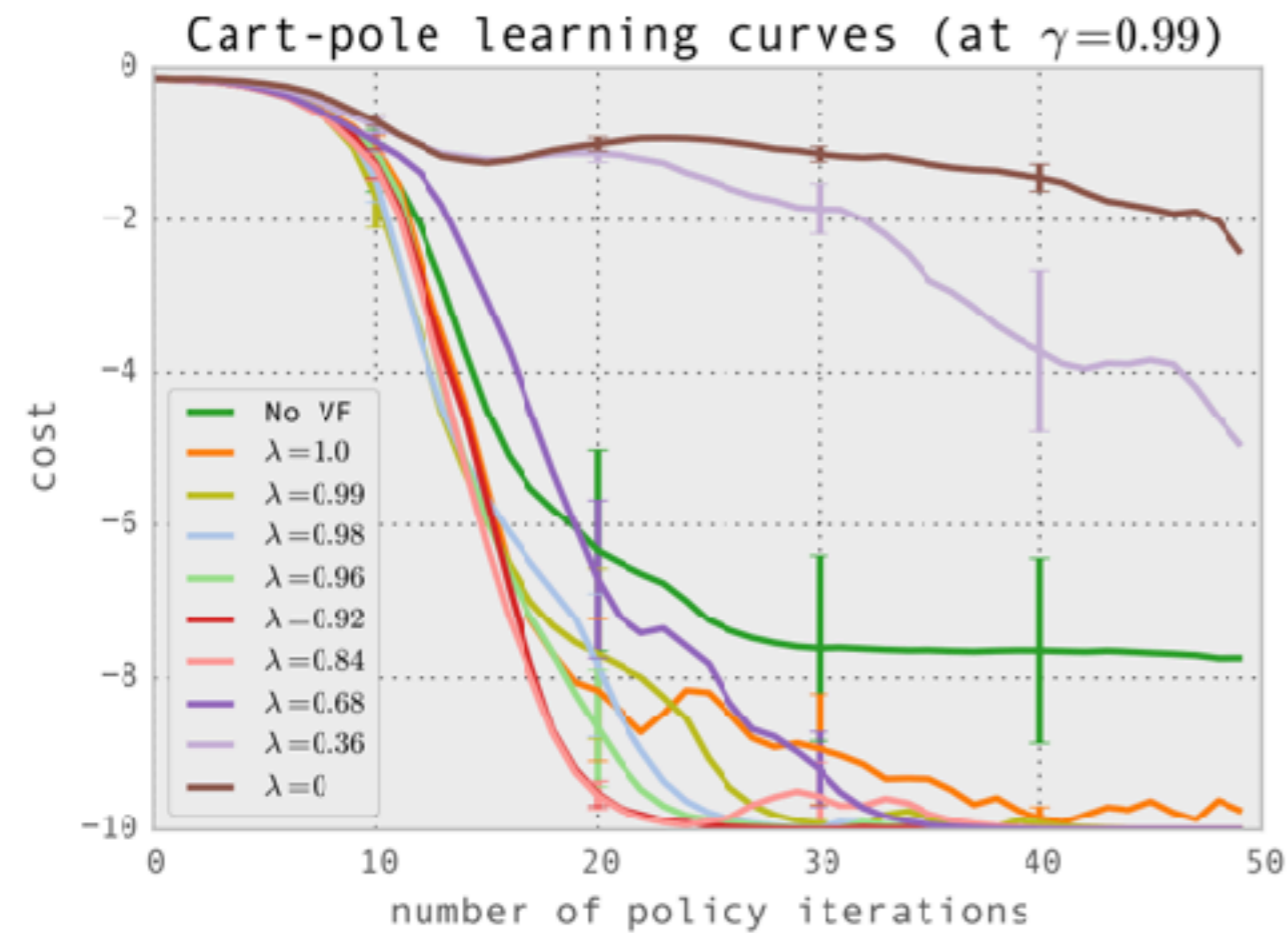
$$\hat{A}^{\lambda, \gamma} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{A}^{(n)} \quad \begin{array}{l} \text{advantage estimate} \\ \text{from state-value function} \end{array}$$

lambda sets trade-off between variance and bias



# Generalized Advantage function Estimation [ICLR 2016]

<https://arxiv.org/pdf/1506.02438.pdf>



- Policy gradient with advantage function
- Results on inverse pendulum



# Primal task - policy gradients for deterministic policy

## DDPG actor-critic method [Lilicrap et al. 2015]

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$ . initialize  $\theta = \text{rand}$
2. Estimate  $y = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_{\theta}(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning  $\mathbf{u}'$

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

Approximated Q-learning



Primal task - policy gradients for deterministic policy

DDPG actor-critic method [Lilicrap et al. 2015]

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$ . initialize  $\theta = \text{rand}$
2. Estimate  $y = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning  $\mathbf{u}'$

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

4. Learn policy  $\pi_\omega$  which do actions maximizing the state-action value function on the collected trajectories

$$\arg \max_{\omega} \sum_{\mathbf{x} \in \tau} Q_\theta(\mathbf{x}, \pi_\omega(\mathbf{x}))$$

Direct policy optimization on Q



Primal task - policy gradients for deterministic policy

DDPG actor-critic method [Lilicrap et al. 2015]

1. Collect trajectories  $\tau_1, \tau_2, \tau_3, \dots$ . initialize  $\theta = \text{rand}$
2. Estimate  $y = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q_\theta(\mathbf{x}', \mathbf{u}')$
3. Update parameters by learning  $\mathbf{u}'$

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

4. Learn policy  $\pi_\omega$  which do actions maximizing the state-action value function on the collected trajectories

$$\arg \max_{\omega} \sum_{\mathbf{x} \in \tau} Q_\theta(\mathbf{x}, \pi_\omega(\mathbf{x}))$$

Direct policy optimization on Q

Considering deterministic policy, gradient is simply chain-rule:

$$\sum_{(\mathbf{x}, \mathbf{a}) \in \tau} \frac{\partial Q_\theta(\mathbf{x}, \mathbf{a})}{\partial \mathbf{a}} \frac{\partial \pi_\omega(\mathbf{x})}{\partial \omega}$$





# Known successes of RL - Starcraft II

- Computer games controlled from pixel inputs
  - Atari 2D plafformers
  - 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



# Known successes of RL - Starcraft II

- Starcraft II (Deepmind AlphaStar 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, units, 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 training small RL agents



## Known successes of RL - Starcraft II

- **Starcraft II** (Deepmind AlphaStar beaten top-end professional human gamers 5:0)
- **Supervised learning** from anonymised human games (performance: (i) humans - gold level, (ii) AI - elite level)
- 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.
- **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:

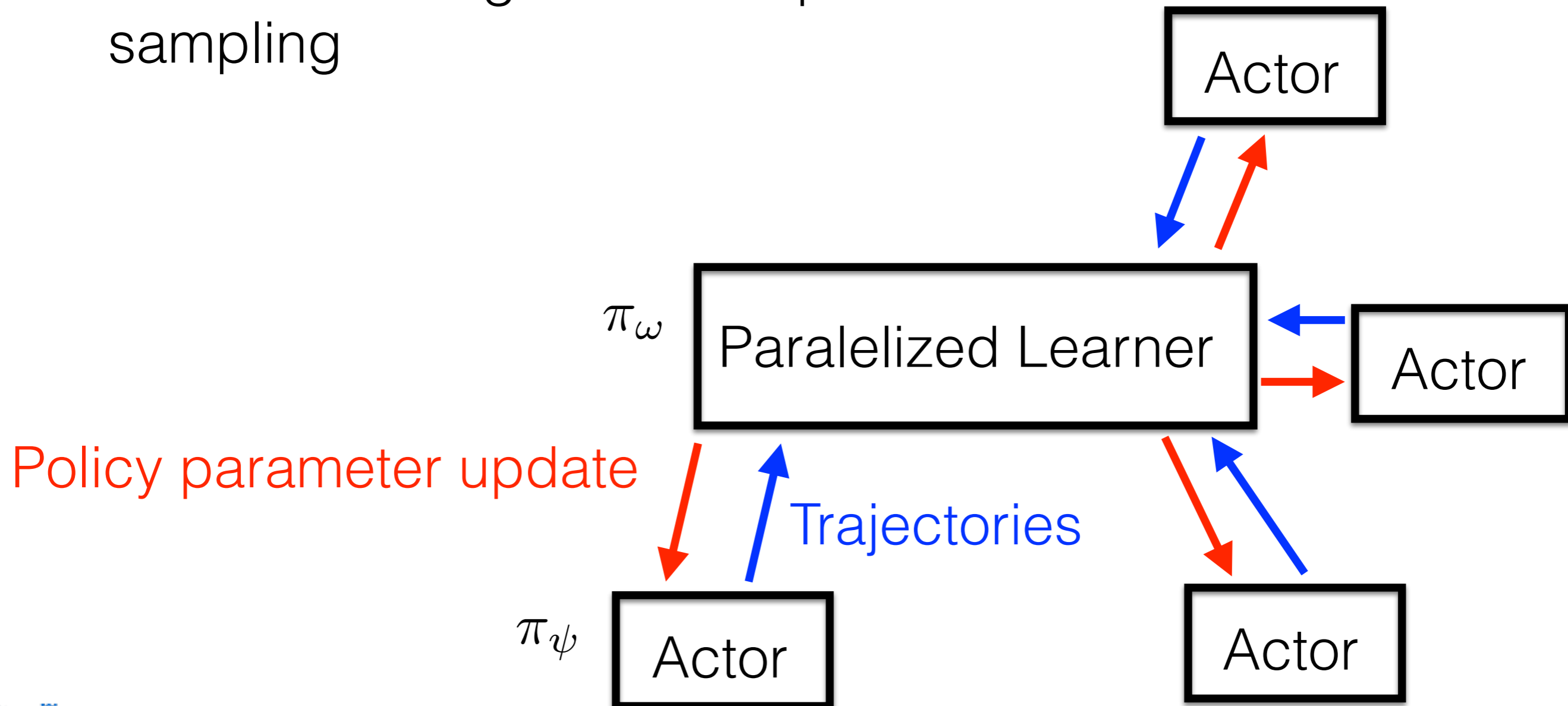
$$\mathbb{E}_{\tau \sim p(\tau | \pi_\omega)} \left[ \sum_{(\mathbf{x}, \mathbf{u}) \in \tau} \frac{\partial \log \pi_\omega(\mathbf{u} | \mathbf{x})}{\partial \omega} \cdot Q_\theta(\mathbf{x}, \mathbf{u}) \right] \approx \sum_k \frac{\partial \log \pi_\omega(\mathbf{u}_k | \mathbf{x}_k)}{\partial \omega}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

- recurrent policy architecture with LSTM blocks
- parallelized learning



# parallelized learning

- parallelized learning
- Actors delayed wrt learner => policy being updated  $\pi_\omega$  is different from the one which collected trajectories  $\pi_\psi$
- rewards re-weighted via importance sampling



# parallelized learning

- importance sampling

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

$\pi_\omega$  ...current  
 $\pi_\psi$  ...old





# parallelized learning

- importance sampling

$$\mathbb{E}_{\tau \sim p(\tau | \pi_\omega)} \left[ \underbrace{\sum_{(\mathbf{x}, \mathbf{u}) \in \tau} \frac{\partial \log \pi_\omega(\mathbf{u} | \mathbf{x})}{\partial \omega} \cdot Q_\theta(\mathbf{x}, \mathbf{u})}_{g(\tau)} \right] \quad \begin{array}{l} \pi_\omega \quad \dots \text{current} \\ \pi_\psi \quad \dots \text{old} \end{array}$$

$$\begin{aligned} \mathbb{E}_{\tau \sim p(\tau | \pi_\omega)} [g(\tau)] &= \int_T p(\tau | \pi_\omega) g(\tau) = \int_T \frac{p(\tau | \pi_\omega)}{p(\tau | \pi_\psi)} p(\tau | \pi_\psi) g(\tau) = \\ &= \mathbb{E}_{\tau \sim p(\tau | \pi_\psi)} \left[ g(\tau) \frac{p(\tau | \pi_\omega)}{p(\tau | \pi_\psi)} \right] \quad \begin{array}{l} \text{recalibrated} \\ \text{estimate} \end{array} \end{aligned}$$



# parallelized learning

- importance sampling

$$\mathbb{E}_{\tau \sim p(\tau|\pi_\omega)} \underbrace{\left[ \sum_{(\mathbf{x}, \mathbf{u}) \in \tau} \frac{\partial \log \pi_\omega(\mathbf{u}|\mathbf{x})}{\partial \omega} \cdot Q_\theta(\mathbf{x}, \mathbf{u}) \right]}_{g(\tau)} \quad \begin{array}{l} \pi_\omega \quad \dots \text{current} \\ \pi_\psi \quad \dots \text{old} \end{array}$$

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

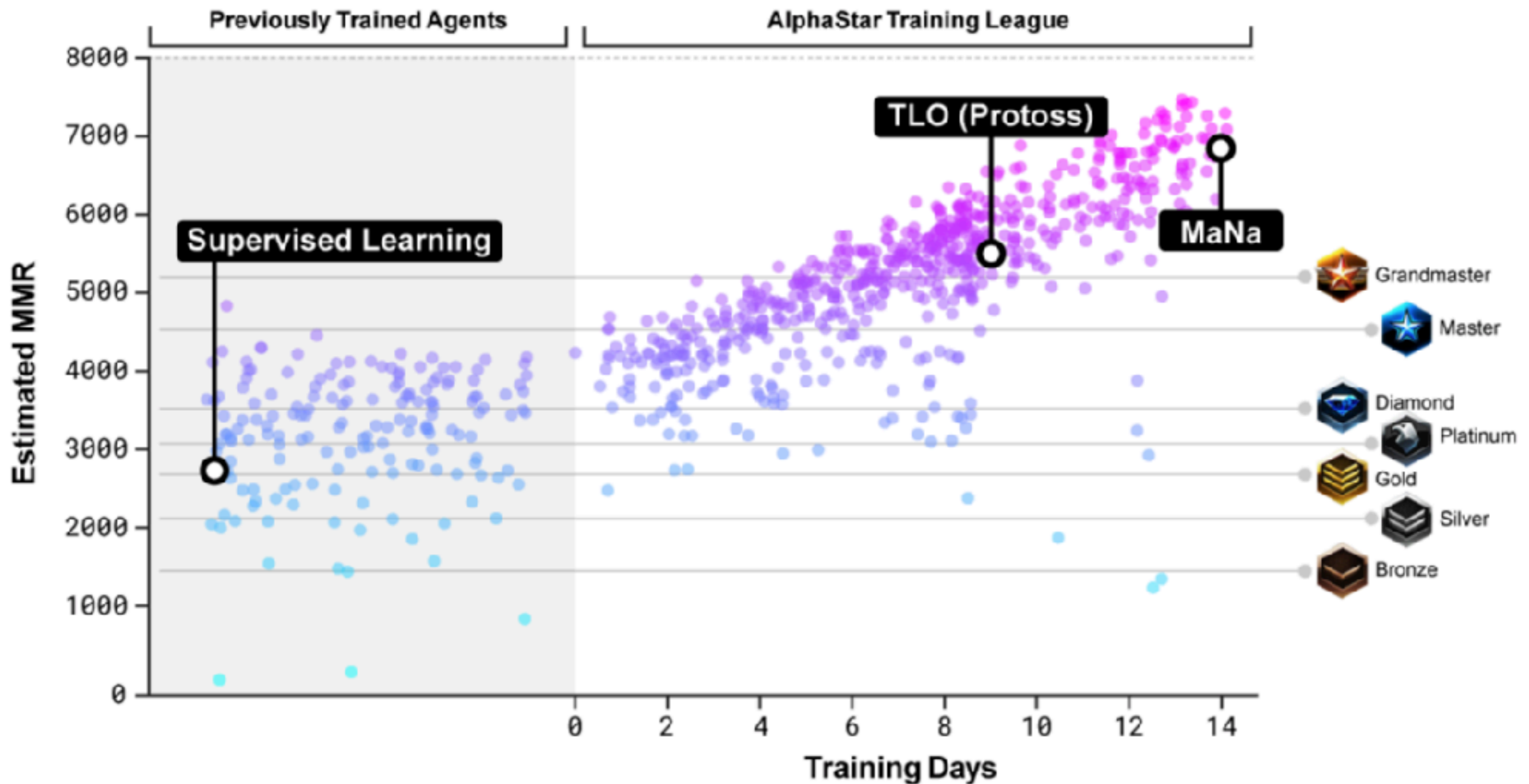
- recalibrated policy gradient estimate

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



# Known successes of RL - Starcraft II

- **Reinforcement learning:** 14 days playing against two grand masters (TLO, MaNa)



## Known successes of RL - Starcraft II

- **Reinforcement learning:** 14 days playing against two grand masters (TLO, MaNa)
- **Evaluation:**
  - When zoomed-out hi-res camera input used AlphaStar learned to consistently win against MaNa.
  - Rematch with the same human camera input as provided for humans AlphaStar lost the game against Mana.



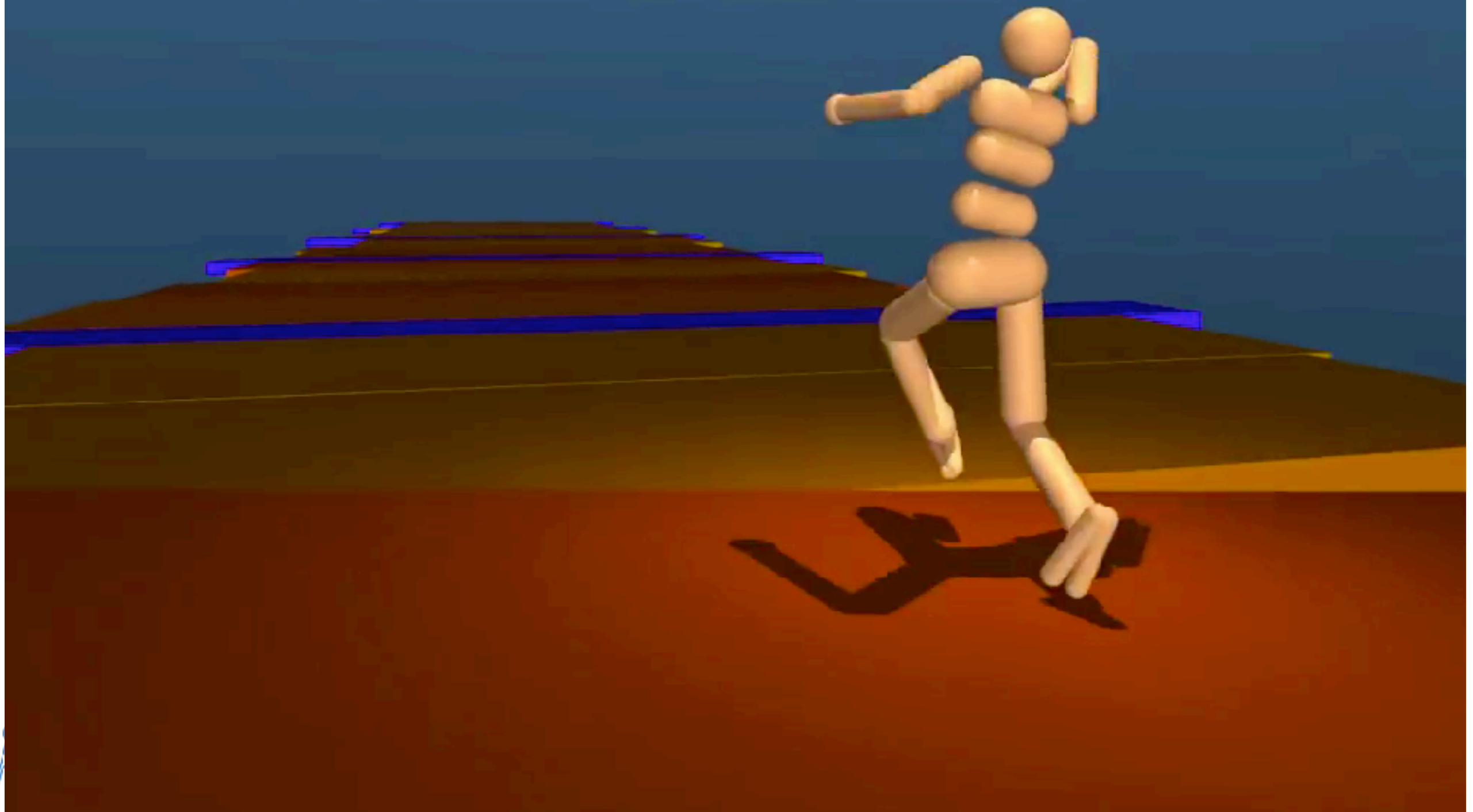
# Known successes of RL

- 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>

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



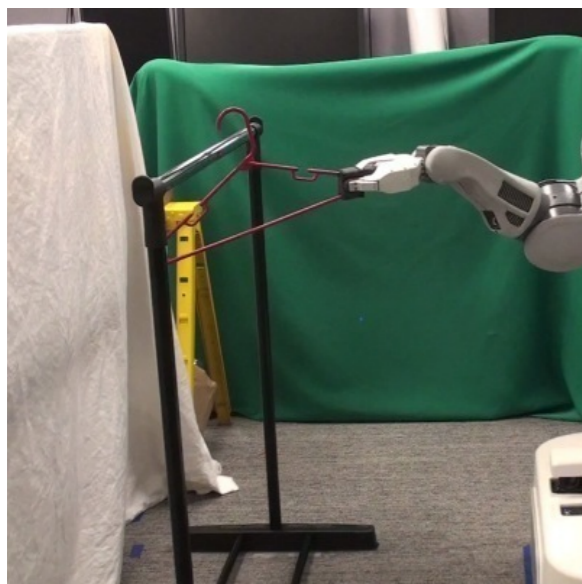
# Known successes of RL

- Application on real robots is still questionable since
  - transfer from simulator suffers from domain bias
  - direct training on robots is impossible due to sample inefficiency of state-of-the-art methods.
- Few known successes in robotics

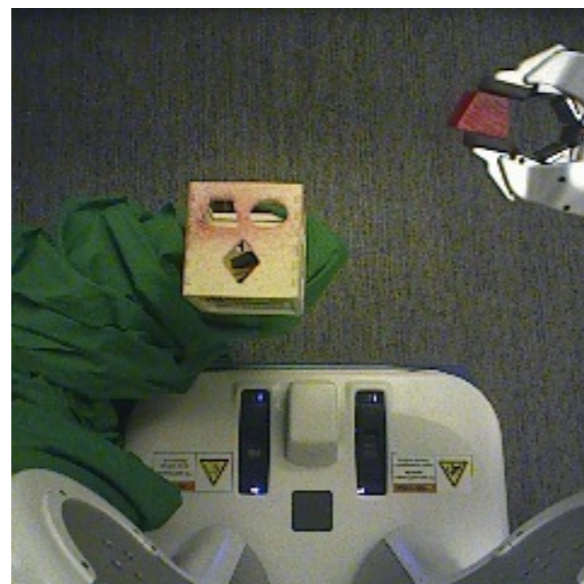


# Levine et al JMLR 2016

- guides policy gradient method by optimal trajectories
- **state space:** RGB camera images
- **action space:** motor torques



(a) hanger



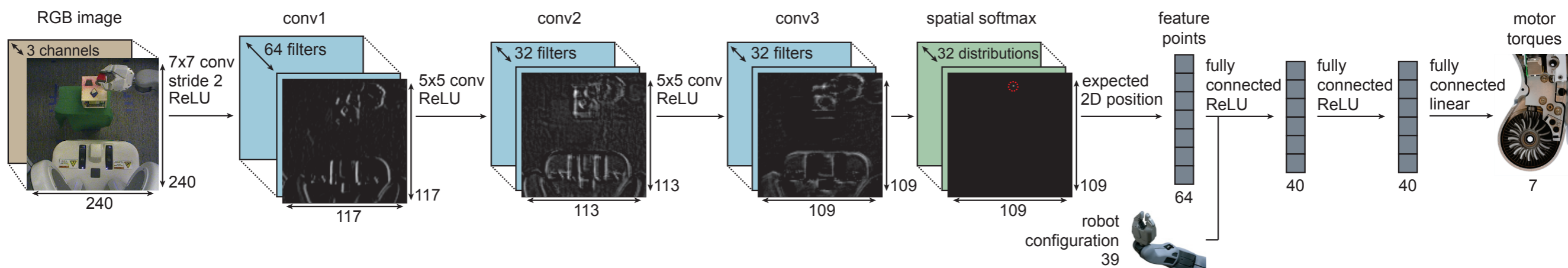
(b) cube



(c) hammer



(d) bottle





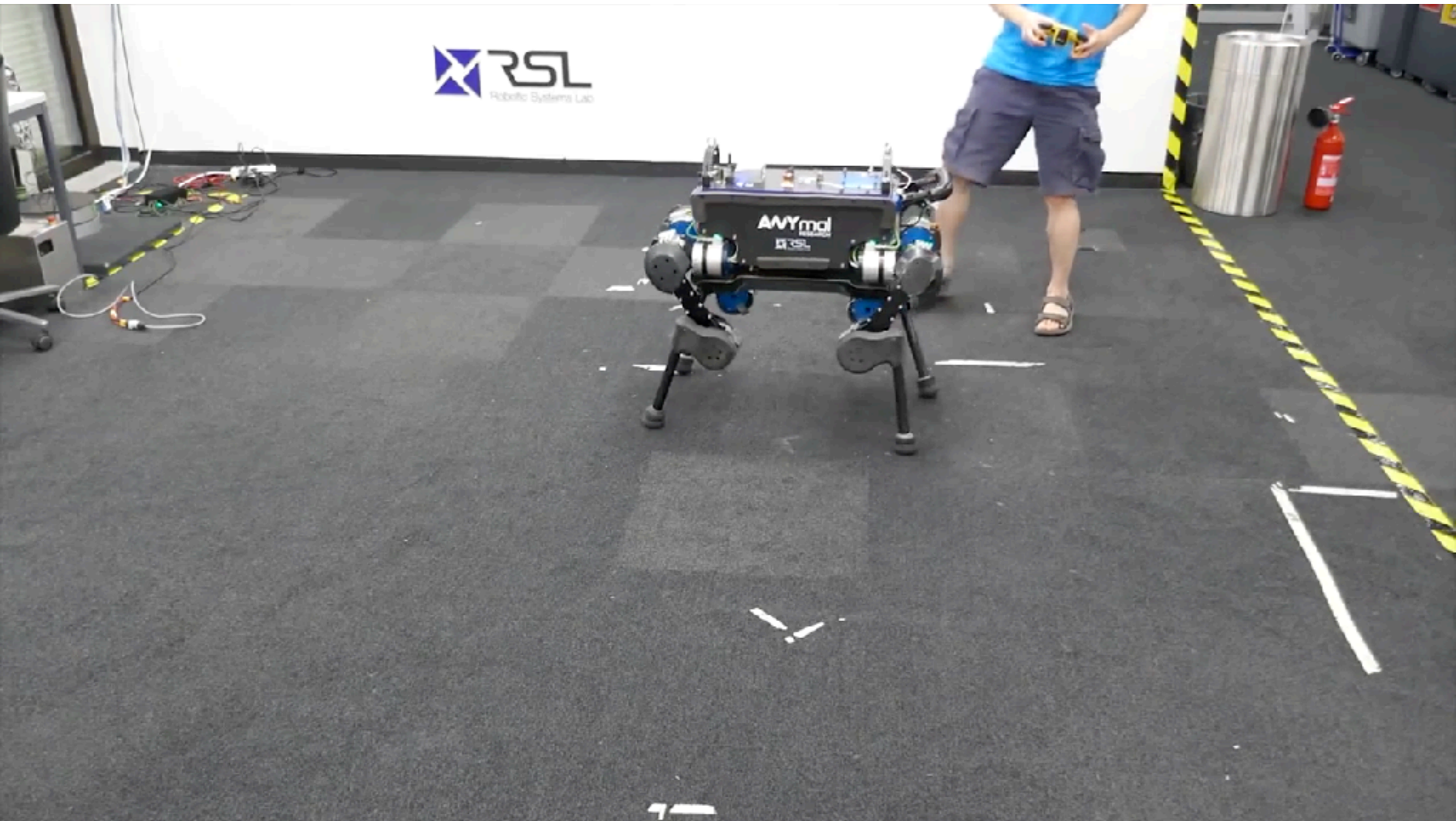
Levine et al JMLR 2016

# Learned Visuomotor Policy: Bottle Task



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

No visual inputs + flat terrain => simple domain transfer



[Hwangbo, ETH Zurich, Science Robotics, 2018]



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

# Motion and compliance control of flippers



[3] Pecka, Zimmermann, Svoboda, et al.

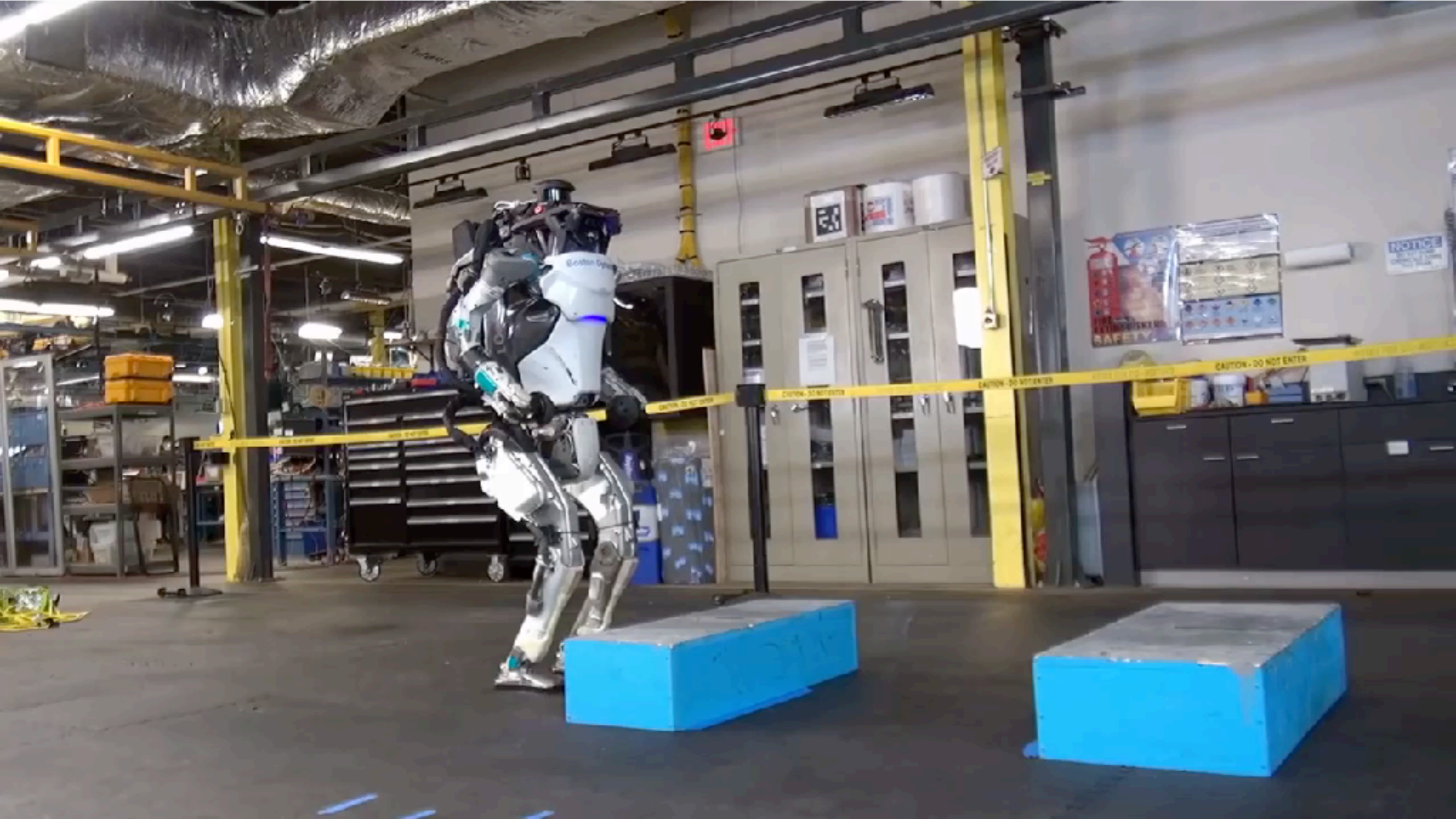
**IROS/RAL/TIE(IF=6)**, 2015-2018

©2018 Technical University of Prague

Faculty of Electrical Engineering, Department of Cybernetics



# Boston dynamics - Atlas - NO RL AT ALL



# Boston dynamics - Big dog - NO RL AT ALL



# Summary

- If accurate differentiable motion model and reward functions are known, than optimal control in MDP is straightforward optimization problem (efficiently tackled by DP or DDP)
- 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)
- **Domain transfer is main bottleneck for real application !!!!**

