SELF-TRANSFER LEARNING

Hélène Plisnier,
Denis Steckelmacher,
Ann Nowé

VUB VRIJE UNIVERSITEIT BRUSSEL

TRANSFER LEARNING IN RL

- two tasks: a **source** task and a **target** task
- goal: make learning the target task **faster** than when learning it from scratch

TRANSFER LEARNING IN RL

- the source is generally somewhat **different** than the target
  - examples: goal, dynamics
- however, no real consensus on **how different**
  - at least same robot ?

TRANSFER LEARNING IN RL

- the agent that has learned the source is the **advisor** of the agent learning the target

TL ALGORITHMS

- different transferables
- different techniques
- $\rightarrow$ we transfer a policy π using Policy Shaping
  - **Probabilistic Policy Reuse (PPR)**
  - **Policy Intersection (PI)**

PROBABILISTIC POLICY REUSE

An advisory policy $\pi_A(s_t)$ gets to decide which action to take with probability ψ:

$$a_t \sim \begin{cases} \pi_A(s_t) & \text{with probability } \psi \\ \pi_L(s_t) & \text{with probability } 1-\psi \end{cases}$$

- looks like ϵ-Greedy

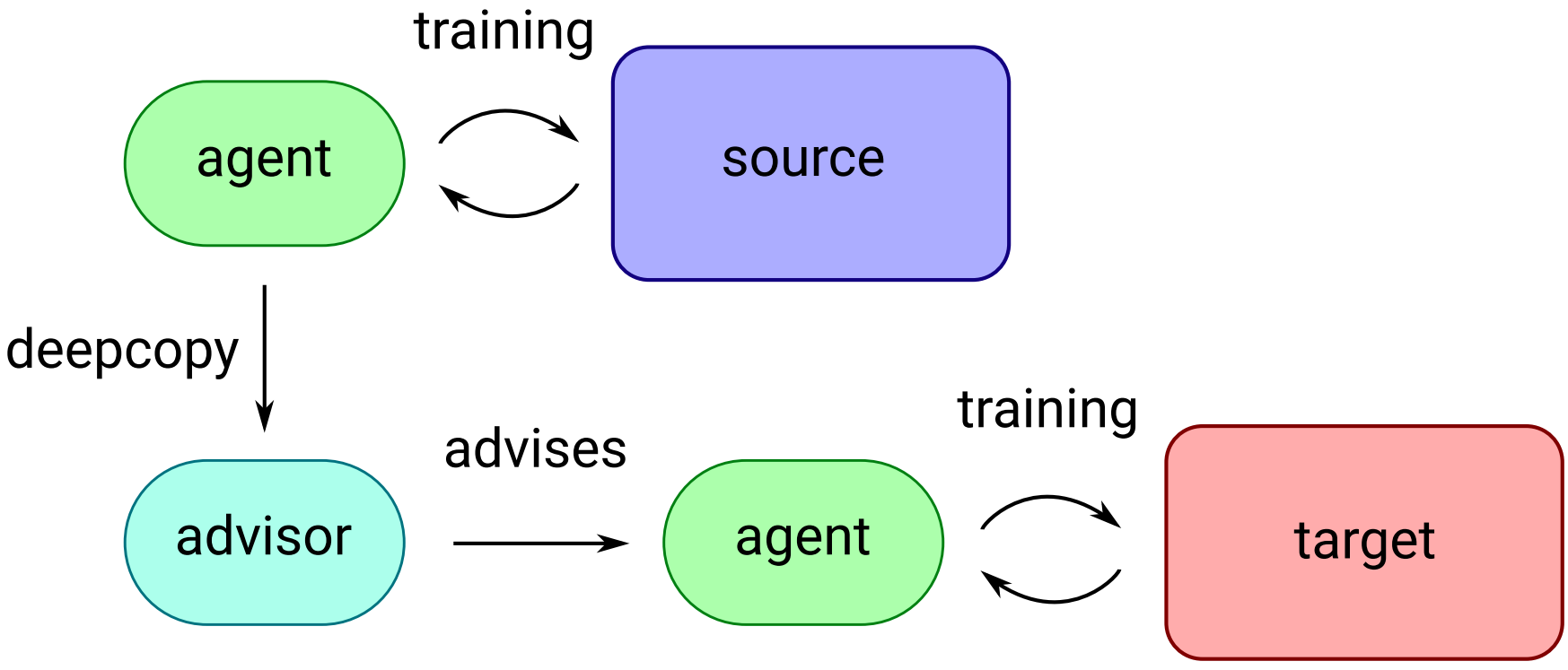
POLICY INTERSECTION

An advisory policy $\pi_A(s_t)$ is mixed with the agent's policy $\pi_L(s_t)$ before an action a_t is sampled:

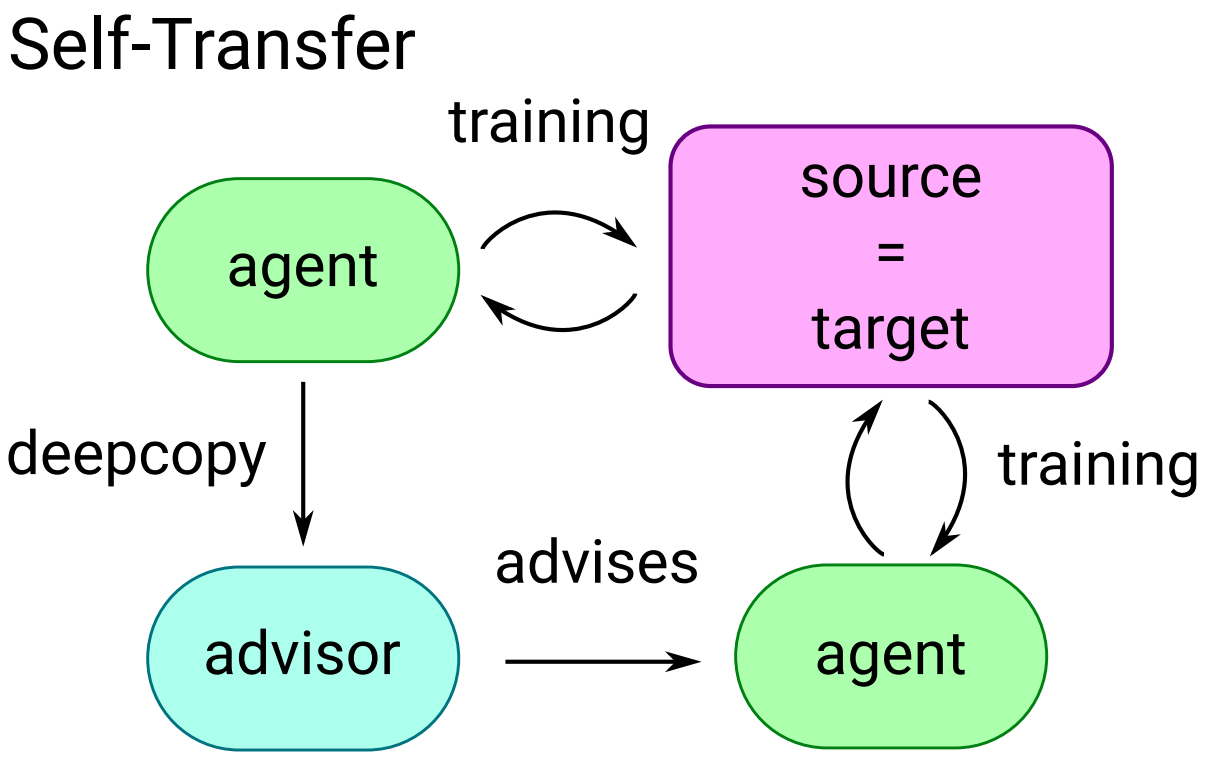
$$a_t \sim \underbrace{\frac{\pi_A(s_t)\pi_L(s_t)}{\pi_A(s_t) \cdot \pi_L(s_t)}}_{\sum_{a \in A} \pi_A(a|s_t)\pi_L(a|s_t)}$$

- acts as an intersection between the advisor and the agent

Transfer


training
agent
source
deepcopy
advises
training
advisor
agent
target

CONTRIBUTION

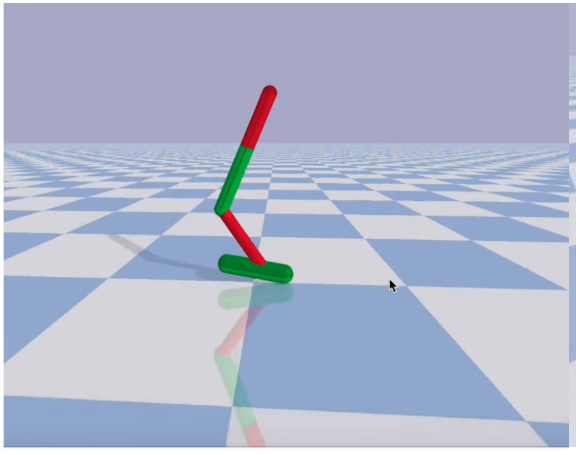


- different levels of training
- different initialization

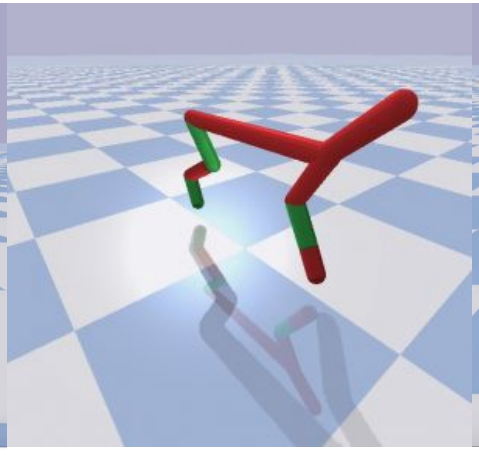
Dual Policy Distillation, Kwei-Herng Lai, Daochen Zha, Yuening LiandXia Hu, IJCAI 2020

- launches two agents in two separate instances of the same environment
- with different initialization
- they regularly distill their policy in each other's policy

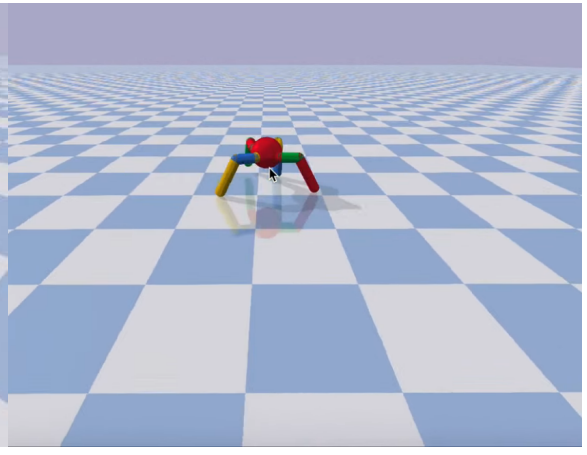
PYBULLET CONTINUOUS CONTROL ENVS



hopper



half-cheetah



ant

- Need to adapt Policy Intersection for continuous actions

POLICY INTERSECTION FOR CONTINUOUS ACTIONS

Require: π_L is the currently learning actor, and π_A is the frozen actor used as advisor

$A^A =$ set of actions sampled from $\pi_A(s_t)$

for every action $a_i^A \in A^A$ **do**

Get probability $\pi_L(a_i^A|s_t)$

Compute possibility $p_i = \frac{\pi_L(a_i^A|s_t)}{max_{a \in A^A} \pi_L(a|s_t)}$

if $p_i > p \sim U(0,1)$ **then**

Execute action a_i^A

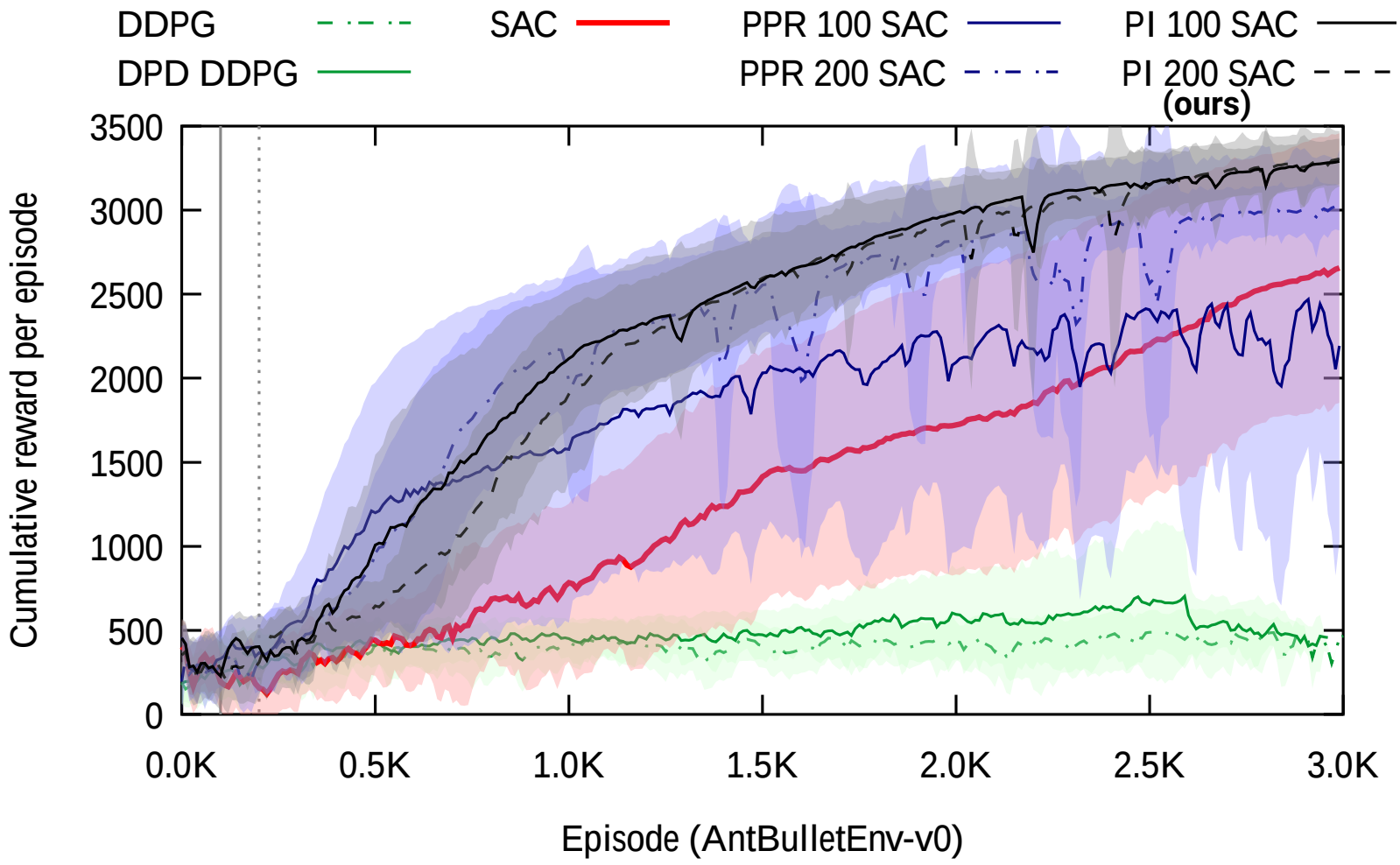
return

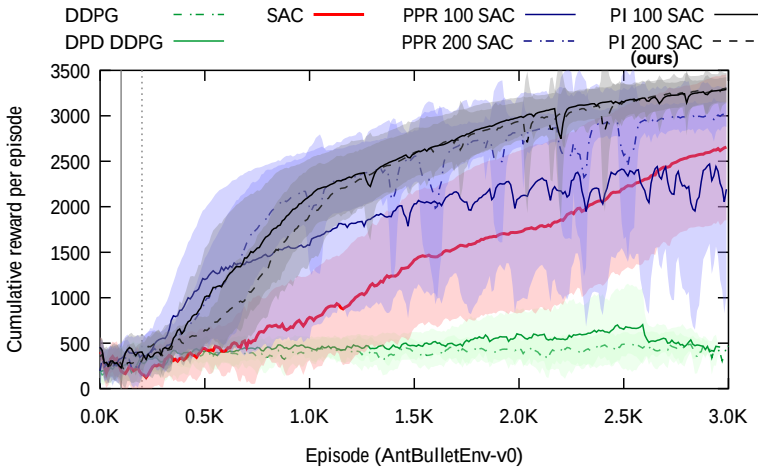
end if

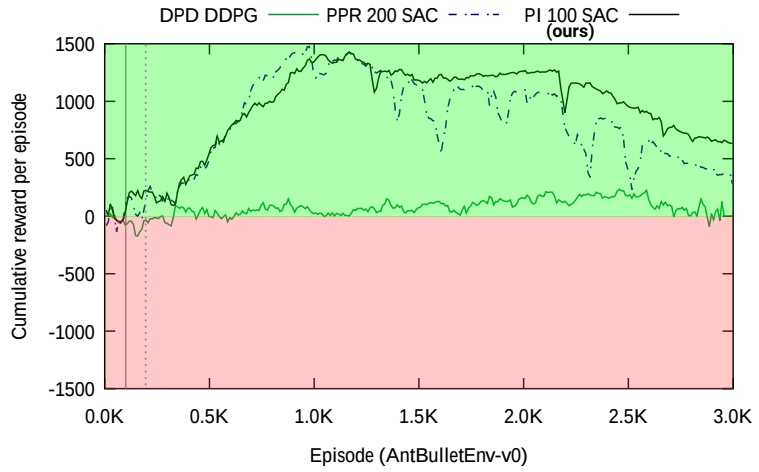
end for

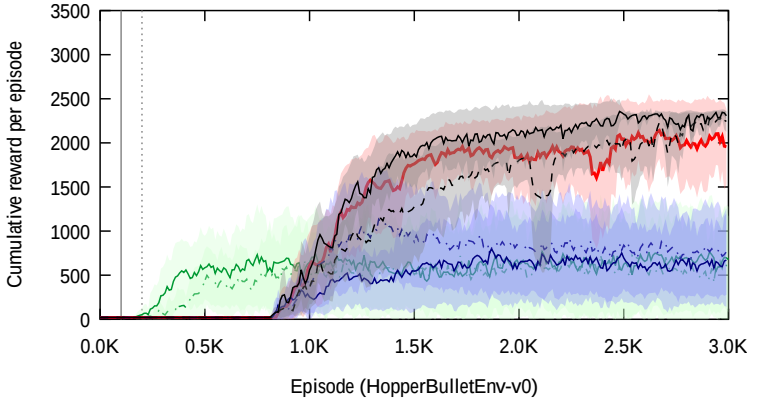
Execute action $a \sim \pi_L(s_t)$

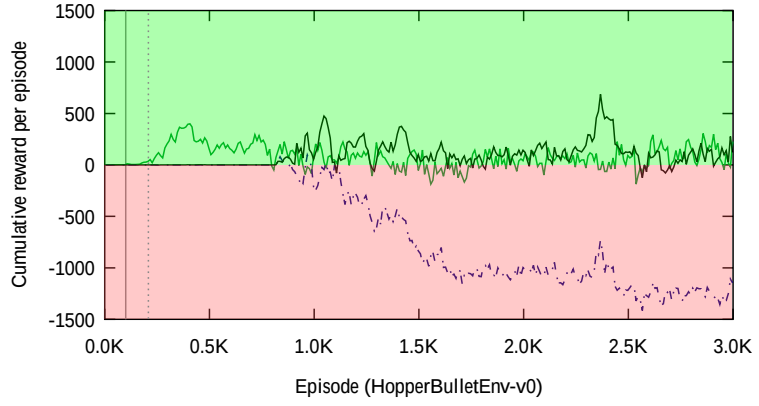
RESULTS

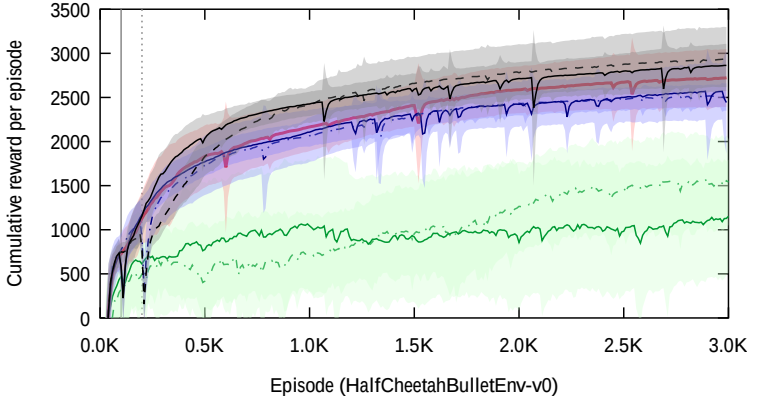

DDPG
SAC
PPR 100 SAC
PI 100 SAC
DPD DDPG
PPR 200 SAC
PI 200 SAC
(ours)
Cumulative reward per episode
3500
3000
2500
2000
1500
1000
500
0
0.0K
0.5K
1.0K
1.5K
2.0K
2.5K
3.0K
Episode (AntBulletEnv-v0)

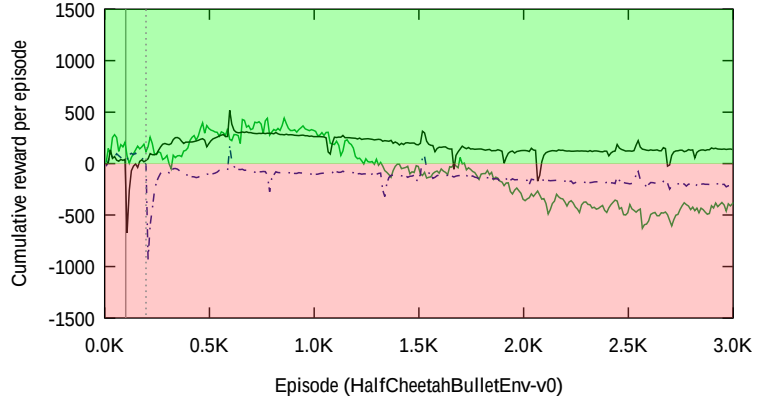

DDPG
SAC
PPR 100 SAC
PI 100 SAC
DPD DDPG
PPR 200 SAC
PI 200 SAC
(ours)
Cumulative reward per episode
Episode (AntBulletEnv-v0)


DPD DDPG
PPR 200 SAC
PI 100 SAC
(ours)
Cumulative reward per episode
Episode (AntBulletEnv-v0)


Cumulative reward per episode
Episode (HopperBulletEnv-v0)


Cumulative reward per episode
Episode (HopperBulletEnv-v0)


Cumulative reward per episode
Episode (HalfCheetahBulletEnv-v0)


Cumulative reward per episode
Episode (HalfCheetahBulletEnv-v0)

SPECULATIONS

Why does Self-Transfer helps?

- the advisor pushes for exploration
- different initialization
- compensate for neural networks errors