Reinforcement Learning

Bootcamp II

Machine Perception

14 May 2020

Advanced Interactive Technologies

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Last week

- Introduction to RL basics
- MDPs
- Bellman equation
- Exact (tabular) methods

This Lecture

First Part:

- TD-Learning

Second Part:

- Policy search methods
- Actor-Critic
- Deep Reinforcement Learning
- Current research in DRL

Dynamic Programming - What if the state space is too big?

- Let's say we are given a policy π
- And the state space is too big to iterate
- How can we estimate the value of those states?

Temporal Difference Learning

TD(0) Learning

Given a policy,

for each action a transitioning from s to s',

compute the difference to current estimate and update value function:

$$\Delta V(s) = r(s,a) + \gamma V(s') - V(s)$$

$$V(s) \leftarrow V(s) + \alpha \Delta V(s)$$

Here $\alpha > 0$ is the learning rate.

We don't update the whole state space, only visited states!

Exploration vs. Exploitation Dilemma

What policy can we use to interact with our environment?

Random policy In each state, choose an action randomly

Is this ideal?

- Visit states close to starting point more often
- Far away states get neglected

Greedy policy In each state, always choose the best action

Is this ideal?

- Can find reward quickly
- Getting stuck in local minimum

Exploration vs. Exploitation Dilemma

We need to find a balance between:

exploration: selecting random actions to visit unknown states

exploitation: selecting actions that are optimal given our current knowledge

What could be a good trade-off?

ε-greedy policy

In each state, with small probability ε choose randomly, else choose greedily

Is this ideal?

- Given its simplicity, it works well in practice
- Decreasing ε can improve it (more exploration in the beginning)
- More sophisticated approaches exist

Two major implemenations

- SARSA

  On-Policy:

  Computes the Q-Value according to a policy and then the agent follows that policy.

- Q-Learning

  Off-Policy:

  Computes the Q-Value according to a greedy policy, but the agent follows a different exploration policy.

Two major implemenations

- SARSA

  On-Policy:

  Computes the Q-Value according to a policy and then the agent follows that policy

- Q-Learning

  Off-Policy:

  Computes the Q-Value according to a greedy policy, but the agent follows a different exploration policy.

for time reason, we mainly focus on Q-learning, as many DRL are built on Q-learning

V-function vs. Q-function

Q-value function

- Explicitly takes actions into account

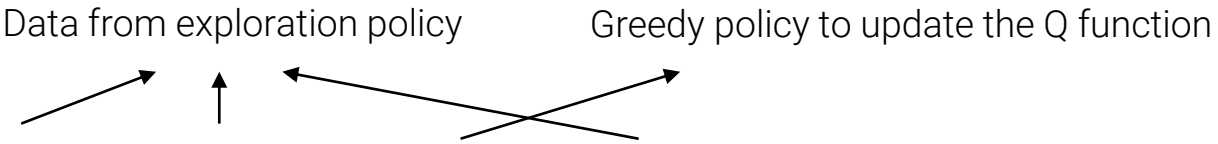
$$v_\pi(s) \doteq E_\pi[G_t|S_t = s]$$

$$q_\pi(s,a) \doteq E_\pi[G_t|S_t = s, A_t = a]$$

- Helps with problems when the transition function of the environment is not available
- When action space is large, this may be problematic

Q Learning

For each action a transitioning from s to s', compute the difference to current estimate and update value function:



$$\Delta Q(S,A) = R_{t+1} + \gamma \max_{a}\{Q(S',a)\} - Q(S,A)$$

$$Q(S,A) \leftarrow Q(S,A) + \alpha \Delta Q(S,A)$$

Here $\alpha > 0$ is the learning rate.

- Q learning is an off-policy algorithm because the policy we use to update the Q function is different from the policy we use to collect data (e.g. ε-greedy).

Example of Q Learning ($\gamma = 1$, step cost=0, $\alpha = 0.1$)

Initial Q function

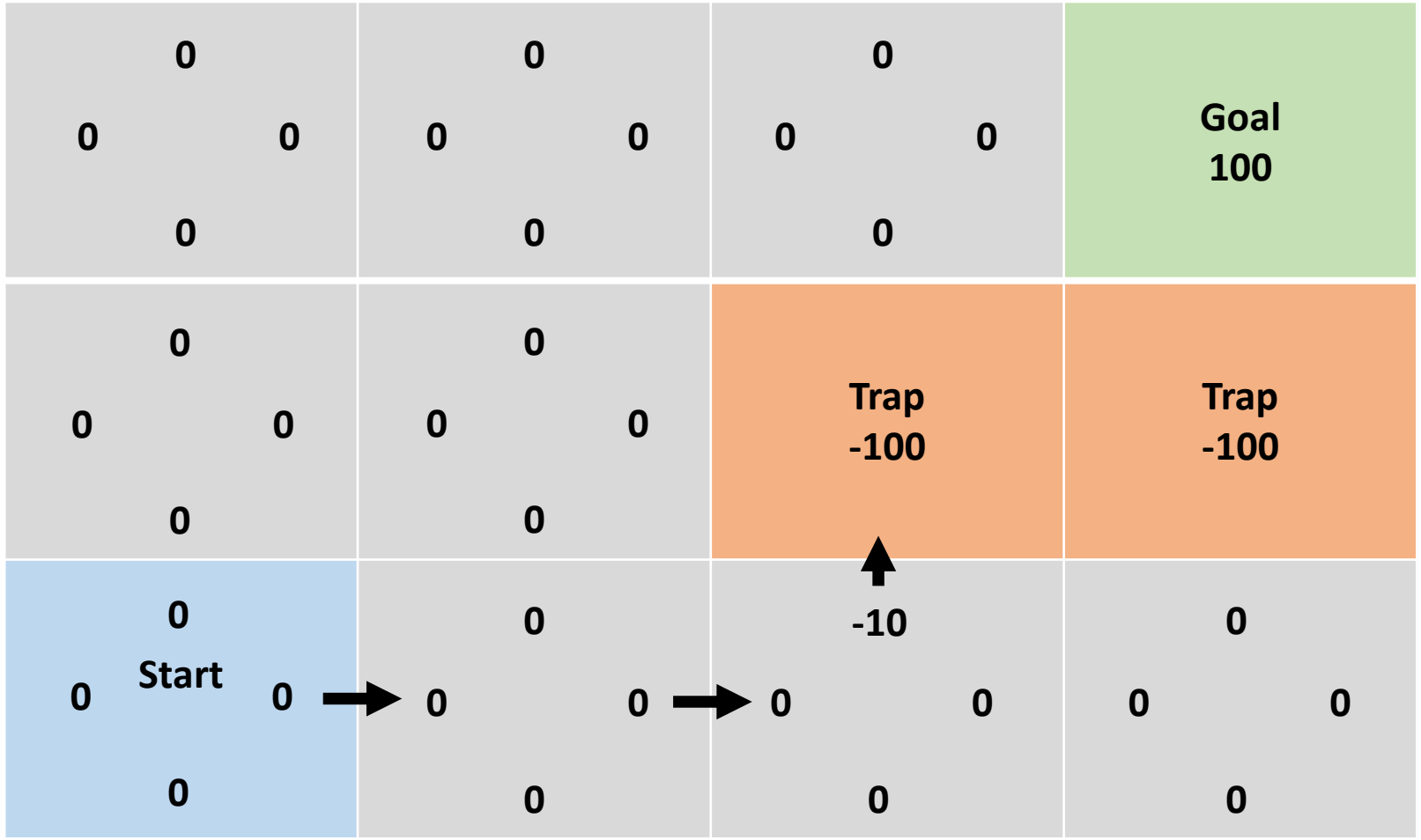
0 0 0 0	0 0 0 0	0 0 0 0	Goal 100
0 0 0 0	0 0 0 0	Trap -100	Trap -100
0 0 Start 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Example of Q Learning ($\gamma = 1$, step cost=0, $\alpha = 0.1$)

Episode 1: Step by step updates of Q-values

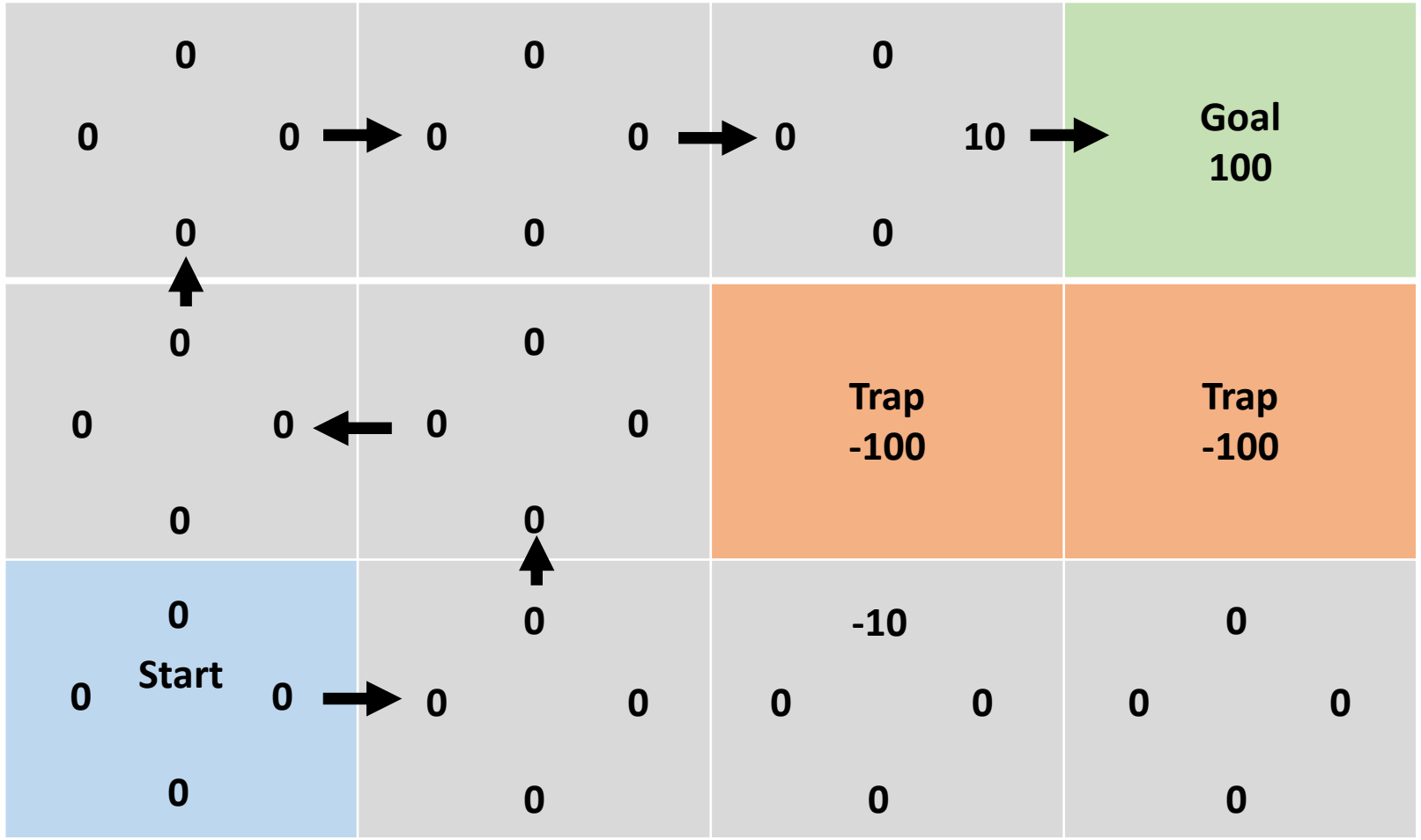
$$\Delta Q(S,A) = R_{t+1} + \gamma \max_{a}\{Q(S',a)\} - Q(S,A)$$

$$Q(S,A) \leftarrow Q(S,A) + \alpha \Delta Q(S,A)$$



Example of Q Learning ($\gamma = 1$, step cost=0, $\alpha = 0.1$)

Episode 2: Step by step updates of Q-values

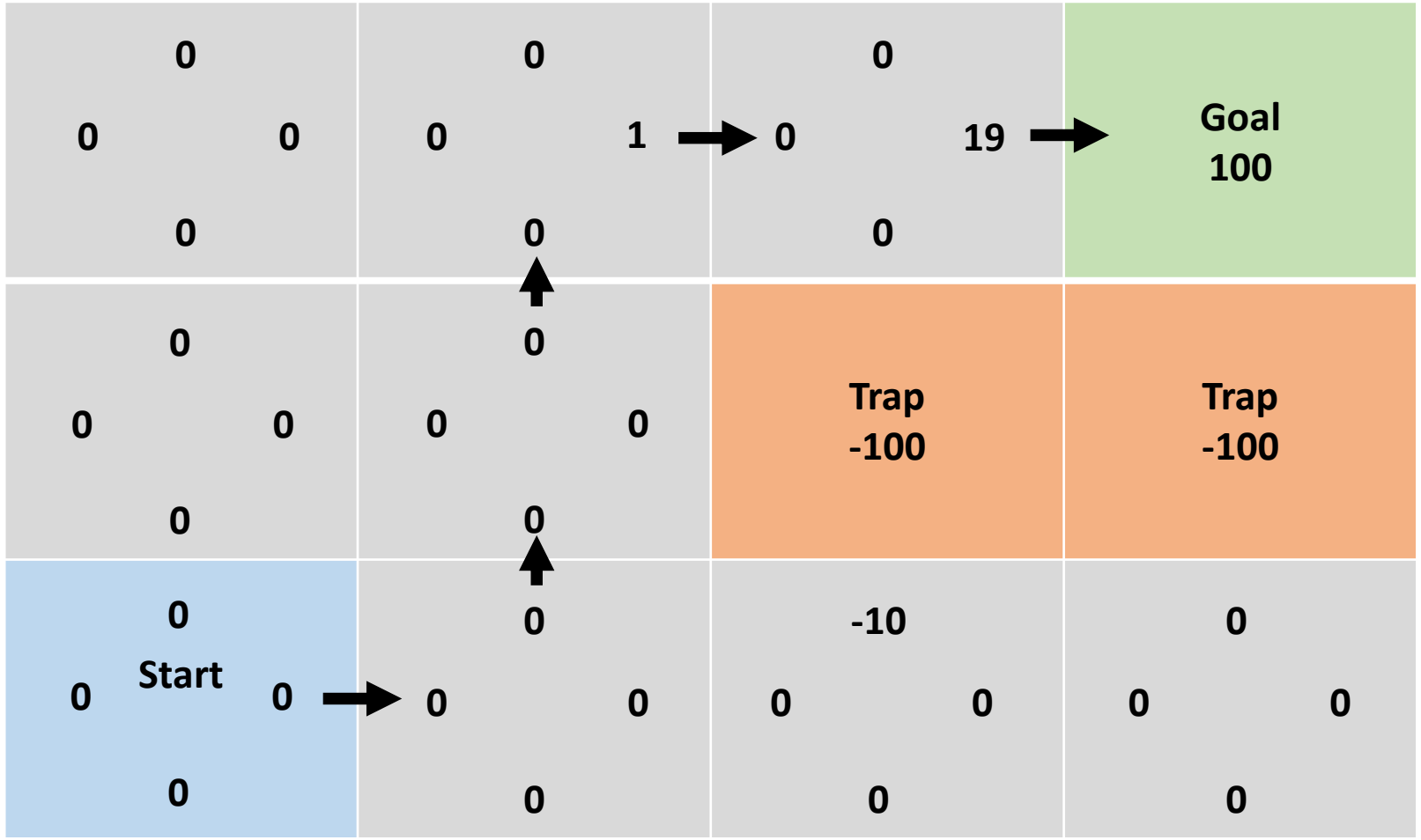


Example of Q Learning ($\gamma = 1$, step cost=0, $\alpha = 0.1$)

$$\Delta Q(S,A) = R_{t+1} + \gamma \max_a \{Q(S',a)\} - Q(S,A)$$

$$Q(S,A) \leftarrow Q(S,A) + \alpha \Delta Q(S,A)$$

Episode 3: Step by step updates of Q-values

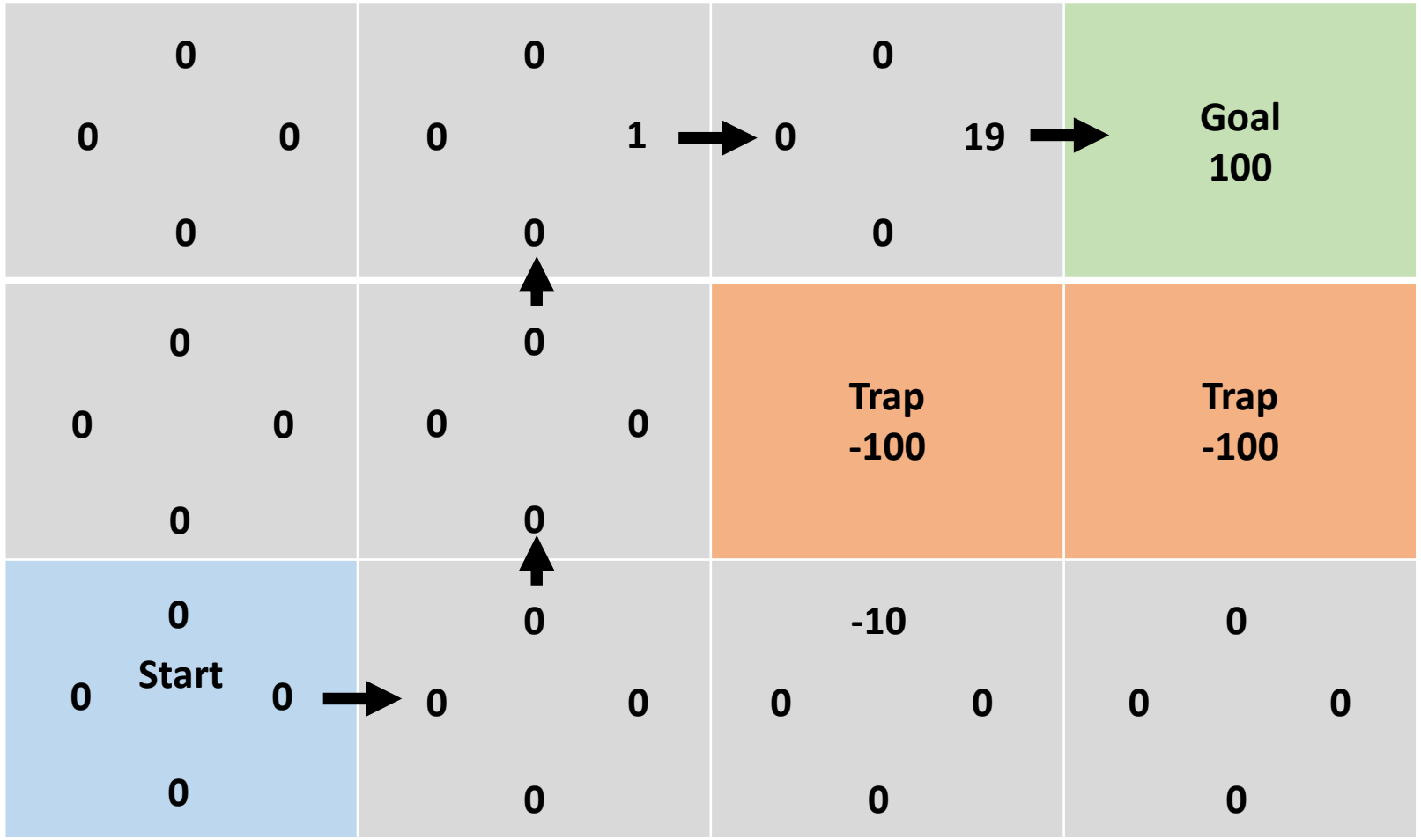


$0 + 0.1 \times (100 - 0)$

$= 10$

Example of Q Learning ($\gamma = 1$, step cost=0, $\alpha = 0.1$)

Keep running until convergence



Pros and Cons of Temporal Difference Learning

Pros:

- Less variance than Monte Carlo Sampling due to bootstrapping
- More sample efficient
- Do not need to know the transition probability matrix.

Cons:

- Biased due to bootstrapping
- Exploration/Exploitation dilemma
- Can behave poorly in stochastic environments

Overview Tabular Learning Methods

Dynamic Programming

- Uses value of neighbouring states to update current state (given π)
- Visits every state
- Requires knowledge of the transition probabilities

Monte Carlo Methods (not covered in detail)

- Run in episodes, update state values at end of episode.
- Experience based (no need to know system dynamics)
- High Variance

Temporal Differences

- Run in episodes, update state values at each step.
- Combination of DP and MC ideas
- Exploration/Exploitation → Local minima

Deep Reinforcement Learning

Introduction

Tabular Learning?

- We learn about each state separately
- No generalization
- For image-input and larger state spaces, Q-Learning is most likely to fail

We need a concept that works for such cases

- Needs to approximate values of states that have not been visited

Introduction

Policy $\pi : S_t \rightarrow A_t$

Value $v : S_t \rightarrow V(S_t)$

These are nothing else than **functions**, we need a way to represent and learn them.

→ Using function approximation to learn the value function!

$$\Delta Q(S,A) = R_{t+1} + \gamma \max_{a}\{Q(S',a)\} - Q(S,A)$$
$$Q(S,A) \leftarrow Q(S,A) + \alpha \Delta Q(S,A)$$

Deep Q-Learning

Previously, we assumed a separate Q-value estimate for each state-action pair (s, a).

We can use a **neural network** to learn the mapping between state-action pairs (s, a) and their values.

The Q-learning updates reduce to **stochastic gradient descent** on:

$$Loss(\theta) = (R + \gamma \max_{a'}\{Q_\theta(S', a')\} - Q_\theta(S, A))^2$$

We do not backprop gradients here

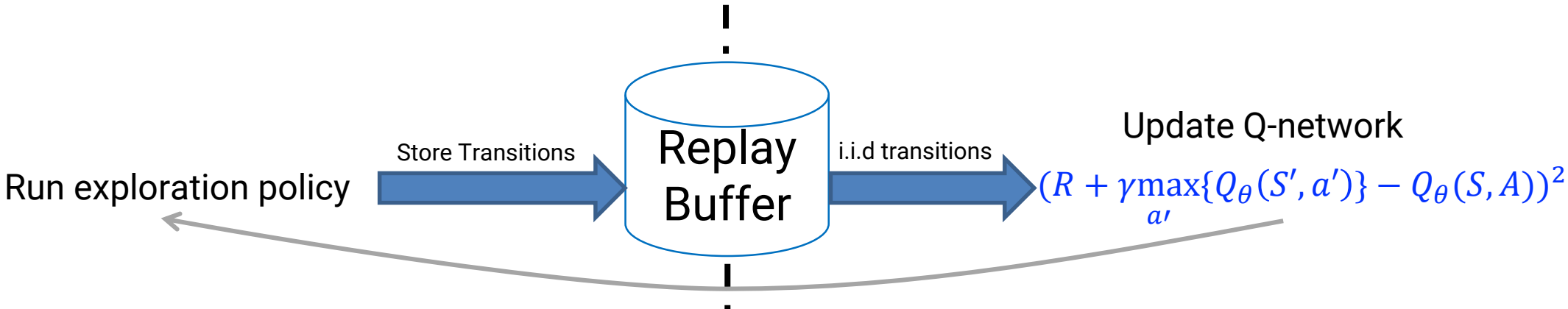
Deep Q-Networks

SGD assumes that the updates we apply are **i.i.d.**

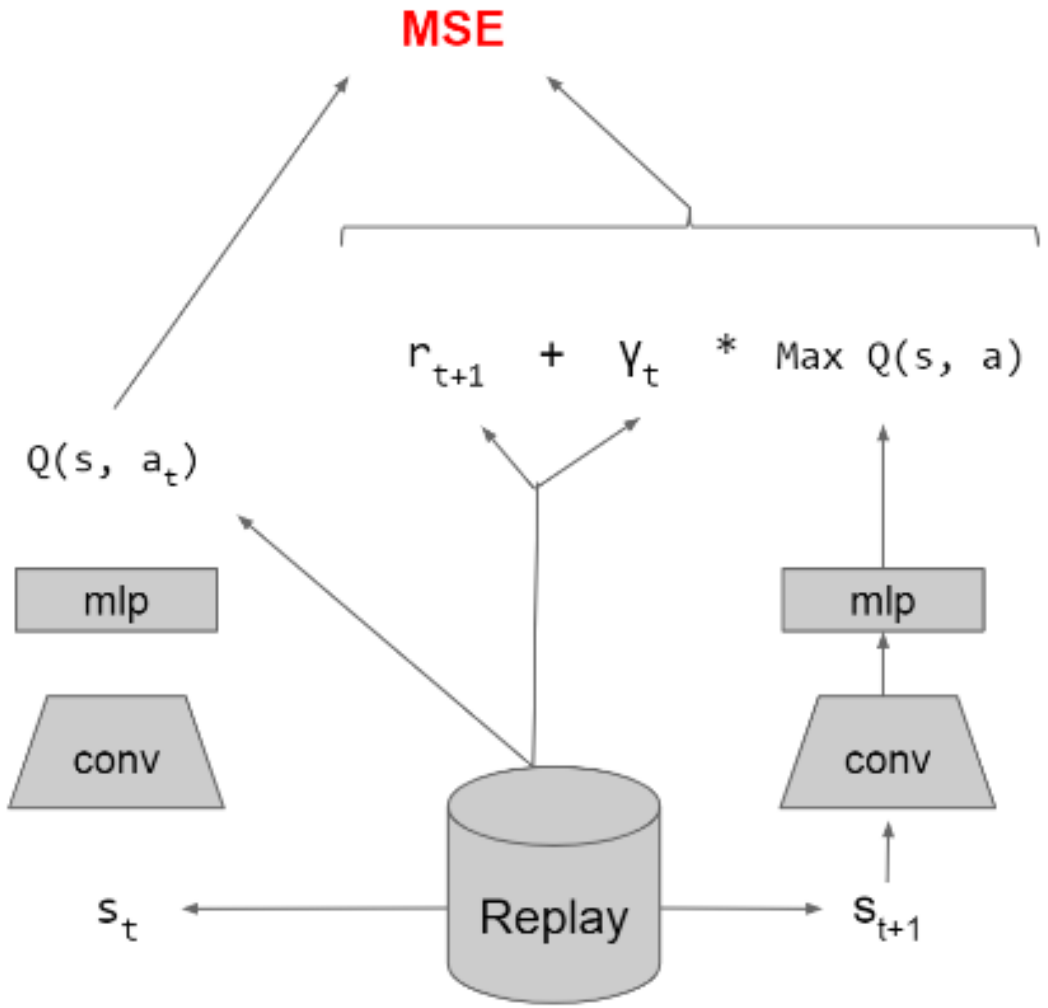
In RL, values of states visited in a trajectory are strongly correlated.

How can we address this?

- Use a **replay buffer** to store generated samples
- During updates, randomly sample transitions from the buffer
- Since Q-learning is off-policy, we are allowed to use old samples



Deep Q-Networks



https://github.com/xuedong/machine-learning-summer-school [Mnih et. al, 2013]

Deep Q-Networks

- Can achieve superhuman performance on benchmark Atari Games
- The observations are high-dimensional and hard to handle
- Often, most pixels in an image are irrelevant to the task at hand

[Mnih et. al, 2013]

Policy Search Methods

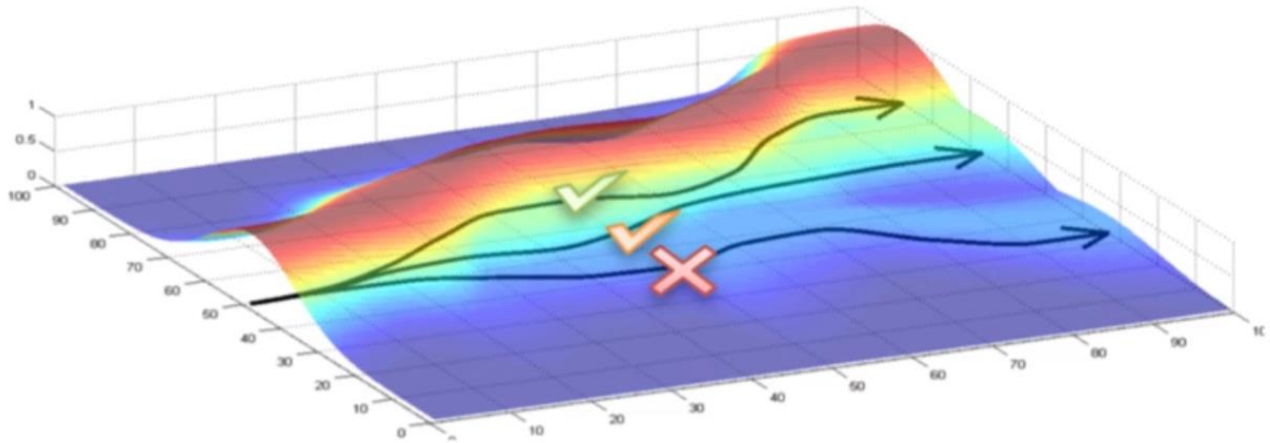
Introduction

- Q-learning is limited to discrete action spaces.
  - For continuous action space the problem is intractable.

- Learning a policy is often much easier. The algorithm directly learns the correct behavior without exploring the value function.

- We could pretrain the policy from human demonstration via supervised learning.

Policy Gradients



- Policy: $\pi(a_t|s_t) = N(\mu_t, \sigma_t^2|s_t)$

- Collect rollout trajectories:

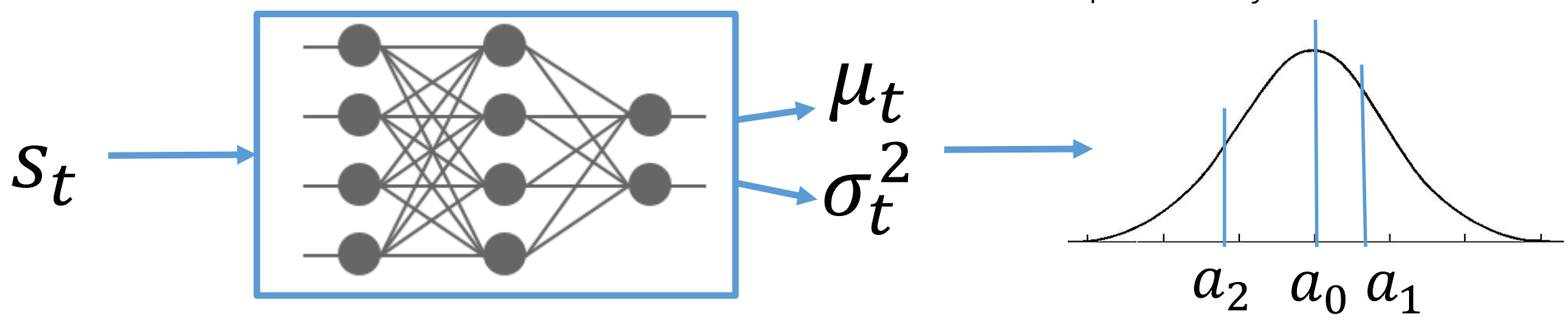
$$p(\tau) \underset{T}{=} p(s_1, a_1, \dots, s_T, a_T) =$$

$$p(s_1) \prod_{t=1} \pi(a_t|s_t) p(s_{t+1}|a_t, s_t)$$

- Update the policy:
  - Good trajectories made more likely
  - Bad trajectories made less likely

Policy

- Policy is a stochastic function approximator:



$$\pi(a_t|s_t,\theta)$$

θ- parameters of the neural network (or other approximator)

Exploration and Evaluation

- Exploration – collecting trajectory data
  - The agent samples action at every time-step from the policy probability distribution *(on-policy methods)* .
- Evaluation
  - We evaluate the policy by computing the expectation of the trajectory reward.

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t \gamma^t r(s_t, a_t) \right]$$

Overview

To Do:

- Policy updates
- Policy gradient derivation
- Algorithm using policy gradients
- Extension of policy gradient algorithm

Overview

To Do:

- Policy updates
  - How should we do updates on the policy parameters?
- Policy gradient derivation
- Algorithm using policy gradients
- Extension of policy gradient algorithm

Policy update

- The goal is to maximize the performance measure:

$$\theta^* = argmax\, J(\theta)$$

- We update parameters using gradient ascent:

$$\theta = \theta + \nabla_\theta J(\theta)$$

So far…

- Policy updates

To Do:

- Policy gradient derivation
  - How can we ensure that an update step through gradient ascent actually improves the policy?
  - Show that updates on the gradients of the performance measure can be done without knowledge of the environment dynamics
- Algorithm using policy gradients
- Extension of policy gradient algorithm

Policy Gradient

$\frac{dy}{dx}$ $y = \ln u \Rightarrow \frac{u'}{u}$, where y is func of x

$$p(\tau) = p(s_1, a_1, \ldots, s_T, a_T) = p(s_1)\prod_{t=1}^{T}\pi(a_t|s_t)p(s_{t+1}|a_t, s_t)$$

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t \gamma^t r(s_t, a_t)\right] = \mathbb{E}_{\tau\sim p(\tau)}[r(\tau)] = \int p(\tau)r(\tau)d\tau$$

$\frac{p(\tau)}{p(\tau)}\nabla p(\tau) = p(\tau)\frac{\nabla p(\tau)}{p(\tau)} = p(\tau)\nabla_\theta \log p(\tau)$

$$\nabla_\theta J(\theta) = \int \nabla_\theta p(\tau)r(\tau)d\tau = \int p(\tau)\nabla_\theta \log p(\tau)r(\tau)d\tau = \mathbb{E}_{\tau\sim p(\tau)}[\nabla_\theta \log p(\tau)r(\tau)]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p(\tau)}\left[\left(\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right)\left(\sum_{t=0}^{T}\gamma^t r(s_t^i, a_t^i)\right)\right]$$

Maximum likelihood

The trajectory reward scales the gradient.

$$\log p(\tau) = \log p(s_1, a_1, \ldots, s_T, a_T) = \log\left[p(s_1)\prod_{t=0}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|a_t, s_t)\right] =$$

∇_θ

$$\cancel{\log p(s_1)} + \log\sum_{t=0}^{T}\pi_\theta(a_t|s_t) + \cancel{\log\sum_{t=0}^{T}p(s_{t+1}|a_t, s_t)}$$

So far…

- Policy updates
- Policy gradient derivation

To Do:

- Algorithm using policy gradients

i is the index of the trajectory, t is the steps along that trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau)} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right) \left(\sum_{t=0}^{T} \gamma^t r(s_t^i, a_t^i) \right) \right]$$

  - How can we use these policy updates and take the expected value?

- Extension of policy gradient algorithm

REINFORCE

- Sample trajectories τ_i by rolling out the policy, i.e. sampling a random action from the current policy until the end of the episode.

- Monte Carlo type of RL method - evaluates full trajectories/episodes to update the policy:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t^i | s_t^i\right)\right)\left(\sum_{t=0}^{T} \gamma^t r\left(s_t^i, a_t^i\right)\right)$$

- Update the policy parameters:

$$\theta = \theta + \nabla_\theta J(\theta)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p(\tau)}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(a_t^i | s_t^i\right)\right)\left(\sum_{t=0}^{T} \gamma^t r\left(s_t^i, a_t^i\right)\right)\right]$$

REINFORCE – the variance problem

- Monte Carlo sampling produces noisy policy gradients! Gradients are computed using only few samples.

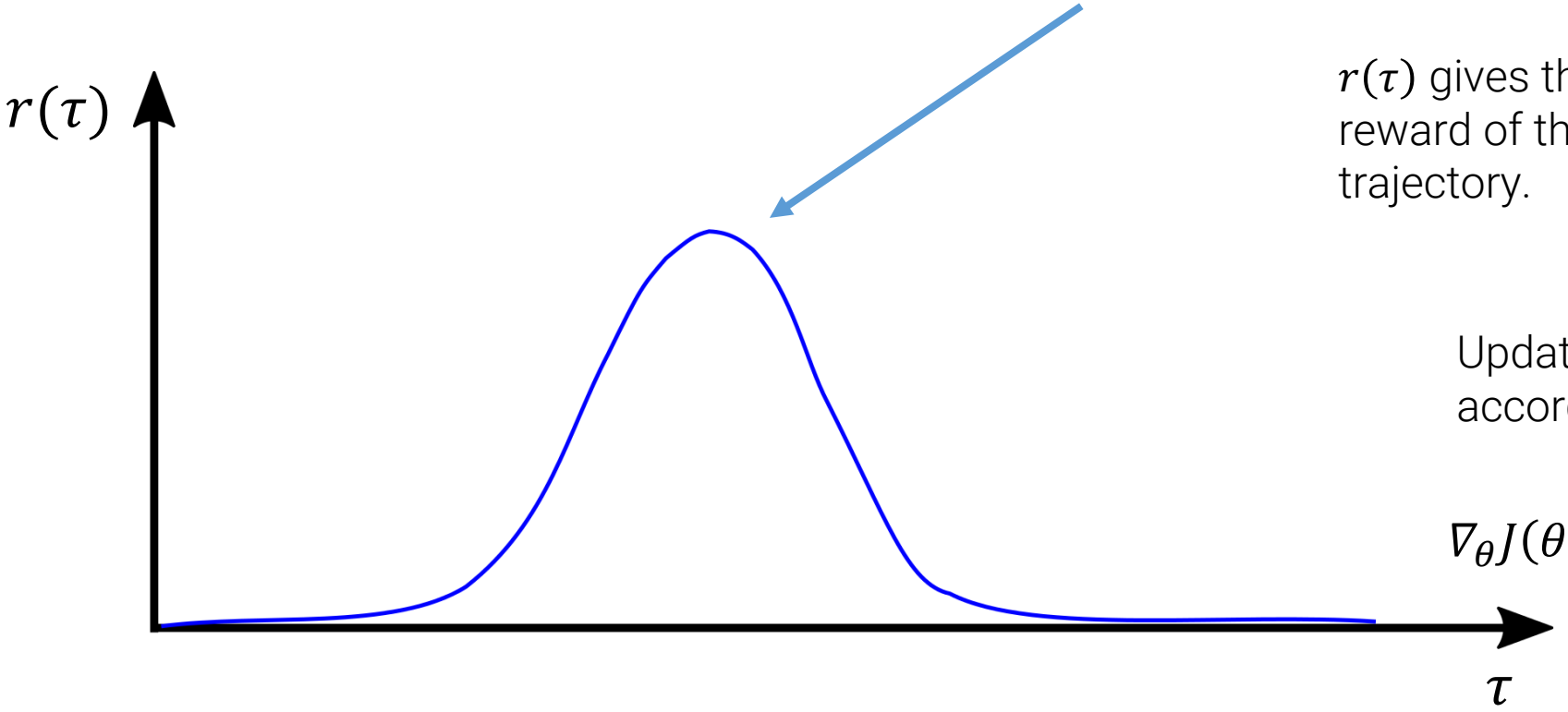
$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right)\left(\sum_{t=0}^{T} \gamma^t r(s_t^i, a_t^i)\right)$$

- What happens if all cumulative rewards are positive?

- What happens if the biggest trajectory reward is 0?

REINFORCE – the variance problem

Sample trajectories according to probability function.

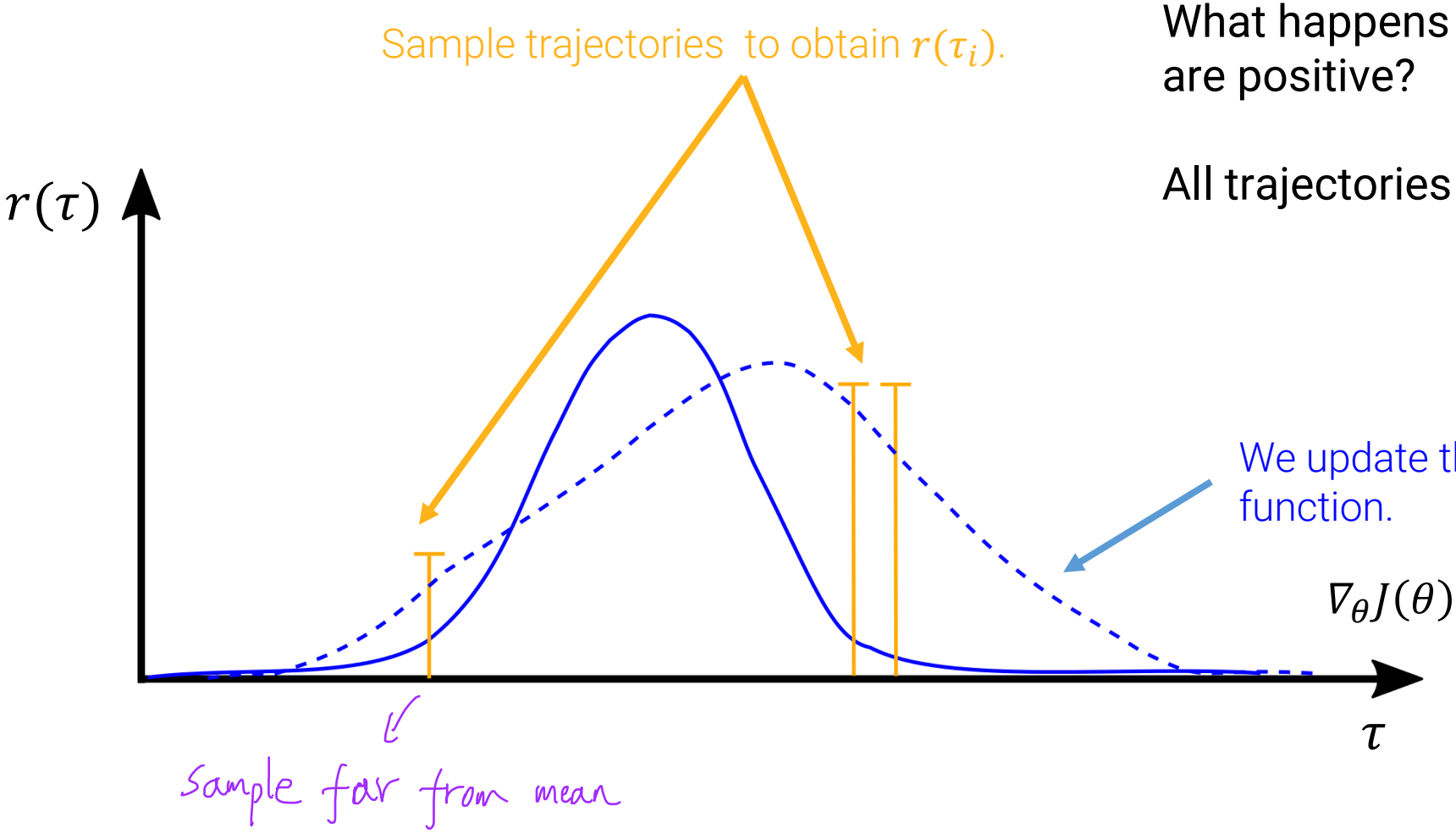


$r(\tau)$ gives the cumulative reward of the sampled trajectory.

Update probability function according to the equation:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\tau)\, r(\tau)$$

REINFORCE – the variance problem



What happens if all cumulative rewards are positive?

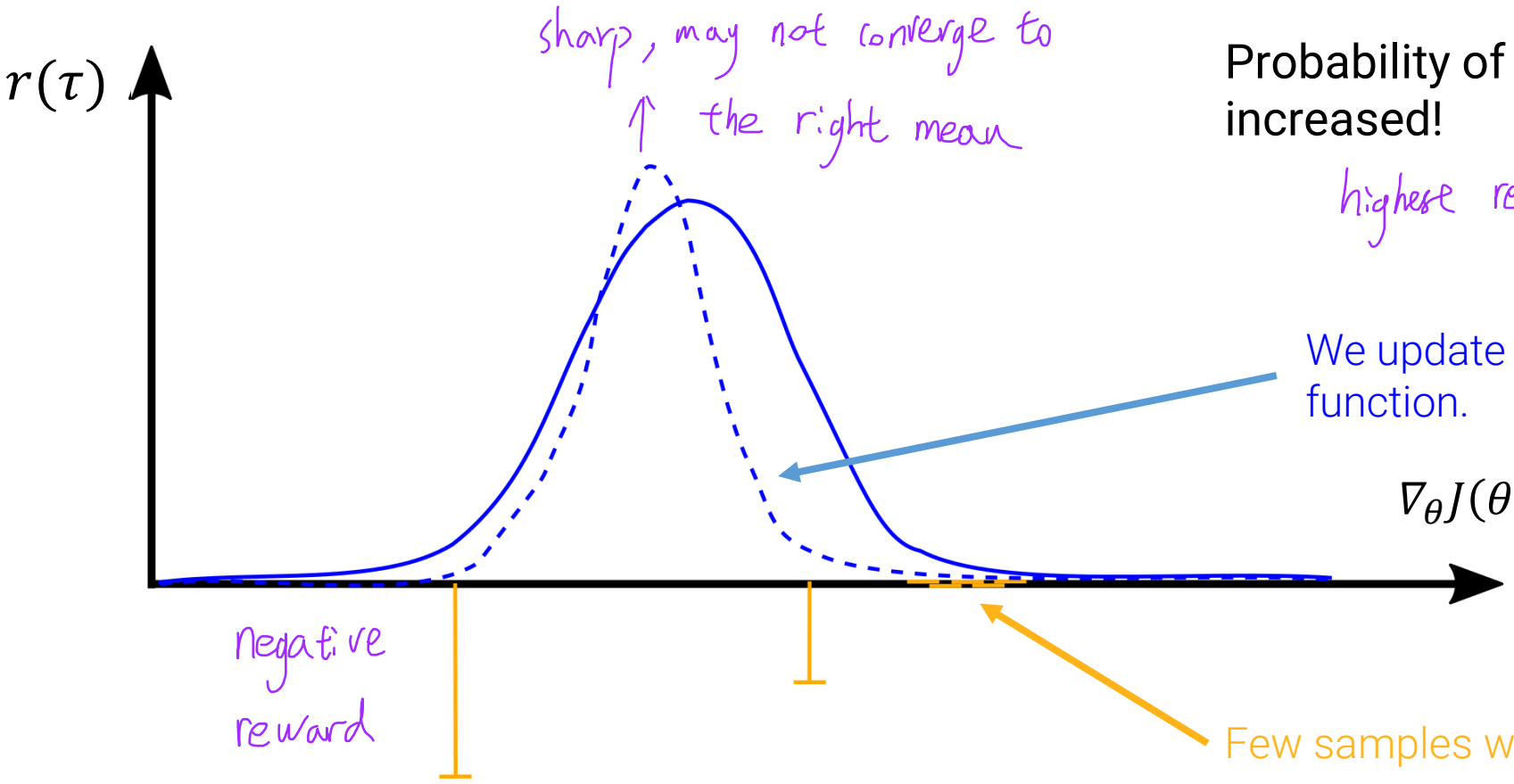
有可能 uniform

All trajectories are made more likely!

We update the trajectory probability function.

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\tau)\, r(\tau)$$

REINFORCE – the variance problem



What happens if the biggest trajectory reward is 0?

Probability of these trajectories is not increased!

highese reward → low prob.

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\tau)\, r(\tau)$$

REINFORCE – baseline

- To reduce the variance we can introduce a baseline $b(s_t^i)$:

$$\nabla J(\theta) = \frac{1}{N}\sum_i \left(\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t^i|s_t^i)\right)\left(\sum_{t=0}^{T} \gamma^t r(s_t^i, a_t^i) - b(s_t^i)\right)$$

- Baseline can be any function that does not depend on the action a_t.
- Examples:
  - Using the average reward as the baseline is one possible solution.
  - Other option is to use the estimate of the state value function as baseline.

- The variance is reduced but the policy gradient estimate stays unbiased.

So far…

- Policy updates
- Policy gradient derivation
- Algorithm using policy gradients

To Do:

- Extension of policy gradient algorithm
  - Can we do better?

Actor-critic

- To introduce bias and reduce the variance even further, we use bootstrapping for estimating the reward:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\left(r(s_t^i, a_t^i) + \gamma\, V(s_{t+1}^i) - V(s_t^i)\right)$$

Function Approximation instead of whole trajectory rollouts

with $v(s_t)$ as the value function in state s_t, i.e., the expected cumulative reward

- The policy (actor) and the value function (critic) are both represented by neural networks

- Most state-of-the-art approaches build upon the actor-critic concept

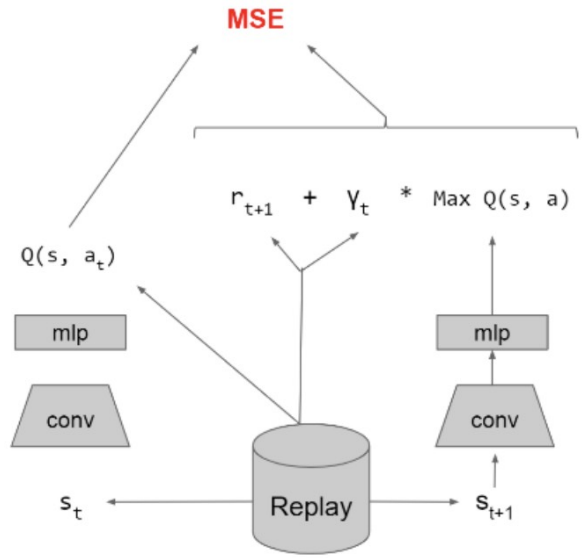
$$Loss(\theta) = (R + \gamma \max_{a'}\{Q_\theta(S', a')\} - Q_\theta(S, A))^2$$

Actor-Critic

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)\left(r(s_t^i, a_t^i) + \gamma\, V(s_{t+1}^i) - V(s_t^i)\right)$$

Does this term look familiar?

- It is the TD-error discussed earlier
- The value function is represented by a neural network as in DQN
- It is also an approximation of the advantage function



Advantage function

$$A_t = Q(s_t, a_t) - V(s_t) \approx r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$$

State-action value estimate, i.e., expected cumulative reward when taking action a_t in state s_t

Reward to go: $$Q(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'})\right] \approx r(s_t, a_t) + \gamma V(s_{t+1})$$

Value function: $$V(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots}\left[\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'})\right]$$

Summary

- Posed optimization $\theta^* = argmax\, J(\theta)$

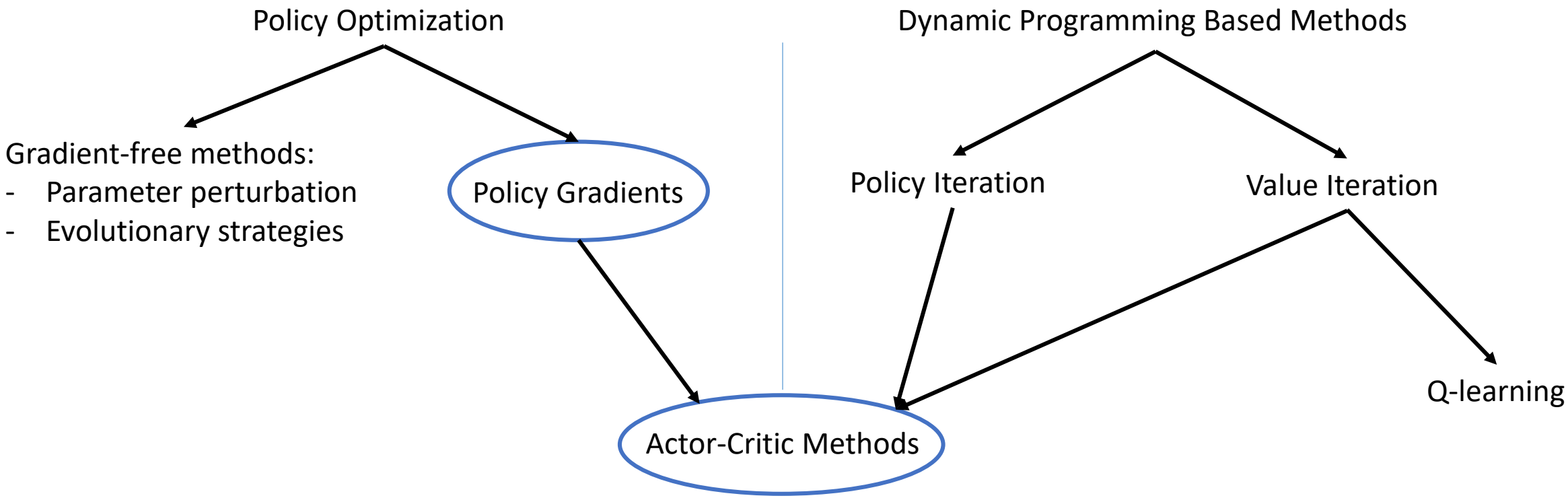
policy gradient

- Showed that updates $\theta = \theta + \nabla_\theta J(\theta)$ can be achieved without knowledge of transition dynamics $p(s_{t+1}|a_t, s_t)$ through policy gradients

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau)} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right) \left(\sum_{t=0}^{T} \gamma^t r(s_t^i, a_t^i) \right) \right]$$

- REINFORCE algorithm that uses sampled trajectory rollouts to do policy gradient updates
- Introduced Actor-Critic

Taxonomy of RL methods



Exe-8.3

In the lecture, we have looked at policy gradient methods in deep reinforcement learning.

1) Explain the main conceptual difference of such methods and Deep Q-learning in maximum 2 sentences.

2) We have also looked into the widely used concept of actor-critic methods. Explain the main difference between policy gradient and actor-critic methods in maximum 2 sentences.

1) In policy gradient methods, we directly optimize the policy to find the desired behavior without having to learn a value function. In Deep-Q-learning, on the other hand, we try to create a model of our environment through the value function and infer a policy from it.
2) Actor-critic methods build on policy gradient methods. The main difference is that in actor-critic, instead of using rewards from trajectory rollouts as in REINFORCE, a value function represented by a neural network is used as reward estimate.

	Policy Optimization	Dynamic Programming
Conceptually:	• Directly optimize desired quantity	• Indirect, exploit problem structure and self-consistency
Empirically:	• More compatible with modern ML machinery, including NN and recurrence • More versatile and flexible • More compatible with auxiliary objectives	• More compatible with *off-policy* and exploration • More sample efficient (*when they work*)

Actor-Critic Algorithms

The following Actor-Critic algorithms we will shortly introduce:

- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimizations (PPO)
- Deep Deterministic Policy Gradients (DDPG)

The following Actor-Critic approaches we leave to the interested reader:

- Asynchronous Advantage Actor-Critic (A3C)
- Soft Actor-Critic (SAC)
- TD3

TRPO

- Adjusting step size is much harder compared to supervised learning
  - Deep RL methods usually compute step size automatically
  - Small steps lead to slow learning
  - One large step in the wrong direction can have disastrous effects

TRPO – Trust Region Policy Optimization

- Pose policy minimization as constrained optimization problem:

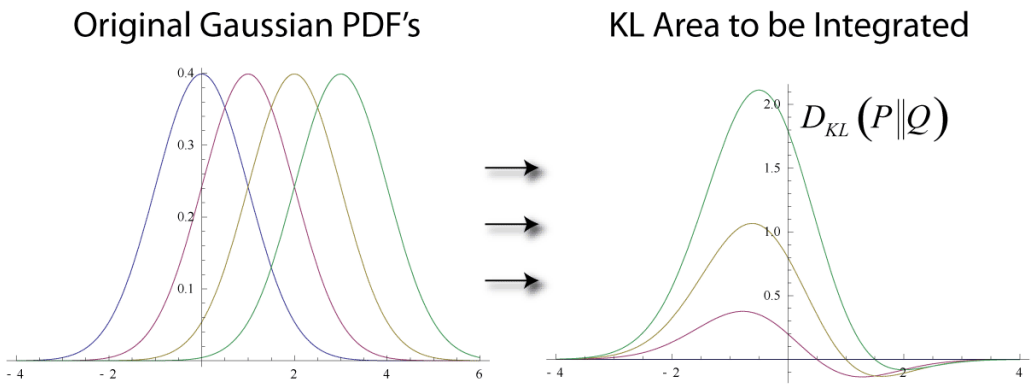
$$\nabla \log \pi(a_t|s_t) = \frac{\nabla \pi(a_t|s_t)}{\pi(a_t|s_t)}$$

$$\max \mathbb{E}_t \left[\frac{\pi(a_t|s_t, \theta)}{\pi(a_t|s_t, \theta_{old})} A_t \right]$$

Advantage function

$$s.t. \mathbb{E}_t \left[D_{KL}\left(\pi(\cdot|s_t, \theta_{old}) || \pi_\theta(\cdot|s_t, \theta) \right) \right] \leq \delta$$

The idea is to control step length by preventing the action distribution to change too much.



[Schulman et. al, 2015]

TRPO

- In the original paper only Q estimate is used instead of advantages. However, most of practical implementations use the advantage function.
- Can be posed in an actor-critic fashion
- The constrained optimization problem can be solved approximately using Fisher Information matrix and conjugate gradient method.
- It is a second order optimization problem and computationally heavy

1.5 优化目标函数（Optimizing the objective function）

正如我们之前提到过的，下界函数M应该是易于优化的。事实上，我们使用泰勒级数(Taylor Series)来估计 L函数 和 KL距离(KL-divergence)的平均值，分别用泰勒将L 展开到一阶 ，KL距离 展开到二阶：

$$L_{\theta_k}(\theta) \approx g^T(\theta - \theta_k)$$
$$g \doteq \nabla_\theta L_{\theta_k}(\theta)|_{\theta_k}$$

$$\overline{D}_{KL}(\theta \| \theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k)$$
$$H \doteq \nabla^2_\theta \overline{D}_{KL}(\theta \| \theta_k)|_{\theta_k}$$

其中g是策略梯度， H 被叫做费雪信息矩阵(FIM,the Fisher Information matrix)，以海森Hessian矩阵的形式(Hessian matrix)

$$H = \nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \end{pmatrix}$$

那么优化问题就变成了：

$$\theta_{k+1} = \arg\max_\theta g^T(\theta - \theta_k)$$
$$subject \quad to \quad \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \le \delta$$

我们可以通过优化二次方程来解决它，并且解集为：

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

https://blog.csdn.net/gsww404/article/details/96709643

https://blog.csdn.net/tanjia6999/article/details/99716133

使用裁剪目标函数的近端策略优化算法PPO

在（使用裁剪目标函数的）PPO中，如果我们希望每次迭代可以较大的改变策略，可以使用类似梯度下降这样的一阶优化方法，并带有软约束正则像的目标来优化我们的问题。

- Natural Policy Gradient：优化 $\max_\theta L_{\pi_{\theta_{old}}}(\pi_\theta) - C\overline{KL}_{\pi_{\theta_{old}}}(\pi_\theta)$，用对优化目标做二次逼近，得到牛顿法更新步：$\theta - \theta_{old} = \frac{1}{C}F^{-1}g$，考虑到计算复杂性，采用近似的二阶优化方法：共轭梯度法（CG），而且使用了小trick，不需要直接求出Hessian矩阵，只需要能计算 $v \to Hv$ 即可
- PPO：“Proximal” Policy Optimization：优化 $\max_\theta \hat{L}_{\pi_{\theta_{old}}}(\pi_\theta) - C\overline{KL}_{\pi_{\theta_{old}}}(\pi_\theta)$，只进行一阶的SGD，但会根据KL大小来改变KL项权重C。
- TRPO：优化 $\max_\theta L_{\pi_{\theta_{old}}}(\pi_\theta)$ 服从 $\overline{KL}_{\pi_{\theta_{old}}}(\pi_\theta) \le \delta$，转化为有限制优化问题，采用Lagrange Multiplier方法，在规划更新方向 $s_{unscaled} = F^{-1}g$，并进行线性搜索。好处是超参数 δ 比C要好确定，且能采用较大步长。

PPO

- Follow up work of TRPO, but using first order optimization
- Relax the hard constraint condition and turn it into a tunable parameter β

$$\max \mathbb{E}_t \left[\frac{\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta_{old})} A_t \right] - \beta \mathbb{E}_t \left[D_{KL}\left(\pi(\cdot\,|s_t,\theta_{old}) || \pi_\theta(\cdot\,|s_t,\theta) \right) \right]$$

Penalizes deviations from the old policy

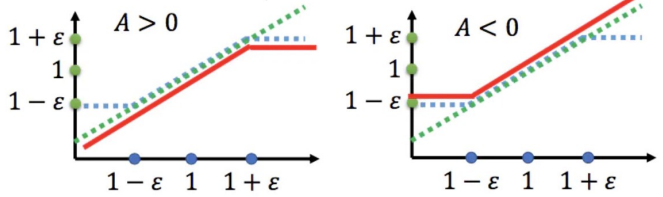
- Allowing some bad decision, but generally optimizing within a trust region

https://openai.com/blog/openai-baselines-ppo/ [Schulman et. al, 2017]

PPO

Proximal Policy Optimization(PPO)算法原理及实现！
https://www.jianshu.com/p/9f113adc0c50

$$J^{\theta^k}_{PPO2}(\theta) \approx \sum_{(s_t,a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}A^{\theta^k}(s_t,a_t),\right.$$
$$\left. clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right)A^{\theta^k}(s_t,a_t)\right)$$

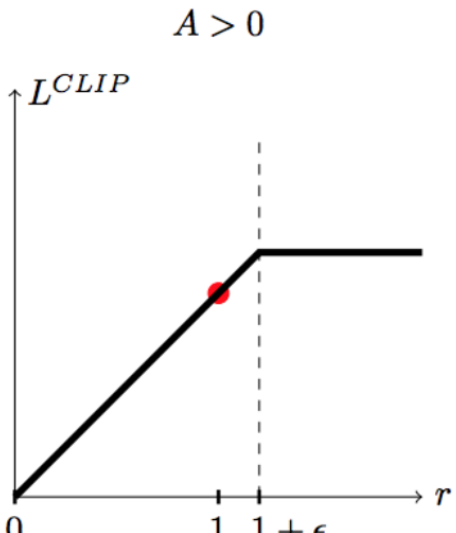


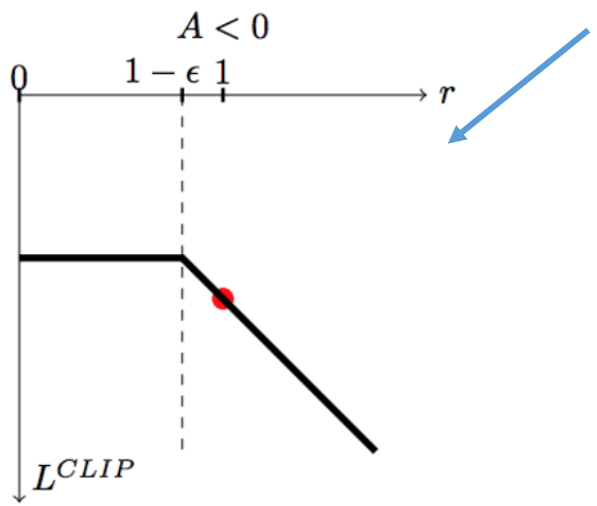
上图中，绿色的线代表min中的第一项，即不做任何处理，蓝色的线为第二项，如果两个分布差距太大，则进行一定程度的裁剪。最后对这两项再取min，防止了θ更新太快。

- But we can even do better by using a clipped objective

$$L^{CLIP} = \max \mathbb{E}_t \left[\sum_{t=0}^{T}\left[\min(r_t(\theta), \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon))\, A_t\right]\right]$$

where the probability ration is $r_t(\theta) = \frac{\pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta_{old})}$





We clip the advantage function to discourage large policy changes outside of our comfort zone

PPO

- Much simpler implementation than TRPO
- Less computationally heavy (first-order)
- Trade-off between fast optimization and allowing some bad updates

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

  by taking K steps of minibatch SGD (via Adam), where

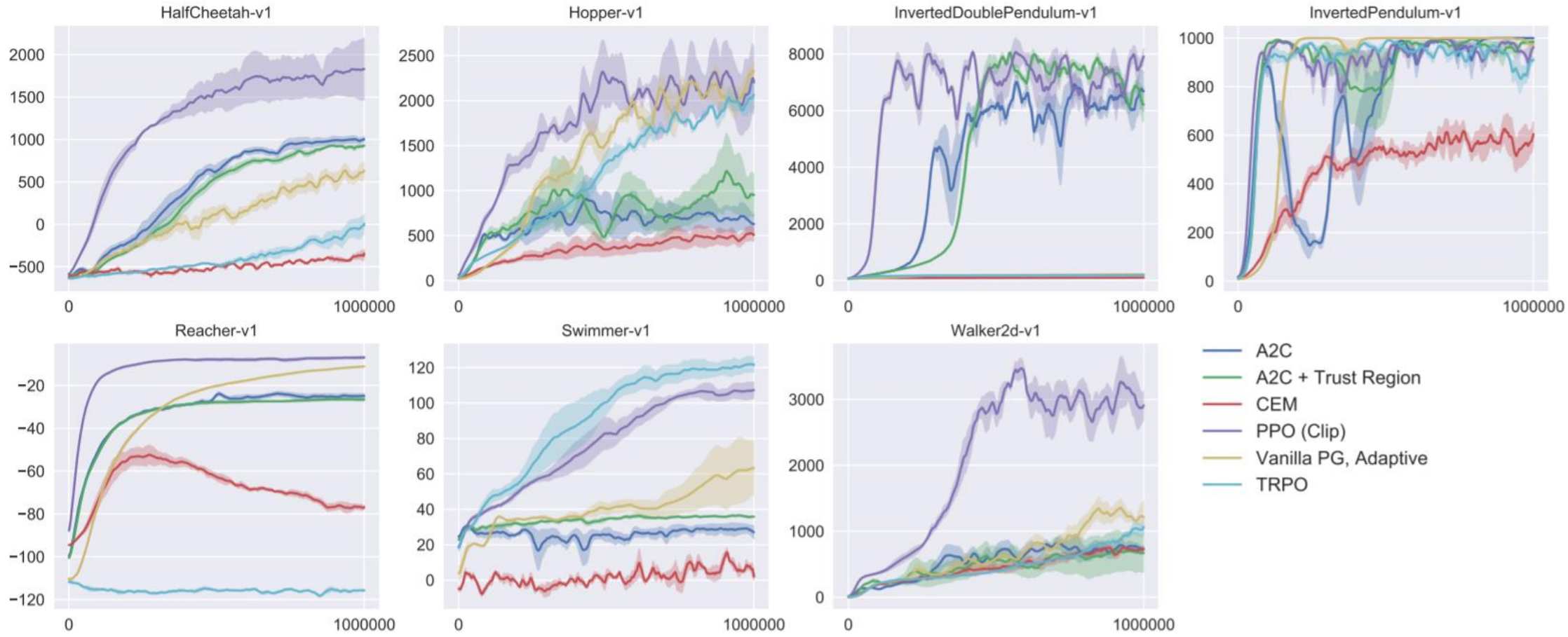
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

PPO

https://openai.com/blog/openai-baselines-ppo/

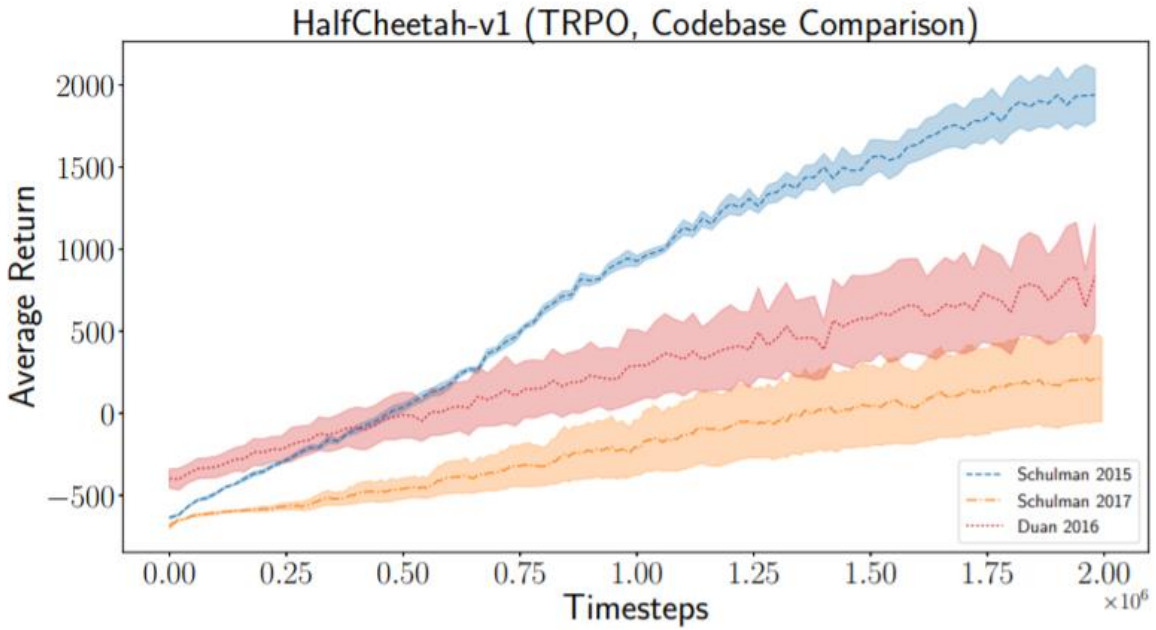
PPO vs. TRPO vs. Actor-Critic vs. Policy Gradients

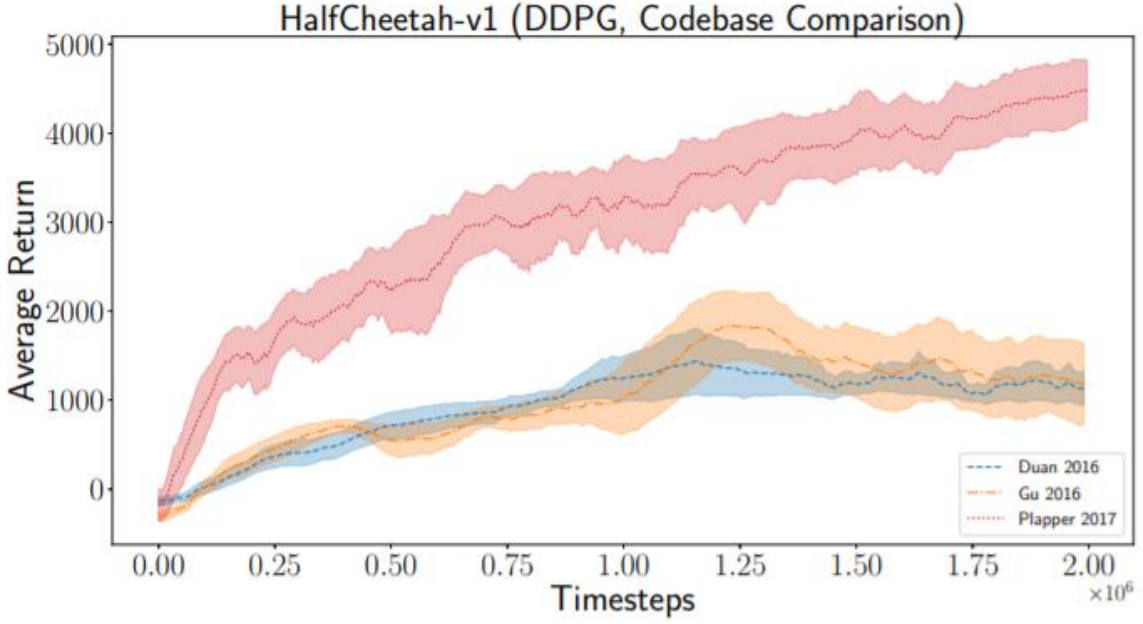


https://openai.com/blog/openai-baselines-ppo/

Issues in DRL research – Domain Knowledge

- Small tricks and experience lead to success
- Even small implementation differences can have a big impact:





https://arxiv.org/abs/1709.06560 [Henderson et. al, 2019]

Deep Deterministic Policy Gradients - DDPG

- What if we want to use an off-policy algorithm for policy gradients?
- DDPG is essentially a combination of insights from DQN and Deterministic Policy Gradients (DPG).
- The actor and critic updates are done in actor-critic fashion with a few tweaks (out of scope to go into details for this lecture).
- It is mostly used for continuous control tasks

https://arxiv.org/abs/1509.02971 [Lillicrap et. al, 2016]

Deep Reinforcement Learning Research

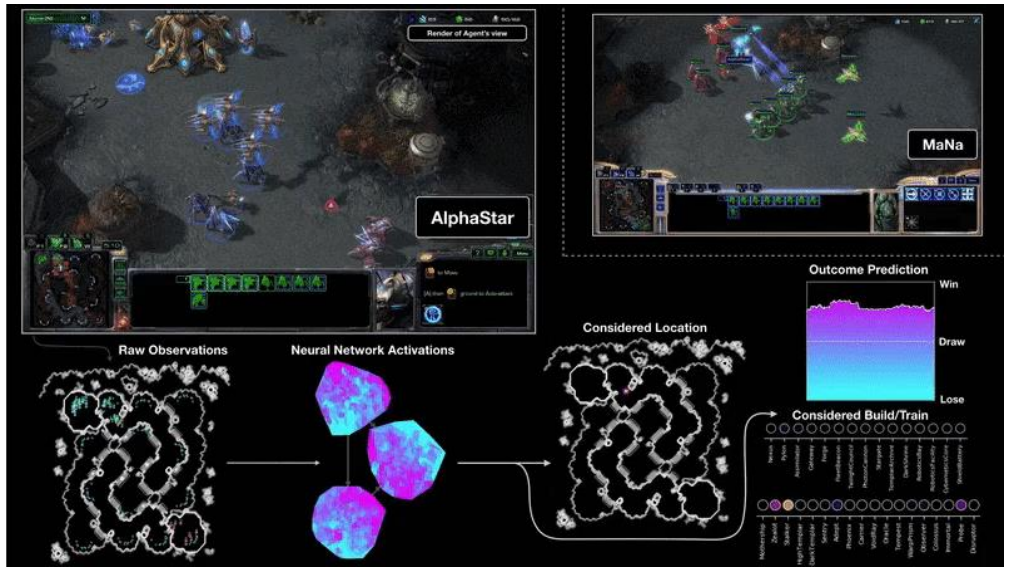
Research Overview - Goals

- Look at current approaches that use the presented algorithms
- Discuss limitations of these approaches
- Talk about current issues in Deep Reinforcement Learning

Examples of Achievements in DRL research

Superhuman performance on video games such as Atari, Super Mario or even modern games like Starcraft

[Mnih et. al, 2013]
Breakout

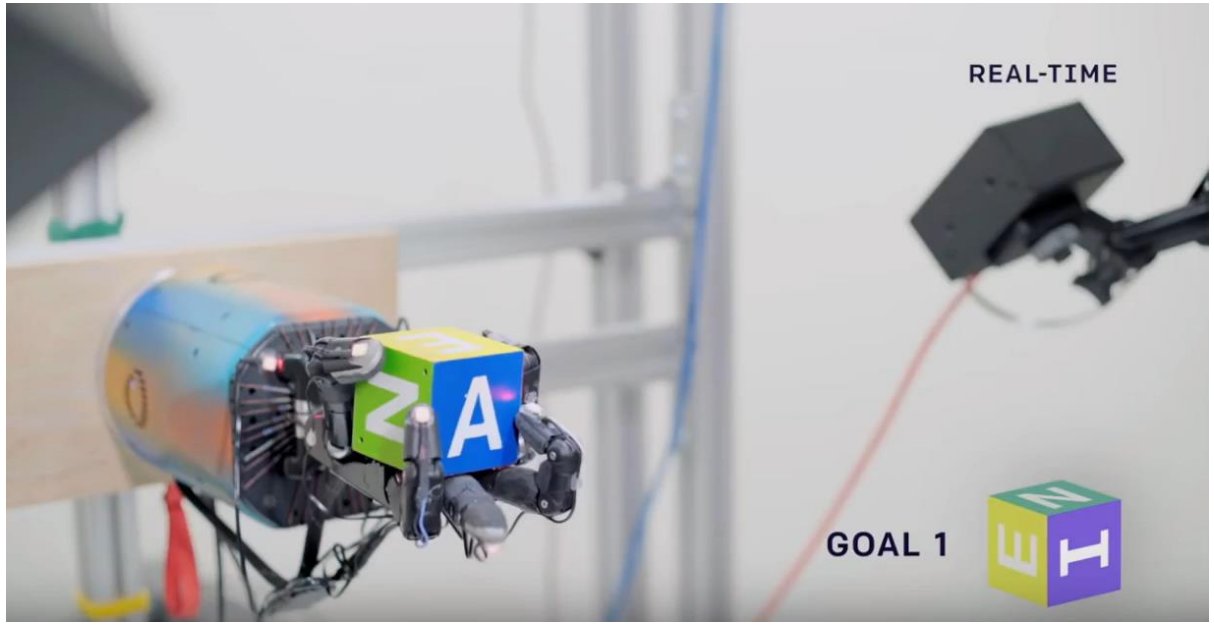


[Vinyals et. al, 2019]
Starcraft

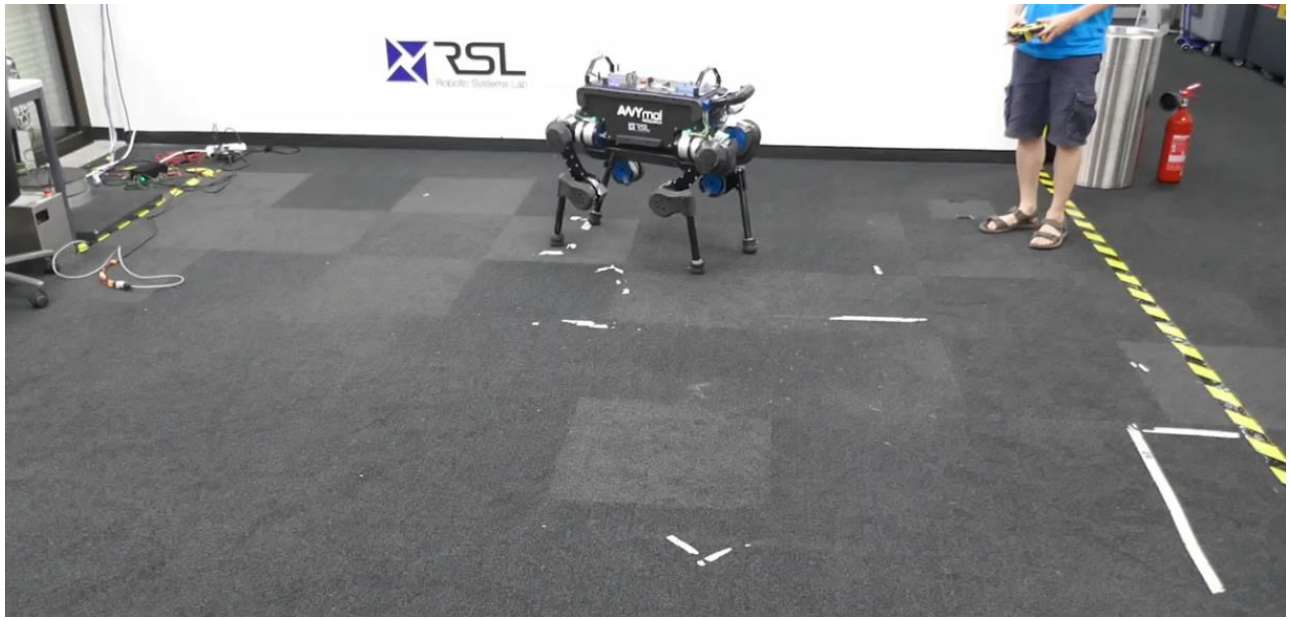
[Schulman et. al, 2017]
Mario

Examples of Achievements in DRL research

Robots being able to solve complex tasks (with a lot of engineering overhead, however)



[OpenAI, 2018]
OpenAI Hand

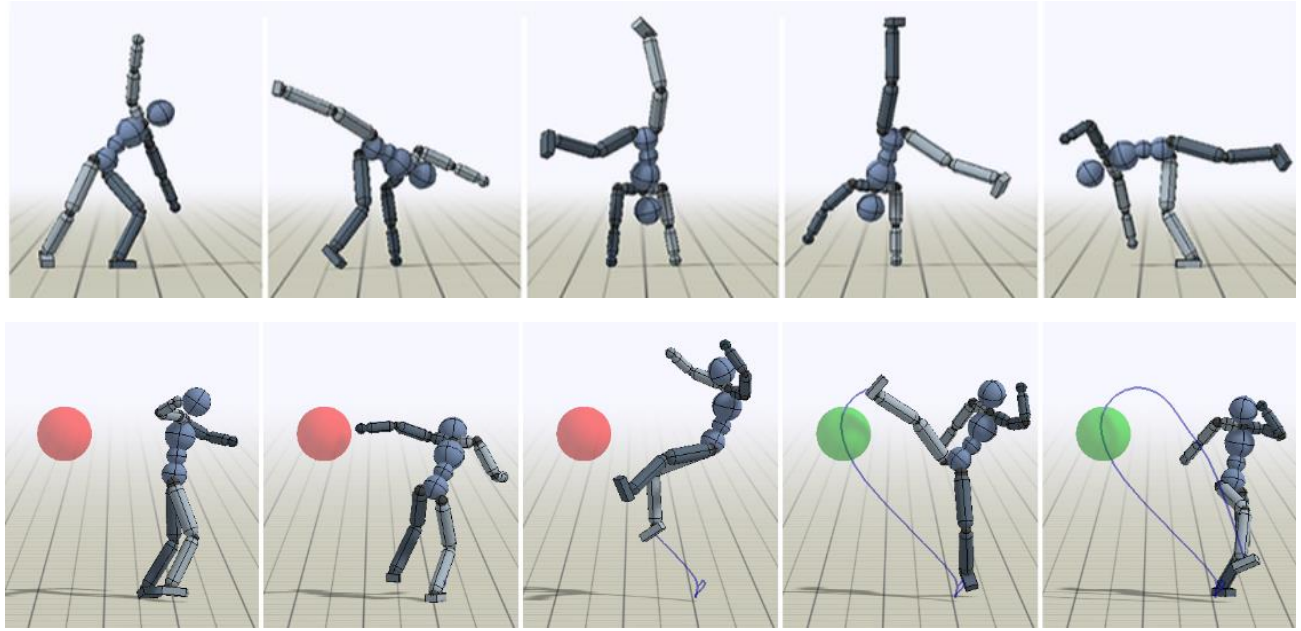


[Hwangbo et. al, 2018]
RSL Anymal

We stopped here
May 14, 2020
remainder of slides for self-study

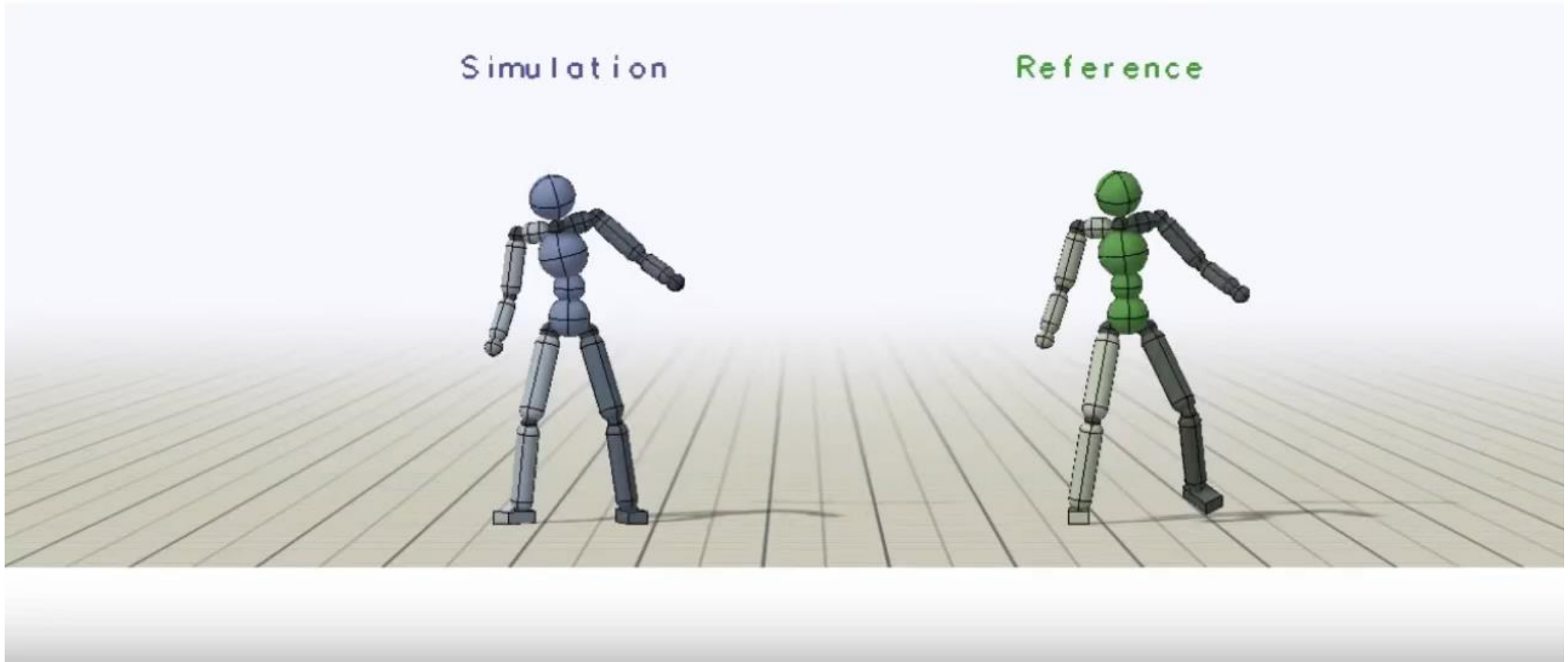
DeepMimic

- Learning physics-based motion policies
- Trying to imitate reference motions while satisfying specific task objectives
- Robust policies that adapt to changes, perturbations and user-specified goals
- Uses PPO as training algorithm



[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

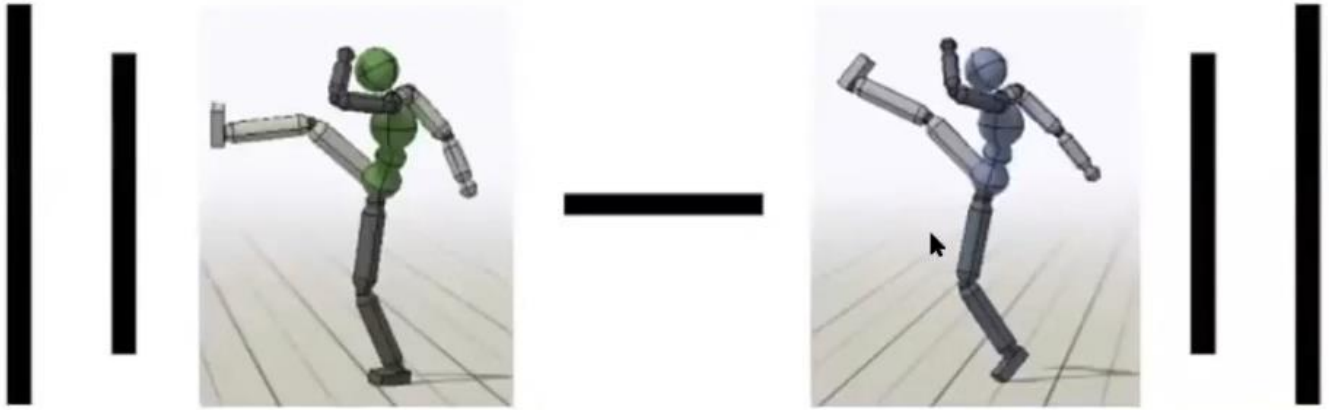
Humanoid: Cartwheel



https://www.youtube.com/watch?v=vppFvq2quQ0

DeepMimic - Reward Function

$$r_t = \omega^I r_t^I + \omega^G r_t^G$$



reference motion executed motion

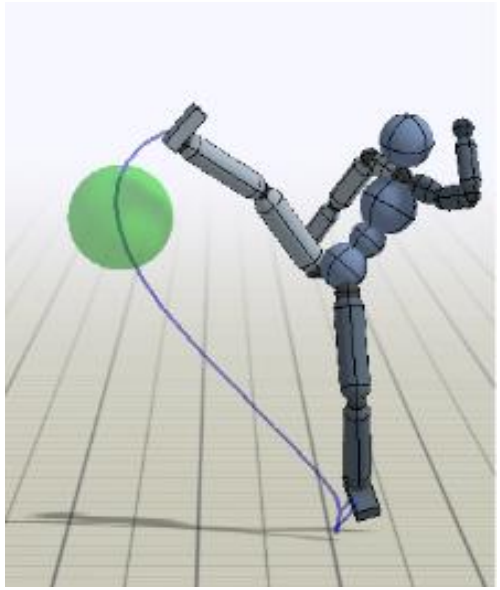
The agent tries to imitate the motion given from a reference motion clip

[Peng et. al, 2018]

https://arxiv.org/abs/1804.02717

DeepMimic - Reward Function

$$r_t = \omega^I r_t^I + \omega^G r_t^G$$



The agent tries to achieve a task objective such as kicking a certain object

[Peng et. al, 2018]

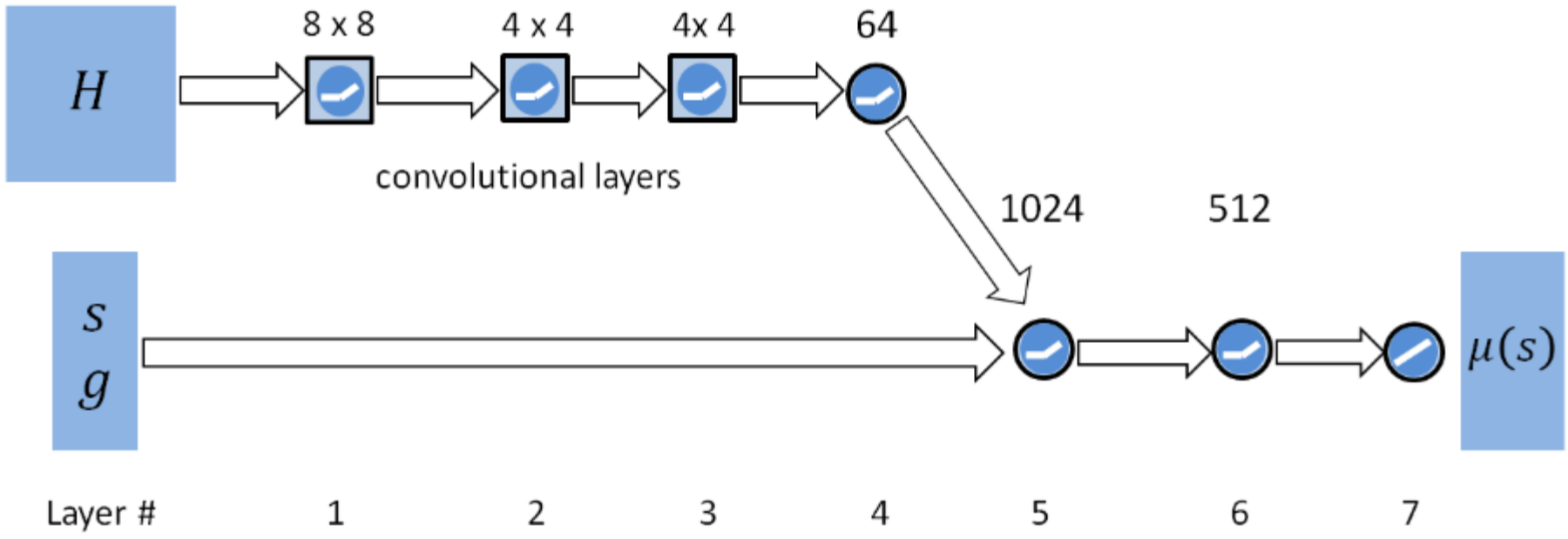
https://arxiv.org/abs/1804.02717

DeepMimic - Policy

- Policy $\pi(a|s,g)$ as neural network that maps state s and goal g to a distribution over actions
- Model action distribution as a Gaussian distribution $\pi(a|s,g)=\mathcal{N}(\mu(s),\Sigma)$ with a state dependent mean $\mu(s)$ and fixed diagonal covariance matrix Σ
- Only difference to regular policy gradient methods is that the goal is also an input to the policy

[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

DeepMimic - Network Structure



H: heightmap
s: input state
g: goal
μ: output

[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

DeepMimic -Novel Techniques

Early termination

- Terminate episodes when failure state is encountered
- Avoid exploitation of local minima
- Requires definition of failure states
- Similar to class imbalance problem in supervised learning

Reference State Initialization

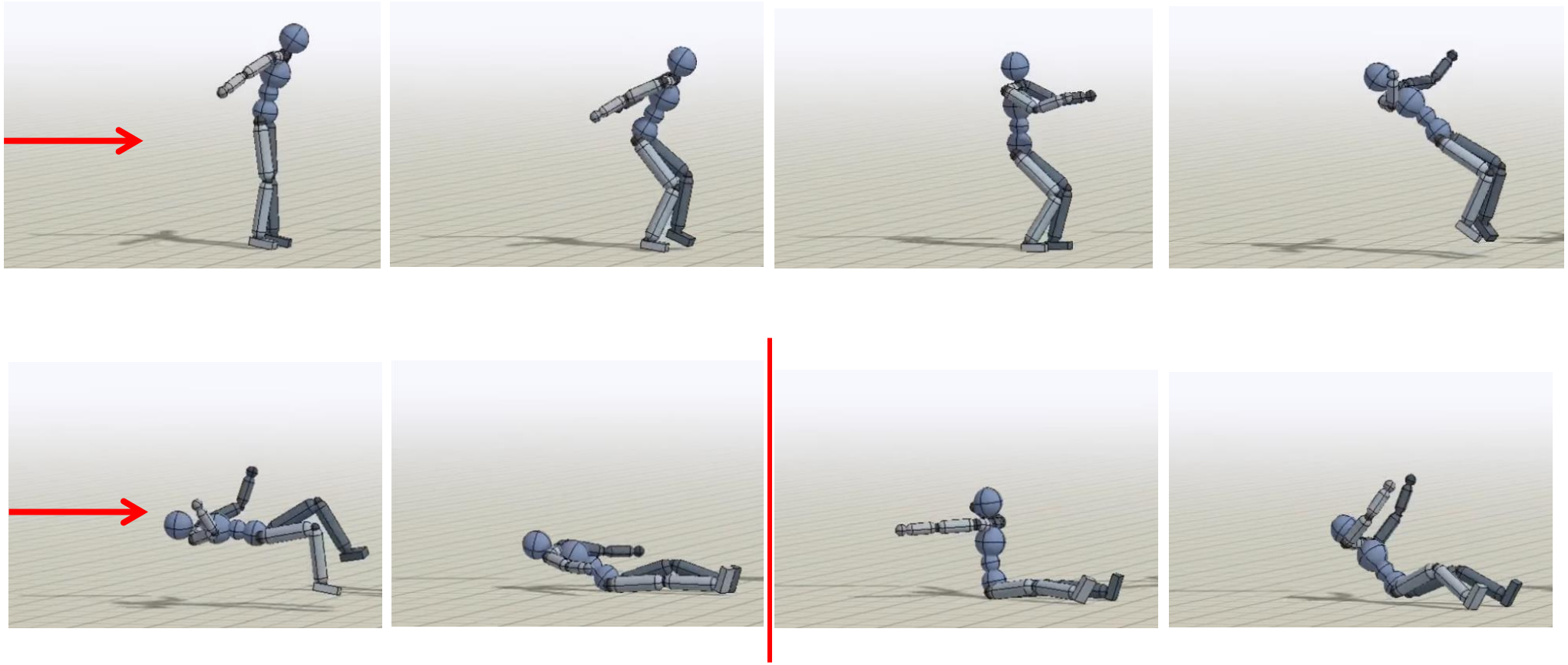
- Initial states along the trajectory of reference motion
- Mitigate problem of sequential learning
- Requires possibility to put agent in different starting states
- Can help overcome reward hacking

[Peng et. al, 2018]

https://arxiv.org/abs/1804.02717

Novel Techniques – Early Termination

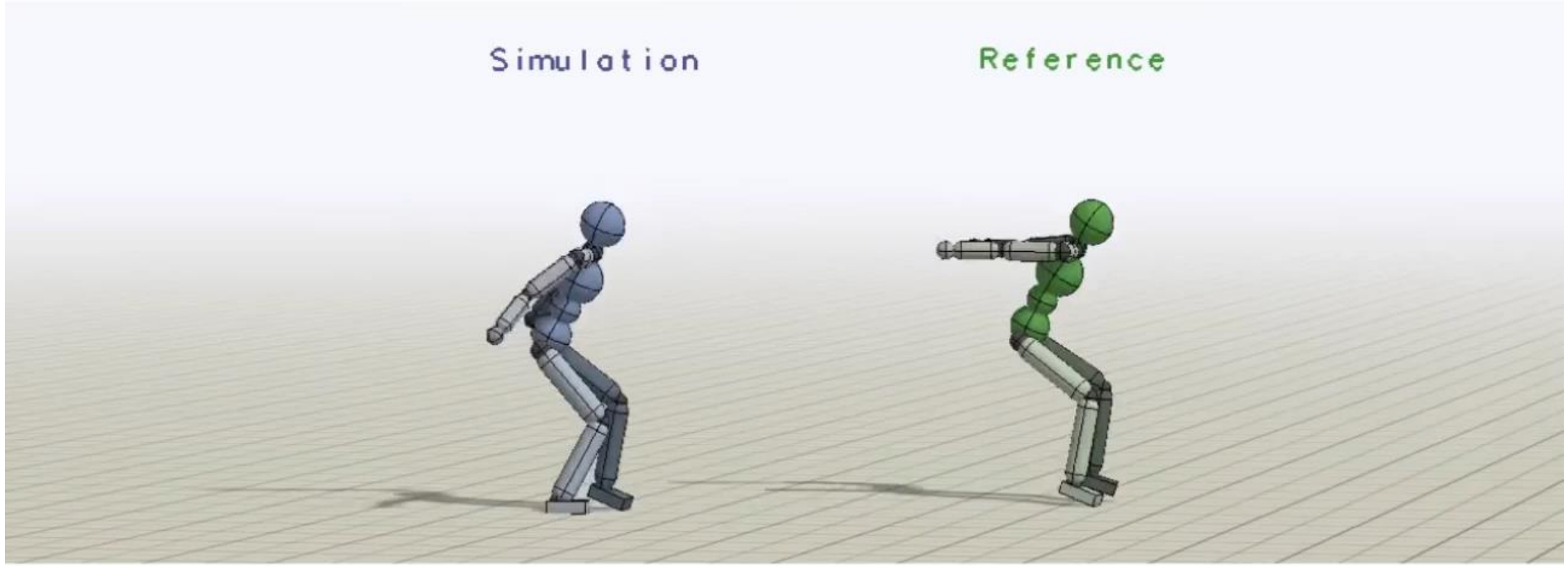
As soon as a stopping criterion is fulfilled, stop the training episode



stopping criterion (torso on ground) == true
→ terminate episode

[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

No Early Termination



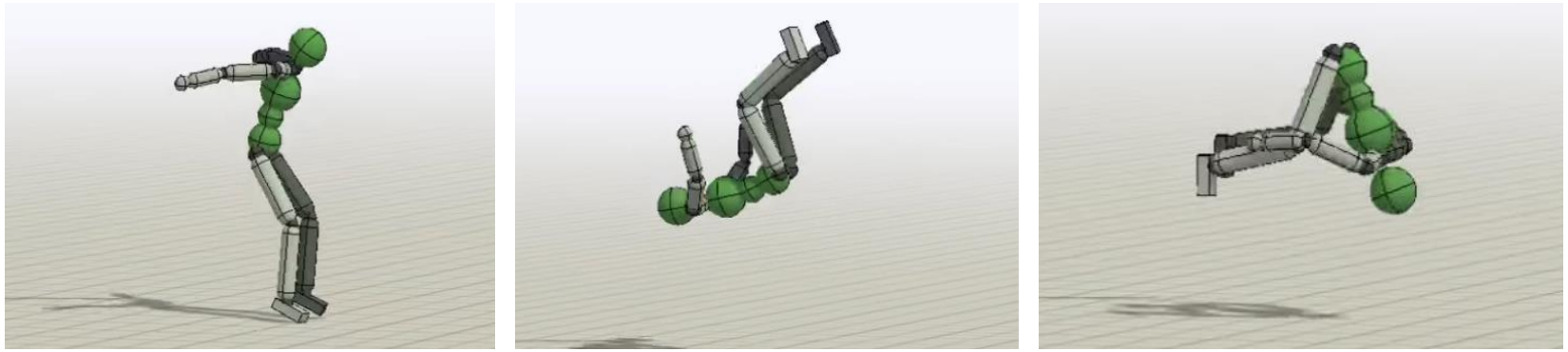
Without early termination (ET), the policy falls and then tries to mime the motion while on the ground.

[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

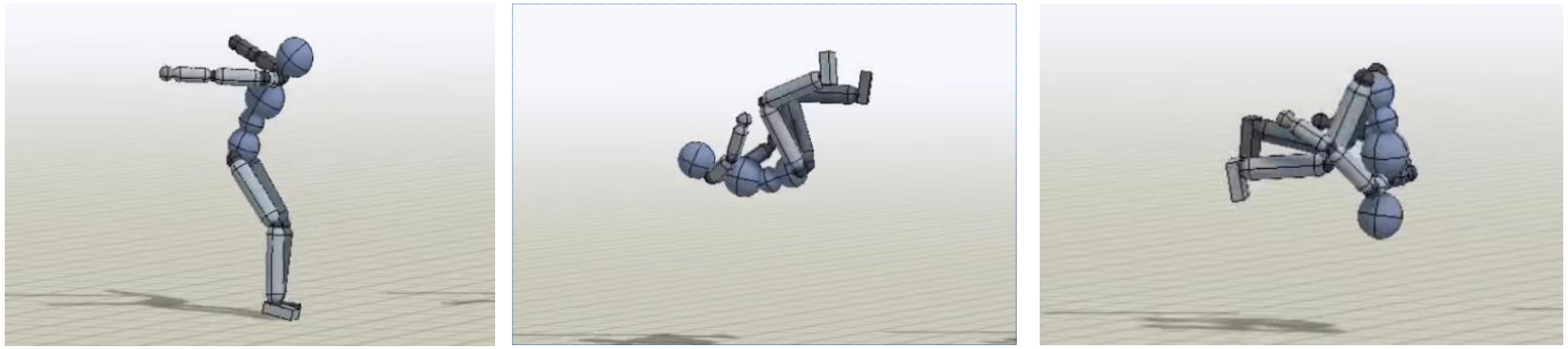
Novel Techniques – RSI

Initialize starting state of training episodes according to states of reference motion

Reference motion



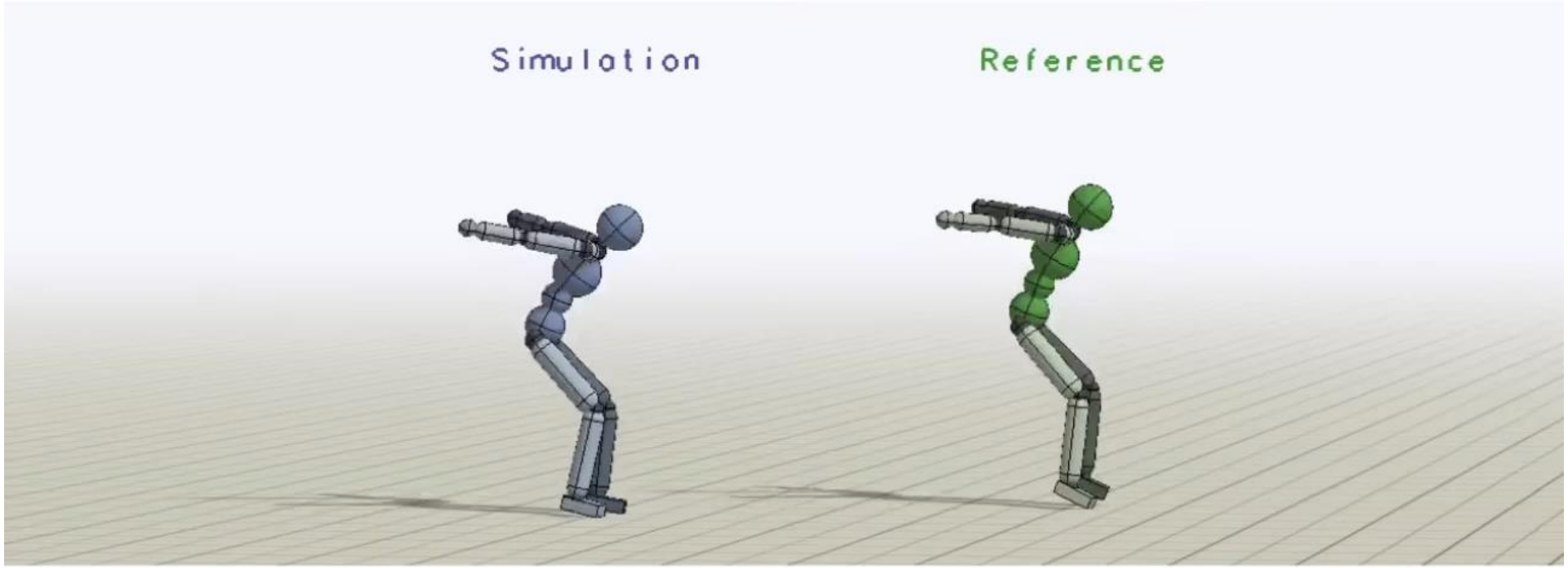
Initial state distribution



[Peng et. al, 2018]

https://arxiv.org/abs/1804.02717

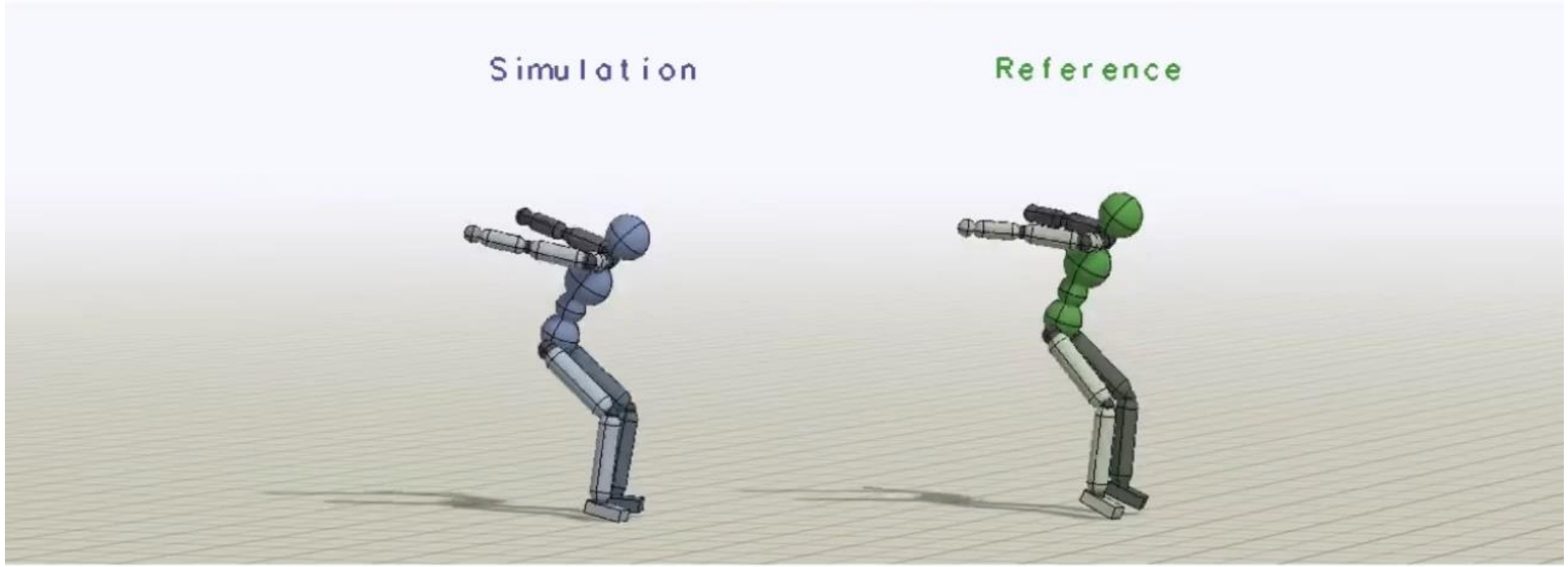
No Reference State Initialization



Without reference state initialization (RSI), the policy never learns to perform a backflip and instead just tries to avoid falling.

[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

RSI + ET (Our Method)



With RSI and ET, the policy learns to perform the flip.

[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

Limitations

- Imitation data costly to obtain
- Performance limited by demonstrator’s performance
- Reverting back to supervised learning
- Carefully shaped reward function

[Peng et. al, 2018]
https://arxiv.org/abs/1804.02717

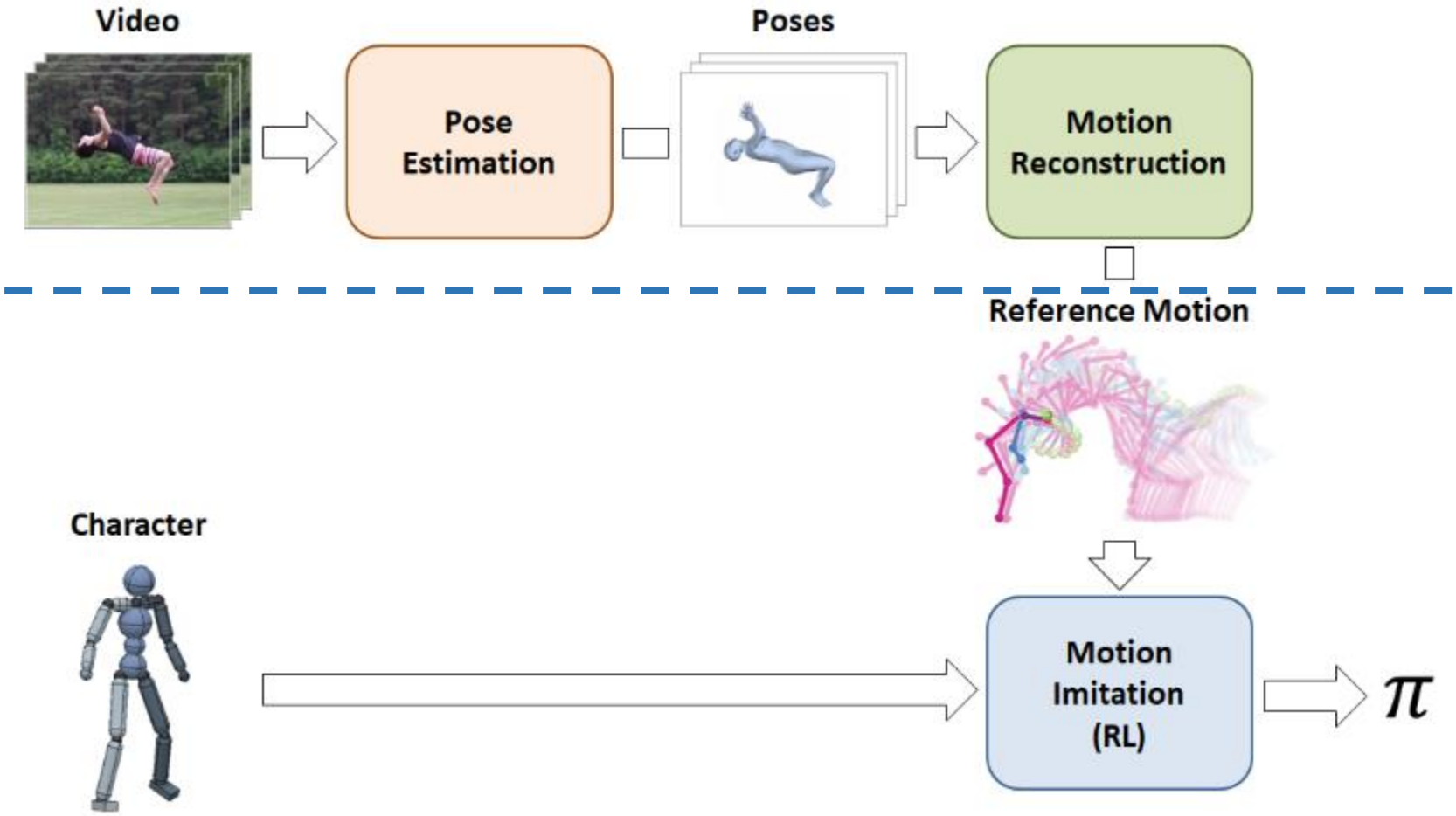
SFV: Reinforcement Learning of Physical Skills from Videos

- Follow-up paper of DeepMimic
- Combines DeepMimic method with deep pose estimation
- Transfers motions from videos to reference motions

[Peng et. al, 2018]

Paper Link

SFV – What's new?

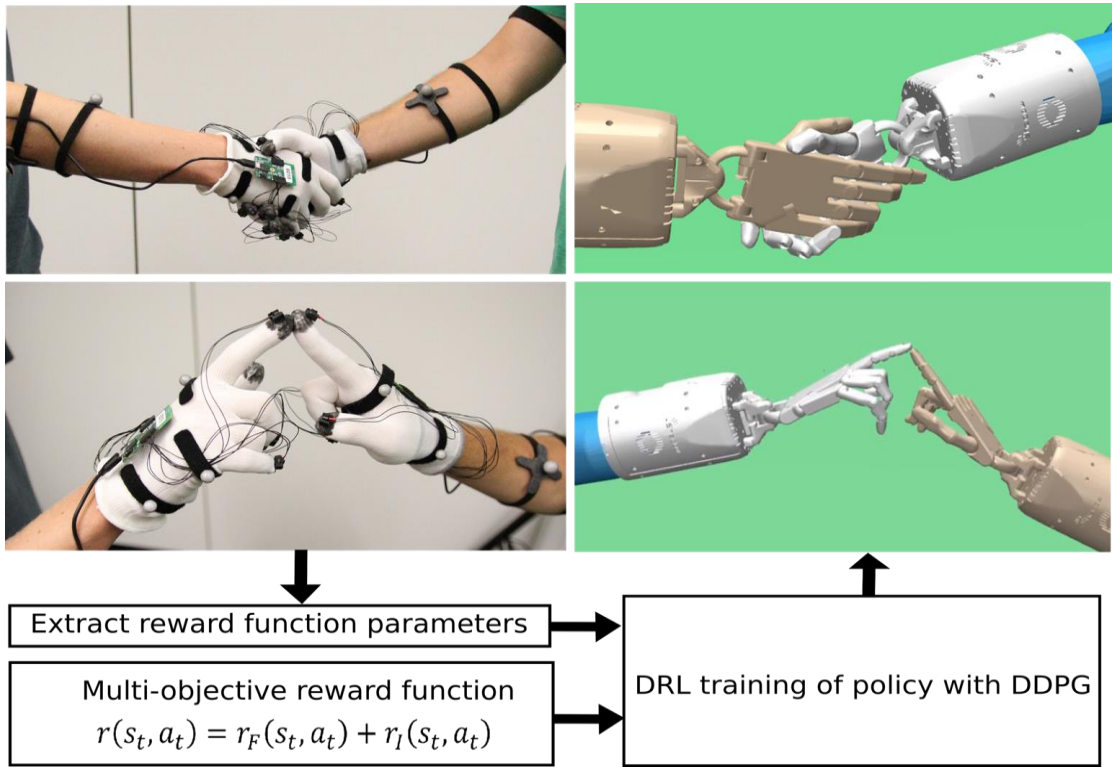


[Peng et. al, 2018]

https://arxiv.org/abs/1810.03599

Work at our Lab – DRL for human-robot interactions

- Multi-objective reward function with parameters extracted from motion capture data to produce human-robot interactions
- Similar to DeepMimic, but considers complex goals such as desired contact patterns
- RSI and Early Termination not possible for hand interactions that involve contacts



[Christen et. al, 2019]
https://arxiv.org/abs/1906.11695

Procedure

Step 1: Collect motion capture data of the hand interaction.
Step 2: Extract the reward function parameters from the dataset.
Step 3: Train the agent using reinforcement learning.
Output: A control policy that performs the interaction.

[Christen et. al, 2019]
https://arxiv.org/abs/1906.11695

Reward Function

$$r(s_t, a_t) = r_F(s_t, a_t) + r_I(s_t, a_t)$$

Final state reward

Imitation reward

[Christen et. al, 2019]
https://arxiv.org/abs/1906.11695

Reward Function

$$r(s_t, a_t) = r_F(s_t, a_t) + r_I(s_t, a_t)$$

final state reward

imitation reward

$$\mathbf{r_F(s_t, a_t) = r_p(s_t) + r_\alpha(s_t) + r_c(s_t)} \quad \mathbf{r_I(s_t, a_t) = r_{pI}(s_t) + r_{\alpha I}(s_t)}$$

position

angle

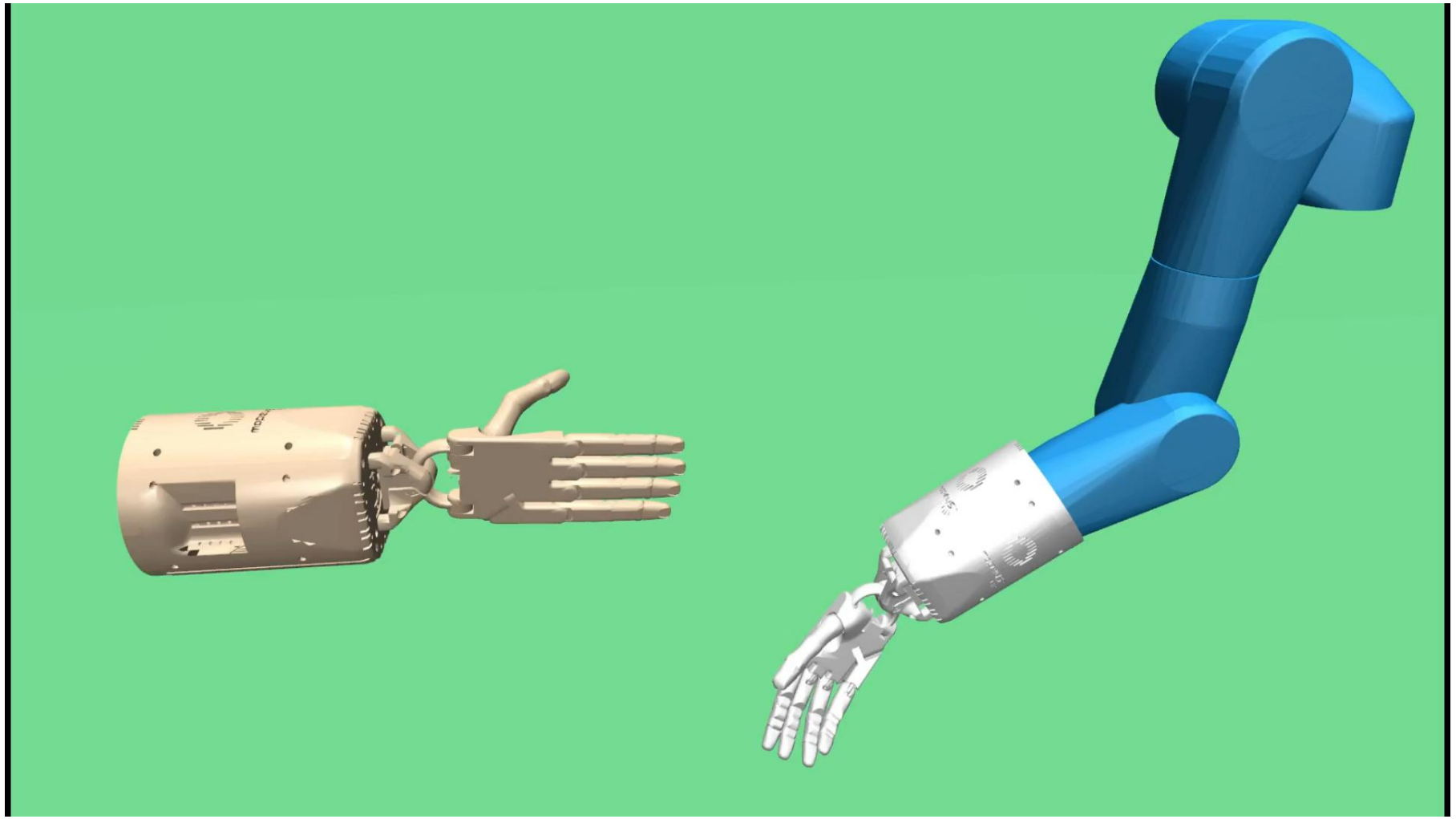
contact

inpposition

angle

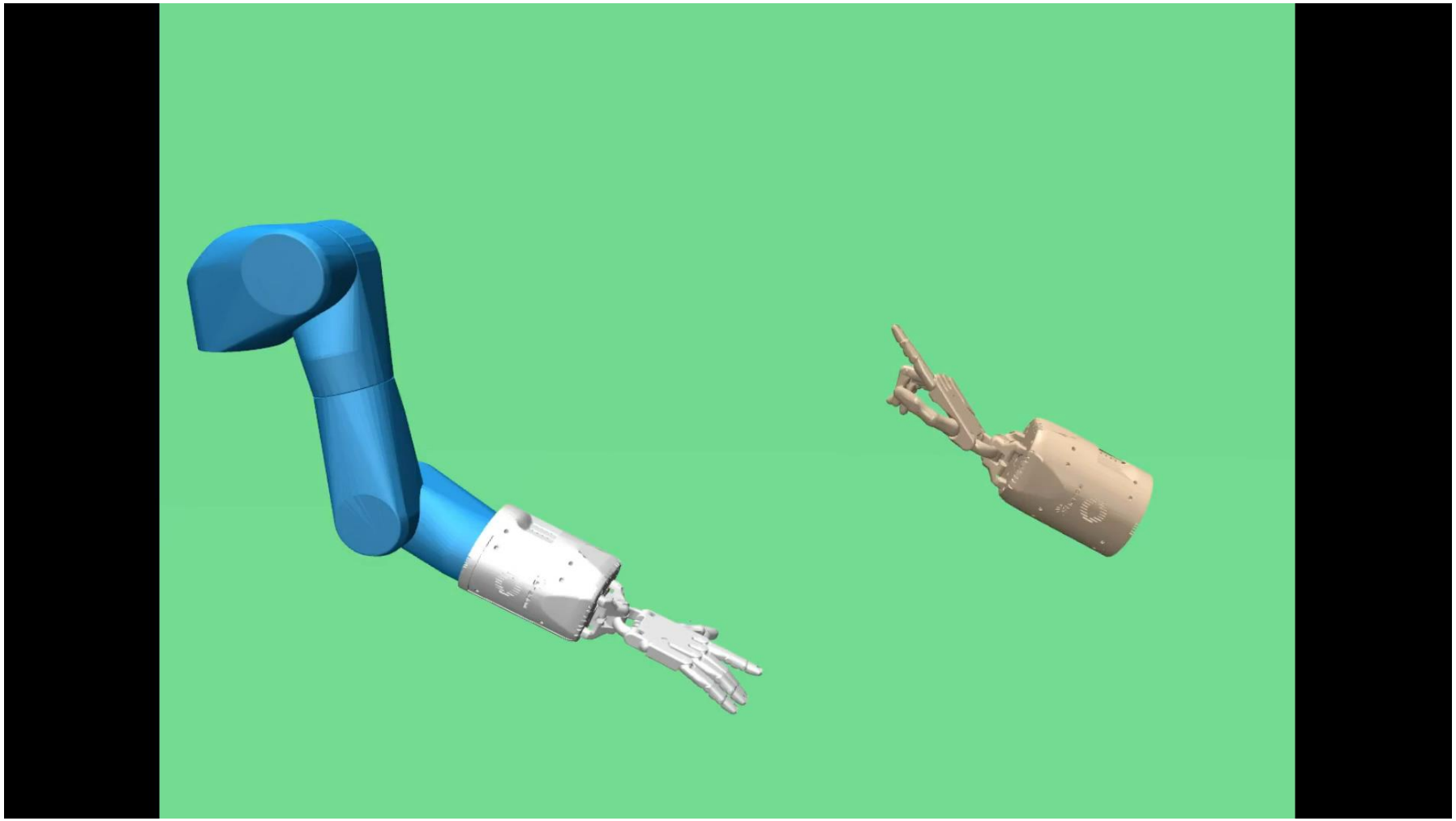
[Christen et. al, 2019]
https://arxiv.org/abs/1906.11695

Results - Handshake



https://www.youtube.com/watch?v=ZSgEqyltaN4&t

Results – E.T.

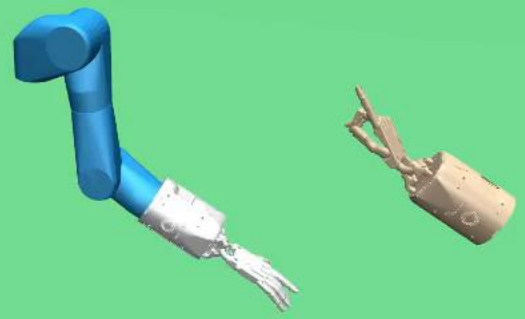


https://www.youtube.com/watch?v=ZSgEqyltaN4&t

Position offset

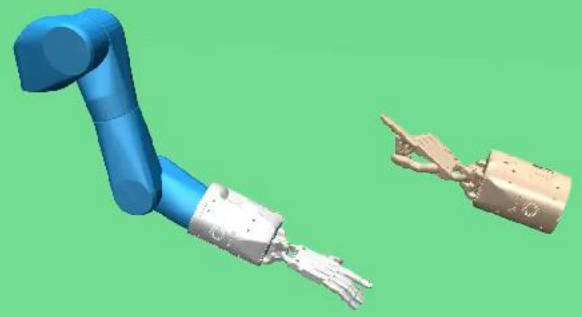
-10 cm

+10 cm

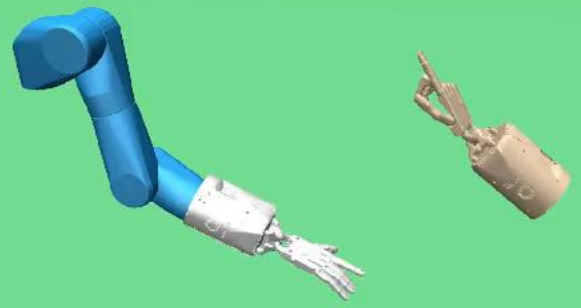


Angle offset

-15°

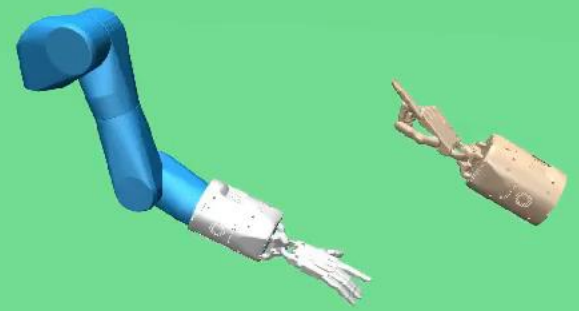


+15°

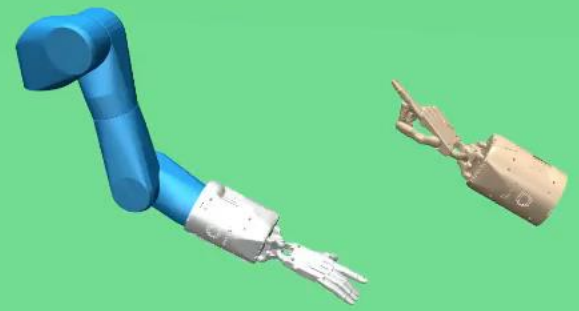


Moving target

2.5 cm/s



5 cm/s



Reinforcement Learning is nice, but...

... it has not been solved (yet)

Issues in DRL research – Exploration vs. Exploitation

- Trade-off between when to explore and when to exploit
- Agents can easily get stuck in local minima, i.e. exploit behavior that is irreversible

Agent exploits behavior to land on its back, very unlikely it learns to stand up again

Issues in DRL research – Reward Hacking

Rewards have to be carefully designed, undesired behavior may occur because of ill-shaped reward

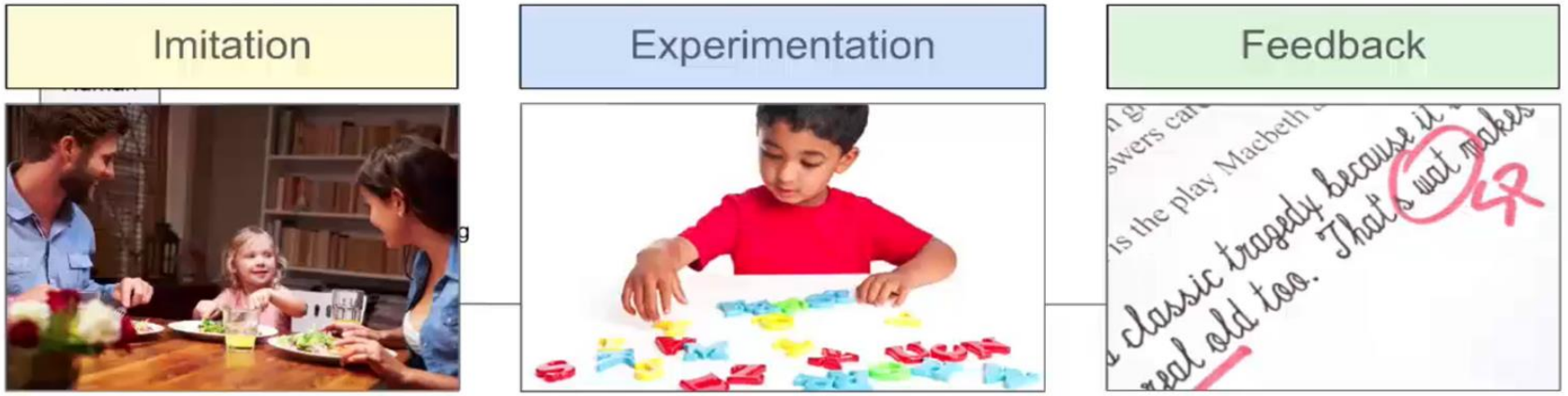
Agent begins to only collect power-ups instead of finishing the race

https://www.youtube.com/watch?v=tlOIHko8ySg

Issues in DRL research – Sample Efficiency

- Deep Reinforcement Learning methods are horribly sample inefficient, often need an excessive amount of samples to learn a policy
- This data may be easy to obtain in some domains (e.g. games), but in real-world scenarios the amount of training samples may be limited
- Especially on-policy methods suffer from sample inefficiency, because the samples have to be “thrown away” after the policy is updated

Research Outlook – How do humans learn?



DeepMimic

???

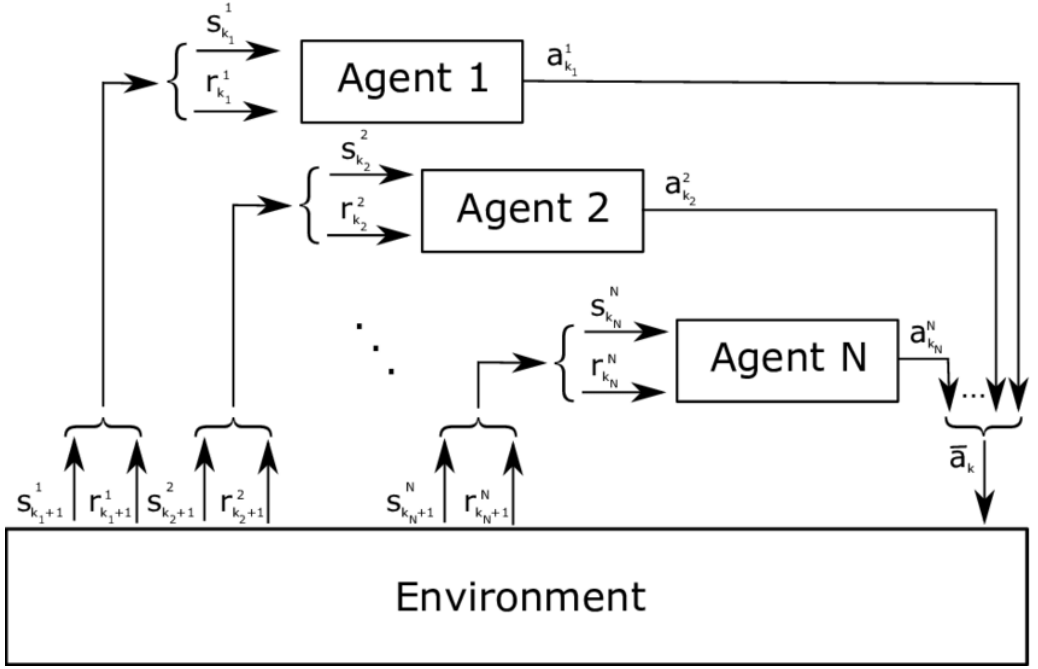
???

field of imitation learning:
- behavior cloning
- direct policy learning
- etc.

methods for simple tasks exist

Research Outlook – Multi Agent Systems

- Until now, we discussed systems that consist of single agents acting in an environment
- In real world scenarios, environments often include many agents that base their actions on the behavior or expected behavior of other agents
- This is especially relevant for cognition tasks



- E.g., an autonomos self-driving car agent bases its behavior largely on the (expected) behavior of other agents/people on the road → Game Theory

References - Papers

TRPO:

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *"Trust Region Policy Optimization",* arXiv:1707.06347, 2017

PPO:

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *"Proximal Policy Optimization Algorithms",* ICML, 2015

DeepMimic:

X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, *"Deepmimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills",* arXiv preprint arXiv:1804.02717, 201

Skills From Video:

X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, S. Levine, *" SFV: Reinforcement Learning of Physical Skills from Videos ",* Transactions on Graphics, 2018

Paper from our lab:

S. Christen, S. Stevsic, and O. Hilliges, *Demonstration-guided Deep Reinforcement Learning of Control Policies for Dexterous Human-robot Interaction,* International Conference on Robotics and Automation (ICRA), 2019

Deep Reinforcement Learning that Matters:

P. Henderson, R. Islam, P. Bachman, J. Pineau. D. Precup, and D. Meger, *Deep Reinforcement Learning that Matters,* Conference on Artificial Intelligence AAAI, 2018

References - Slides

Literature:

R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction,* MIT Press, 1998.

Links:

http://rail.eecs.berkeley.edu/deeprlcourse/

https://www.alexirpan.com/2018/02/14/rl-hard.html

https://sites.google.com/view/deep-rl-bootcamp/lectures

https://www.youtube.com/watch?v=RLa5XqH36c8

https://www.youtube.com/watch?v=rQssCd_d8ZA

https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9