


OpenAI Spinning Up

**OpenAI**
Spinning Up

latest

USER DOCUMENTATION


- Introduction
- Installation
- Algorithms
- Running Experiments
- Experiment Outputs
- Plotting Results

INTRODUCTION TO RL

- Part 1: Key Concepts in RL
- Part 2: Kinds of RL Algorithms
- Part 3: Intro to Policy Optimization

[Docs](#) » Welcome to Spinning Up in Deep RL![Edit on GitHub](#)

Welcome to Spinning Up in Deep RL!



[Welcome to Spinning Up in Deep RL! — Spinning Up documentation \(openai.com\)](#)

Reinforcement Learning

- RL is an area of ML concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward.
- RL differs from supervised learning in not needing labelled input/output pairs, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

HW2 Train virtual human to walk using Unity ML agent

HW1, 2 Train regression and classification model

Reinforcement Learning

- The environment is typically stated in the form of a Markov decision process (MDP), because many reinforcement learning algorithms for this context use dynamic programming techniques.[3] The main difference between the classical dynamic programming methods and RL algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

Markov decision process (MDP)

- In mathematics, a Markov decision process (MDP) is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming.

Markov decision process (MDP)

- MDPs were known at least as early as the 1950s;[1] a core body of research on Markov decision processes resulted from Ronald Howard's 1960 book, Dynamic Programming and Markov Processes.[2] They are used in many disciplines, including robotics, automatic control, economics and manufacturing. The name of MDPs comes from the Russian mathematician Andrey Markov as they are an extension of Markov chains.

Core RL algorithms

- Vanilla Policy Gradient (VPG)
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)
- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed DDPG (TD3)
- Soft Actor-Critic (SAC)

They are all implemented with MLP (non-recurrent) actor-critics, making them suitable for fully-observed, non-image-based RL environments.

- These core deep RL algorithms reflect useful progressions of ideas from the recent history of the field, culminating in two algorithms in particular - PPO and SAC-which are close to state of the art on reliability and sample efficiency among policy-learning algorithms.

On-policy algorithms

- On-policy algorithms don't use old data, which makes them weaker on sample efficiency. But this is for a good reason: these algorithms directly optimize the objective you care about-policy performance-and it works out mathematically that you need on-policy data to calculate the updates. So, this family of algorithms trades off sample efficiency in favor of stability—but you can see the progression of techniques (from VPG to TRPO to PPO) working to make up the deficit on sample efficiency.

Unity ML Agent
implementation (PPO)

num_epoch: 3

Off-policy algorithms

- Algorithms like DDPG and Q-Learning are off-policy, so they are able to reuse old data very efficiently. They gain this benefit by exploiting Bellman's equations for optimality, which a Q-function can be trained to satisfy using any environment interaction data (as long as there's enough experience from the high-reward areas in the environment).

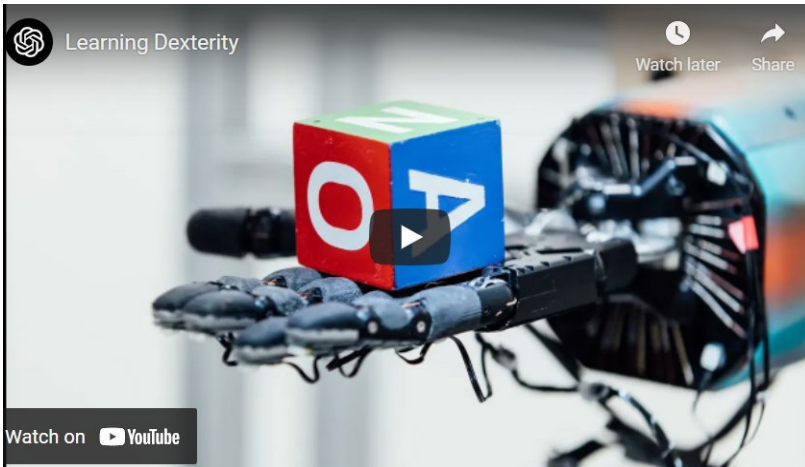
Unity ML Agent
implementation (SAC)

batch_size: 1024
buffer_size: 2000000

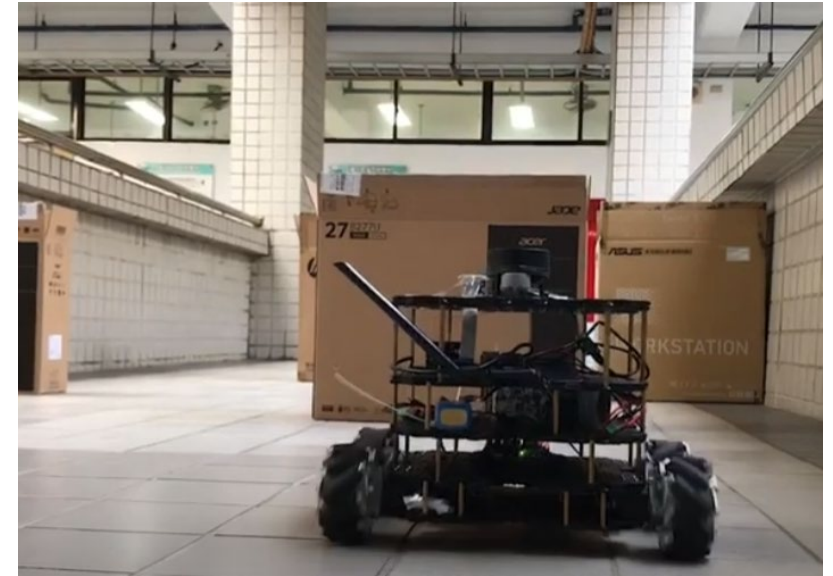
Off-policy algorithms

- But problematically, there are no guarantees that doing a good job of satisfying Bellman's equations leads to having great policy performance. Empirically one can get great performance-and when it happens, the sample efficiency is wonderful-but the absence of guarantees makes algorithms in this class potentially brittle and unstable. TD3 and SAC are descendants of DDPG which make use of a variety of insights to mitigate these issues.

What RL can do?



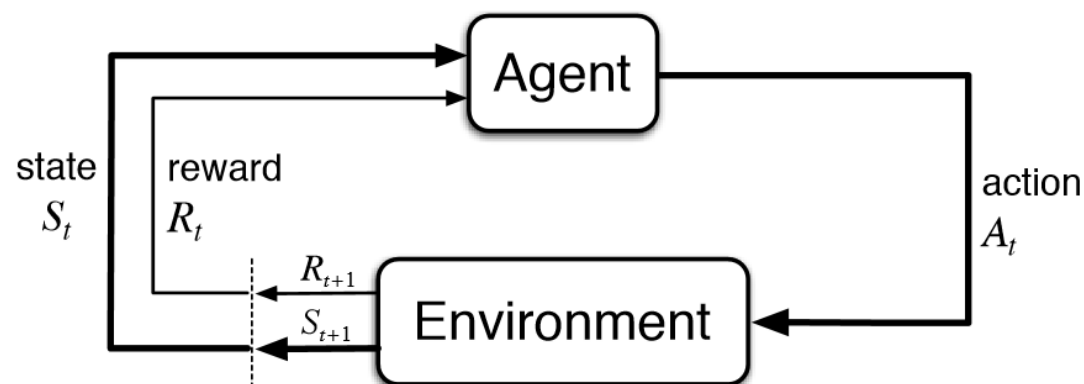
<https://youtu.be/5fVBeqpFqpw>



<https://youtu.be/vtp7AGEBbDM>

https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

Agent-environment interaction loop



(Sutton and Barto, 1998)

- The main characters of RL are the agent and the environment. At every step of interaction, the agent sees an observation of the state of the world, and then decides on an action to take.
- The agent also perceives a reward signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called return.

Key concepts

Policies	$a_t \sim \pi_\theta(s_t)$
Trajectories	$\tau = (s_0, a_0, s_1, a_1, \dots)$
Reward	$r_t = R(s_t, a_t, s_{t+1})$
Return	$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$

$$P(\tau|\pi) = \rho_0(s_0) \cdot \pi(a_0|s_0) \cdot P(s_1|s_0, a_0) \cdot \pi(a_1|s_1) \cdot P(s_2|s_1, a_1) \cdot \dots$$

$$J(\pi) = E_{\tau \sim \pi} (R(\tau))$$

$$V^\pi(s) = E_{\tau \sim \pi} (R(\tau) | s_0 = s)$$

$$Q^\pi(s, a) = E_{\tau \sim \pi} (R(\tau) | s_0 = s, a_0 = a)$$

$$V^\pi(s) = E_{\substack{\tau \sim \pi \\ s' \sim P}} [r(s, a) + \gamma V^\pi(s')]$$

$$Q^\pi(s, a) = E_{s' \sim P} \left[r(s, a) + \gamma E_{a' \sim \pi} [Q^\pi(s', a')] \right]$$

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$