Vilnius University
Faculty of Mathematics and Informatics
Institute of Computer Science

Course Work

Agent training using deep reinforcement learning

(Agento apmokymas naudojant gilųjį skatinamąjį mokymąsi)

By: 3rd course, 4th group student
Jokūbas Kondrackas (Signature)
Supervisor:
Andrius Grevys (Signature)

Vilnius
2020

Table of Contents

Introduction

Reinforcement learning (RL) is a computational approach to understanding and automating goal-directed learning and decision making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, without requiring exemplary supervision or complete models of the environment. [SB18a]

The advantage of reinforcement learning versus other machine learning areas is that it does not need any training data beforehand and learns as it goes along, similar to how humans or animals would learn to handle given tasks.

In this study, I'll shortly describe Reinforcement Learning, its history, algorithms and then compare the tools provided by Unity Technologies and OpenAI to create virtual sandboxes capable of training agents based on DRL.

Reinforcement Learning

The history of reinforcement learning has two main threads, both long and rich, that were pursued independently before intertwining in modern reinforcement learning. One thread concerns learning by trial and error and the other thread concerns the problem of optimal control and its solution using value functions and dynamic programming.

Firstly, I'll discuss the term "optimal control" thread that I've mentioned before. This term came into use in the late 1950s to describe the problem of designing a controller to minimize a measure of a dynamical system's behavior over time. One of the approaches to this problem was developed in the mid-1950s by Richard Bellman and others through extending a nineteenth century theory of Hamilton and Jacobi. This approach uses concepts of a dynamical system's state and of a value function, of "optimal return function," to define a functional equation, now often called the Bellman equation. The class of methods for solving optimal control problems by solving this equation came to be known as dynamic programming.

The other thread leading to the modern field of reinforcement learning, centered on the idea of trial and error learning. The thread began in psychology, where "reinforcement" theories of learning are common. Perhaps the first to succinctly express the essence of trial-and-error learning was Edward Thorndike. We take this essence to be the idea that actions followed by good or bad outcomes have their tendency to be reselected altered accordingly. [SB18b]

The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the agent. The thing it interacts with, comprising everything outside the agent is the environment. These continually interact with each other. The agent selects actions and the environment responds to those actions and presents a new situation to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time.

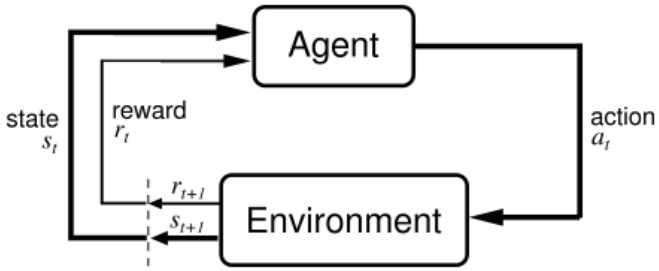


Image: Agent-Environment Interface, Richard S. Sutton and Andrew G. Barto, Reinforcement Learning - An Introduction (second edition)

Simply put, the agent and environment interact at each of a sequence of discrete time steps. At each time step t, the agent receives some representation of the environments state, and on that basis selects an action. One-time step later, in part as a consequence of its action, the agent receives a numerical reward, and finds itself in a new state. [SB18c]

Model-Free vs Model-Based Reinforcement Learning

One of the most important branching points in a Reinforcement Learning algorithm is the question of whether the agent has access to a model of the environment.

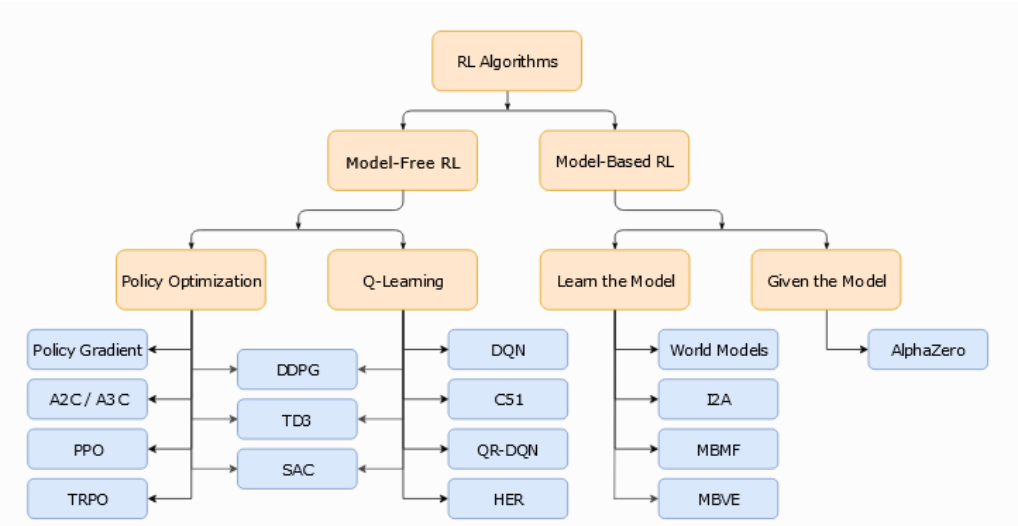


Image: taxonomy of algorithms in modern Reinforcement Learning, spinningup.openai.com

Model is the agent's view of the environment, which maps state-action pairs to probability distributions over states. The main upside of having a model is that it allows the agent to plan by thinking ahead. Seeing what would happen for a range of possible choices, and explicitly deciding between its options. Agents can then distill the results from planning ahead into a learned policy. However, the main downside is that a ground-truth model of the environment is usually not available to the agent. If an agent wants to use a model in this case, it has to learn the model purely from experience.

Model-based Reinforcement Learning uses experience to construct an internal model of the transitions and immediate outcomes in the environment. Appropriate actions are then chosen by searching or planning in this world model.

Model-free Reinforcement Learning, on the other hand, uses experience to learn directly one or both of two simpler quantities (state/action values or policies) which can achieve the same optimal behavior but without estimation or use of a world model. Given a policy, a state has a value, defined in terms of the future utility that is expected to accrue starting from that state.

Model-free methods are statistically less efficient than model-based methods, because information from the environment is combined with previous, and possibly

erroneous, estimates or beliefs about state values, rather than being used directly. A critical branching point in a Reinforcement Learning algorithm is the question of what to learn. This will be discussed next. [Oai18a]

What to Learn in Model-Free Reinforcement Learning

There are two main approaches to representing and training agents with a model-free RL:

Policy Optimization.

Q-Learning.

In Policy optimization methods the agent learns directly the policy function that maps state to action. The policy is determined without using a value function. There are two types of policies: deterministic and stochastic. Deterministic policy maps state to action without uncertainty, while the Stochastic policy outputs a probability distribution over actions in a given state. Some of the methods include:

Policy Gradient

Asynchronous Advantage Actor-Critic

Trust Region Policy Optimization

Proximal Policy Optimization

Q-learning learns the action-value function Q(s, a): how good is it to take an action at a particular state. Basically a scalar value is assigned over an action given the state s. Methods used with Q-learning:

Deep Q Neural Network

C51

Distributional Reinforcement Learning with Quantile Regression

Hindsight Experience Replay

Policy Optimization Algorithms

Policy Gradient:

In this method, we have the policy π that has the parameter θ. This π outputs a probability distribution of actions.

$$\pi_\theta(a|s) = P[a|s]$$

Then we must find the best parameters to maximize a score function J(θ), given the discount factor γ and the reward r.

$$J(\theta) = E_{\pi\theta}[\sum \gamma r]$$

Main steps:

Measure the quality of a policy with the policy score function.

Use policy gradient ascent to find the best parameter that improves the policy. [Sim18a]

Asynchronous Advantage Actor-Critic (A3C)

This method was published by Google's DeepMind group and covers the following key concept embedded in it's naming:

Asynchronous: Several agents are trained in it's own copy of the environment and the model from these agents are gathered in a master agent.

Advantage: Similarly to PG where the update rule used the discounted returns from a set of experiences in order to tell the agent which actions were "good" or "bad"

Actor-critic**:** combines the benefits of both approaches from policy-iteration method as PG and value-iteration method as Q-learning. The network will estimate both a value function and a policy. [Jul16]

Trust Region Policy Optimization (TRPO)

An on-policy algorithm that can be used on environments with either discrete or continuous action spaces. TRPO updates policies by taking the largest step possible to improve performance, while satisfying a special constraint on how close the new and old policies are allowed to be. [Oai18b]

Proximal Policy Optimization (PPO)

An on-policy algorithm which similarly to TRPO can perform on discrete or continuous action spaces. PPO shares motivation with TRPO in the task of answering the question: how to increase policy improvement without the risk of performance collapse? The idea is that PPO improves stability of the actor training by limiting the policy update at each training step. [Sim18b]

Q-Learning Algorithms

Deep Q Neural Network (DQN)

DQN is a RL technique that is aimed at choosing the best action for given circumstances. Each possible action for each possible observation has its 'Q' value, where 'Q' stands for a quality of a given move. To end up with accurate Q values we need to use neural networks and linear algebra. For each state experienced by our agent, we will have to remember it and perform an experience replay. An experience replay is a biologically inspired process that uniformly samples experiences from memory and for each entry updates its Q value. Simply put, we are calculating the new q by taking the maximum q for a given action, multiplying it by the discount factor and ultimately adding it to the current state reward.

Formal notation:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Image: Q-learning, https://en.wikipedia.org/wiki/Q-learning

[Sim18c]

C51

C51 is a feasible algorithm proposed by Bellemare et al. to perform iterative approximation of the value distribution Z using the Distributional Bellman equation. The number 51 represents the use of 51 discrete values to parameterize the value distribution Z(s, a). [Yu17]

Distributional Reinforcement Learning with Quantile Regression (QR-DQN)

In QR-DQN for each state-action pair instead of estimating a single value a distribution of values is learned. The distribution of the values, rather than just the average can improve the policy. This means that quantiles are learned which threshold values are attached to certain probabilities in the cumulative distribution function. [Ful17]

Hindsight Experience Replay (HER)

In HER method, a DQN is supplied with a state and a desired end-state. It allows quick learning when the rewards are sparse. [Kim18]

What to learn in model based RL

Unlike model-free RL, there aren't a small number of easy-to-define clusters of methods for model-based RL: there are many ways of using models. [Oai18a]

Pure Planning

The most basic approach never explicitly represents the policy, and instead, uses pure planning techniques like model-predictive control (MPC) to select actions. In MPC, each time the agent observes the environment, it computes a plan which is optimal with the respect to the model, where the plan describes all actions to take over some fixed window of time. The agent then executes the first action of the plan, and immediately discards the rest of it. It computes a new plan each time it prepares to interact with the environment, to avoid using an action from a plan with a shorter planning horizon. [Oai18a]

Expert Iteration

A follow-on to Pure planning involves using and learning an explicit representation of the policy. The agent uses a planning algorithm in the model, generating candidate actions for the plan by sampling from its current policy. The planning algorithm produces an action which is better than what the policy alone would have produced. The policy is afterwards updated to produce an action more like the planning algorithm's output. [Oai18a]

Unity

Unity is a powerful cross-platform 3D engine. It is a complete 3D environment, suitable for laying out levels, creating menus, doing animation, writing scripts, organizing projects and so on. Thanks to the Unity Machine Learning agent package, it is an excellent tool for training agents and creating custom environments for those agents to learn in.

Simplified, Unity environment is based around Game Objects. These game objects on their own are completely empty objects with no features, but thanks to the component system, we can add an incredible amount of functionality to those Game Objects, for example renderers, colliders, animation systems, audio systems and custom components that we can code ourselves. Due to this design, we can create any kind of objects that we might want. This system allows us to have complete control over our agents and create them however we want.

Training agents with deep reinforcement learning using Unity Machine learning agents.

Unity Machine Learning Agents (ML-Agents) trains intelligent agents with reinforcement learning and evolutionary methods via a Python API. It is an open-source project that enables games and simulations to serve as environments for training agents. Many of the used algorithms in ML-Agents toolkit leverage some form of deep learning. More specifically, Unity implementations are built on top of the open-source library Tensor Flow. [MSP+G20]

TensorFlow

TensorFlow is an open source library for performing computations using data flow graphs. It facilitates training and inference on CPUs and GPUs in a desktop, server, or even a mobile device. When training the behavior of agents on ML-Agents toolkit, the output is a model (.nn) file that one can associate with an agent. However, the use of TensorFlow is mostly abstracted away and behind the scenes in Unity. [Uni20a]

Built-in Training and Inference

The ML-Agents toolkit ships with several implementations of state-of-the-art algorithms for training intelligent agents. More specifically, during training, all of the

medics in the scene send their observations to the Python API through the External Communicator. The Python API processes these observations and sends back actions for each medic to take. During training these actions are mostly exploratory to help the Python API learn the best policy for each medic. Once training concludes, the learned policy for each medic can be exported as a model file. Then during the inference phase, the medics still continue to generate their observations, but instead of being sent to the Python API, they will be fed into their model to generate the optimal action for each medic to take at every point in time. [Uni20b]

Custom Training and Inference

Any user of the ML-Agents Toolkit can leverage their own algorithms for training. In this case, the behaviors of all the agents in the scene will be controlled within Python. You can even turn your environment into a gym. [Uni20b]

Deep reinforcement learning within the ML-Agents package

ML-Agents provide an implementation of two reinforcement learning algorithms:

- Proximal Policy Optimization (PPO)
- Soft Actor-Critic (SAC)

The default algorithm is PPO. This is a method that has been shown to be more general purpose and stable than many other RL algorithms. [Uni20b]

Example Unity Environments

The Unity ML-Agents Toolkit includes an expanding set of example environments that highlight the various features of the toolkit. At the moment there are 15+ example Unity Environments. These environments can also serve as templates for new environments or as ways to test new ML algorithms.

- Basic
- 3DBall: 3D Balance Ball
- GridWorld
- Tennis
- Push Block
- Wall Jump
- Reacher
- Crawler
- Worm
- Food Collector
- Hallway
- Bouncer
- Soccer Twos
- Strikers Vs. Goalie
- Walker
- Pyramids

In Addition to those, it is possible to create an environment from scratch within the Unity Engine. [Uni20c]

Example Environment: 3DBall

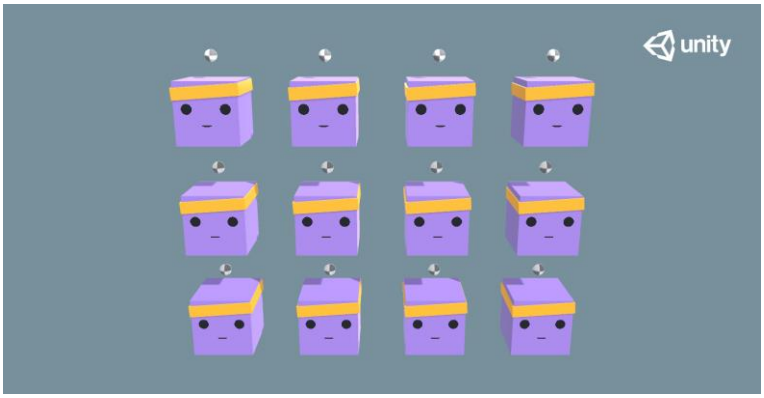


Image: 3D Ball example project screenshot in the Unity Editor

- Set-up: A balance-ball task, where the agent balances the ball on its head.
- Goal: The agent must balance the ball on its head for as long as possible.
- Agents: The environment contains 12 agents of the same kind, all using the same Behavior Parameters.
- Agent Reward Function:
    - +0.1 for every step the ball remains on its head.
    - -1.0 if the ball falls off.
- Float Properties: Three
    - scale: Specifies the scale of the ball in the 3 dimensions (equal across the three dimensions)
        - Default: 1
        - Recommended Minimum: 0.2
        - Recommended Maximum: 5
    - gravity: Magnitude of gravity
        - Default: 9.81
        - Recommended Minimum: 4
        - Recommended Maximum: 105
    - mass: Specifies mass of the ball
        - Default: 1
        - Recommended Minimum: 0.1
        - Recommended Maximum: 20

Training 3D Ball:

Instead of using a pre-trained model provided by Unity, I trained the model myself. I stopped the training when the Cumulative Reward was converging near 100, which is the maximum that the agent can achieve in this environment.

Hyper Parameters:

Batch size: 1200

Buffer size: 12000

Learning rate: 0.0003

Beta: 0.001

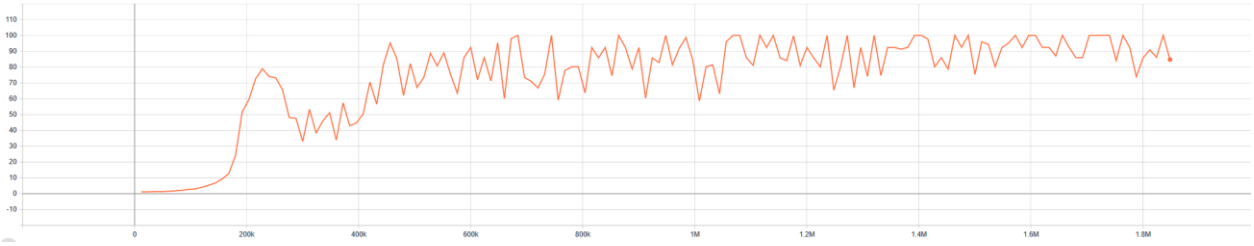
Epsilon: 0.2

Lambda: 0.95

Epoch number: 3

Training results:

Cumulative Reward:



x axis - steps

y axis - cumulative reward

I've successfully trained my agents to converge on the maximum amount of cumulative reward that is possible in this environment.

Unity Conclusions

The Unity ML-Agents package is still in early development, but it already has a lot of useful features and support for multiple environment configurations and training scenarios. It has great documentation with step by step tutorials on how to set up the environment, train agents and create new environments from scratch. The most important feature is that it is very simple to begin with and it's possible to integrate the agents into your already existing project.

The only issues that I've encountered with unity can be explained by the fact that it's still a preview package which is in active development. These issues can range from minor annoyances to complete breakage of an example environment.

OpenAI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong of Pinball. It makes no assumptions about the structure of your agent, and is compatible with any numerical computation library, such as TensorFlow.

The Gym library is a collection of test problems - environments - that you can use to work out your reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms. This means that the user has to write training algorithms from scratch. [Gym16a]

Available Environments:

Classic control, complete small-scale tasks, mostly from the reinforcement learning literature. Examples include:

- Acrobot-v1
- CartPole-v1
- MountainCar-v0
- Pendulum-v0
- Algorithms:

Algorithmic, perform computations such as adding multi-digit numbers and reversing sequences. One might object that these tasks are easy for a computer, but the challenge is to learn these algorithms purely from examples. These tasks have the nice property that it is easy to vary the difficulty by varying the sequence length. Examples include:

- Copy-v0
- DuplicatedInput-v0
- RepearCopy-v0
- Reverse-v0

Atari, play classic Atari games. Examples include:

- AirRaid-v0
- Alien
- Amidar
- Assault

- Asteroids

2D and 3D robots, control a robot in simulation. These tasks use the MuJoCo physics engine, which was designed for fast and accurate robot simulation. Included are some environments from a recent benchmark by UC Berkeley researchers. MuJoCo is proprietary software, but offers free trial licenses

- Ant-v2
- Humanoid-v2
- Walker2d-v2
- HalfCheetah-v2

Example Environment: CartPole-v1

A pole is attached by an un-actuated joint to a card, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. [Gym16b]

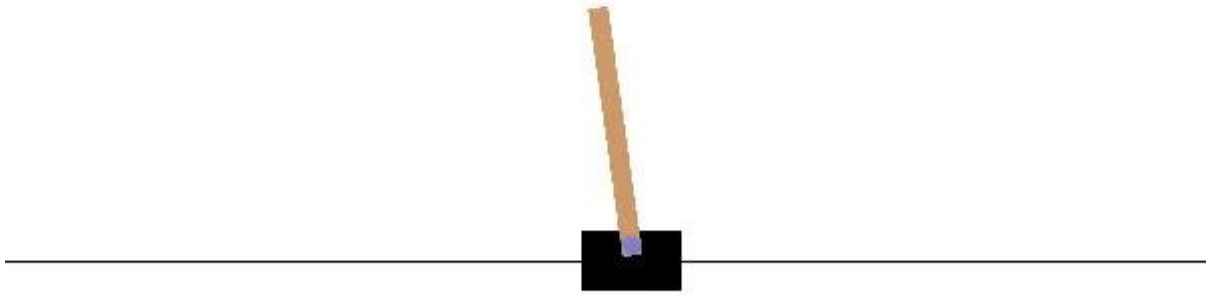


Image: CartPole-v1, https://gym.openai.com/envs/CartPole-v1/

Training CartPole-v1 agent

Initially we start with the initial environment. It doesn't have any associated reward yet, but it has a state. Then, for each iteration, an agent takes the current state, picks the best action and executes it in an environment. Subsequently, the environment returns a reward for a given action, a new state and an information if the new state is terminal. Afterwards, we remember our state, action, reward, next state, terminal (SARS) and perform Experience replay. To train our agent, I'll be using Deep Q-Learning (DQN). [Sur18]

Hyperparameters:

Gamma = 0.95

Learning Rate = 0.001

Memory Size= 1000000

Batch Size = 20

Exploration Max = 1.0

Exploration Min = 0.01

Exploration Decay = 0.995

Training results:

x axis - iterations

y axis - Cumulative reward

I managed to successfully train my agent in this environment.

Open AI Gym Conclusions

Gym is already a mature library, the official status of it is Maintenance, so they won't be introducing any new features or environments. It is a very reliable library and the environments that they provide are really good, however, the documentation is lackluster. In my opinion, Gym is more reliable than the Unity ML-agents package, however it also lacks the freedom and simplicity that the Unity Engine offers.

Gym vs ML-Agents

Gym:

- More stable than ML-Agents
- Simple setup
- Targeted at researchers and scientists
- More difficult to start with
- A lot of environments ranging from simple examples to complex Atari games

Unity ML-Agents:

- Incredible documentation
- Complex setup
- Targeted at regular developers (but can still be used for research)
- Not as stable as Gym
- 15 Example Unity environments
- Thanks to the Unity Engine it's easy to create new environments from scratch

Literature:

[Bon17] Why Reinforcement Learning Might Be the Best AI Technique for Complex Industrial Systems, https://medium.com/@BonsaiAI/why-reinforcement-learning-might-be-the-best-ai-technique-for-complex-industrial-systems-fde8b0ebd5fb

[Ful17] Evan Fuller, Quantile Reinforcement Learning, https://medium.com/@fuller.evan/quantile-reinforcement-learning-56f8b3c3f134

[Gym16a] OpenAI, Gym documentation, https://gym.openai.com/docs/

[Gym16b] OpenAI, CartPole-v1, https://gym.openai.com/envs/CartPole-v1/

[Jul16] Arthur Julianim Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C), https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

[Kim18] Min Sang Kim, Learning from mistakes with Hindsight Experience Replay https://becominghuman.ai/learning-from-mistakes-with-hindsight-experience-replay-547fce2b3305

[LHI+18] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, Joelle Pineau, An Introduction to Deep Reinforcement Learning 2018

[MSP+G20] Marwan Mattar, Jeffrey Shih, Vincent-Pierre Berges, Chris Elion and Chris Goy, Announcing ML-Agents Unity Package v1.0! https://blogs.unity3d.com/2020/05/12/announcing-ml-agents-unity-package-v1-0/

[Oai18a] OpenAI, Spinning Up Docs, Part 2: Kinds of RL Algorithms, https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[Oai18b] OpenAI, Spinning Up Docs, Trust Region Policy Optimization https://spinningup.openai.com/en/latest/algorithms/trpo.html

[SB18a] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning - An Introduction (second edition), Chapter 1.6 2018;

[SB18b] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning - An Introduction (second edition), Chapter 1.7 2018;

[SB18c] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning - An Introduction (second edition), Chapter 3.1 2018;

[Sim18a] Thomas Simonini, An introduction to Policy Gradients with Cartpole and Doom, https://www.freecodecamp.org/news/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f/

[Sim18b] Thomas Simonini, Proximal Policy Optimization (PPO) with Sonic the Hedgehog 2 and 3 https://towardsdatascience.com/proximal-policy-optimization-ppo-with-sonic-the-hedgehog-2-and-3-c9c21dbed5e

[Sim18c] Thomas Simonini, An introduction to Deep Q-Learning: let's play Doom, https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/

[Sur18] Greg Surma, Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning), https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288

[Sur19] gsurma, Reinforcement Learning solution of the OpenAI's CartPole, https://github.com/gsurma/cartpole

[Uni20a] Unity Technologies, ml-agents Documentation Background: TensorFlow, https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-TensorFlow.md

[Uni20b] Unity Technologies, ml-agents Documentation ML-Agents-Overview https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md

[Uni20c] Unity Technologies, ml-agents Documentation Learning-Environment-Examples, https://github.com/Unity-Technologies/ml-agents/blob/release_2/docs/Learning-Environment-Examples.md

[Yu17] Felix Yu, Distributional Bellman and the C51 Algorithm https://flyyufelix.github.io/2017/10/24/distributional-bellman.html