Introduction to DRL

Introduction to DRL

P. Durdevic[1]

[1]Department of Energy Technology
Aalborg University

Wednesday 13th July, 2022

Table of Contents

Introduction

Introduction

Agenda

- What is Reinforcement Learning & Deep Reinforcement Learning?
- Why DRL?
  - *"When evaluating a policy is inexpensive, as in board games; there are sufficient resources to perform a near brute-force optimization, as in evolutionary optimization; no other control strategy works."* Brunton and Kutz [2022]

Goal of this lecture

1. Basic concepts and theory of RL
2. Give an example of RL through most fundamental method **Q-learning**
   - Introduce foundation theory for understanding **Q-learning**
   - e.g. Dynamic programing, Markov Decision Process, Monte Carlo Learning, Temporal-Difference Learning, etc.
3. Demonstrate **OpenAI GYM**
   - Show an example of the environment
   - Show an example of Q-learning

Introduction

Introduction

Idea	Which actions to take in order to improve the accumulated reward
Application	Many areas: Control, multi-agent systems, etc.
Basic concept	modeled as a MDP *(More details later)*
Scope	Different families of RL *(We look at some)*
Impressive	Can control systems from Pixel inputs only

Example: Riding a bicycle

- Define the *Action* and *State Space*
- Try $\rightarrow$ fail $\rightarrow$ *'reward'* $\rightarrow$ try $\rightarrow$ fail $\rightarrow$ *'reward'* . . . try $\rightarrow$ Succeed $\rightarrow$ *'reward'*

TODO - Finish

Challenges of RL

- *Many theoretical convergence results, and many fundamental RL algorithms, only apply to finite MDPs, characterized by finite actions A and states S* brunton2022data
- Rewards are often delayed and very rare
  - Result = high computation expense
- Example is to find the goal in a maze, where the reward is only given at the exit. (better version to give at each turn)
- Optimizing the policy often requires **MANY** action sequences *(computationally expensive)*

Overview of RL Algorithms

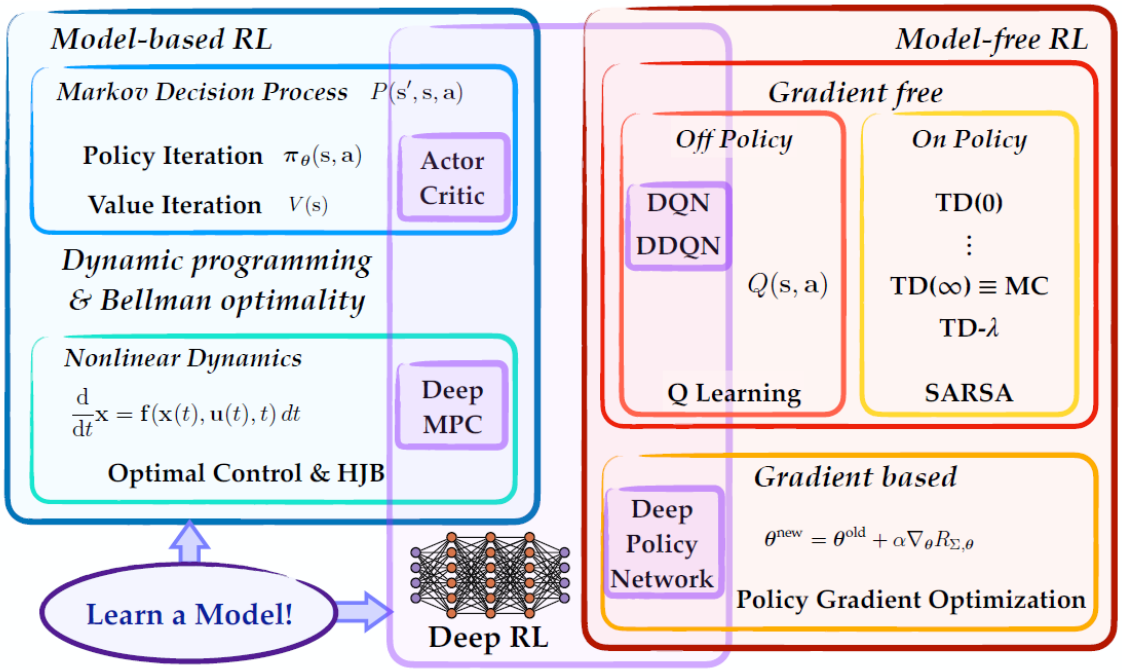


Figure: Figure borrowed from Brunton and Kutz [2022] make my own figure

- Two main categories (**Model based** and **Model free**)
- Our focus will be on the model free (explanation follows)

Quick Intro to Model Based

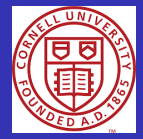
Here explain when to use model based, then argue why it is not good for robotics where we use high dimensional space with a high dimensional input from the camera

- In many cases, such as in robotics with non-linearities and high dimentional state and action space (camera, UAV,...) what is the problem
- The solution is to use model-free optimization techniques

Model Free Methods - Our Focus

Here explain when to use model based, then argue why it is not good for robotics where we use high dimentional space with a high dimention input from the camera

Theory of Reinforcement Learning



Bandits

Introduction

TODO maybe short about bandits

Mathematical Formulation

The RL Process & Markov Decision Process (MDP)

RL in a few simple steps

- An agent observes a state $s_k \in \mathcal{S}$ in its environment at time step k
- The agent takes actions $a_k \in \mathcal{A}$ in state s_k
- The environment and agent transition to state s_{k+1} based on the action a_k and the current state s_t
- At every transition the agent receives a reward $r_{k+1} \in \mathcal{R}$ (might be a delayed or immidiate)

RL Policy

- The goal of the agent is to learn a policy π that maximizes the expected return
- An optimal policy is a policy which maximizes the expected return in the environment

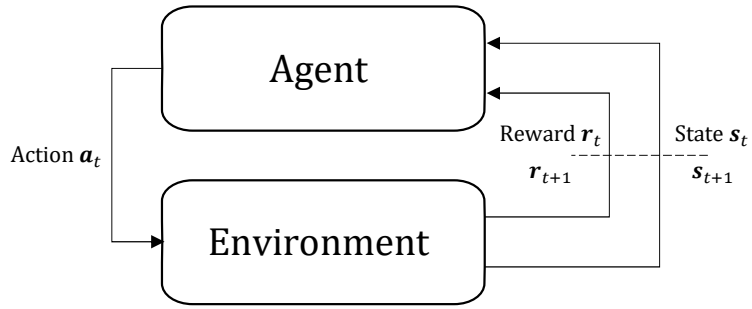


Figure: **The RL process**

Arulkumaran et al. [2017], Brunton and Kutz [2022]

Mathematical Formulation

The RL Process & Markov Decision Process (MDP)

RL in a few simple steps

- An agent observes a state $s_k \in \mathcal{S}$ in its environment at time step k
- The agent takes actions $a_k \in \mathcal{A}$ in state s_k
- The environment and agent transition to state s_{k+1} based on the action a_k and the current state s_t
- At every transition the agent receives a reward $r_{k+1} \in \mathcal{R}$ (might be a delayed or immidiate)

RL Policy

- The goal of the agent is to learn a policy π that maximizes the expected return
- An optimal policy is a policy which maximizes the expected return in the environment

Relation To control Theory

RL	*agent, environment and action*
Control	*controller, controlled, system (plant), control signal*

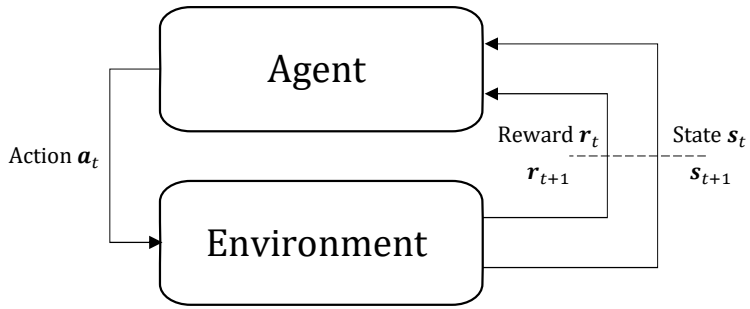


Figure: **The RL process**

Arulkumaran et al. [2017], Brunton and Kutz [2022]

Mathematical Formulation

The Policy

- An agent senses s and takes a action through policy π
- Discrete time steps t, (k)
- The policy π is optimized by learning to maximize future rewards r
- I.e. optimization problem - learn the policy $\pi(\mathbf{s}, \mathbf{a})$

$$\pi(\mathbf{s}, \mathbf{a}) = \Pr(\mathbf{a} = \mathbf{a} | \mathbf{s} = \mathbf{s}) \tag{1}$$

- May be a look-up table of the discrete $\mathcal{S}$ and $\mathcal{A}$. [but this is **EXPENSIVE** in most cases]

INSTEAD represent π as an approximate function parametrized by a lower-dimensional vector θ as follows:

$$\pi(\mathbf{s}, \mathbf{a}) \approx \pi(\mathbf{s}, \mathbf{a}, \theta) \tag{2}$$

It is possible to represent $\pi(\mathbf{s}, \mathbf{a}, \theta)$ with a deep neural network.

Brunton and Kutz [2022], check formulation in Sutton and polish the language

Mathematical Formulation

The Environment

- In its simplest form, the environment, follows the Markov Decision Process (MPD)
  - Thus the probability of a system state is determined by it previous state.
- In most cases, especially in robotics, this is not the case and in many cases the model is too complex to be known **(A motivation for Model Free)**
  - **Model Based**: model is either found using first principles or using a data driven approach (e.g. Deep Learning)
  - The latter is preferred for complex, non-linear, coupled, time varying, dynamical systems.

Next a short introduction to MDP

Brunton and Kutz [2022], check formulation in Sutton and polish the language

Mathematical Formulation

The Environment **[MDP]**

MDP control loop on page 9 the algorithm pseudo code is really a good idea and the entire MDP description is really good.

Graesser and Keng [2019]

Mathematical Formulation

The Environment **[MDP]**

The MDP is defined as a 4-tuple, $[\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}]$, where:

$\mathcal{S}$ = A set of *states*, i.e. the state space

$\mathcal{A}$ = A set of *actions*, i.e. the action space

$\mathcal{R}$ = The immediate/expected immediate - reward,
received after transitioning from state $\mathbf{s}_k$ to state $\mathbf{s}_{k+1}$ after performing an action $\mathbf{a}_k$

P = The probability that action $\mathbf{a}_k$ in state $\mathbf{s}_k$ at time t will lead to state $\mathbf{s}_{k+1}$ at time t_{k+1}, [1]

Where $\mathcal{R}$, P are given as:

$$\mathcal{R}(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \Pr(\mathbf{r}_{k+1} | \mathbf{s}_{k+1} = \mathbf{s}', \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a}) \tag{3}$$

$$P(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \Pr(\mathbf{s}_{k+1} = \mathbf{s}' | \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a}) \tag{4}$$

Alternative notation for the transition probability $P(\mathbf{s}', \mathbf{s}, \mathbf{a})$ can be given as $P(\mathbf{s}' | \mathbf{s}, \mathbf{a})$ Brunton and Kutz [2022]
In addition, $\mathcal{A}$ can be defined as:

$\mathcal{A}_s$ which is the set of actions available from the state $\mathbf{s}$

[1] function P defines the dynamics of the MDP

[1] function P defines the dynamics of the MDP

Brunton and Kutz [2022], check formulation in Sutton and polish the language

Mathematical Formulation

The Environment **[MDP]**

This sequence of state, action and reward, is a *trajectory*:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \ldots \tag{5}$$

Where in a *finite* **MDP**, $[S, A, R]$ have a finite number of elements.
Sutton page 48

Brunton and Kutz [2022],Sutton and Barto [2018]

Mathematical Formulation

The Environment **[MDP]**

In some cases the action a can be represented by the policy π
Extra material, maybe add later, see equation 11.5 on page 503
probably a good idea to add as it involves the policy, but investigate in Sutton if there is a better transition to the value function

Brunton and Kutz [2022],Sutton and Barto [2018]

Mathematical Formulation

The Environment **[Markov Process and MPD]**

- A Markov process (Markov Chain) is stochastic model.
- sequence of events occurring with some probability
- The events probability of occurring is determined only by the state attained in the previous event

Example of a two state Markov Process

Given a finte set of states S and a vector of probility of bin in each state $[n]$, $s \in \mathcal{R}^n$.
Then the markov process $P(s', s)$ can be written using a ptobability matrix T as

$$s' = \mathsf{T}s \tag{6}$$

and an MDP can be written as

$$s' = \sum_{a \in A} \pi(s, a)\mathsf{T}_a s \tag{7}$$

Mathematical Formulation - The Environment **[POMDP]**

TODO

"A partially observable Markov decision process (POMDP) is a generalization of a Markov decision process (MDP). A POMDP models an agent decision process in which it is assumed that the system dynamics are determined by an MDP, but the agent cannot directly observe the underlying state."

Mathematical Formulation

Value Function

- For the **Value Function**, we estimate the **value**, i.e. the **Expected Return**
- The value function $V_\pi(\mathbf{s})$, (*State-value function arulkumaran2017deep*), given a policy π, is a measure of *"desirability of being in a given state"* brunton2022data.

$$V_\pi(\mathbf{s}) = \mathbb{E}\left(\sum_k \gamma^k \mathbf{r}_k | \mathbf{s}_0 = \mathbf{s}\right) \tag{8}$$

- That is, V_π is given by the expected value of the accumulated reward $\mathbf{r}_k$ discounted by a factor γ.
- Note that γ^k and that $\gamma \in [0, 1]$, this results in more emphasis on immediate rewards arulkumaran2017deep.

We can find the best possible policy*(optimal policy)* by reformulating the value function:

$$V(\mathbf{s}) = \max_\pi \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_k | \mathbf{s}_0 = \mathbf{s}\right) \tag{9}$$

Explain equation

Brunton and Kutz [2022]

Mathematical Formulation

Value Function

We can find the best possible policy*(optimal policy)* by reformulating the value function:

$$V(\mathbf{s}) = \max_{\pi} \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_k | \mathbf{s}_0 = \mathbf{s}\right) \tag{8}$$

The value at a state **s** can be define recursively as follows:

$$V(\mathbf{s}) = \max_{\pi} \mathbb{E}\left(\mathbf{r}_0 + \sum_{k=1}^{\infty} \gamma^k \mathbf{r}_k | \mathbf{s}_1 = \mathbf{s}_{k+1}\right) \tag{9}$$

Which is the same as:

$$V(\mathbf{s}) = \max_{\pi} \mathbb{E}\left(\mathbf{r}_0 + \gamma V(\mathbf{s}_{k+1})\right) \tag{10}$$

Which is the Bellman equation, and a statement of Bellman's principle of optimality.
Now the policy π can be found as follows:

$$\pi = \arg\max_{\pi} \mathbb{E}\left(\mathbf{r}_0 + \gamma V(\mathbf{s}_{k+1})\right) \tag{11}$$

Brunton and Kutz [2022]

Mathematical Formulation

Bellman Equation in MDP

The bellman equation expected reward for taking an action following some policy π

Bellman optimality equation

Brunton and Kutz [2022]

Mathematical Formulation - Short on max and argmax

Formulation
TODO

Brunton and Kutz [2022], check formulation in Sutton and polish the language

Model Based

Model Based - Quality function

todo

value learning vs. policy learning

Model Free

Model Free Methods - Introduction

- When a model is not available
- Many different methods
- We will start with Q-learning
  - Learns the Q function from interaction with the environment
  - Before we introduce Q-**learning**, we start by introducing **Monte Carlo learning** and **Temporal difference learning**

Quality Function TO BE MOVED TO MODEL BASED METHODS

$$Q(\mathbf{s}, \mathbf{a}) = \mathbb{E}(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') \tag{12}$$

$$= \sum_{\mathbf{s}'} P(\mathbf{s}'|\mathbf{s}, \mathbf{a})(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')) \tag{13}$$

where $s' = s_{k+1}$

add policy and value iteration to show that they are very similar in order for the following to make sense

From earlier we can now see some similarity between the optimal policy and the optimal value function, and they are related by the quality function, as follow.

$$\pi(\mathbf{s}, \mathbf{a}) = \arg\max_a Q(\mathbf{s}, \mathbf{a}) \tag{14}$$

$$V(\mathbf{s}) = \max_a Q(\mathbf{s}, \mathbf{a}) \tag{15}$$

Monte Carlo Learning

- Based on Monte Carlo random sampling
  - short example page 30 in Graesser and Keng [2019]
- *"Learn the Q and V function using Monte Carlo random sampling of the state-action space through repeated evaluation of many policies"*
- requires an *episodic RL task*
- Brute force search
- Typically it is performed *on-policy*

Temporal Difference Learning (TD)-learning

Also Check Laura page 56

- Important to RL
- Combination of Monte Carlo (MC) and Dynamic programming (DP)
  - learn directly from raw experience without a model of the environment's dynamics, **like MC**
  - update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap), **like DP**
- Sample-based learning strategy
- not restricted to episodic tasks finish based on notes
- finish based on notes

TD(0): **one-step** look ahead

- Policy evaluation, (*Prediction problem*)
- Estimating the *value function* v_π for a given π
- To find the optimal policy we use *Generalized policy iteration* (GPI)
- **TD Method:** Get estimate for $V(S_t)$ based on the observed reward R_{t+1} and the estimate $V(S_{t+1})$
- Its simplest form:

$$V(S_t) \leftarrow V(S_t) + [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{16}$$

- TD(0) bases its update in part on an existing estimate, thus it is a *bootstrapping method*, like **DP**
- The TD target samples the expected values in $v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$ and it uses the current estimate V instead of the true v_π. sutton2018reinforcement
- TD methods combine the sampling of Monte Carlo with the bootstrapping of DP.
- Where α is a constant *step-size*parameter

TD(0): **one-step** look ahead

- Important to note is that $[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$ represents a form for error
- But note that this error in $V(S_t)$ is available only at time $t + 1$

State–action–reward–state–action learning (SARSA) *[On-policy TD Control]*

- Learn the Q-function on-policy

State–action–reward–state–action learning (SARSA) *[On-policy TD Control]*

- Learn the Q-function on-policy

on-policy

"An off-policy learner learns the value of the optimal policy independently of the agent's actions. Q-learning is an off-policy learner. An on-policy learner learns the value of the policy being carried out by the agent including the exploration steps." `http://artint.info/html/ArtInt_268.html`

State–action–reward–state–action learning (SARSA) *[On-policy TD Control]*

- Learn the Q-function on-policy
- transitions from state–action pair to state–action pair and learn the values of state–action pairs
- The Q update equation

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha \left(\mathbf{r}_k + \gamma Q^{\text{old}}(\mathbf{s}_{k+1}, \mathbf{a}_{k+1}) - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) \right) \tag{16}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right] \tag{17}$$

- Uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ **(SARSA)**
- finish highlighted in Brunton (similar to solving exercise 6.8 in sutton)
- explain old and new from sutton

Q-learning *[Off-policy TD Control]*

sutton p. 131

Q-learning *[Off-policy TD Control]*

- "*Off-policy TD learning scheme for the Q function*"
- Q update equation:

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha \left(\mathbf{r}_k + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{k+1}, \mathbf{a}) - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) \right) \tag{18}$$

Where α is the learning rate and γ is the discount factor.

- *"Q-learning applies to discrete action spaces A and state spaces S governed by a finite MDP."*
- *"However, this tabular approach doesn't scale well to large state spaces, and so typically function approximation is used to represent the Q function, such as a neural network in deep Q-learning"*
- **Example shown later**

Experience replay and Imitation learning

Experience Replay

Remove this as this comes in last lecture with DQN

- Store the agents experience in a replay memory
- use stored experience for many updates
- Finish

Imitation Learning

- Finish

page 440 Sutton and Barto [2018] Mnih et al. [2015]

Exploration vs exploitation: $\epsilon-$greedy actions

- **random exploration**
- $\epsilon-$greedy algorithm to select the next action

Based on the current Q function the action is computed with the probability $1-\epsilon$, $\epsilon \in [0,1]$, i.e.:

$$\mathbf{a}_k = \max_a Q(\mathbf{s}_k, \mathbf{a}) \tag{19}$$

Thus a random action is performed by the agent with probability ϵ

- *"Balance exploration with the random actions and exploitation with the optimal actions"* Brunton and Kutz [2022]
- Larger $\epsilon =$ more random exploration.
- Often, $\epsilon = 1$ at $t = 0$, and ϵ decays over time

finish based on Sutton and Barto [2018] page 100 finish based on Graesser and Keng [2019] page 66,69-70

Policy Gradient Optimization

READ Sutton and Barto [2018] page 321, Chapter 13
the entire policy gradient is derived on page 28 in Graesser and Keng [2019].

- Learn a **parametrized policy**, which can select actions without "consulting" a value function sutton2018reinforcement
- *most common and powerful techniques to optimize a policy that is parametrized*
- Directly optimize parameters θ using gradient descent or stochastic gradient descent.

add one or two slides about cons and pros, and a discussion of policy gradien vs. value functions

Policy Gradient Optimization

- Let $\boldsymbol{\theta} \in \mathcal{R}^{d'}$ be the policies parameter vector, then we have

add one or two slides about cons and pros, and a discussion of policy gradien vs. value functions

$$\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a | S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\} \tag{20}$$

- Learn the policy parameter based on the gradient of some scalar performance measure $J(\boldsymbol{\theta})$ wrt. the policy parameter.
- i.e. methods which **maximizes** the performance, such that their updates approximate gradient **ascent** in J:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)} \tag{21}$$

- All methods that follow this general schema are referred to as *policy gradient methods* whether or not they also learn an approximate value function
- *Methods that learn approximations to both policy and value functions are often called actor–critic methods, where 'actor' is a reference to the learned policy, and 'critic' refers to the learned value function, usually a state-value function* sutton2018reinforcement
- $\pi(a|s, \boldsymbol{\theta})$ must be **differentiable with respect to its parameters**

Policy Gradient Optimization - previous slide and this one should be made following Laura

Getting started with OpenAI GYM

Getting started with OpenAI GYM

Prerequisites

To start we need to install python, and the OpenAI GYM environment

- Python installed using e.g. Anaconda [2]
  - Anaconda is an open source package and environment management system
  - Helps with: Installation, running, and updating packages and dependencies.

`https://www.anaconda.com/products/distribution`

- OpenAi Gym can be installed using pip, as follows: [3]

```
pip install gym
```

- Or using Anaconda, as follows: [4]

```
conda install -c conda-forge gym
```

[2] `https://www.anaconda.com/`
[3] `https://gym.openai.com/docs/`
[4] `https://anaconda.org/conda-forge/gym`

Getting started with OpenAI GYM

Prerequisites

Useful packages:

```
conda install -c conda-forge opencv

conda install -c anaconda numpy

conda install -c conda-forge pillow
```

Others???

Getting started with OpenAI GYM

The Environment

- OpenAI Gym contains the `ENV` class
- A simulator for the environment - it is here we train our agent
  - A lot of inbuilt environments
  - Custom environments can be created
- Here is an example with the CartPole environment

In the following the environment will be described step-by-step.

```
1 import gym
2 env = gym.make('CartPole-v0')
3 env.reset()
4 for _ in range(1000):
5     env.render()
6     observation, reward, done, info = env.step(env.
          action_space.sample())
7     print(observation, reward, done, info)
8 env.close()
```

https://blog.paperspace.com/getting-started-with-openai-gym/
https://gym.openai.com/docs/

Getting started with OpenAI GYM

The Environment - Step-By-Step

```
gym.Env.step(self, action: gym.core.ActType) -> tuple[ObsType, float, bool, dict]
```

- Run one time step of the environment's dynamics.
- When end of episode is reached, you are responsible for calling `reset()` to reset this environment's state. Accepts an action and returns a tuple `(observation, reward, done, info)`

PARAMETERS

action **(object)** an action provided by the agent, either continuous or discrete.

RETURNS

observation **(object)** an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game. Either continuous or discrete.

reward **(float)** amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.

done **(boolean)** whether it's time to reset the environment again. Most **but not all** tasks are divided up into well-defined episodes, and done being True indicates the episode has terminated. **For example, perhaps the pole tipped too far, or you lost your last life.**

info **(dictionary)** diagnostic information, **it might contain the raw probabilities behind the environment's last state change**.

---

https://www.gymlibrary.ml/content/api/
https://gym.openai.com/docs/

Getting started with OpenAI GYM

The Environment - Step-By-Step

`gym.Env.reset(self, *, seed: Optional[int] = None, return_info: bool = False, options: Optional[dict] = None)-> Union[gym.core.ObsType, tuple[ObsType, dict]]`

- Resets the environment to an initial state and returns the initial observation.

PARAMETERS

seed **(opt. int)** The seed that is used to initialize the environment's PRNG. If the environment does not already have a PRNG and seed=None (the default option) is passed, a seed will be chosen from some source of entropy (e.g. timestamp or /dev/urandom). However, if the environment already has a PRNG and seed=None is passed, the PRNG will not be reset. If you pass an integer, the PRNG will be reset even if it already exists. Usually, you want to pass an integer right after the environment has been initialized and then never again. Please refer to the minimal example above to see this paradigm in action.

return_info **(bool)** If true, return additional information along with initial observation. This info should be analogous to the info returned in `step()`

options **(opt. dict)** Additional information to specify how the environment is reset (optional, depending on the specific environment)

RETURNS

observation (object) Observation of the initial state. This will be an element of `observation_space` (typically a numpy array) and is analogous to the observation returned by `step()`.

info (opt. dictionary) This will only be returned if `return_info=True` is passed. It contains auxiliary information complementing observation. This dictionary should be analogous to the info returned by `step()`.

most of this is directly from the webpage `https://www.gymlibrary.ml/content/api/#gym.Env.step`

`https://www.gymlibrary.ml/content/api/`

Getting started with OpenAI GYM

The Environment - Step-By-Step

`gym.Env.render(self, mode='human')`

- Renders the environment.

A set of supported modes varies per environment. (And some third-party environments may not support rendering at all.) By convention, if mode is:

human: render to the current display or terminal and return nothing. Usually for human consumption.

rgb_array Return a `numpy.ndarray` with shape `(x, y, 3)`, representing RGB values for an x-by-y pixel image, suitable for turning into a video.

ansi: Return a string (str) or `StringIO.StringIO` containing a terminal-style text representation. The text can include newlines and ANSI escape sequences (e.g. for colors).

`https://www.gymlibrary.ml/content/api/`
`https://github.com/openai/gym/blob/master/gym/core.py`

Getting started with OpenAI GYM

The Environment - Additional Environment API

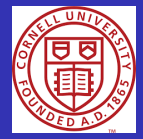
Attributes `Env.action_space: gym.spaces.space.Space[gym.core.ActType]` This attribute gives the format of valid actions. It is of datatype Space provided by Gym. For example, if the action space is of type Discrete and gives the value Discrete(2), this means there are two valid discrete actions: 0 & 1.
`Env.observation_space: gym.spaces.space.Space[gym.core.ObsType]`
`Env.reward_range = (-inf, inf)` **Methods** `gym.Env.close(self)` `gym.Env.close(self)` finish this explenation from `https://www.gymlibrary.ml/content/api/#gym.Env.step`

Getting started with OpenAI GYM

The Environment - Wrappers

finish this explenation from `https://www.gymlibrary.ml/content/api/#gym.Env.step`

Q-learning




Q-learning in Open-AI GYM

Introduction

todo

Q-learning in Open-AI GYM

Introduction

`https://www.gocoder.one/blog/rl-tutorial-with-openai-gym`
Q-learning example `https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/`

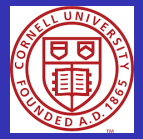
- Taxi environment
- Try to brute force
  - Agent randomly explores the environment
  - No memory of state-action-to-reward
- Q-learning - Will enable the agent to have some memory

test the code using OpenAI

- Reward Table
-

Q-learing method

- We make a Q-table [state $\times$ action]
- Initialized at 0, and updated through training
- Represent the path producing he highest rewards




Q-learning in Open-AI GYM

TODO: analyse ode line by line and explain with notes
Go thrugh code in IDE

Exercise

Exercise Open-AI GYM

Build a custom environment

Explanation of the exercise:
This is an idea, but work with it and maka a fun exercise, some inspiration can be found in the note Given structure of an Open-AI environment in the right column, expand upon the environment to enable it to train a child to pick a ball.

Listing of example code

```
from gym import Env
class Learning(Env):
    def __init__(self):
        # define your environment
        # action space, observation space
    def step(self, action):
        # take some action
    def render(self, mode="dog"):
        # render your environment (can be a
        visualisation/print)
    def reset(self):
        # reset your environment
```

References I

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Brunton, S. L. and Kutz, J. N. (2022). *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press.

Graesser, L. and Keng, W. L. (2019). *Foundations of deep reinforcement learning: theory and practice in Python*. Addison-Wesley Professional.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.