Deep Reinforcement Learning (DRL)

An introduction to using DRL in robotics

@ENSEIRB-MATMECA – Bordeaux INP

Jean-Luc.Charles@ENSAM.EU

Arts et Métiers Institute of Technology

November 2022

An introduction to using DRL in Robotics

An introduction to get familiarised with...

- Unsupervised / Supervised / Reinforcement learning.
- DRL Training & operating of Artificial Neural Networks
- PyTorch module for DRL training.
- Applications of DRL to robotics.

An introduction to using DRL in Robotics

An introduction to get familiarised with...

- Unsupervised / Supervised / Reinforcement learning.
- DRL Training & operating of Artificial Neural Networks
- PyTorch module for DRL training.
- Applications of DRL to robotics.

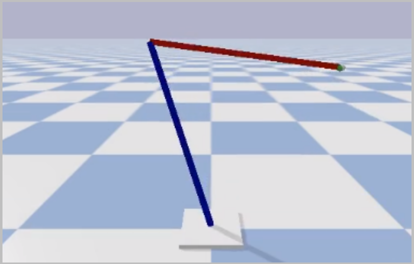
Ressources

- 2 hours of lecture and 1 $\times$ practical work Python session (4h) on **your laptop**
- Dedicated github repository with all the course material (PDF, notebooks...)

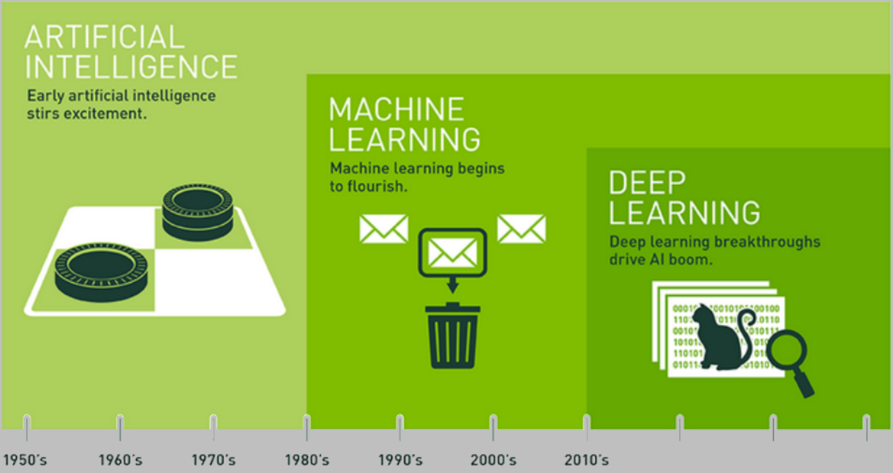
Practical Work: 1 $\times$ 4h

DRL pactical work:

- Build the PyBullet simulation of a 2 DOF robot arm based on its URDF description.
- train a PPO (Proximal Policy Optimisation) network to drive the displacement of the end effector of a 2 DOF robot arm.



The historical way...



(from : developer.nvidia.com/deep-learning)

Artificial Intelligence ?

Artificial Intelligence [1]: remains an ambiguous term with multiple definitions varying with time:

- *"...the science of making computers do things that require intelligence when done by humans."* Alan Turing, 1940
- *"the field of study that gives computers the ability to learn without being explicitly programmed."* Arthur Samuel, 1960
- *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."* Tom Mitchell, 1997
- Notion of *intelligent agent* or *rational agent*
  "...agent that acts in such a way as to reach the best solution or, in an uncertain environment, the best predictable solution." Stuart Russel, Peter Norvig, "Intelligence Artificielle" 2015

[1] first used in 1956 by John McCarthy, researcher at Stanford during the Dartmouth conference

Artificial Intelligences ?

Qualifiers often encountered:

Strong AI

Weak AI

General AI

Narrow AI

Artificial Intelligences ?

Qualifiers often encountered:

Strong AI

- Build systems that think exactly the same way that people do.
- Try also to explain how humans think... Whe are not yet here.

Weak AI

- Build systems that can behave like humans.
- The results will tell us nothing about how humans think.
- We already are there... We use it every day!
  (anti-spam, facial/voice recognition, language translation...)

General AI

Narrow AI

Artificial Intelligences ?

Qualifiers often encountered:

Strong AI

- Build systems that think exactly the same way that people do.
- Try also to explain how humans think... Whe are not yet here.

Weak AI

- Build systems that can behave like humans.
- The results will tell us nothing about how humans think.
- We already are there... We use it every day!
  (anti-spam, facial/voice recognition, language translation...)

General AI

- AI systems designed for the ability to reason in general.

Narrow AI

- AI systems designed for specific tasks.

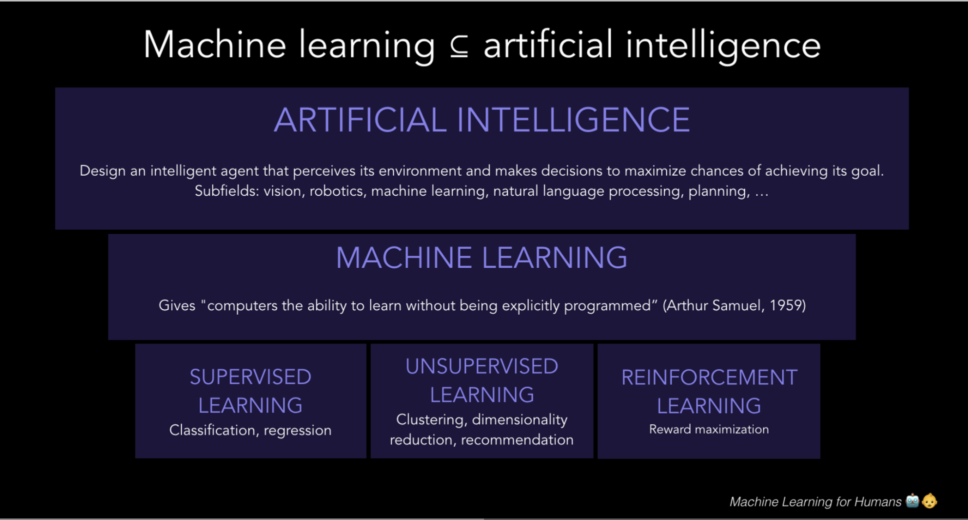
Artificial Intelligence

already a reality:

- Runs in much of our present technology (smartphone apps...)
- Powered by rapid advances in data storage, computer processing power.
- Powered by **free dataset acces via Internet** and **code publishing as open source** environments.
- Rate of acceleration is already astounding.
- Will likeky shape our future more powerfully than any other innovation this century.

Machine Learning and AI

Page from medium.com/machine-learning-for-humans/...



Branches of Machine Learning

Supervised learning applications

labeled dataset is used to train algorithms to classify data:

- **Classification**
  - Images classification
  - Objects detection
  - Speech recognition...
- **Regression**
  - Predict a value...
- **Anomaly detection**
  - Spam detection
  - Manufacturing: finding known (learned) defects
  - Weather prediction
  - Diseases classification...

Branches of Machine Learning

Unsupervised learning application

Analyze and cluster **unlabeled datasets**:

- **Clustering** & **Grouping**
  - Data mining, web data grouping, news grouping...
  - Market segmentation
  - DNA grouping
  - Astronomical data analysis...
- **Anomaly Detection**
  - Fraud detecion
  - Manufacturing: finding defects even new ones
  - Monitoring abnormal activity: failure, hacker, fraud...
  - Fake account on Internet...
- **Dimensionality reduction**
  - Compress data using fewer numbers...

Branches of Machine Learning

Reinforcement learning

An agent learns how to drive an environment by maximising a **reward**:

- **Control/command**
  - Controlling robots, drones...
  - Factory optimization
  - Financial (stock) trading...
- **Decision making**
  - games (video games)
  - financial analysis...

Various approaches for ML algorithms

Supervised learning:

- Neural Networks
- Bayesian inference
- Random forest
- Decision Tree
- Support Vector Machine
- K-Nearest Neighbor
- Linear regression
- Logistic regression...

Unsupervised learning:

- Neural Networks
- Principal Composant Analysis
- Singular Value Decomposition
- K-mean & Probabilistic clustering

Reinforcement learning:

- Monte Carlo
- SARSA
- Neural Networks (Q-learning, Actor-Critic...)

Various approaches for ML algorithms

Supervised learning:

- Neural Networks
- Bayesian inference
- Random forest
- Decision Tree
- Support Vector Machine
- K-Nearest Neighbor
- Linear regression
- Logistic regression...

Unsupervised learning:

- Neural Networks
- Principal Composant Analysis
- Singular Value Decomposition
- K-mean & Probabilistic clustering

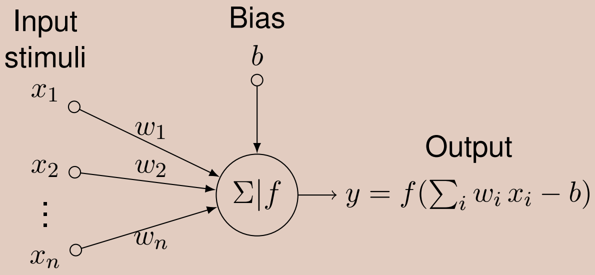
Reinforcement learning:

- Monte Carlo
- SARSA
- Neural Networks (Q-learning, Actor-Critic...)

The following deals only with **Artificial Neural Networks**.

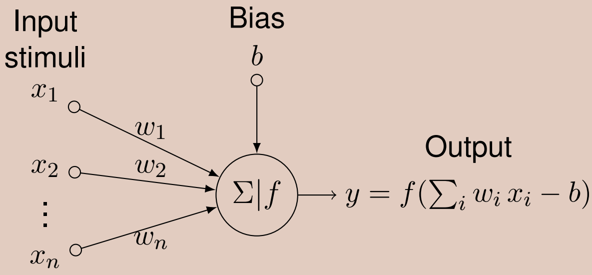
Artificial Neural Networks

The Artificial neuron model



Artificial Neural Networks

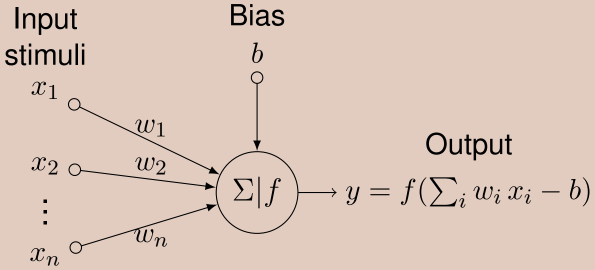
The Artificial neuron model



An **artificial neuron**:

Artificial Neural Networks

The Artificial neuron model

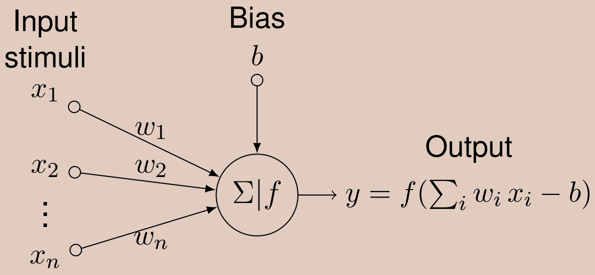


An **artificial neuron**:

- receives the input stimuli $(x_i)_{i=1..n}$ with **weights** (w_i)

Artificial Neural Networks

The Artificial neuron model

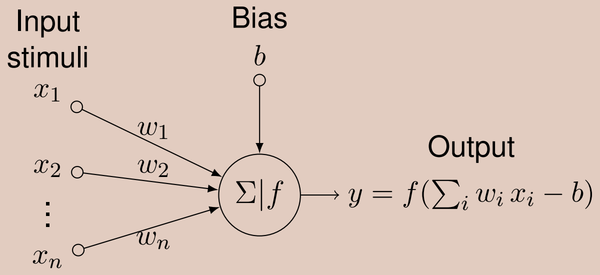


An **artificial neuron**:

- receives the input stimuli $(x_i)_{i=1..n}$ with **weights** (w_i)
- computes the **weighted sum** of the input $\sum_i w_i\, x_i - b$

Artificial Neural Networks

The Artificial neuron model

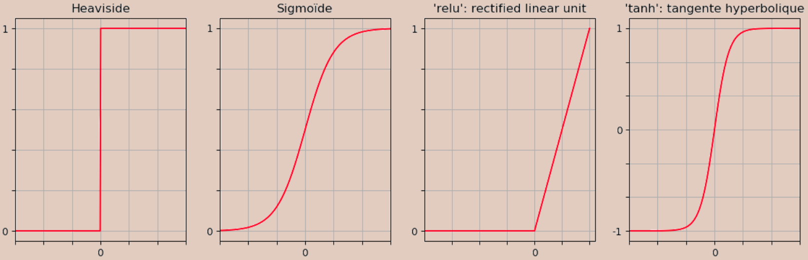


An **artificial neuron**:

- receives the input stimuli $(x_i)_{i=1..n}$ with **weights** (w_i)
- computes the **weighted sum** of the input $\sum_i w_i\, x_i - b$
- outputs its activation $f(\sum_i w_i\, x_i - b)$, computed with a non-linear **activation function** f .

Artificial Neural Networks

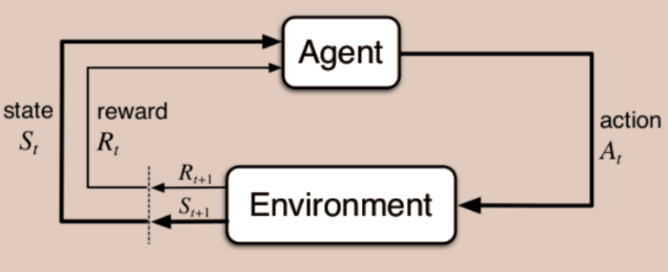
Common activation functions



- Introduces a non-linear behavior.
- Sets the range of the neuron output: $[-1, 1]$, $[0, 1]$, $[0, \infty[$...
- The bias b sets the activation threshold of the neuron.

Reinforcement Learning

RL main ingredients



(Source: Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto)

The **agent** (the neural network) learns how to take the right **action** on the **environment** (the system to be controlled) in order to maximize its long-term **reward**.

RL ingredients: **Agent**

The **Agent** is the **algorithm**:

RL ingredients: **Agent**

The **Agent** is the **algorithm**:

- Monitors the **Environment**
- Decides wich **action** to be taken
- Action can be
  discrete: on/off, left/right...
  continuous: force/velocity applied....
- Goal: maximize the total reward it receives in the long run.

Discrete versus continuous action involves different algorithms for the learning stage

RL ingredients: **Environment**

The **Environment** is what the Agent wants to monitor:

RL ingredients: **reward** funtion

RL ingredients: **Environment**

The **Environment** is what the Agent wants to monitor:

- Receives **actions** from the Agent.
- Takes a new **state** under the Agent's action.
- Gives back its new **state** and computed **reward** to the Agent.
- Modelized as a **Partially Observable Markov Process**.

RL ingredients: **reward** funtion

RL ingredients: **Environment**

The **Environment** is what the Agent wants to monitor:

- Receives **actions** from the Agent.
- Takes a new **state** under the Agent's action.
- Gives back its new **state** and computed **reward** to the Agent.
- Modelized as a **Partially Observable Markov Process**.

RL ingredients: **reward** funtion

- Maps each (state, action) pair of the environment to a number indicating the intrinsic desirability of that state.
- The environment sends a **scalar value** as a **reward** in response to the agents's action.

Model-free versus Model-based DRL algorithms

Model-free

- No explicit representation of the environment
- Learning rely only on experiences using **trial and error**.
- Examples : Monte Carlo, SARSA, Q-learning, and Actor-Critic algorithms.
- Applies to car driving, robot control...

Model-based algorithms

- The agent has access to (or learns) a **model** of the environment.
- Applies to environment like Chess, Go...

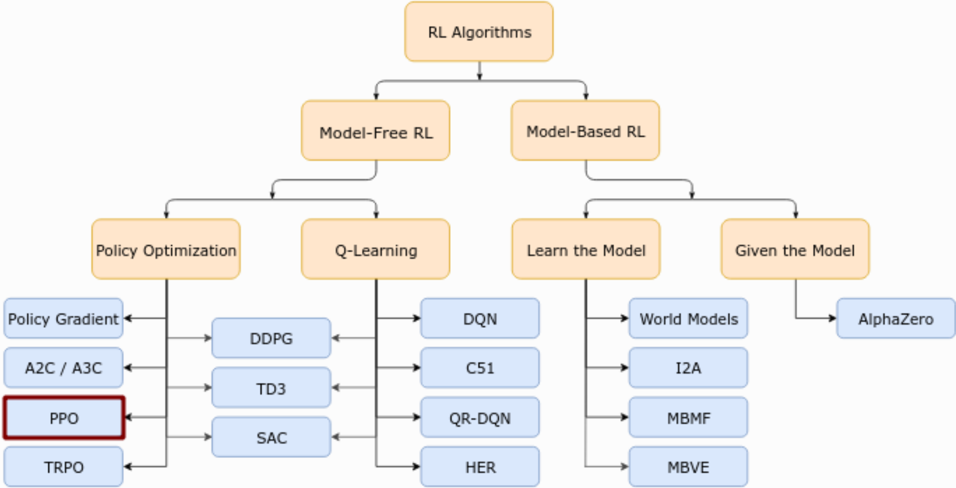
RL reference sites

Stable-baseline3

A set of reliable implementations of reinforcement learning algorithms in **PyTorch**.

- github.com/DLR-RM/stable-baselines3
- docs: stable-baselines3.readthedocs.io spinningup.openai.com
- install: `pip install stable-baselines3`

A Taxonomy of RL Algorithms (Source: spinningup.openai.com)



The DRL practical work

- Goal: train a **PPO** (Proximal Policy Optimisation) neural network to drive the position of the end effector of a 2 DOF robot arm.

The DRL practical work

- Goal: train a **PPO** (Proximal Policy Optimisation) neural network to drive the position of the end effector of a 2 DOF robot arm.
- The training of the network involves the simulation of the robot with the PyBullet simulator.

The DRL practical work

- Goal: train a **PPO** (Proximal Policy Optimisation) neural network to drive the position of the end effector of a 2 DOF robot arm.
- The training of the network involves the simulation of the robot with the PyBullet simulator.
- The robot is described in the simulator thanks to its URDF file.

The DRL practical work

- Goal: train a **PPO** (Proximal Policy Optimisation) neural network to drive the position of the end effector of a 2 DOF robot arm.
- The training of the network involves the simulation of the robot with the PyBullet simulator.
- The robot is described in the simulator thanks to its URDF file.
- The PyTorch Python module provides classes to instanciate and train the PPO neural network.

The DRL practical work

- Goal: train a **PPO** (Proximal Policy Optimisation) neural network to drive the position of the end effector of a 2 DOF robot arm.
- The training of the network involves the simulation of the robot with the PyBullet simulator.
- The robot is described in the simulator thanks to its URDF file.
- The PyTorch Python module provides classes to instanciate and train the PPO neural network.
- To ensure the robustness and maintainability of Python developments, the work is done in a **Python Virtual Environment** (PVE).

The DRL practical work

- The pedagogical material of the practical work is on the Git repository github.com/cjlux/DRL_at_ENSEIRB-MATMECA

The DRL practical work

- The pedagogical material of the practical work is on the Git repository github.com/cjlux/DRL_at_ENSEIRB-MATMECA
- Session sequencing :

The DRL practical work

- The pedagogical material of the practical work is on the Git repository github.com/cjlux/DRL_at_ENSEIRB-MATMECA
- Session sequencing :
  1. From the Git repository download the file `Create_PVE.pdf`

The DRL practical work

- The pedagogical material of the practical work is on the Git repository github.com/cjlux/DRL_at_ENSEIRB-MATMECA
- Session sequencing :
  1. From the Git repository download the file `Create_PVE.pdf`
  2. Following the PDF document, create & activate the PVE `pydrl`, download the repository and install the required Python modules.

The DRL practical work

- The pedagogical material of the practical work is on the Git repository github.com/cjlux/DRL_at_ENSEIRB-MATMECA
- Session sequencing :
  1. From the Git repository download the file `Create_PVE.pdf`
  2. Following the PDF document, create & activate the PVE `pydrl`, download the repository and install the required Python modules.
  3. Within the PVE `pydrl` run `jupyter notebook` or `jupyter lab` or whatever you want to open the notebook `1-Getting_started_with_pybullet.ipynb`.

The DRL practical work

- The pedagogical material of the practical work is on the Git repository github.com/cjlux/DRL_at_ENSEIRB-MATMECA
- Session sequencing :
  1. From the Git repository download the file `Create_PVE.pdf`
  2. Following the PDF document, create & activate the PVE `pydrl`, download the repository and install the required Python modules.
  3. Within the PVE `pydrl` run `jupyter notebook` or `jupyter lab` or whatever you want to open the notebook `1-Getting_started_with_pybullet.ipynb`.
  4. Play with the notebok...

The DRL practical work

Stable-baseline3 & Gym for DRL

- stable-baselines3 works with the Gym workbench.
- The main class defined in `Gym` is Env.

`Env` is an **abstract** class $\leadsto$ we must derive it and define:

- methods:
  - `step`
  - `reset`
  - `render` ($\leadsto$ we will use PyBullet rendering instead)
  - `close`
- attributes:
  - `action_space`
  - `observation_space`

The DRL practical work

Gym.Env class

```
from abc import abstractmethod
import gym
from gym import error
from gym.utils import closer

class Env(object):
    """The main OpenAI Gym class. It encapsulates an environment with
    arbitrary behind-the-scenes dynamics. An environment can be
    partially or fully observed.

    The main API methods that users of this class need to know are:

     step, reset, render, close & seed

    And set the following attributes:

     action_space:      The Space object corresponding to valid actions
     observation_space: The Space object corresponding to valid observations
     reward_range:      A tuple corresponding to the min and max possible rewards

    Note: a default reward range set to [-inf,+inf] already exists.
          Set it if you want a narrower range.
    """

    # Set this in SOME subclasses
    metadata     = {"render.modes": []}
    reward_range = (-float("inf"), float("inf"))

    # Set these in ALL subclasses
    action_space      = None
    observation_space = None
```

The DRL practical work

Gym.Env class

```
    @abstractmethod
    def step(self, action):
        """
        Run one timestep 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).

        Args:
            action (object): an action provided by the agent

        Returns:
            observation (object): agent's observation of the current environment
            reward (float) :      amount of reward returned after previous action
            done (bool):          whether the episode has ended, in which case further
                                  step() calls will return undefined results
            info (dict):          contains auxiliary diagnostic information
                                  (helpful for debugging, and sometimes learning)
        """
        raise NotImplementedError
```

The DRL practical work

Gym.Env class

```python
@abstractmethod
def reset(self):
    """
    Resets the environment to an initial state and returns an initial
    observation.

    Note that this function should not reset the environment's random
    number generator(s); random variables in the environment's state should
    be sampled independently between multiple calls to `reset()`.
    In other words, each call of `reset()` should yield an environment suitable for
    a new episode, independent of previous episodes.

    Returns:
        observation (object): the initial observation.
    """
    raise NotImplementedError
```

The DRL practical work

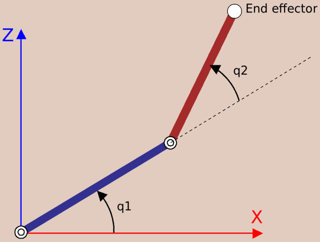
Gym.Env class

```
    @abstractmethod
    def render(self, mode="human"):
        """Renders the environment.

        The set of supported modes varies per environment. (And some
        environments do 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 an 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).
        ...
        """
        raise NotImplementedError

    def close(self):
        """Override close in your subclass to perform any necessary cleanup.

        Environments will automatically close() themselves when
        garbage collected or when the program exits.
        """
        pass
```

The DRL practical work

PPO training and Gym.Env

The robot simulated in PyBullet is connected to the `RoboticArm_2DOF_PyBullet` class



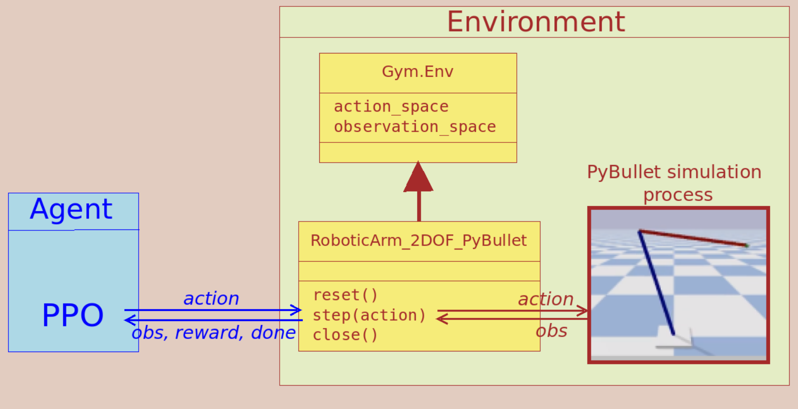
L1=L2=1m
M1=M2=1kg

Defined in the `RoboticArm_2DOF_PyBullet` class:

- state of the environment:
  $(q_1, \dot{q}_1, q_2, \dot{q}_2, x_T, z_T, x_{EE}, z_{EE})$
  T: target, EE: End Effector
- action given by the agent:
  $(\text{Torque}_{motor1}, \text{Torque}_{motor2})$

The DRL practical work

PPO training and Gym.Env



The DRL practical work

PPO training and the Gym.Env

Training is done over `total_timestep` time steps:

- Run training episodes...
  - the Agent chooses and sends `action` to the Environment
  - the Environment sends back its new state `(state, reward, done)`
  - End of episode: done=True or time step reaches `max_episode_steps`
- Every `nb_steps` steps PPO is updated `nb_epochs` times using `batch_size` experiences (state, action, newstate, reward)

The DRL practical work

PPO training and the Gym.Env

Training is done over `total_timestep` time steps:

- Run training episodes...
    - the Agent chooses and sends `action` to the Environment
    - the Environment sends back its new state `(state, reward, done)`
    - End of episode: done=True or time step reaches `max_episode_steps`
- Every `nb_steps` steps PPO is updated `nb_epochs` times using `batch_size` experiences (state, action, newstate, reward)

Now, within the PVE `pydrl` open `2-DRL_training.ipynb` and play with the notebook...