Project on Reinforcement Learning

Vincent François-Lavet

Outline

Overview of the course

Poll to get to know you (see Menti.com)

Project

Schedule

Very short intro to RL

Some initial pointers for the project (more in the tutorials)

Overview of the course

Reinforcement learning

Reinforcement learning is about learning sequential decision-making tasks from data (i.e. trial and error).

Overall view

- Project-based course in teams of 2 or 3
- Mostly based on "self-study" (including one test) and "self-do" (project).

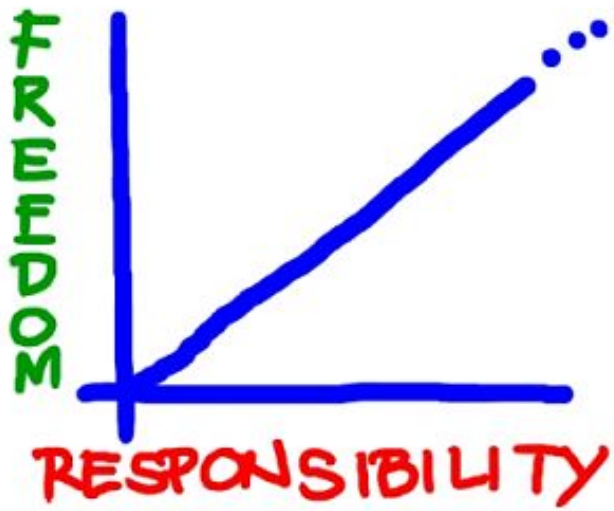


Figure: With freedom comes responsibility

Project

Two tracks

- Suggested project: learn the operation of a datacenter to minimize costs (given two years of time series and equations that govern the dynamics)
- Fully free topic: you have already a problem in mind. Needs approval (send a one page description and to be discussed before Friday 13th)

All information is on Canvas

Poll to get to know you (see Menti.com)

Project

Description of the project

Data centers are significant consumers of electricity, playing a critical role in supporting digital infrastructure worldwide.

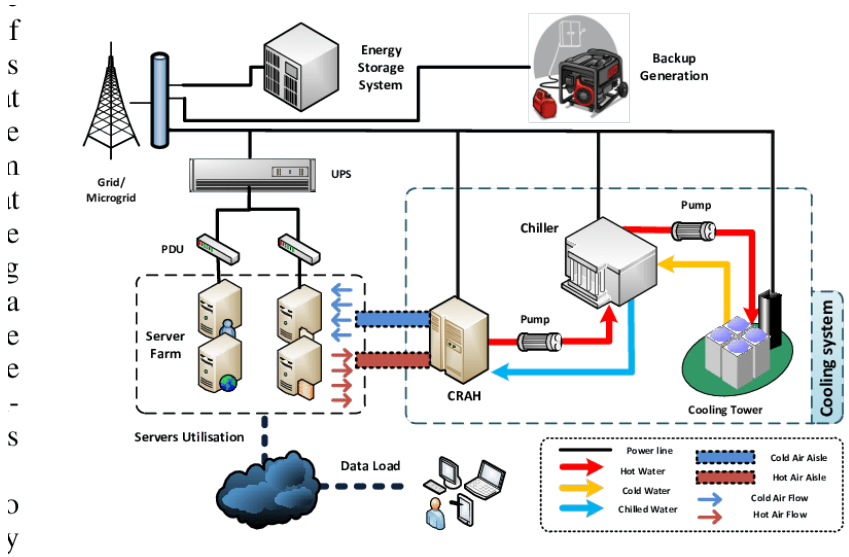


Figure: Illustration of a dam

One key challenge for data centers is the daily management of their power consumption.
In this project, you are in the role of the data science team and need to define a control strategy for a new data center that has been constructed to optimize its energy usage.

How can one improve the datacenter operation?

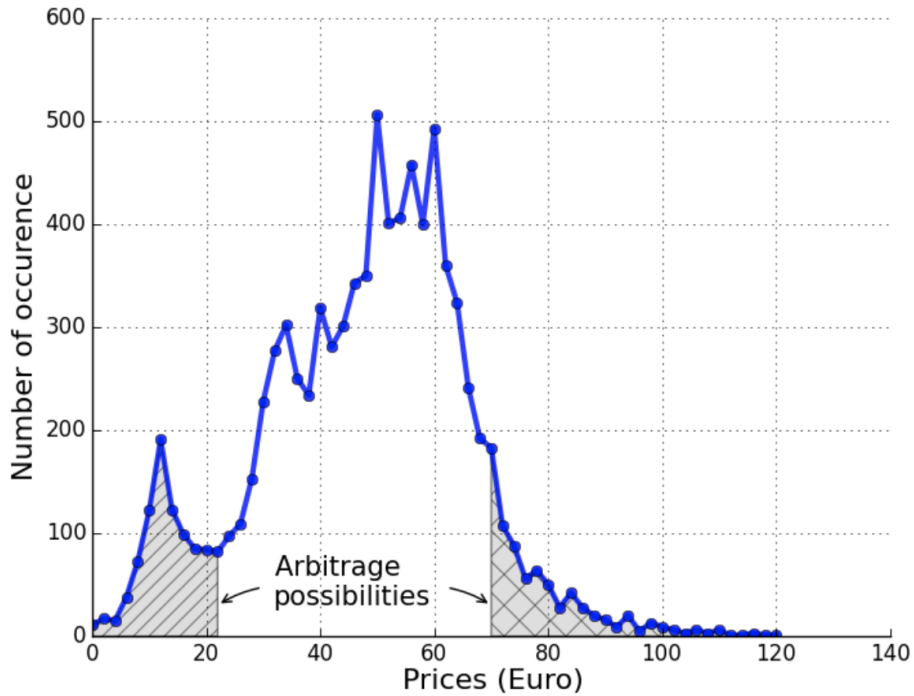


Figure: Histogram of the prices of electricity on the local market for one of the year in the training data.

What can the operation look like?

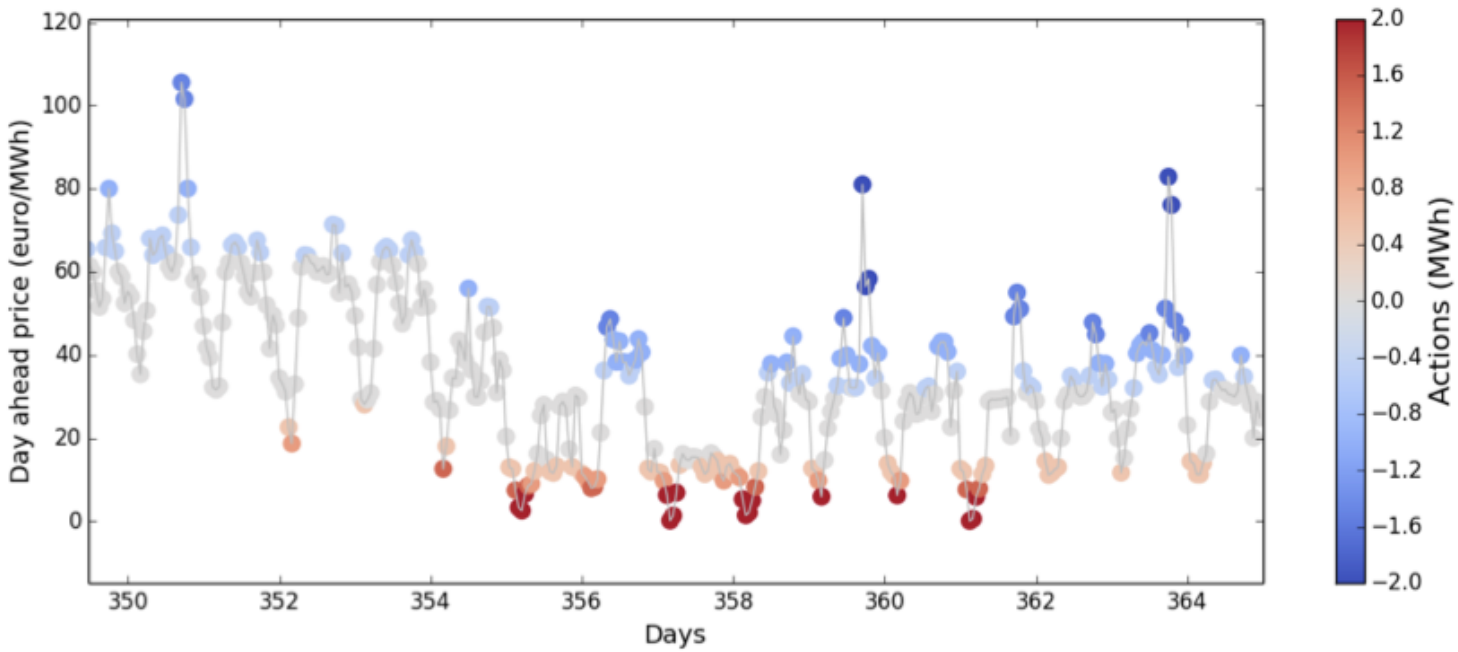


Figure: Illustration of the electricity prices with time.

How should we do this?

Main challenge: We only have access to past prices, current price, time, date but we do not have access to future electricity prices. The problem of optimally buying electricity can be formulated as a sequential decision making problem under uncertainty where, at every time-step, the uncertainty comes from the lack of knowledge about future electricity prices.

$\rightarrow$ What would be basic strategies?

How can one improve the datacenter operation?

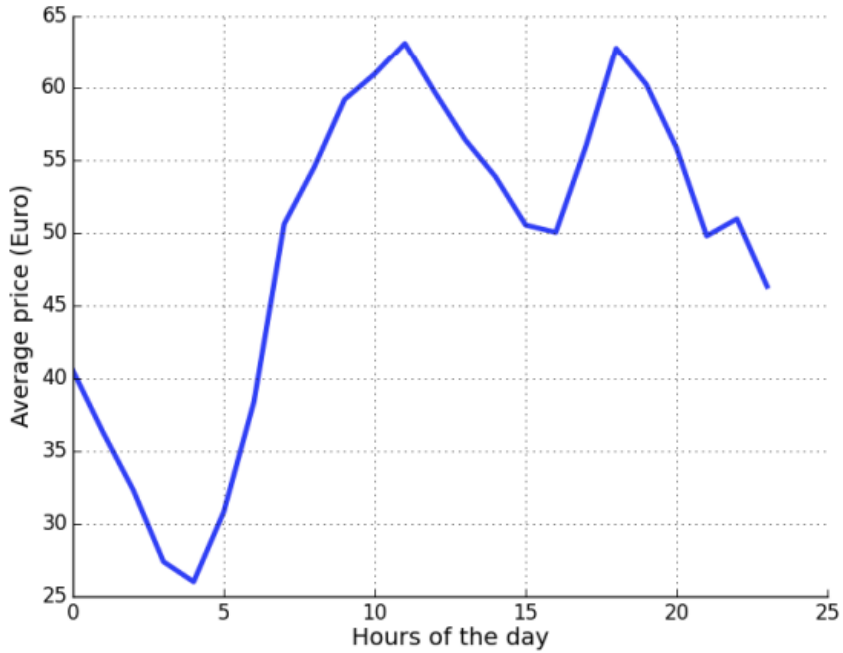


Figure: Histogram of the prices of electricity (euro per MWh) on the local market as a function of the hour of the day.

How can one improve the datacenter operation?

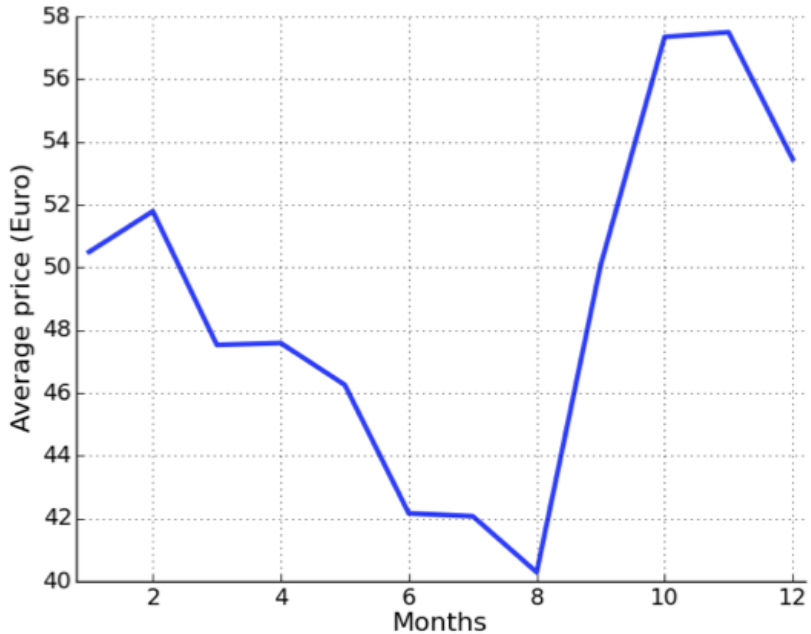


Figure: Histogram of the prices of electricity (euro per MWh) on the local market as a function of the month.

Questions on the problem?

Schedule

Overall view

Schedule on Canvas and Rooster (except changes, here it is):

- "Seminar groups" 3 days a week through the 4 weeks
- Lectures first week: Intro (Mon) + tutorials (Wednesday-Thursday)
- Lectures Second week: Lecture+Q&A (Monday), and Wednesday (test).
- There are backup slots on Rooster but check the schedule on Canvas (not all slots have a lecture).
- Friday 17 Jan: intermediate presentation
- Friday 31 Jan: final presentation

Attendance

Attendance is advised.

- Participation to the group work (seminar group) sessions is part of the grade via the attendance sheet (10% of the grade). Min 50% participation is required otherwise the whole project is failed and the maximum of points is obtained from 80% participation.

First presentation

You need to make a presentation where

- you have implemented the environment as a gym environment,
- you have implemented at least one baseline algorithm (not necessarily RL),
- you can provide visualization of the operation of the dam as well as estimated performance,
- you explain the key next steps to improve the model.

You need to make a presentation (5minutes presentation + 5minutes feedback/questions).

Final project and final presentation

The final report and presentation need to show

- an RL solution that you have developed for the problem at hand,
- how you have validated the RL algorithm and what the estimated performance are,
- the code with a pre-trained model that can be run on new time series that will be provided. The additional data will be provided in the same format but possibly less days (e.g. with 1 year of data). The code should also have a README file that explains how to run the code for training the model and how one can use the pre-trained model.

You need to hand in a report (max 8 pages, appendix allowed) along with source code. You also need to make a presentation (5minutes presentation + 5minutes questions).

Grade system for the course

The final grade takes the following elements into account:

- Participation to the group work (seminar group) sessions via the attendance sheet (10% of the grade). The maximum of points is obtained from 80% participation. Min 50% participation is required (if specific problems to reach these 50%, please reach out and we can discuss your particular situation)
- Test on Wed 18th (10%)
- Intermediate presentation (10%)
- Report and final presentation (70%)

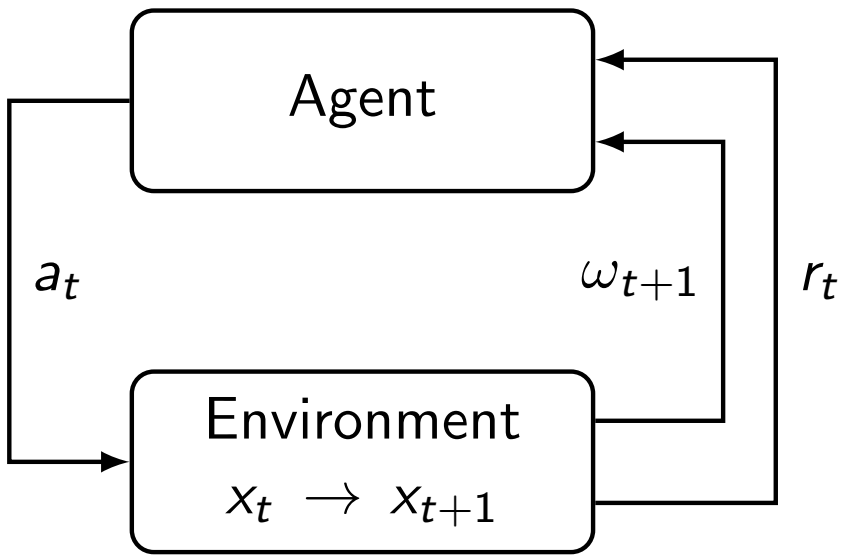
Very short intro to RL

Resources on RL

- Richard Sutton and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.
- Vincent François-Lavet et al. *"An introduction to deep reinforcement learning"*. Foundations and Trends in ML.
- Videos
- Other (RL Course by David Silver on Youtube : https://www.youtube.com/watch?v=2pWv7GOvuf0)

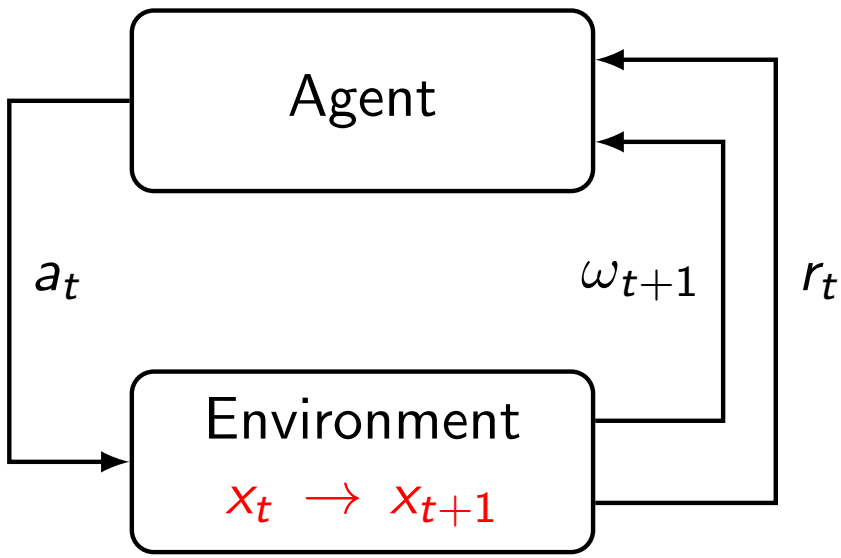
Objective

From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Objective

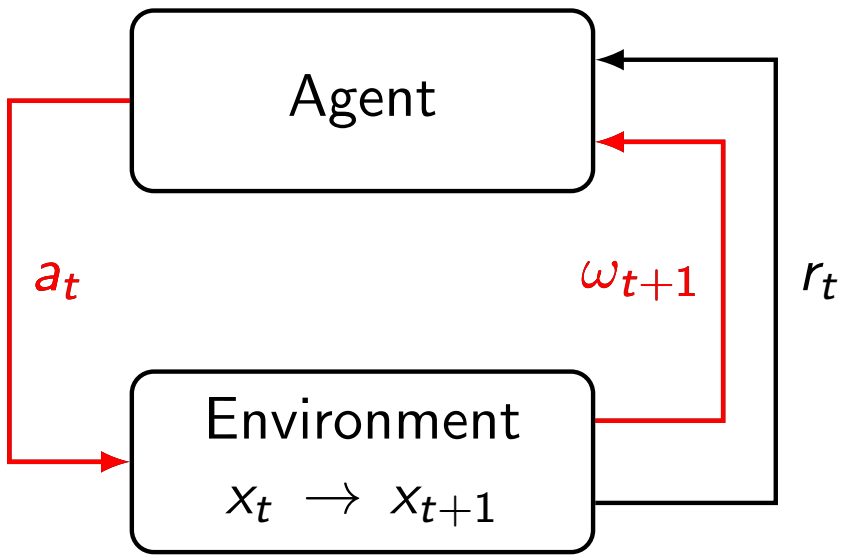
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



transitions
are usually
stochastic

Objective

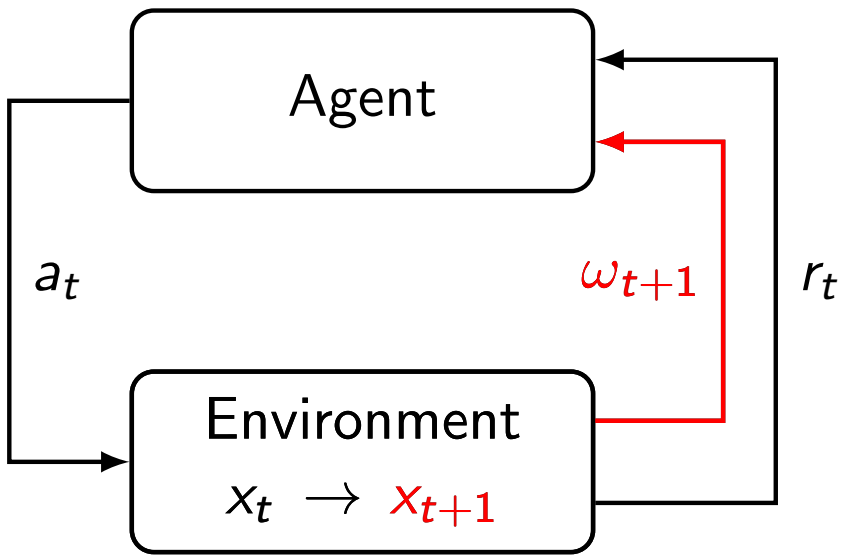
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Observations and
actions may be
high dimensional

Objective

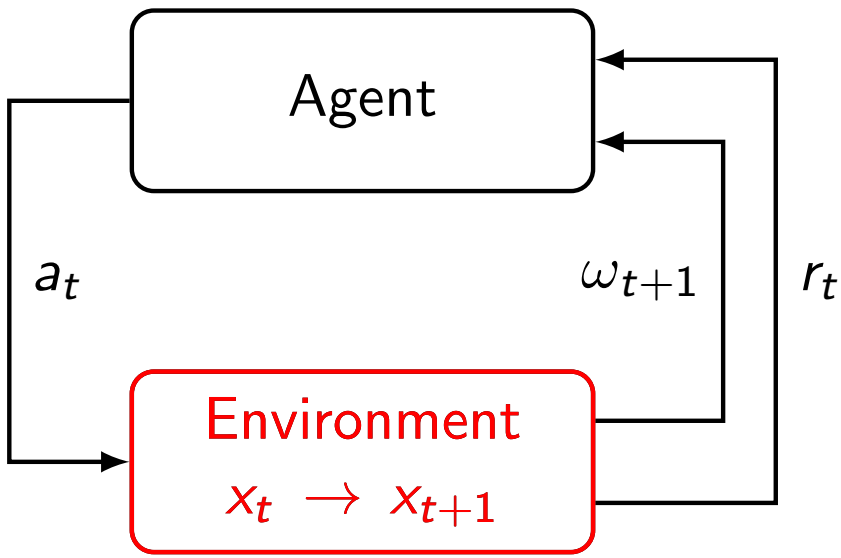
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Observations may not
provide full knowledge
of the underlying
state: $\omega_t \neq x_t$

Objective

From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.

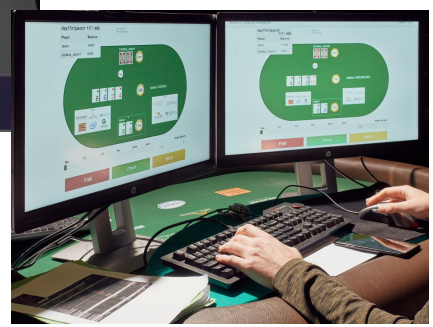


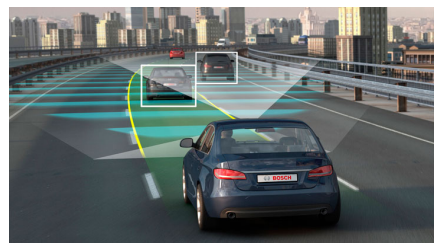
Experience may be constrained
(e.g., not access to an accurate simulator or limited data)

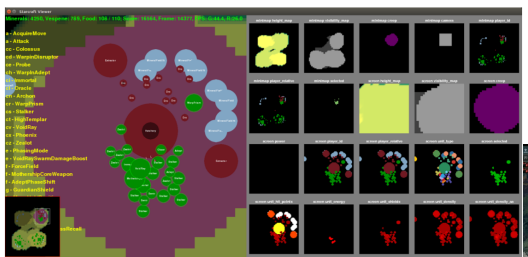
Motivation

Figure: Example of an ATARI game: Seaquest

Motivation: Overview







Motivation

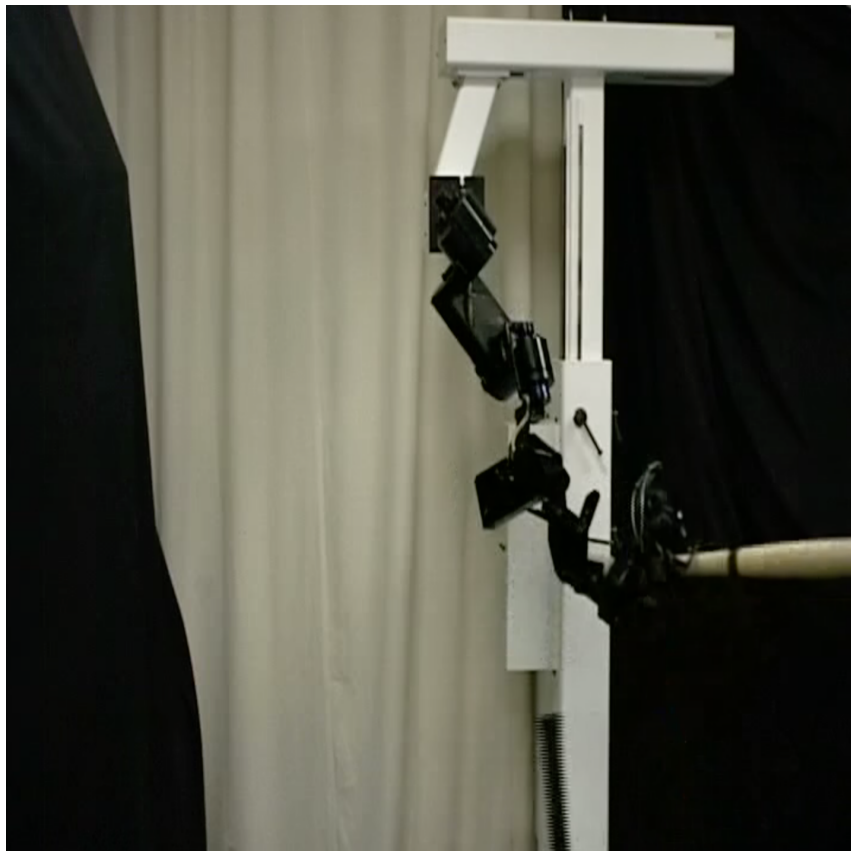


Figure: Application in robotics (credits: Jan Peters'team, Darmstadt)

Challenges of applying RL to real-world problems

In real-world scenarios, it is often not possible to let an agent interact freely and sufficiently in the actual environment:

1. The agent may not be able to interact with the true environment but only with an inaccurate simulation of it. This is known as the **reality gap**.
2. The agent might have access to only **limited data**. This can be due to safety constraints (robotics, medical trials, etc.), compute constraints or due to limited exogenous data (e.g., weather conditions, trading markets).

Challenges of applying RL to real-world problems

In order to deal with the reality gap and limited data, different elements are important:

- One can aim to develop **a simulator that is as accurate as possible**.
- One can design the learning algorithm so as to **improve generalization** (and/or use specific transfer learning methods).

Generalization

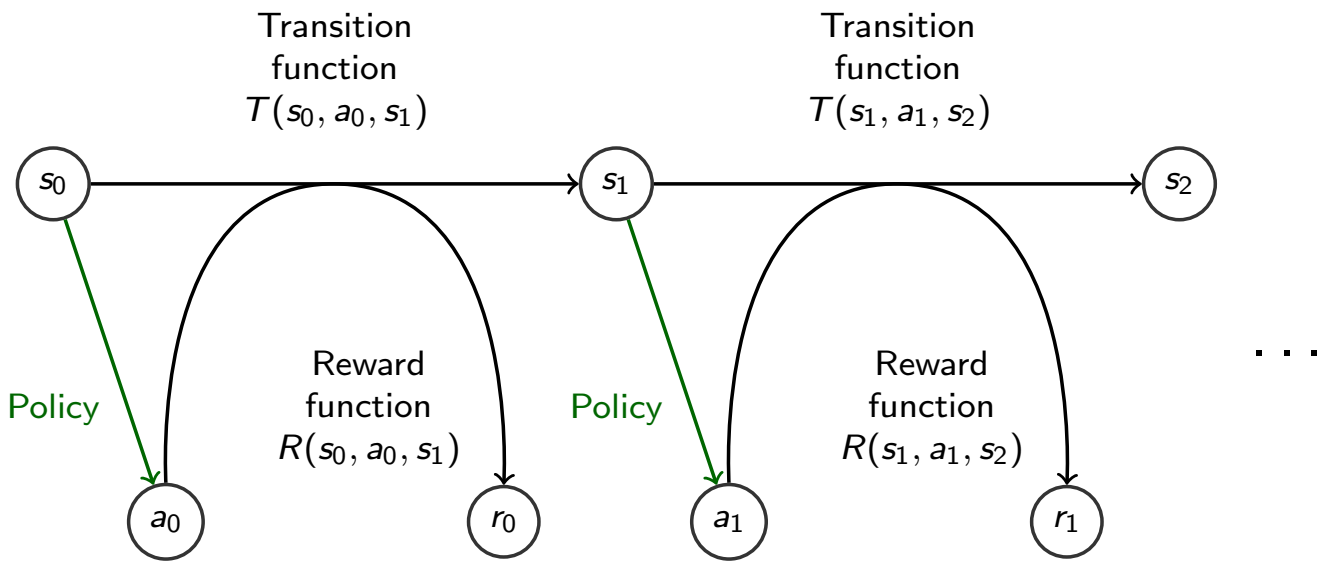
In an RL algorithm, *generalization* refers to either

- the capacity to achieve good performance in an environment where limited data has been gathered, or
- the capacity to obtain good performance in a related environment. This latter case can be tackled with specific *transfer learning* techniques.

Definition of an MDP

An MDP can be defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ where:

- $\mathcal{S}$ is a finite set of states $\{1, \ldots, N_{\mathcal{S}}\}$,
- $\mathcal{A}$ is a finite set of actions $\{1, \ldots, N_{\mathcal{A}}\}$,
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function (set of conditional transition probabilities between states),
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the reward function, where $\mathcal{R}$ is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$ (e.g., $[0, R_{max}]$),
- $\gamma \in [0, 1)$ is the discount factor.



Performance evaluation

In an MDP $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, the expected return $V^{\pi}(s) : \mathcal{S} \rightarrow \mathbb{R}$ ($\pi \in \Pi$, e.g., $\mathcal{S} \rightarrow \mathcal{A}$) is defined such that

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \mid s_t = s, \pi\right], \tag{1}$$

with $\gamma \in [0, 1)$.

From the definition of the expected return, the optimal expected return can be defined as

$$V^{*}(s) = \max_{\pi \in \Pi} V^{\pi}(s). \tag{2}$$

and the optimal policy can be defined as:

$$\pi^{*}(s) = \operatorname*{argmax}_{\pi \in \Pi} V^{\pi}(s). \tag{3}$$

Some initial pointers for the project (more in the tutorials)

Example 1: Mountain car

A car tries to reach the top of the hill but the engine is not strong enough.

- State: position and velocity
- Action: accelerate forward, accelerate backward, coast.
- Goal: get the car to the top of the hill (e.g., reward $= 1$ at the top).

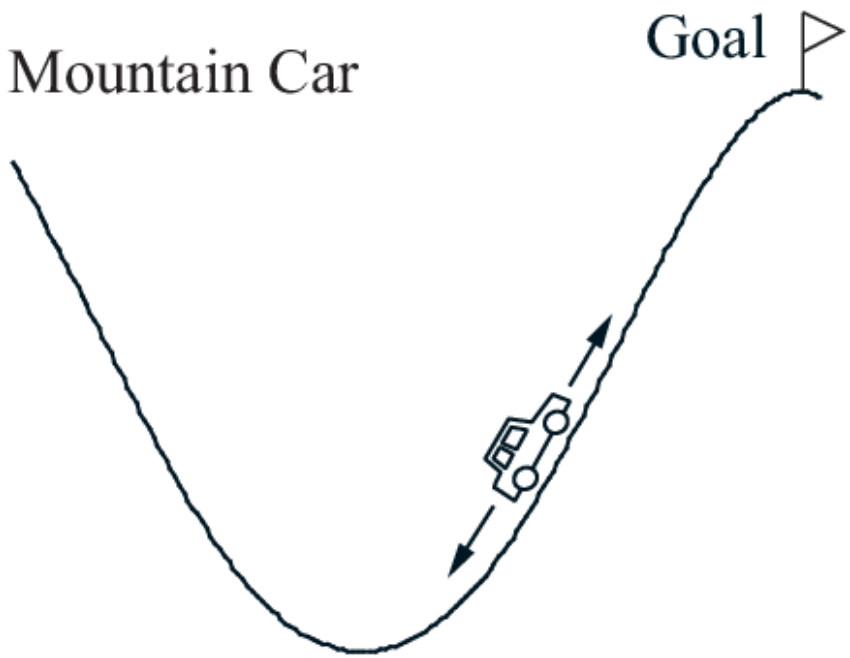


Figure: Mountain car

Example 1: Mountain car

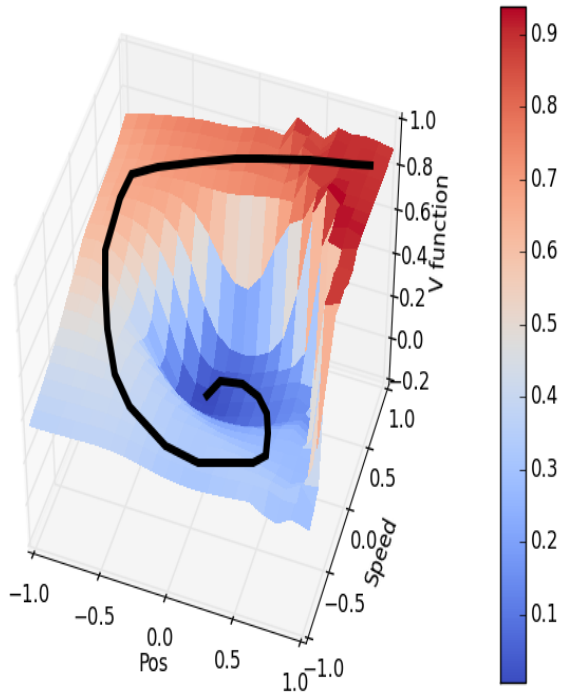


Figure: Application to the mountain car domain: $V = \max_a Q(x, a)$, where the state space has been discretized finely.

Example 1: Mountain car

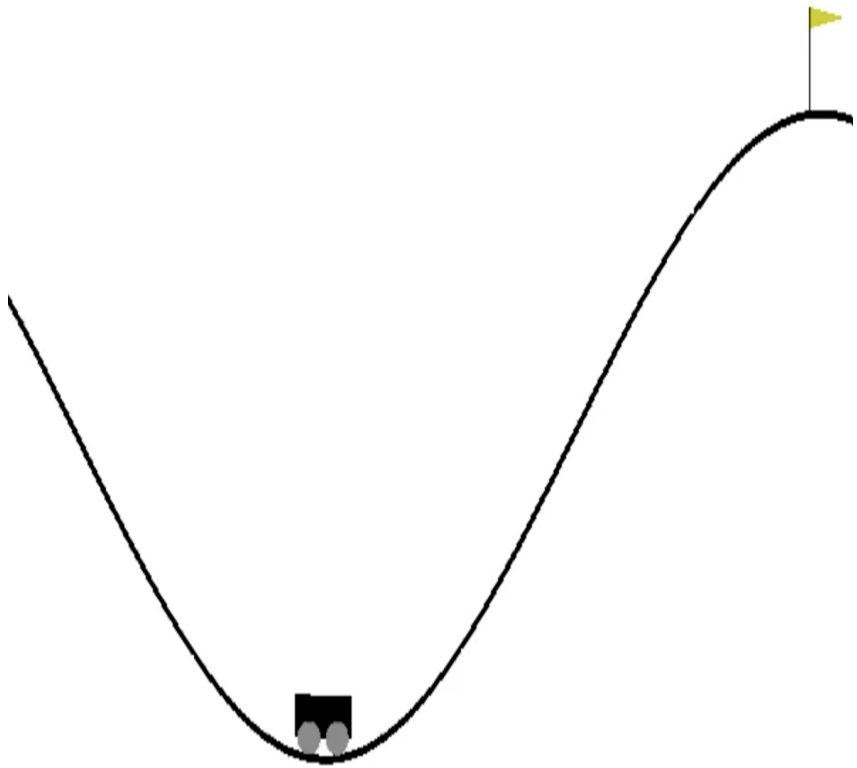


Figure: Mountain car optimal policy

Why function approximators?

A tabular approach with discretization fails due to the curse of dimensionality when the number of (initially continuous) dimensions for the state $\gtrapprox 10$ or for a large number of states.

When do we need function approximators?

- large and/or continuous state space $\rightarrow$ DQN
- (large and/or continuous action space) $\rightarrow$ next week we'll see the continuous action space

Example 2: toy example in finance

This environment simulates the possibility of buying or selling a good. The agent can either have one unit or zero unit of that good. At each transaction with the market, the agent obtains a reward equivalent to the price of the good when selling it and the opposite when buying. In addition, a penalty of 0.5 (negative reward) is added for each transaction. The price pattern is made by repeating the following signal plus a random constant between 0 and 3:

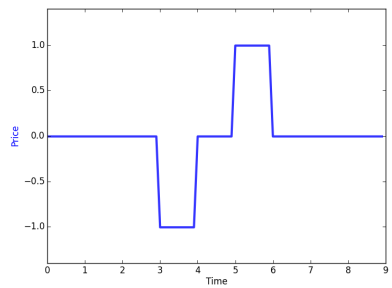


Figure: Price signal

- State: current price and price at the last five time steps+ whether the agent has one item of the good. (This problem becomes very complex without function approximator)
- Action: buy, sell, do nothing.
- Goal: get as much $$$ as possible.

Example using the DeeR library

```
git clone -b master https://github.com/VinF/deer.git
```

Assuming you already have a python environment with `pip`, you can automatically install all the dependencies (except specific dependencies that you may need for some examples) with:

```
pip install -r requirements.txt
```

And you can install the framework as a package using the mode `develop` so that you can make modifications and test without having to re-install the package.

```
python setup.py develop
```

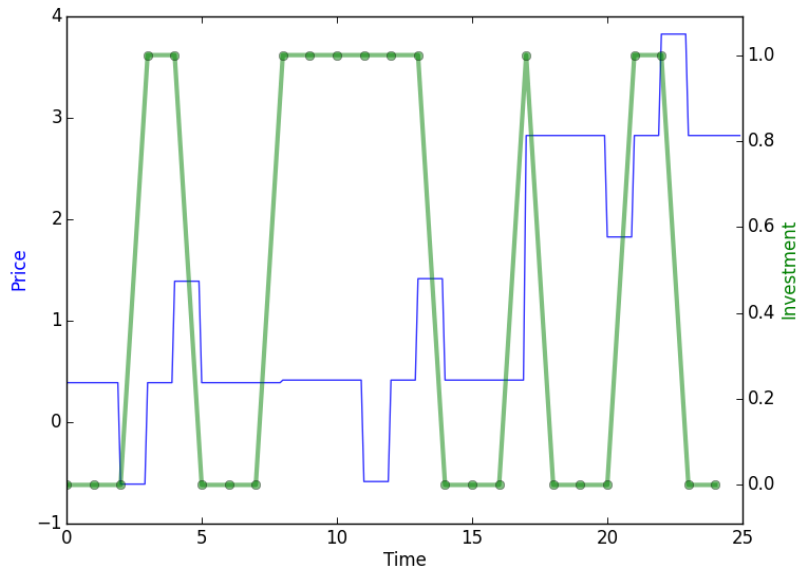
You can then launch "run_toy_env_simple.py" in the folder "examples/toy_env/".

Example: run_toy_env_simple.py

```
rng = np.random.RandomState(123456)

# --- Instantiate environment ---
env = Toy_env(rng)

# --- Instantiate qnetwork ---
qnetwork = MyQNetwork(
    environment=env,
    random_state=rng)

# --- Instantiate agent ---
agent = NeuralAgent(
    env,
    qnetwork,
    random_state=rng)

# --- Bind controllers to the agent ---
# Before every training epoch, we want to print a summary of important elements.
agent.attach(bc.VerboseController())

# During training epochs, we want to train the agent after every action it takes.
agent.attach(bc.TrainerController())

# We also want to interleave a "test epoch" between each training epoch.
agent.attach(bc.InterleavedTestEpochController(epoch_length=500))

# --- Run the experiment ---
agent.run(n_epochs=100, epoch_length=1000)
```

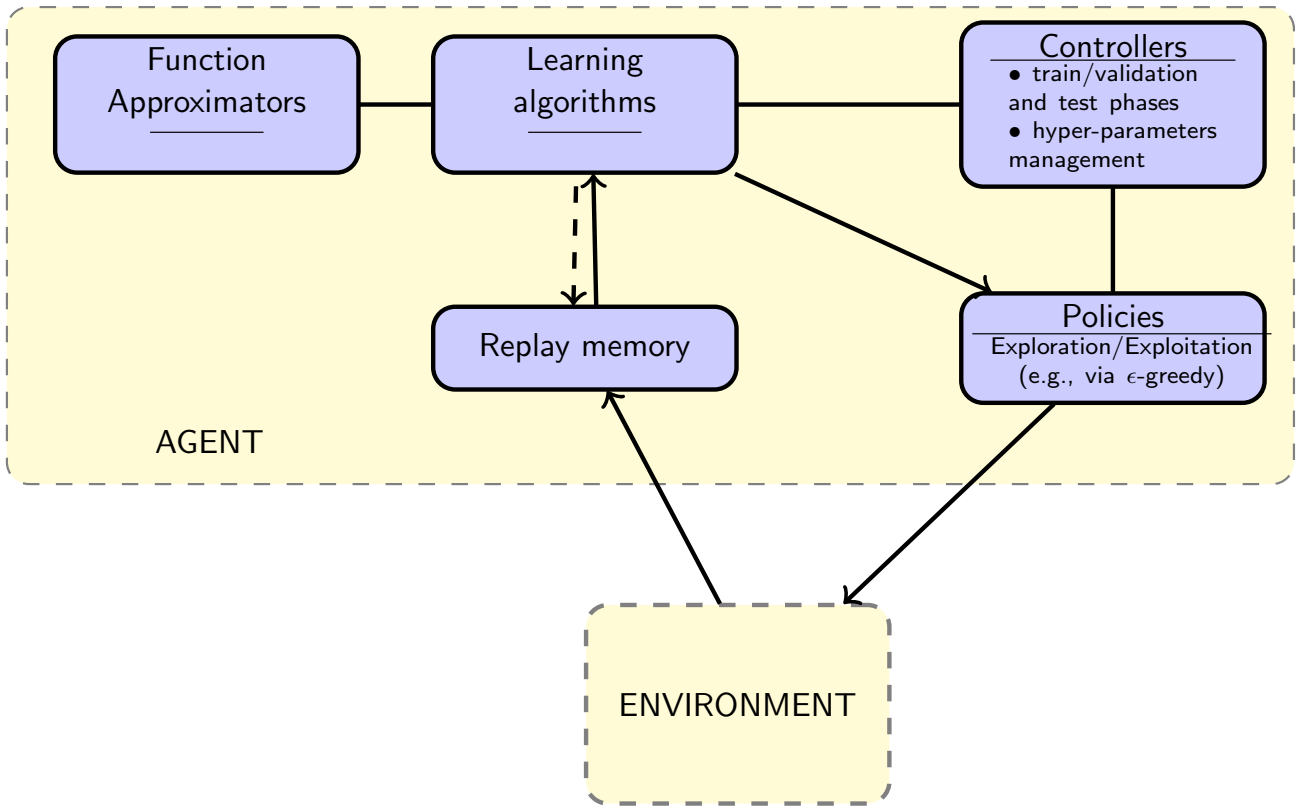
Example: run_toy_env_simple.py

Every 10 epochs, a graph is saved in the "toy_env" folder:



In this graph, you can see that the agent has successfully learned to take advantage of the price pattern. It is important to note that the results shown are made on a validation set that is different from the training and we can see that learning generalizes well. For instance, the action of buying at time step 7 and 16 is the expected result because in average this will allow to make profit since the agent has no information on the future.

Further ressources



Implementation : https://github.com/VinF/deer

Questions?