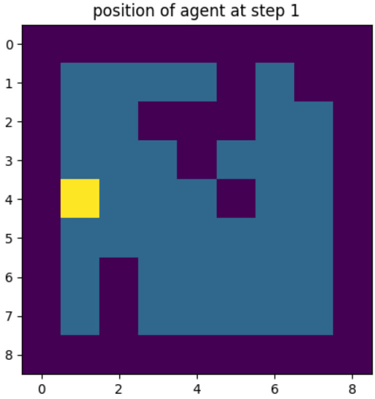
Machine learning II, unsupervised learning and agents: reinforcement learning



- RL has many applications and is quite a hot topic.
- **Deep Reinforcement Learning** has received a lot of attention recently.

- Atari games

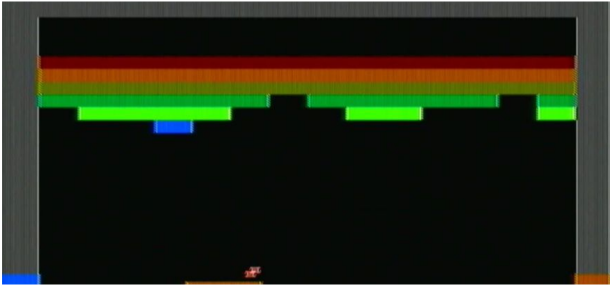


Figure – [?]

- AlphaGo

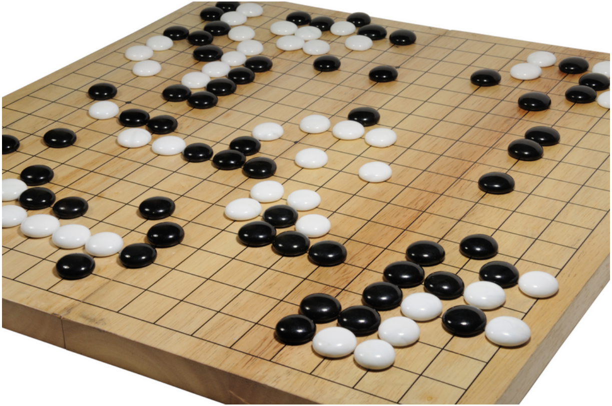


Figure – Go game, beaten by AlphaGo in 2017 [?]

- Reinforcement Learning is also being used in the community of **Computationnal neuroscience.**

Overview

Supervised learning and Correction

- In **supervised learning**, the supervisor indicates the **expected answer** the agent should answer.
- The feedback does not depend on the action performed by the agent (for instance the prediction from the agent)

Supervised learning and Correction

- In **supervised learning**, the supervisor indicates the **expected answer** the agent should answer.
- The feedback does not depend on the action performed by the agent (for instance the prediction from the agent)
- We say that the agent receives an **instructive feedback**.

Supervised learning Correction

- In **supervised learning**, the supervisor indicates the **expected answer** the agent should answer.
- The agent must then **correct its model** based on this answer.

Cost sensitive learning

- In **Cost sensitive learning**, the situation is different.
- The agent receives an **evaluative feedback**. The feedack depends on the action performed by the agent.

Cost sensitive learning

- In **Cost sensitive learning**, the situation is different.
- The agent receives an **evaluative feedback**. The feedack depends on the action performed by the agent.
- **Examples :**
  - AI playing a game and receiving "victory" or "defeat" as a feedback.
  - Child playing with an animal.

Reinforcement learning

- **Reinforcement learning** is a particular case of cost-sensitive learning.

Reinforcement learning

- **Reinforcement learning** is a particular case of cost-sensitive learning.
- In reinforcement learning, the feedback is a **real number**.

Reinforcement learning

- **Reinforcement learning** is a particular case of cost-sensitive learning.
- In reinforcement learning, the feedback is a **real number**.
- **Example :** amount of coins won after a poker turn.

Reinforcement learning

- First, the agent does not know if a reward is good or bad *per se*.
- A reward of -10 can be good or bad depending on the other rewards that are possible to obtain !

Reinforcement learning

- First, the agent does not know if a reward is good or bad per se.
- A reward of -10 good be good or bad depending on the other rewards that are possible to obtain.
- Most of the time, the objective of the agent will be to optimize the **agregation of rewards**.

Reinforcement learning

- The agent lives in a world E, and can be in several states s. The agent performs **actions** a and receives rewards r.

Reinforcement learning

- The agent lives in a world E, and can be in several states s. The agent performs **actions** a and receives rewards r.
- **Examples :**
  - world $= \mathbb{R}^2$
  - state $=$ position
  - actions $=$ moving somewhere
  - reward $=$ amount of food found

Formalization

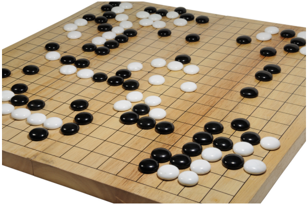
- There are many aspects of the problem that we need to formalize. Several formalizations are possible depending on the situation.

Formalization

- There are many aspects of the problem that we need to formalize. Several formalizations are possible depending on the situation.
- We will consider **discrete spaces** :
  - the time will be discrete
  - the number of possible states will be **finite**
  - the number of possible actions will be **finite**
- Continuous spaces are also available for RL. In those cases the objects are slightly different, and the optimization procedures also differ. For an introductory course, discrete spaces are more suitable.

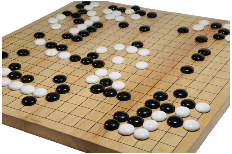
Question

- We will consider **discrete spaces** :
  - the time will be discrete
  - the number of possible states will be **finite**
  - the number of possible actions will be **finite**
- Are these hypotheses valid in the case of AlphaGo ?



Question

- We will consider **discrete spaces** :
  - the time will be discrete
  - the number of possible states will be **finite**
  - the number of possible actions will be **finite**
- Are these hypothesis valid in the case of AlphaGo ?



- Yes ! This shows that discrete spaces can still describe very complex problems.

Formalization

- we will write :
  - s_t : state at time t
  - a_t : action performed at time t
  - r_t : reward received at time t
- how is the action chosen ?

Let us continue with the formalization

- we will write :
  - s_t : state at time t
  - a_t : action performed at time t
  - r_t : reward received at time t
- the actions are chosen according to a **policy** π

Policies

- The policy π is a function of the current state.
- It can be **deterministic** : the action chosen is chosen with probability 1.

Policies

- The policy π is a function of the current state.
- It can be **deterministic :** the action chosen is chosen with probability 1.
- Or **stochastic :** the action performed in a given state is drawn from a **distribution**.

Two levels of randomness

- The policy can be deterministic or stochastic.
- But the result of an action could also be stochastic ! This is called a **stochastic transition function**.

Two levels of randomness

- The policy can be deterministic or stochastic.
- But the result of an action could also be stochastic! This is called a **stochastic transition function**.

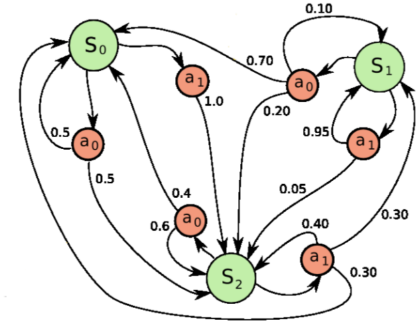


Figure – A stochastic policy with a stochastic transition function.

Exercice 1 :

- What is the probability of staying in state S_0 when performing an action from S_0 ? and from S_1 and S_2 ?

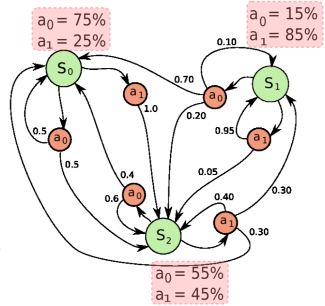


Figure – A stochastic policy with a stochastic transition function.

Agregation of rewards

- Remember that our agent want to optimize the **agregation of the rewards.**
- There are several ways to agregate the rewards.

Value function : episodic case

- If the *horizon* is finite (number of steps in the simulation), we can compute the value function of policy π, (assuming the actions are always taken according to policy π)

$$V^{\pi}(s_0) = r_0 + \cdots + r_N \tag{1}$$

- s_0 to the state and V to the *value*.

Value function : episodic case

- If the *horizon* is finite, we can take the sum

$$V^\pi(s_0) = r_0 + \cdots + r_N \tag{2}$$

- We could also average a window. For instance a window of size 3 :

$$V^\pi(s_0) = \frac{r_0 + r_1 + r_2}{3} \tag{3}$$

Value function : general case

- if the horizon is infinite, the **discount factor** $\gamma \in [0, 1[$ weights the rewards r_k

$$V^{\pi}(s_0) = \sum_{t=t_0}^{+\infty} \gamma^{t-t_0} r_t \tag{4}$$

More considerations

- The Markov hypothesis
- Exploitation exploration compromise

ϵ-greedy policy

A way to tackle the exploitation-exploration compromise.

- with probability $1 - \epsilon$: go to the best known reward (exploitation).
- with probability ϵ : perform a random action (exploration).

Art

"RL is a science, but dealing with the exploration-exploitation compromise is an art" (Sutton)

Dynamic programming

- Today we will study a simple case of Reinforcement learning
- Deterministic transition function.

World

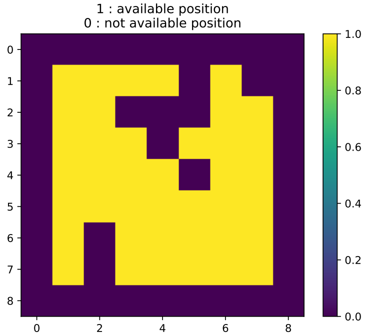


Figure – 2 dimensional world.

Reward

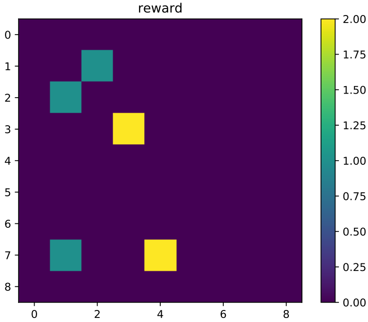


Figure – Reward function.

2D world

- Our agent can move in the 4 directions, one step at a time.
- We will progressively build an agent that learns to evaluate the states and then learns how to go to the best state.

Optimal policy

We look for the value of the **optimal policy** π^* .

$$V^*(s_0) = \max_{(a_t)_{t \in [0,+\infty]}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{5}$$

$R(s_t, a_t)$ is the reward of doing action a_t in state s_t.

Bellamn optimality equation

Exercice 2 :

- For each state s_0,

$$V^*(s_0) = \max_{(a_t)_{t\in[0,+\infty]}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{6}$$

- Can you express $V(s_0)$ as a function of $V(s_1)$?

Bellman optimality equation

$$V^*(s_0) = \max_a \left[r_1 + \gamma V^*(s_1(a)) \right] \tag{7}$$

with $s_1(a)$ being the state reached when choosing the action a in state s_0.
This is one of the many forms of Bellman equations (see Sutton book.)

Value Iteration

Value iteration belongs to dynamic programming methods. They differ from RL in that a perfect model of the environment is assumed.
These methods are building blocks for RL.

Value Iteration

- First, the initial Value function for all the states is 0.

Value Iteration

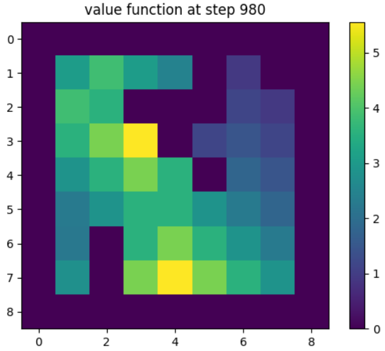
- First, the initial Value function for all the states is 0.
- Then we propagate the information about the rewards between the states, in order to **update the value function**
- We can find an optimal policy in the following way :

$$\forall s \in V(s_t) \leftarrow \max_{a_t} \left(r_{s_t} + \gamma V(s_{t+1}) \right) \tag{8}$$

(s_{t+1} depends on a_t).

Value iteration

- After learning, we will obtain a value function



- **cd reinforcement_learning/**
- Use the file **create_world.py** in order to generate your own environment.
- You can also use the one that is already there if you prefer.
- We store the data about the world in **.npy** files.

Exercice 3 :

- In **value_iteration.py**, modify the function **move_agent()** so that the agent is randomly moved at each time step.

Exercice 4 :

- In **value_iteration.py**, modify the function **update_value_function()** in order to update the value function according to the Bellman equation, and run the algorithm until convergence of the value function.

Exercice 5 :

- Use the file **value_iteration_policy** in order to design an optimal policy for our agent.

Optimal policy

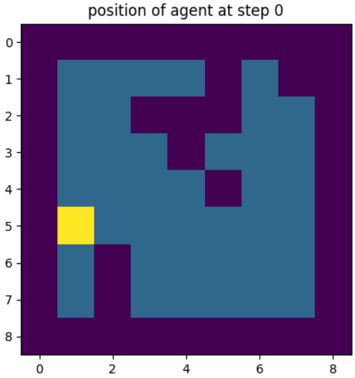


Figure – After learning hte optimal policy, the agent can go to the reward.

Optimal policy

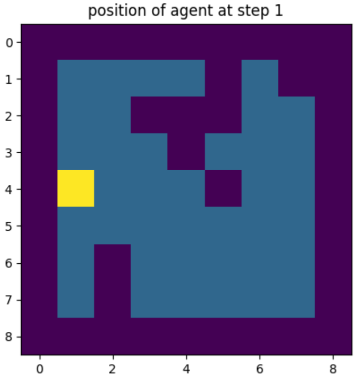


Figure – After learning, the agent can go to the reward.

Optimal policy

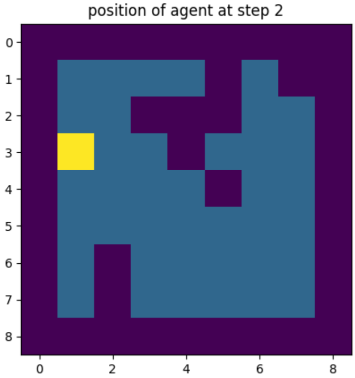


Figure – After learning, the agent can go to the reward.

Optimal policy

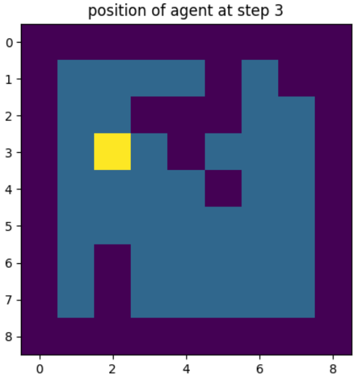


Figure – After learning, the agent can go to the reward.

Optimal policy

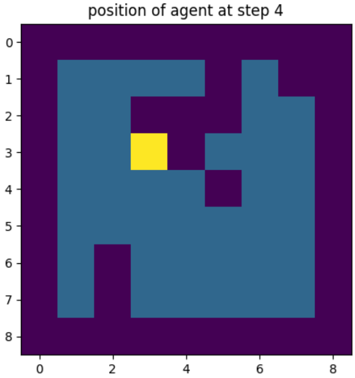


Figure – After learning, the agent can go to the reward.

Optimal policy

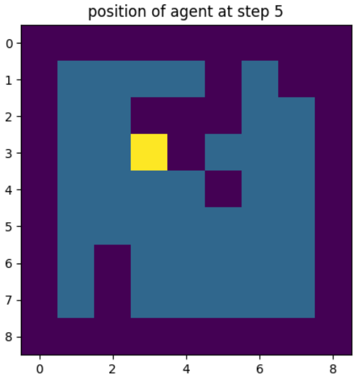


Figure – After learning, the agent can go to the reward.

Policy Iteration

- **Policy iteration** is another method that is slightly different.
- It consists in two steps :
  - **Policy evaluation**

Policy Iteration

- **Policy iteration** is another method that is slightly different.
- It consists in two steps :
  - **Policy evaluation**
  - **Policy improvement**

Exercice 6 :

- Pease use the file **policy_iteration.py** in order to perform the algorithm.
- Add randomness to the actions of the agent to **guarantee exploration**.

Multiple paradigms

- Reinforcement learning has many variants.
- In the ones we studied, a model of the effect of our actions were known.
- This is not always de case.

Temporal difference learning

- In temporal difference learning, the agent does not know a **model** of its world.
- But it can still learn the value function with the **TD updates**

Temporal difference learning

- In temporal difference learning, the agent does not know a **model** of its world.
- But it can still learn the value function with the **TD updates**

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{9}$$

Monte Carlo methods

Monte Carlo methods can be used in Reinforcement Learning to estimate the expected values of some random variables (such as the expected reward in a given state).

Actor critic methods

- Sometimes you can use **two** policies
  - the **behavior policy** provides actions and guarantees exploration
  - the **target polivy** is the optimal policy learned in parallel by the agent, that would be used in exploitation mode.

Tabular case and continous case

- We studied **finite** (and thus discrete situations).
- However, RL can also be applied to continuous state / discrete action spaces (DQN).

Tabular case and continous case

- We studied **finite** (and thus discrete situations).
- However, RL can also be applied to continuous state / discrete action spaces (DQN)
- And even to continous state / continous action spaces (DDPG) [?] .

References I

Bengio, Y. (2009).
CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING.
Foundations and Trends® in Machine Learning.

Mnih, V., Silver, D., and Riedmiller, M. (2013).
Deep Q Network (Google).
Arxiv.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016).

References II

Mastering the Game of Go with Deep Neural Networks and Tree Search.
Nature, 529(7587) :484–489.