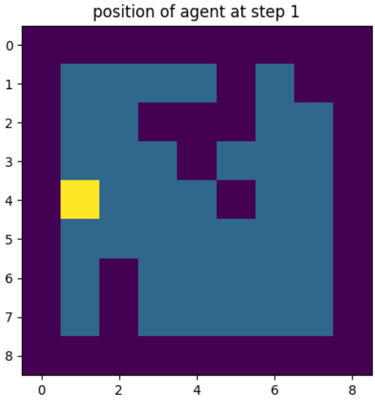
Machine learning II, unsupervised learning and agents: reinforcement learning



- RL has many applications and **Deep Reinforcement Learning** has received a lot of attention for more than 10 years now.
- Sutton textbook : `http://incompleteideas.net/book/the-book-2nd.html`

- Atari games

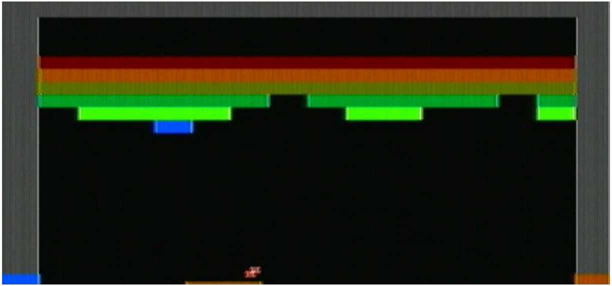


Figure – [Mnih et al., 2013]

- AlphaGo

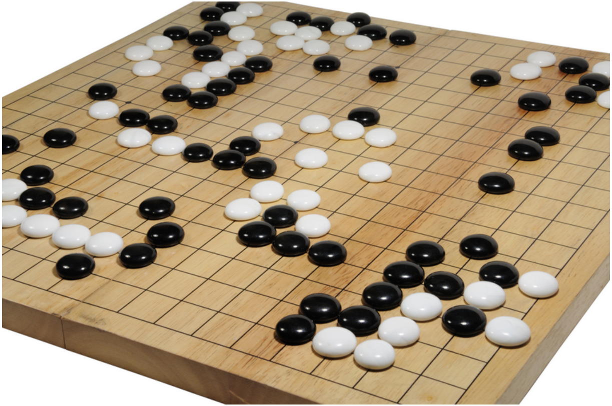


Figure – Go game, beaten by AlphaGo in 2017 [Silver et al., 2016]

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

Supervised learning and Correction

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

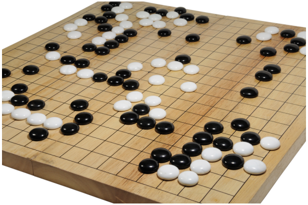
- The agent lives in a world E, and can be in several states s. The agent performs **actions** a and receives rewards r.
- **Examples :**
  - world $= \mathbb{R}^2$
  - state $=$ position
  - actions $=$ moving somewhere
  - reward $=$ amount of food found

Formalization

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

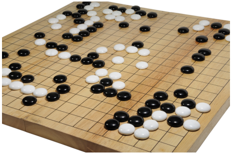
Question

- 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
  - R_t : reward received at time t
  - A_t : action performed at time t
- the actions are chosen according to a **policy** π

Action - reward loop

54 CHAPTER 3. FINITE MARKOV DECISION PROCESSES

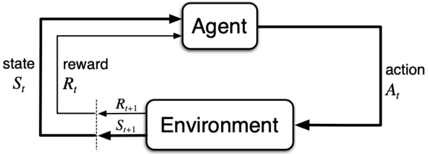


Figure 3.1: The agent–environment interaction in reinforcement learning.

Figure – Image taken from the Sutton book, page 68.

http://incompleteideas.net/book/the-book-2nd.html

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**.

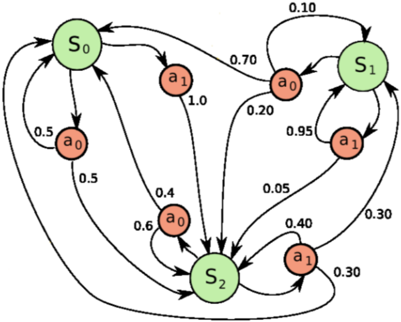


Figure – A stochastic policy with a stochastic transition function.

Exercice 1 :

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

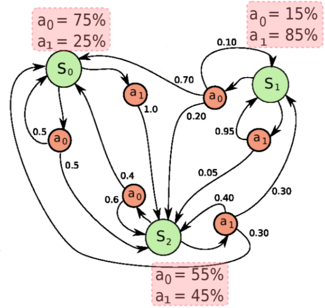


Figure – A stochastic policy with a stochastic transition function.

Agregation of rewards

- The agent wants to optimize the **agregation of the rewards.**
- There are several ways to agregate the rewards.

Returns

We introduce the **return** G_t.

- Episodic case (finite number N of steps) :

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_{t+N} \tag{1}$$

- Continuing tasks :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2}$$

$\gamma \in [0, 1[$ is the **discount factor**

Value function

Given a fixed policy π, the value function $v_\pi(s)$ quantifies how good a state s is.

$$v_\pi(s) = E[G_t | S_t = s] \tag{3}$$

(the expectation is taken over the next actions, following the policy π).

The Bellman equation

Given a fixed policy π, and a state s, we can write a recursive relationship between $v_\pi(s)$ and the values v_π of the next possible successor states (see the Sutton book for the general form of the equation 3.12 page 85).

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).

"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

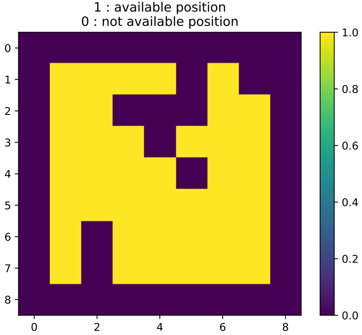


Figure – 2 dimensional world.

Reward

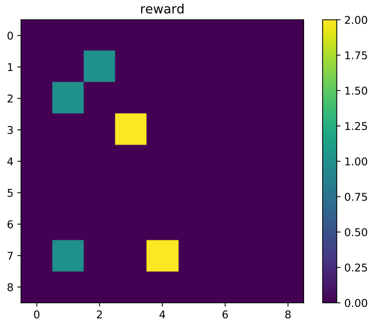


Figure – Reward function.

2D world

- Our agent can move in the 4 directions, one step at a time.

Optimal value functions

We look for the value of the **optimal policy** π^*, defined by the fact that it has the best value function among all policies.

$$
\begin{aligned}
v_*(s) &= \max_{a \in \text{available actions}} E[G_t | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[\sum_{k \geq 0} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[R_{t+1} + \gamma \sum_{k \geq 1} \gamma^{k-1} R_{t+k+1} | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[R_{t+1} + \gamma \sum_{k \geq 0} \gamma^k R_{t+k+2} | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]
\end{aligned} \tag{4}
$$

Bellman optimality equation

In the case of our simple 2D deterministic world, the Bellman optimality equation 4 takes a simpler form !

$$v_*(s) = \max_{a \in \text{available actions}} R_{t+1} + \gamma v_*(S_{t+1}) \tag{5}$$

The expected values are replaced by deterministic values.

Value Iteration

- Value iteration belongs to dynamic programming methods. They are a specific case of RL where a perfect model of the environment is assumed.
- In value iteration, equation 4 is used as an update rule at each time step.

Value Iteration

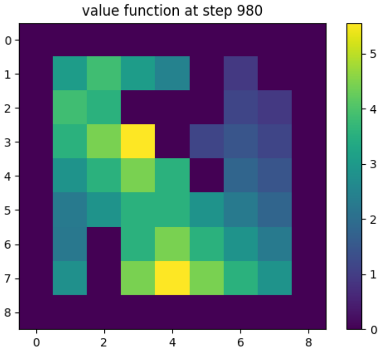
- First, the initial Value function for all the states is 0.
- Then we propagate the information about the rewards between the states, in order to **update the value function** (sweep)

$$\forall s \in V(s) \leftarrow \max_a \left(R_{t+1} + \gamma V(s_{t+1}) | S_t = s, A_t = a \right) \tag{6}$$

- In parallel, we explore the world to learn about the distribution of rewards.

Value iteration

- After learning, we will obtain a value function



Exercice 2 :

- **cd reinforcement_learning/**
- We store the data about the world in **.npy** files (optionally, you can generate a different world !)
- In **value_iteration.py**, modify :
  - **move_agent()** so that the agent is randomly moved at each time step.
  - **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.

Optimal policy

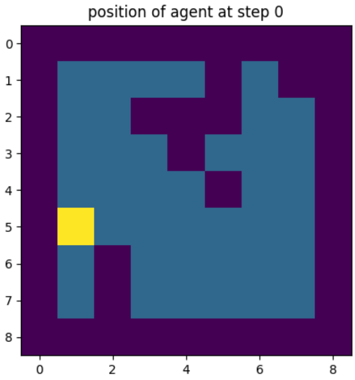


Figure – After learning hte optimal policy, the agent can go to the reward.

Optimal policy

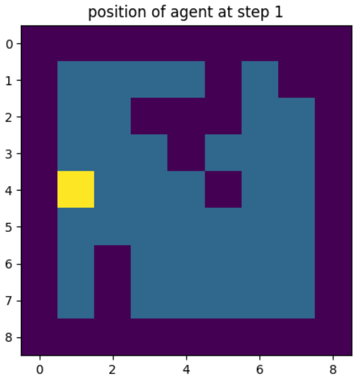


Figure – After learning, the agent can go to the reward.

Optimal policy

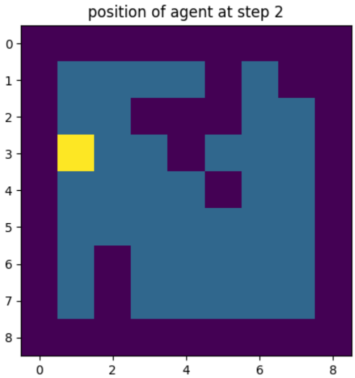


Figure – After learning, the agent can go to the reward.

Optimal policy

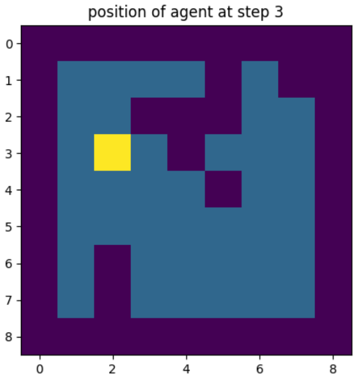


Figure – After learning, the agent can go to the reward.

Optimal policy

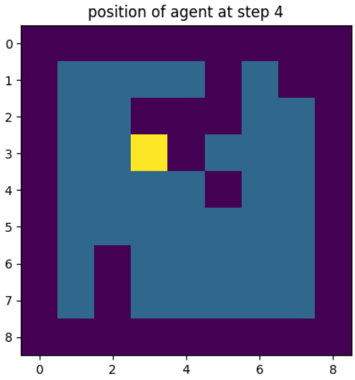


Figure – After learning, the agent can go to the reward.

Optimal policy

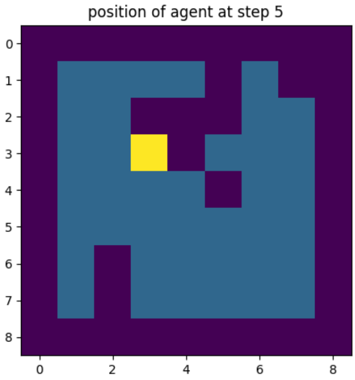


Figure – After learning, the agent can go to the reward.

Multiple paradigms

- Reinforcement learning has many variants.
- In the ones we studied, a model of the consequence of our actions was known.
- This is not always the 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 (temporal difference) updates**

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \qquad (7)$$

Monte Carlo methods

Monte Carlo methods can be used in Reinforcement Learning to estimate some expected values (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 policy** 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) [Bengio, 2009] .

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.