

Course 5: Reinforcement Learning



Last session

- 1 Combinatorial game theory
- 2 Definition of a game
- 3 Proof of determined games

Today's session

- 1 Reinforcement Learning
- 2 Value and Policy Functions
- 3 Q-Learning

Note: reinforcement learning and combinatorial game theory share a common mathematical framework. But to ease access to online resources, we will adopt a new vocabulary.

Learning Approaches

Supervised L

- 1 **Data:**
(x =data, y =labels)
- 2 **Goal:** Learn a mapping $x \rightarrow y$

This thing is a cheese



Reinforcement L

- 1 **Data:** (state, action)
- 2 **Goal:** Maximize future rewards

Eat this thing for reward



Unsupervised L

- 1 **Data:** x =data, no labels
- 2 **Goal:** Learn structure in x

This thing is like the other thing



Outline of the course

1 Definitions of Reinforcement Learning (RL)

- Fundamentals
- Example: PyRat
- Policy and values

2 Q-learning

- Q-learning definitions
- Example
- Approximate Q-learning
- Exploration/Exploitation

1 Definitions of Reinforcement Learning (RL)

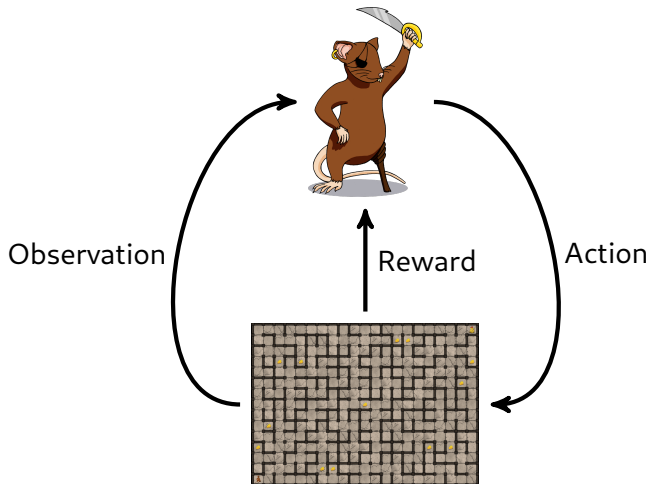
- Fundamentals
- Example: PyRat
- Policy and values

2 Q-learning

- Q-learning definitions
- Example
- Approximate Q-learning
- Exploration/Exploitation

Agent and environment

Our objective is to train an **agent** to maximize its **reward** through **actions** that affect an **environment**.



Reward hypothesis

- *All goals can be described by the maximization of expected cumulated reward over time.*

Specificities of reinforcement learning

- No supervision, only a reward signal,
- Delayed feedback, the reward can come (much) later,
- Importance of the temporal dimension,
- Agent's actions affect the subsequent data it receives.

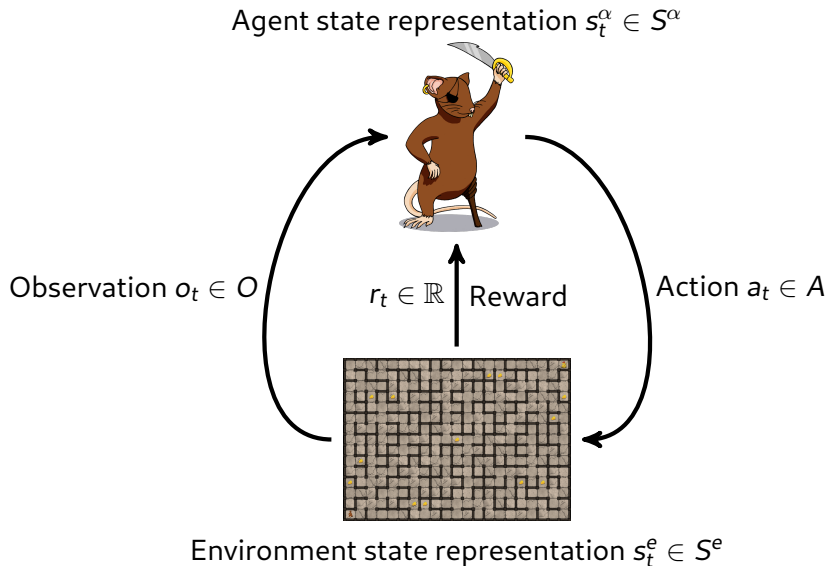
Reward hypothesis

- *All goals can be described by the maximization of expected cumulated reward over time.*

Specificities of reinforcement learning

- No supervision, only a reward signal,
- Delayed feedback, the reward can come (much) later,
- Importance of the temporal dimension,
- Agent's actions affect the subsequent data it receives.

Agent and environment



Definitions

■ The agent α ...

- 1 analyzes previous actions, states, rewards and observations,
- 2 computes action a_t ,
- 3 obtains reward r_t ,
- 4 obtains an observation o_{t+1} ,
- 5 deduce a new state s_{t+1}^α .

■ The environment...

- 1 receives action a_t ,
- 2 produces reward r_t ,
- 3 deduce a new state s_{t+1}^e ,
- 4 produces o_{t+1} .

■ Observability:

- **Perfect:** $s_t^\alpha = s_t^e = o_t$
- **Imperfect:** no access to full environment state:
 - The agent indirectly observes the environment through o_t ,
 - s_t^α is estimated by the agent and may differ from s_t^e .

Definitions

■ The agent α ...

- 1 analyzes previous actions, states, rewards and observations,
- 2 computes action a_t ,
- 3 obtains reward r_t ,
- 4 obtains an observation o_{t+1} ,
- 5 deduce a new state s_{t+1}^α .

■ The environment...

- 1 receives action a_t ,
- 2 produces reward r_t ,
- 3 deduce a new state s_{t+1}^e ,
- 4 produces o_{t+1} .

■ Observability:

- **Perfect:** $s_t^\alpha = s_t^e = o_t$
- **Imperfect:** no access to full environment state:
 - The agent indirectly observes the environment through o_t ,
 - s_t^α is estimated by the agent and may differ from s_t^e .

Definitions

■ The agent α ...

- 1 analyzes previous actions, states, rewards and observations,
- 2 computes action a_t ,
- 3 obtains reward r_t ,
- 4 obtains an observation o_{t+1} ,
- 5 deduce a new state s_{t+1}^α .

■ The environment...

- 1 receives action a_t ,
- 2 produces reward r_t ,
- 3 deduce a new state s_{t+1}^e ,
- 4 produces o_{t+1} .

■ Observability:

- **Perfect:** $s_t^\alpha = s_t^e = o_t$
- **Imperfect:** no access to full environment state:
 - The agent indirectly observes the environment through o_t ,
 - s_t^α is estimated by the agent and may differ from s_t^e .

Definitions

■ The agent α ...

- 1 analyzes previous actions, states, rewards and observations,
- 2 computes action a_t ,
- 3 obtains reward r_t ,
- 4 obtains an observation o_{t+1} ,
- 5 deduce a new state s_{t+1}^α .

■ The environment...

- 1 receives action a_t ,
- 2 produces reward r_t ,
- 3 deduce a new state s_{t+1}^e ,
- 4 produces o_{t+1} .

■ Observability:

- **Perfect:** $s_t^\alpha = s_t^e = o_t$
- **Imperfect:** no access to full environment state:
 - The agent indirectly observes the environment through o_t ,
 - s_t^α is estimated by the agent and may differ from s_t^e .

Example: PyRat

Definitions

- The agent is either the Rat or the Python,
- The opponent becomes part of the environment,
 - Note that the game can be with perfect observability if the opponent strategy is known,
- Seen this way, the game becomes sequential.

RL-based PyRat versus supervised approach

- Reward signal: number of picked up pieces of cheese,
- Delayed feedback: several moves required to reach a reward,
- Character's moves affect subsequent data it receives,
- Importance of the temporal dimension.

Example: PyRat

Definitions

- The agent is either the Rat or the Python,
- The opponent becomes part of the environment,
 - Note that the game can be with perfect observability if the opponent strategy is known,
- Seen this way, the game becomes sequential.

RL-based PyRat versus supervised approach

- Reward signal: number of picked up pieces of cheese,
- Delayed feedback: several moves required to reach a reward,
- Character's moves affect subsequent data it receives,
- Importance of the temporal dimension.

Observability examples

- Perfect: $s_t^{rat} = s_t^o = o_t$
 - a_t : Last move of the rat,
 - o_t : The entire maze with all cheese locations and python position,
 - r_t : Binary variable which is 1 if the rat just got a piece of cheese.
- Imperfect:
 - a_t : Last move of the rat,
 - o_t : Neighboring cells of the rat,
 - r_t : Binary variable which is 1 if the rat just got a piece of cheese.

To represent the strategy of the rat, we use a **policy function**.

Observability examples

- Perfect: $s_t^{rat} = s_t^o = o_t$
 - a_t : Last move of the rat,
 - o_t : The entire maze with all cheese locations and python position,
 - r_t : Binary variable which is 1 if the rat just got a piece of cheese.
- Imperfect:
 - a_t : Last move of the rat,
 - o_t : Neighboring cells of the rat,
 - r_t : Binary variable which is 1 if the rat just got a piece of cheese.

To represent the strategy of the rat, we use a **policy function**.

Observability examples

- Perfect: $s_t^{rat} = s_t^o = o_t$
 - a_t : Last move of the rat,
 - o_t : The entire maze with all cheese locations and python position,
 - r_t : Binary variable which is 1 if the rat just got a piece of cheese.
- Imperfect:
 - a_t : Last move of the rat,
 - o_t : Neighboring cells of the rat,
 - r_t : Binary variable which is 1 if the rat just got a piece of cheese.

To represent the strategy of the rat, we use a [policy function](#).

Policy Function

Definition

The policy function of an agent α is:

$$\pi : \begin{cases} S^\alpha & \rightarrow A \\ s_t^\alpha & \mapsto a_t^\alpha \end{cases}$$

- π can be deterministic or stochastic.

Playout

The playout $(s_t^{\alpha, \pi})_{t \in \mathbb{N}}$ associated with a policy π and initial state s_0 , is defined by considering agent α takes his/her actions using π .

Policy Function

Definition

The policy function of an agent α is:

$$\pi : \begin{cases} S^\alpha & \rightarrow A \\ s_t^\alpha & \mapsto a_t^\alpha \end{cases}$$

- π can be deterministic or stochastic.

Playout

The playout $(s_t^{\alpha, \pi})_{t \in \mathbb{N}}$ associated with a policy π and initial state s_0 , is defined by considering agent α takes his/her actions using π .

Value Function

Definition

Fix $\gamma \in [0, 1[$, the value function v^π is defined as:

$$v^\pi : \begin{cases} S^\alpha & \rightarrow \mathbb{R} \\ s_{t_0}^{\alpha, \pi} & \mapsto \sum_{t=t_0}^{+\infty} \gamma^{t-t_0} r_t \end{cases}$$

- The value of a policy function is thus an expectation of cumulative future rewards, weakened by the geometrical coefficient γ to avoid divergence and prioritize short term reward,
- The best possible policy $\pi^*(s)$ is defined as:

$$\forall s \in S^\alpha, \forall \pi, V^{\pi^*}(s) \geq V^\pi(s).$$

Value Function

Definition

Fix $\gamma \in [0, 1[$, the value function v^π is defined as:

$$v^\pi : \begin{cases} S^\alpha & \rightarrow \mathbb{R} \\ s_{t_0}^{\alpha, \pi} & \mapsto \sum_{t=t_0}^{+\infty} \gamma^{t-t_0} r_t \end{cases}$$

- The value of a policy function is thus an expectation of cumulative future rewards, weakened by the geometrical coefficient γ to avoid divergence and prioritize short term reward,
- The best possible policy $\pi^*(s)$ is defined as:

$$\forall s \in S^\alpha, \forall \pi, V^{\pi^*}(s) \geq V^\pi(s).$$

1 Definitions of Reinforcement Learning (RL)

- Fundamentals
- Example: PyRat
- Policy and values

2 Q-learning

- Q-learning definitions
- Example
- Approximate Q-learning
- Exploration/Exploitation

Q-learning

Definition

In Q-learning, we aim to find V^{π^*} as a solution to the recursive system of equations (Bellman equation):

$$\forall s \in S^\alpha, \forall a \in A, Q(s, a) = r_{s,a} + \gamma \max_{a'} Q(s(a), a'),$$

where $r_{s,a}$ is the reward agent α performs action a in state s and $s(a)$ is the state observed by agent α after performing action a .

Pros and cons

Pros:

- Can be learned even if the agent is not following any specific π ,
- Self training is possible,

Cons:

- Scalability issues when S^α is large.

Q-learning

Definition

In Q-learning, we aim to find V^{π^*} as a solution to the recursive system of equations (Bellman equation):

$$\forall s \in S^\alpha, \forall a \in A, Q(s, a) = r_{s,a} + \gamma \max_{a'} Q(s(a), a'),$$

where $r_{s,a}$ is the reward agent α performs action a in state s and $s(a)$ is the state observed by agent α after performing action a .

Pros and cons

Pros:

- Can be learned even if the agent is not following any specific π ,
- Self training is possible,

Cons:

- Scalability issues when S^α is large.

Q-learning

Definition

In Q-learning, we aim to find V^{π^*} as a solution to the recursive system of equations (Bellman equation):

$$\forall s \in S^\alpha, \forall a \in A, Q(s, a) = r_{s,a} + \gamma \max_{a'} Q(s(a), a'),$$

where $r_{s,a}$ is the reward agent α performs action a in state s and $s(a)$ is the state observed by agent α after performing action a .

Pros and cons

Pros:

- Can be learned even if the agent is not following any specific π ,
- Self training is possible,

Cons:

- Scalability issues when S^α is large.

Q-learning: value iteration

Definition

In Q-learning, we aim to find V^{π^*} as a solution to the recursive system of equations (Bellman equation):

$$\forall s \in S^{\alpha}, \forall a \in A, Q(s, a) = r_{s,a} + \gamma \max_{a'} Q(s(a), a'),$$

where $r_{s,a}$ is the reward agent α performs action a in state s and $s(a)$ is the state observed by agent α after performing action a .

	UP	DOWN	RIGHT	LEFT
s1	+2	0	+1	+2.5
s2	0	+1	+4	+1
s3	0	+1	0	0
s4	0	+1	+2.5	+2.5

initialize $Q[s,a]$ arbitrarily

observe initial state s

repeat

 select action a

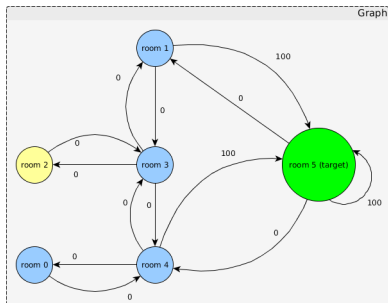
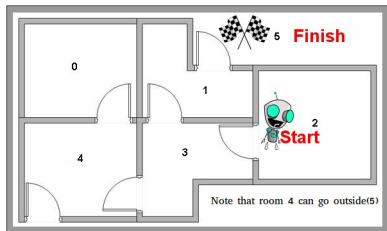
 observe reward r and new state s'

$Q[s,a] = r + \gamma \max_{a'} Q[s',a']$

$s = s'$

until end

Example



https://leonardoaraujasantos.gitbook.io/artificial-intelligence/artificial_intelligence/reinforcement_learning/qlearning_simple

Example

Step 1: state 1, random action 5

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$
$$R = \begin{matrix} & \begin{matrix} \text{State} & 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} \text{Action} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{matrix}$$
$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Update Q(1,5):

$$Q(1,5) = R(1,5) + 0.8 \cdot \max([Q(5,1), Q(5,4), Q(5,5)]) = 100 + 0.8(0) = 100$$

Step 2: state 3, random action 1

$$Q(3,1) = R(3,1) + 0.8 \cdot \max([Q(1,3), Q(1,5)]) = 0 + 0.8(100) = 80$$

Step 3:continue to better estimate the Q table

Approximated Q-learning

Definition

- Train a model to approximate Q ,
 - Input is a state s and output is made of values of $Q(s, \cdot)$,
 - Representation learning can be used to compress S^α .

Problems

- Almost always needs a simulator for the game,
- Game duration can be bottleneck for training,
- Catastrophic forgetting and adversary specialization,
 - These effects can be alleviated by training using experience replay.

Experience replay

- Instead of using only the last decision to train, sample at random from the m previous decisions,
- Decisions taken before should remain considered now.

Approximated Q-learning

Definition

- Train a model to approximate Q ,
 - Input is a state s and output is made of values of $Q(s, \cdot)$,
 - Representation learning can be used to compress S^α .

Problems

- Almost always needs a simulator for the game,
- Game duration can be bottleneck for training,
- Catastrophic forgetting and adversary specialization,
 - These effects can be alleviated by training using experience replay.

Experience replay

- Instead of using only the last decision to train, sample at random from the m previous decisions,
- Decisions taken before should remain considered now.

Approximated Q-learning

Definition

- Train a model to approximate Q ,
 - Input is a state s and output is made of values of $Q(s, \cdot)$,
 - Representation learning can be used to compress S^α .

Problems

- Almost always needs a simulator for the game,
- Game duration can be bottleneck for training,
- Catastrophic forgetting and adversary specialization,
 - These effects can be alleviated by training using experience replay.

Experience replay

- Instead of using only the last decision to train, sample at random from the m previous decisions,
- Decisions taken before should remain considered now.

Dilemma

- Repeat with existing strategy (Exploitation)...
- ... or try a new strategy (Exploration)?

Example

- Always eating in restaurants that you know is exploitation,
- While that is a good heuristic, you have no way of knowing if you have the maximum reward possible,
- So exploring new restaurants from time to time may be needed to find the maximum reward.

Dilemma

- Repeat with existing strategy (Exploitation)...
- ... or try a new strategy (Exploration)?

Example

- Always eating in restaurants that you know is exploitation,
- While that is a good heuristic, you have no way of knowing if you have the maximum reward possible,
- So exploring new restaurants from time to time may be needed to find the maximum reward.

TP4 - PyRat with reinforcement learning

- Approximate Q-learning algorithm using experience replay and linear regression to beat the greedy algorithm,
- Approximation method (linear regression) and experience replay routine are given,
- Improve the performances of Reinforcement Learning to play PyRat.

Challenge

You can continue working in the challenge after finishing the RL practical. You can now integrate reinforcement learning in your solution.