Course 5: Reinforcement Learning

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

2023-11-23

Course 5: Reinforcement Learning

Summary

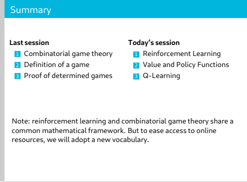
Last session

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.



Summary

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.

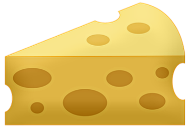
It is good to tell them that we chose not to use the same notations as in the CGT lesson, although we could have.

Learning Approaches

Supervised L

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

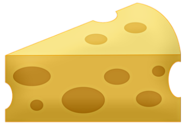
This thing is a cheese



Reinforcement L

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

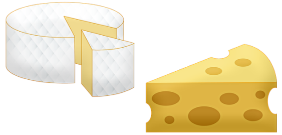
Eat this thing for reward

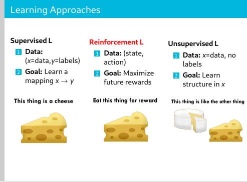


Unsupervised L

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

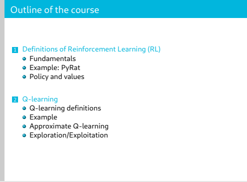
This thing is like the other thing





As for UL in Reinforcement learning the algorithm does not know what the object is (cheese) –> it learns by trial and error. RL stands in between supervised and unsupervised: kind of semi-supervised–> learns from sparse rewards, exploiting information about the world, state, actions

Outline of the course


Outline of the course

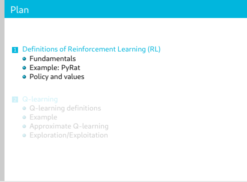
1 Definitions of Reinforcement Learning (RL)
- Fundamentals
- Example: PyRat
- Policy and values

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

Plan

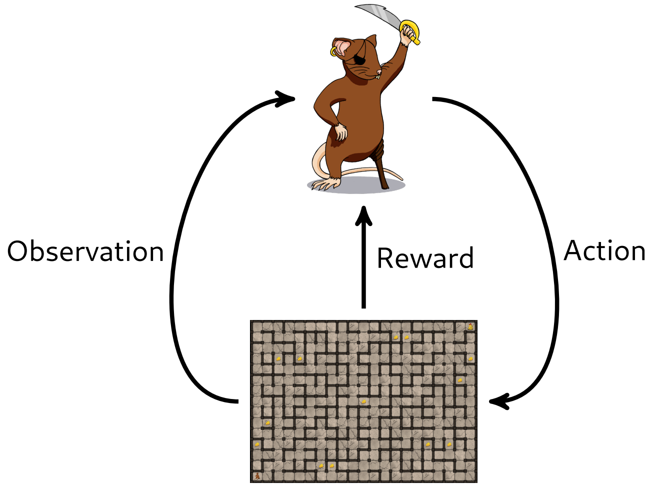
2023-11-23

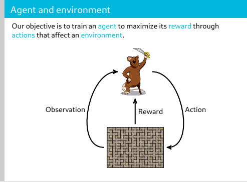
Course 5: Reinforcement Learning
└Definitions of Reinforcement Learning (RL)
└Plan



Agent and environment

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





Agent and environment

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

Pretty much self-explanatory. At this point, there is no need to talk about the specificity of Pyrat, because we will detail all this in the next slides. Just go through the definition : the agent is the rat, the actions are its moves, the reward could be something related to winning the game, and the observation is what the rat sees. (don't detail here)

GL: Let's set the Vocabulary, little different from other approaches.

Agent: takes actions. Environment:where agent operates and exists.

Observation: data that the agent sees in return from the environment.

Reward: feedback on the success of agent action

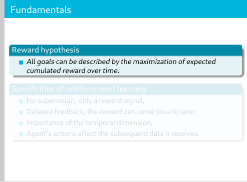
Fundamentals

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.


Fundamentals

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.

It is important to emphasize the fact that reinforcement learning is built upon the reward hypothesis, which is still an hypothesis. Some goals may not be described by maximizing reward.
For each of the specificities of RL, you may want to illustrate it with a very short example.GL: reward signal numeric (+1 if you eat a cheese). Delayed: you don't get a reward after each action, can be much later: in some applications you only have a reward at the end. Temporal: RL is based on step: start state s at time t, take an action, reach state s t+1... Actions modify the environment, changes observations the agent receives

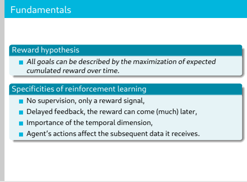
Fundamentals

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.

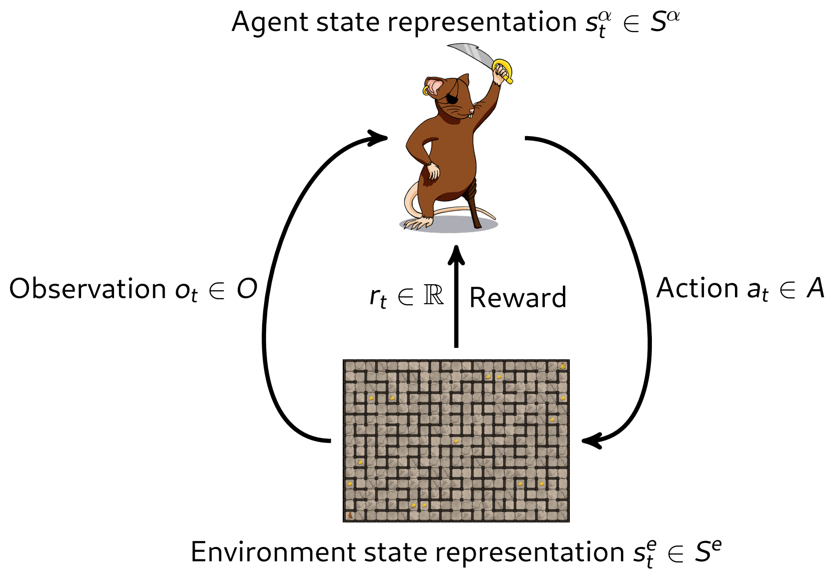

Fundamentals

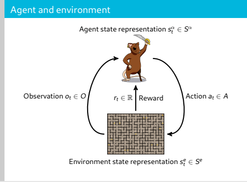
Reward hypothesis
- All goals can be described by the maximization of expected cumulated reward over time.

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.

It is important to emphasize the fact that reinforcement learning is built upon the reward hypothesis, which is still an hypothesis. Some goals may not be described by maximizing reward.
For each of the specificities of RL, you may want to illustrate it with a very short example.GL: reward signal numeric (+1 if you eat a cheese). Delayed: you don't get a reward after each action, can be much later: in some applications you only have a reward at the end. Temporal: RL is based on step: start state s at time t, take an action, reach state s t+1... Actions modify the environment, changes observations the agent receives

Agent and environment


Agent state representation $s_t^\alpha \in S^\alpha$
Observation $o_t \in O$
$r_t \in \mathbb{R}$
Reward
Action $a_t \in A$
Environment state representation $s_t^e \in S^e$

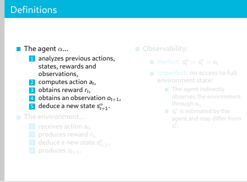


Now we can finally put some notations on Pyrat in the context of Reinforcement Learning. At this stage, it is important to note that the state of the rat and the state of the environment are not the same.

GL: state: situation that agent perceives

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^{α}_{t+1}.
- The environment...
  1. receives action a_t,
  2. produces reward r_t,
  3. deduce a new state s^{e}_{t+1},
  4. produces o_{t+1}.
- Observability:
  - Perfect: $s^{\alpha}_t = s^{e}_t = o_t$
  - Imperfect: no access to full environment state:
    - The agent indirectly observes the environment through o_t,
    - s^{α}_t is estimated by the agent and may differ from s^{e}_t.

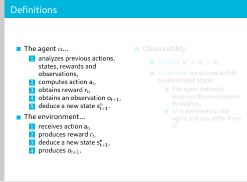


And now we desrcibe with even more detail what those definitions relate to exactly. In the right part of the slide, the discussion about observability can be related to perfect and imperfect information in the CGT lesson. The important (and sometimes tricky) thing to understand is that in the case of perfect (aka full) observability, the state of the environement, of the rat and the observation is the SAME thing ; bascially, there is no need in differentiating them mathematically.

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

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


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

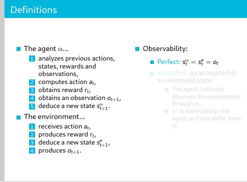
- Observability:
  - Perfect: $s^{\alpha}_t = s^{e}_t = o_t$
  - Imperfect: no access to full environment state:
    - The agent indirectly observes the environment through o_t,
    - s^{α}_t is estimated by the agent and may differ from s^{e}_t.

And now we desrcibe with even more detail what those definitions relate to exactly. In the right part of the slide, the discussion about observability can be related to perfect and imperfect information in the CGT lesson. The important (and sometimes tricky) thing to understand is that in the case of perfect (aka full) observability, the state of the environement, of the rat and the observation is the SAME thing ; bascially, there is no need in differentiating them mathematically.

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^α_{t+1}.
- The environment...
  1. receives action a_t,
  2. produces reward r_t,
  3. deduce a new state s^e_{t+1},
  4. produces o_{t+1}.

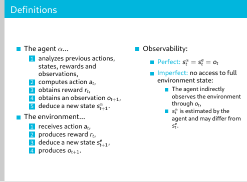
- Observability:
  - Perfect: $s^\alpha_t = s^e_t = o_t$
  - Imperfect: no access to full environment state:
    - The agent indirectly observes the environment through o_t,
    - s^α_t is estimated by the agent and may differ from s^e_t.



And now we desrcibe with even more detail what those definitions relate to exactly. In the right part of the slide, the discussion about observability can be related to perfect and imperfect information in the CGT lesson. The important (and sometimes tricky) thing to understand is that in the case of perfect (aka full) observability, the state of the environement, of the rat and the observation is the SAME thing ; bascially, there is no need in differentiating them mathematically.

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^{α}_{t+1}.
- The environment...
  1. receives action a_t,
  2. produces reward r_t,
  3. deduce a new state s^{e}_{t+1},
  4. produces o_{t+1}.
- Observability:
  - Perfect: $s^{\alpha}_t = s^{e}_t = o_t$
  - Imperfect: no access to full environment state:
    - The agent indirectly observes the environment through o_t,
    - s^{α}_t is estimated by the agent and may differ from s^{e}_t.



And now we desrcibe with even more detail what those definitions relate to exactly. In the right part of the slide, the discussion about observability can be related to perfect and imperfect information in the CGT lesson. The important (and sometimes tricky) thing to understand is that in the case of perfect (aka full) observability, the state of the environement, of the rat and the observation is the SAME thing ; bascially, there is no need in differentiating them mathematically.

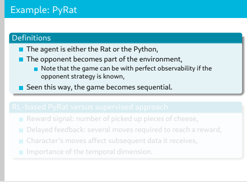
Example: PyRat

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.

In the final part of this slide (observability examples), we fully describe the case of PyRat with precise definitions. This enables us to define the policy function.

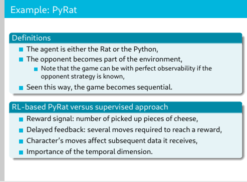
Example: PyRat

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.



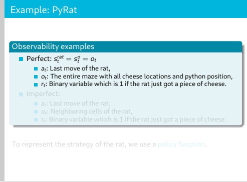
In the final part of this slide (observability examples), we fully describe the case of PyRat with precise definitions. This enables us to define the policy function.

Example: PyRat

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.



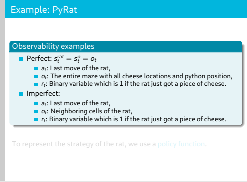
In the final part of this slide (observability examples), we fully describe the case of PyRat with precise definitions. This enables us to define the policy function.

Example: PyRat

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.



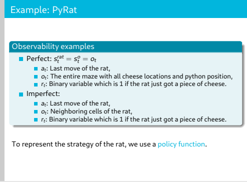
In the final part of this slide (observability examples), we fully describe the case of PyRat with precise definitions. This enables us to define the policy function.

Example: PyRat

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.


Example: PyRat

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.

In the final part of this slide (observability examples), we fully describe the case of PyRat with precise definitions. This enables us to define the policy function.

Policy Function

Definition

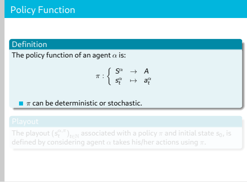
The policy function of an agent α is:

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

- 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.



A policy is a function from the set of states to the set of actions. We can mirror here the previous course on CGT by defining the playout associated with a policy (in CGT, we were talking about strategies, but it is important to make the difference because strategies were defined on sequences of vertices in the arena).

Policy Function

Definition

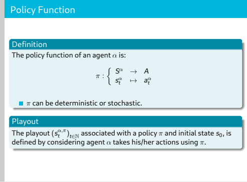
The policy function of an agent α is:

$$\pi : \left\{ \begin{array}{lcl} S^\alpha & \to & A \\ s_t^\alpha & \mapsto & a_t^\alpha \end{array} \right.$$

- 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:
$$\pi : \left\{ \begin{array}{lcl} S^\alpha & \to & A \\ s_t^\alpha & \mapsto & a_t^\alpha \end{array} \right.$$
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.

A policy is a function from the set of states to the set of actions. We can mirror here the previous course on CGT by defining the playout associated with a policy (in CGT, we were talking about strategies, but it is important to make the difference because strategies were defined on sequences of vertices in the arena).

Value Function

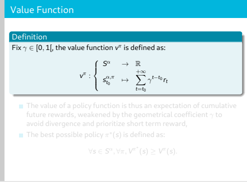
Definition

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

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

- 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:

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



This definition is quite self explanatory ; it is the core of RL as value is directly the sum of future rewards. The value function captures the expected total future reward an agent in state S can obtain applying an action a associated to the policy pi. The best policy is the one that maximize the value function. This basic principle is the basis of RL algorithm, such as Q-learning.GL: the total expected reward is not just the sum of the rewards associated to subsequent actions, but future rewards are multiplied but a factor that gets smaller for future rewards. This is also called DISCOUNTED future rewards: the closest reward in time counts more than future ones. Gamma is also called discounted factor.

Value Function

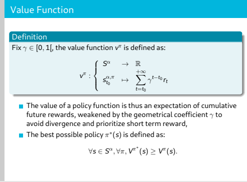
Definition

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

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

- 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:

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

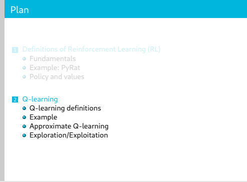

Value Function
Definition
Fix $\gamma \in [0,1[$, the value function v^π is defined as:
The value of a policy function is thus an expectation of cumulative future rewards, weakened by the geometrical coefficient γ to avoid divergence and prioritize short term reward,
The best possible policy $\pi^*(s)$ is defined as:
$\forall s \in S^\alpha, \forall \pi, V^{\pi^*}(s) \geq V^\pi(s).$

This definition is quite self explanatory ; it is the core of RL as value is directly the sum of future rewards. The value function captures the expected total future reward an agent in state S can obtain applying an action a associated to the policy pi. The best policy is the one that maximize the value function. This basic principle is the basis of RL algorithm, such as Q-learning.GL: the total expected reward is not just the sum of the rewards associated to subsequent actions, but future rewards are multiplied but a factor that gets smaller for future rewards. This is also called DISCOUNTED future rewards: the closest reward in time counts more than future ones. Gamma is also called discounted factor.

Plan

2023-11-23

Course 5: Reinforcement Learning
└Q-learning
└Plan


Plan

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.

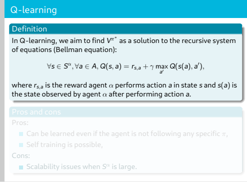
Pros and cons

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.

The tricky part of the definition of Q learning is that students have to understand about (1) the fact that it is defined iteratively and (2) that Q is defined on couples (s,a) and not just s, and (3) that $r_{s,a}$ is the reward that has been obtained by performing action a in state s. So, we are estimating Q(s,a) by summing the obtained reward with the maximum possible Q value across all actions from the next state.

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.

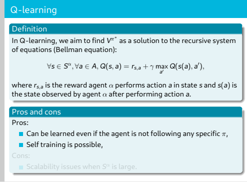
Pros and cons

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.

The tricky part of the definition of Q learning is that students have to understand about (1) the fact that it is defined iteratively and (2) that Q is defined on couples (s,a) and not just s, and (3) that $r_{s,a}$ is the reward that has been obtained by performing action a in state s. So, we are estimating Q(s,a) by summing the obtained reward with the maximum possible Q value across all actions from the next state.

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.

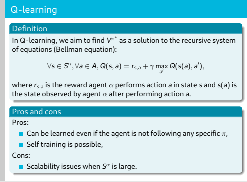
Pros and cons

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.

The tricky part of the definition of Q learning is that students have to understand about (1) the fact that it is defined iteratively and (2) that Q is defined on couples (s,a) and not just s, and (3) that $r_{s,a}$ is the reward that has been obtained by performing action a in state s. So, we are estimating Q(s,a) by summing the obtained reward with the maximum possible Q value across all actions from the next state.

Q-learning: value iteration

Definition

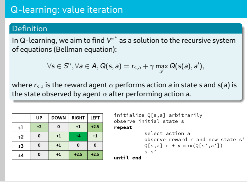
In Q-learning, we aim to find V^{π^*} as a solution to the recursive system of equations (Bellman equation):

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

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.

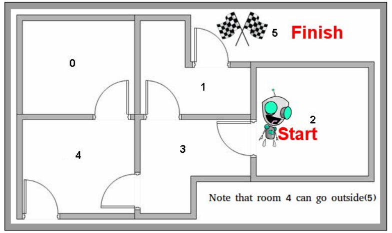
	UP	DOWN	RIGHT	LEFT
s1	+2	0	+1	+2.5
s2	0	+1	+4	+1
s3	0	+1	0	0
s4	0	+1	+2.5	+2.5

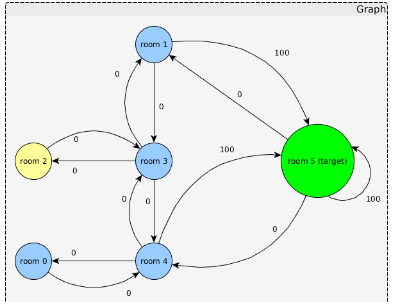
```
initialize Q[s,a] arbitrarily
observe initial state s
repeat
        select action a
        observe reward r and new state s'
        Q[s,a]=r + γ max(Q[s',a'])
        s=s'
until end
```



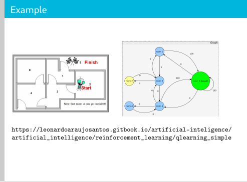
This table associates a reward (Q) for each couple action state it is randomly initialized. Then at every exploration it is updated with the Q function: Q function is recursively updated following this rule. When the space of action becomes large, updating this table can become intractable—this relates to the scalability issue

Example





https://leonardoaraujosantos.gitbook.io/artificial-inteligence/artificial_intelligence/reinforcement_learning/qlearning_simple



Actions: 0,1,2,3,4,5 States: 0,1,2,3,4,5 Rewards:0,100

Example

Step 1: state 1, random action 5

$$Q=\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$R=\begin{array}{c|cccccc} \text{State} \backslash \text{Action} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & \boxed{0} & -1 & \boxed{100} \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

$$R=\begin{array}{c|cccccc} \text{State} \backslash \text{Action} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & \boxed{0} & -1 & -1 & \boxed{0} & \boxed{100} \end{array}$$

$$Q=\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 100 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

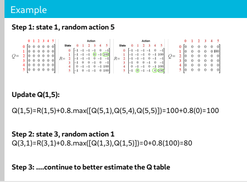
Update Q(1,5):

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

Step 3: ....continue to better estimate the Q table


Example
Step 1: state 1, random action 5
Update Q(1,5):
Q(1,5)=R(1,5)+0.8.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.max([Q(1,3),Q(1,5)])=0+0.8(100)=80
Step 3: ....continue to better estimate the Q table

The agent does not know the matrix of reward: it will learn it by experience by updating the Q matrix. At the beginning of the exploration all Q are zero. Let's consider our initial state Room 1 and $\gamma = 0.8$... Different steps, or episodes are needed to estimate the Q table.

Approximated Q-learning

Definition

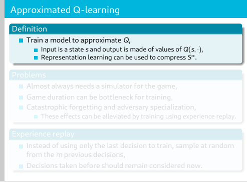
- Train a model to approximate Q,
  - Input is a state s and output is made of values of $Q(s, \cdot)$,
  - Representation learning can be used to compress S^{α}.

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.



GL: Estimating the Q table becomes soon intractable, even for simple games! In the majority of RL application we learn a model that, given state and action approximate the Q function. AlphaGo,.. Experience Replay is one of the tricks that researchers have discovered to make (Deep) RL efficient and stable: by randomly sampling a batch of experiences from a buffer/memory, we avoid catastrophic forgetting (last experience cancel impact of first ones) and correlation between series of consequent experiences.

Approximated Q-learning

Definition

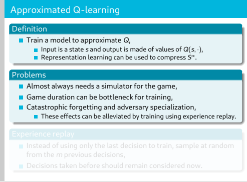
- Train a model to approximate Q,
  - Input is a state s and output is made of values of $Q(s, \cdot)$,
  - Representation learning can be used to compress S^{α}.

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.

GL: Estimating the Q table becomes soon intractable, even for simple games! In the majority of RL application we learn a model that, given state and action approximate the Q function. AlphaGo,.. Experience Replay is one of the tricks that researchers have discovered to make (Deep) RL efficient and stable: by randomly sampling a batch of experiences from a buffer/memory, we avoid catastrophic forgetting (last experience cancel impact of first ones) and correlation between series of consequent experiences.

Approximated Q-learning

Definition

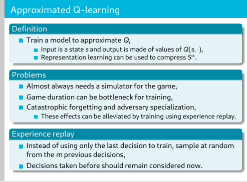
- Train a model to approximate Q,
  - Input is a state s and output is made of values of $Q(s, \cdot)$,
  - Representation learning can be used to compress S^{α}.

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.



GL: Estimating the Q table becomes soon intractable, even for simple games! In the majority of RL application we learn a model that, given state and action approximate the Q function. AlphaGo,.. Experience Replay is one of the tricks that researchers have discovered to make (Deep) RL efficient and stable: by randomly sampling a batch of experiences from a buffer/memory, we avoid catastrophic forgetting (last experience cancel impact of first ones) and correlation between series of consequent experiences.

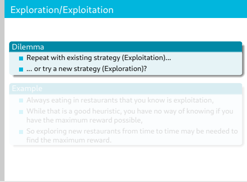
Exploration/Exploitation

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.



Exploration/Exploitation
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.

GL we have seen that in RL algorithm we get to know the best possible actions by exploring the world, by playing the game.. Q-table is filled by randomly choosing actions, then when we get a good estimation we can be more confident in our policy, and repeat it to get max reward. This describes well the exploitation/exploration dilemma: usually at the beginning we do more exploration, then more exploitation: and it s generally a good trick to have a SOFT policy

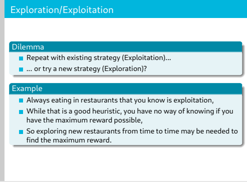
Exploration/Exploitation

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.


Exploration/Exploitation
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.

GL we have seen that in RL algorithm we get to know the best possible actions by exploring the world, by playing the game.. Q-table is filled by randomly choosing actions, then when we get a good estimation we can be more confident in our policy, and repeat it to get max reward. This describes well the exploitation/exploration dilemma: usually at the beginning we do more exploration, then more exploitation: and it s generally a good trick to have a SOFT policy

Lab Session 5

TP4 - PyRat with reinforcement learning

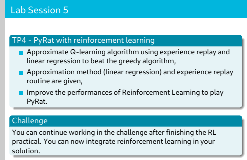
- Approximate Q-learning algorithm using experience replay and linear regression to beat the greedy algorithm,
- Approximation method (linear regression) and experience replay routine are given,
- Improve the performances of Reinforcement Learning to play PyRat.

Challenge

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

2023-11-23

Course 5: Reinforcement Learning
└Q-learning
  └Exploration/Exploitation
    └Lab Session 5


Lab Session 5

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.