Course 5: Reinforcement Learning

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

2018-11-27

Course 5: Reinforcement Learning

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.

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

2018-11-27
Course 5: Reinforcement Learning
└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

1 Definitions of Reinforcement Learning (RL)

- Fundamentals
- Example: PyRat
- Policy and values

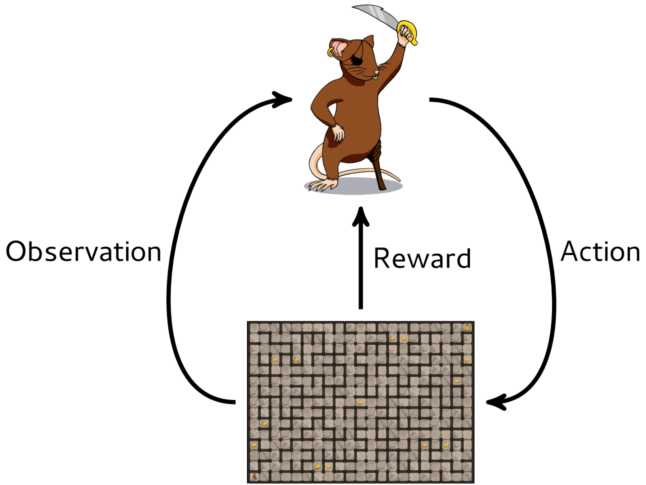
2 Q-learning

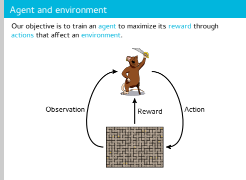
- Q-learning definitions
- Example
- Approximate Q-learning
- Exploration/Exploitation

Plan
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.





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)

Fundamentals

Reward hypothesis

- *All goals can be described by the maximization of excepted 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 excepted 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.

Fundamentals

Reward hypothesis

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

Specificities of reinforcement learning

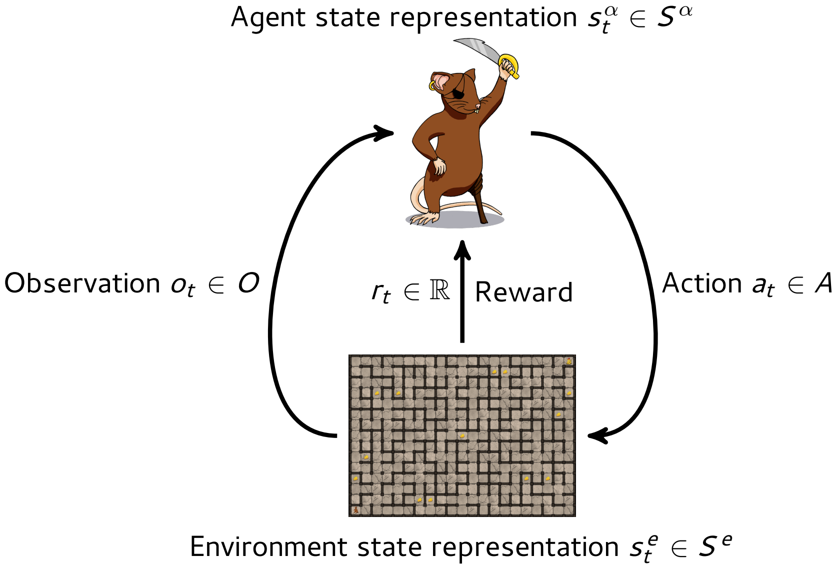
- No supervision, only a reward signal,
- Delayed feedback, the reward can come (much) later,
- Importance of the temporal dimension,
- Agent's actions affect the subsequent data it receives.

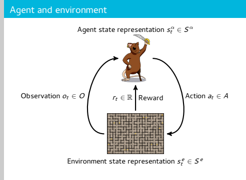
Fundamentals
Reward hypothesis
All goals can be described by the maximization of excepted cumulated reward over time.
Specificities of reinforcement learning
No supervision, only a reward signal,
Delayed feedback, the reward can come (much) later,
Importance of the temporal dimension,
Agent's actions 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.

Agent and environment

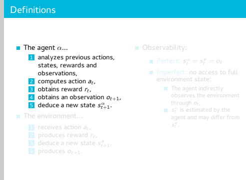




Now we can finally put some notations on Pyrat in the context of Reinforcement Learning. At this stage, it is important to note that the state of the rat and the state of the environment are not the same.

- 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 descibe 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 descibe 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… analyzes previous actions, states, rewards and observations, computes action a_t, obtains reward r_t, obtains an observation o_{t+1}, deduce a new state s^{α}_{t+1}.
The environment… receives action a_t, produces reward r_t, deduce a new state s^{e}_{t+1}, produces o_{t+1}.
Observability: Perfect: $s^{\alpha}_t = s^{e}_t = o_t$

And now we descibe 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.

- 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 descibe 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 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.

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.

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.

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:

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

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

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.

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:

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

- The value of a policy function is thus an expectation of cumulative future rewards, weakened by the geometrical coefficient γ to avoid divergence,
- 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.

Definition

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

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

- The value of a policy function is thus an expectation of cumulative future rewards, weakened by the geometrical coefficient γ to avoid divergence,
- 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:
$$v^\pi : \begin{cases} S^\alpha & \to \mathbb{R} \\ s_{t_0}^{\alpha,\pi} & \mapsto \displaystyle\sum_{t=t_0}^{+\infty} \gamma^{t-t_0} r_t \end{cases}$$
- The value of a policy function is thus an expectation of cumulative future rewards, weakened by the geometrical coefficient γ to avoid divergence,
- 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.

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

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

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,

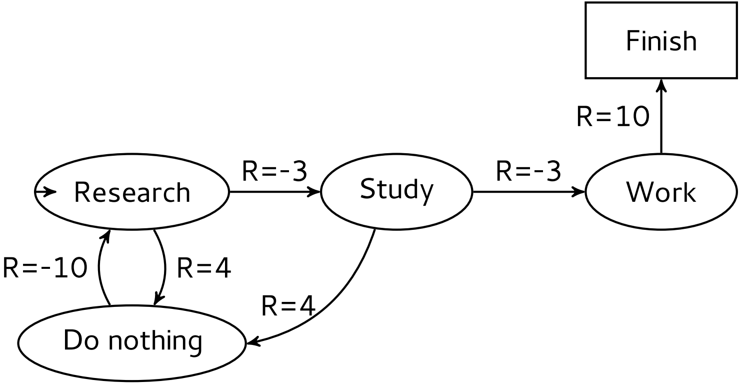
Cons:

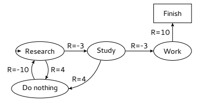
- Scalability issues when S^α is large.

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 example

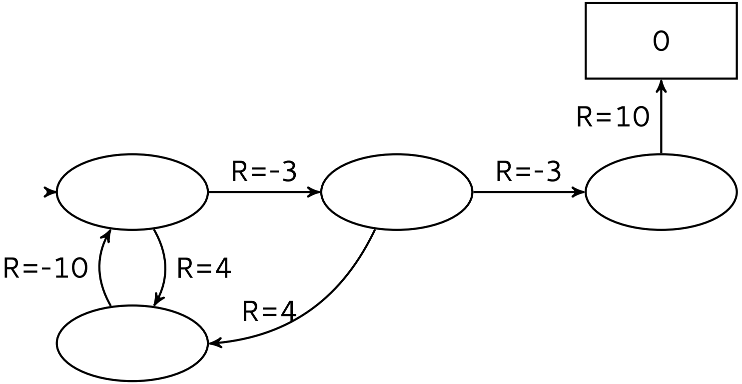


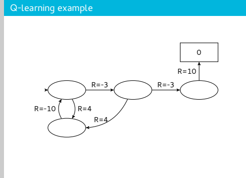


Here is an example of what could be said on this example.
Let's imagine we have the following situation, which is a typical state-space model about studying to finish a work. This is hard, so it is broken down in several steps, but as each step is itself hard, they are associated with negative rewards. However, going back to the state in which we do nothing is "nice", so it is associated with a positive reward. Let's see how to apply Q-learning from this example.
To simplify things, let's assume we start from the end position, and let's set a max Q-value of 0 to this state.
Let's note Q(s,s') where s' is the next state. Inside each state, we will note the max Q-value in this state (ie the maximum value that can be obtained throughout possible actions reaching this state).
From the final state we get $Q(10, 0) = R + Q_{max}(0) = 10 + 0 = 10$
and the following steps iteratively.
$Q(7, 10) = R + Q_{max}(10) = -3 + 10 = 7$
$Q(4, 7) = R + Q_{max}(7) = -3 + 7 = 4$
$Q(-6, 4) = R + Q_{max}(4) = -10 + 4 = -6$
$Q(4, -6) = R + Q_{max}(-6) = 4 - 6 = -2$
$Q(7, -6) = R + Q_{max}(-6) = 4 - 6 = -2$

In the end, we just have to follow the path that maximises the Q value.

Q-learning example

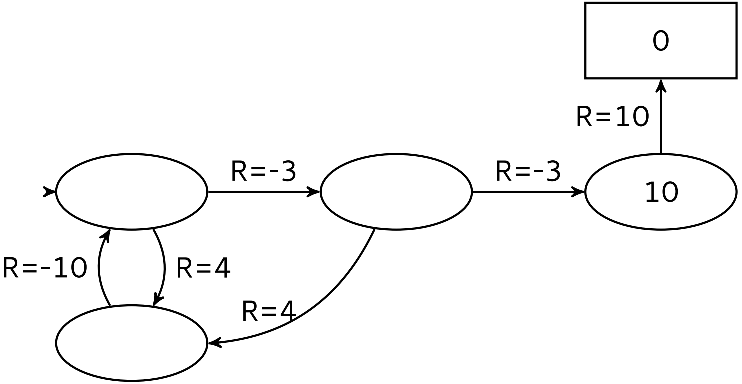


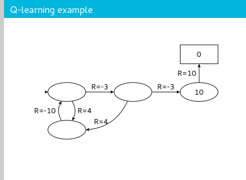


Here is an example of what could be said on this example.
Let's imagine we have the following situation, which is a typical state-space model about studying to finish a work. This is hard, so it is broken down in several steps, but as each step is itself hard, they are associated with negative rewards. However, going back to the state in which we do nothing is "nice", so it is associated with a positive reward. Let's see how to apply Q-learning from this example.
To simplify things, let's assume we start from the end position, and let's set a max Q-value of 0 to this state.
Let's note Q(s,s') where s' is the next state. Inside each state, we will note the max Q-value in this state (ie the maximum value that can be obtained throughout possible actions reaching this state).
From the final state we get $Q(10, 0) = R + Q_{max}(0) = 10 + 0 = 10$
and the following steps iteratively.
$Q(7, 10) = R + Q_{max}(10) = -3 + 10 = 7$
$Q(4, 7) = R + Q_{max}(7) = -3 + 7 = 4$
$Q(-6, 4) = R + Q_{max}(4) = -10 + 4 = -6$
$Q(4, -6) = R + Q_{max}(-6) = 4 - 6 = -2$
$Q(7, -6) = R + Q_{max}(-6) = 4 - 6 = -2$

In the end, we just have to follow the path that maximises the Q value.

Q-learning example

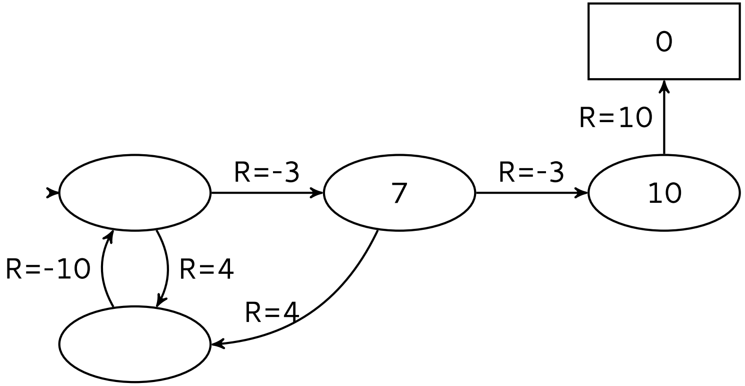


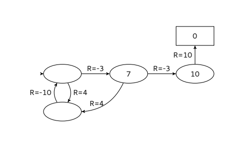


Here is an example of what could be said on this example.
Let's imagine we have the following situation, which is a typical state-space model about studying to finish a work. This is hard, so it is broken down in several steps, but as each step is itself hard, they are associated with negative rewards. However, going back to the state in which we do nothing is "nice", so it is associated with a positive reward. Let's see how to apply Q-learning from this example.
To simplify things, let's assume we start from the end position, and let's set a max Q-value of 0 to this state.
Let's note Q(s,s') where s' is the next state. Inside each state, we will note the max Q-value in this state (ie the maximum value that can be obtained throughout possible actions reaching this state).
From the final state we get $Q(10,0) = R + Q_{max}(0) = 10 + 0 = 10$
and the following steps iteratively.
$Q(7,10) = R + Q_{max}(10) = -3 + 10 = 7$
$Q(4,7) = R + Q_{max}(7) = -3 + 7 = 4$
$Q(-6,4) = R + Q_{max}(4) = -10 + 4 = -6$
$Q(4,-6) = R + Q_{max}(-6) = 4 - 6 = -2$
$Q(7,-6) = R + Q_{max}(-6) = 4 - 6 = -2$

In the end, we just have to follow the path that maximises the Q value.

Q-learning example

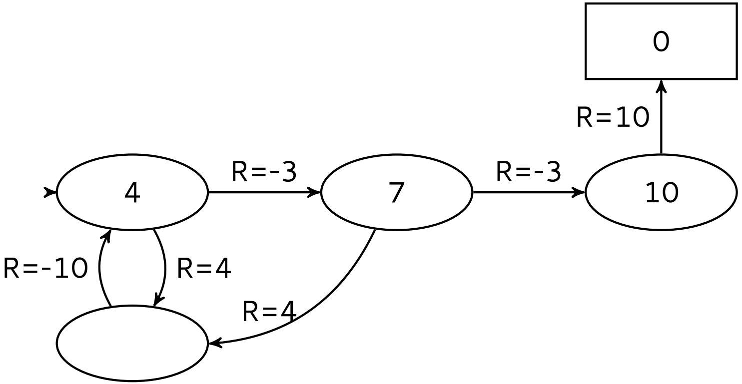




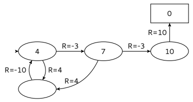
Here is an example of what could be said on this example.
Let's imagine we have the following situation, which is a typical state-space model about studying to finish a work. This is hard, so it is broken down in several steps, but as each step is itself hard, they are associated with negative rewards. However, going back to the state in which we do nothing is "nice", so it is associated with a positive reward. Let's see how to apply Q-learning from this example.
To simplify things, let's assume we start from the end position, and let's set a max Q-value of 0 to this state.
Let's note Q(s,s') where s' is the next state. Inside each state, we will note the max Q-value in this state (ie the maximum value that can be obtained throughout possible actions reaching this state).
From the final state we get $Q(10, 0) = R + Q_{max}(0) = 10 + 0 = 10$
and the following steps iteratively.
$Q(7, 10) = R + Q_{max}(10) = -3 + 10 = 7$
$Q(4, 7) = R + Q_{max}(7) = -3 + 7 = 4$
$Q(-6, 4) = R + Q_{max}(4) = -10 + 4 = -6$
$Q(4, -6) = R + Q_{max}(-6) = 4 - 6 = -2$
$Q(7, -6) = R + Q_{max}(-6) = 4 - 6 = -2$

In the end, we just have to follow the path that maximises the Q value.

Q-learning example



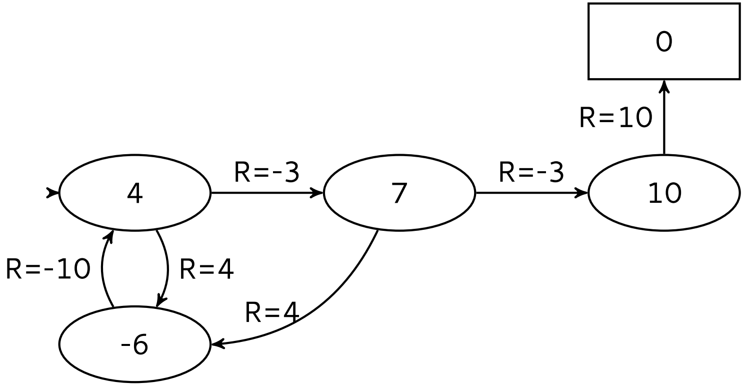
Q-learning example

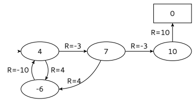


Here is an example of what could be said on this example.
Let's imagine we have the following situation, which is a typical state-space model about studying to finish a work. This is hard, so it is broken down in several steps, but as each step is itself hard, they are associated with negative rewards. However, going back to the state in which we do nothing is "nice", so it is associated with a positive reward. Let's see how to apply Q-learning from this example.
To simplify things, let's assume we start from the end position, and let's set a max Q-value of 0 to this state.
Let's note Q(s,s') where s' is the next state. Inside each state, we will note the max Q-value in this state (ie the maximum value that can be obtained throughout possible actions reaching this state).
From the final state we get $Q(10,0) = R + Q_{max}(0) = 10 + 0 = 10$
and the following steps iteratively.
$Q(7,10) = R + Q_{max}(10) = -3 + 10 = 7$
$Q(4,7) = R + Q_{max}(7) = -3 + 7 = 4$
$Q(-6,4) = R + Q_{max}(4) = -10 + 4 = -6$
$Q(4,-6) = R + Q_{max}(-6) = 4 - 6 = -2$
$Q(7,-6) = R + Q_{max}(-6) = 4 - 6 = -2$

In the end, we just have to follow the path that maximises the Q value.

Q-learning example

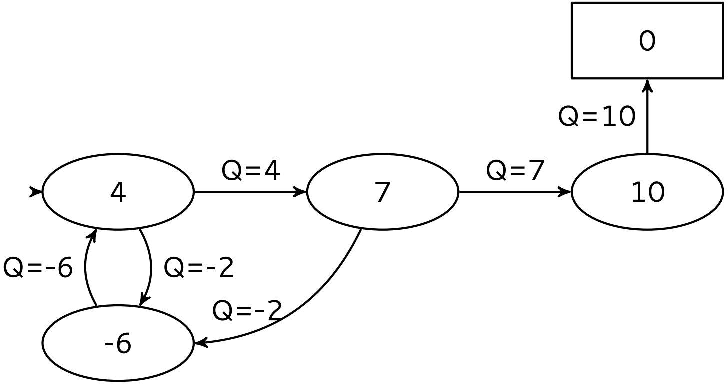




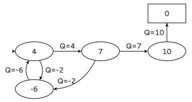
Here is an example of what could be said on this example.
Let's imagine we have the following situation, which is a typical state-space model about studying to finish a work. This is hard, so it is broken down in several steps, but as each step is itself hard, they are associated with negative rewards. However, going back to the state in which we do nothing is "nice", so it is associated with a positive reward. Let's see how to apply Q-learning from this example.
To simplify things, let's assume we start from the end position, and let's set a max Q-value of 0 to this state.
Let's note Q(s,s') where s' is the next state. Inside each state, we will note the max Q-value in this state (ie the maximum value that can be obtained throughout possible actions reaching this state).
From the final state we get $Q(10, 0) = R + Q_{max}(0) = 10 + 0 = 10$
and the following steps iteratively.
$Q(7, 10) = R + Q_{max}(10) = -3 + 10 = 7$
$Q(4, 7) = R + Q_{max}(7) = -3 + 7 = 4$
$Q(-6, 4) = R + Q_{max}(4) = -10 + 4 = -6$
$Q(4, -6) = R + Q_{max}(-6) = 4 - 6 = -2$
$Q(7, -6) = R + Q_{max}(-6) = 4 - 6 = -2$

In the end, we just have to follow the path that maximises the Q value.

Q-learning example



Q-learning example



Here is an example of what could be said on this example.
Let's imagine we have the following situation, which is a typical state-space model about studying to finish a work. This is hard, so it is broken down in several steps, but as each step is itself hard, they are associated with negative rewards. However, going back to the state in which we do nothing is "nice", so it is associated with a positive reward. Let's see how to apply Q-learning from this example.
To simplify things, let's assume we start from the end position, and let's set a max Q-value of 0 to this state.
Let's note Q(s,s') where s' is the next state. Inside each state, we will note the max Q-value in this state (ie the maximum value that can be obtained throughout possible actions reaching this state).
From the final state we get $Q(10,0) = R + Q_{max}(0) = 10 + 0 = 10$
and the following steps iteratively.
$Q(7,10) = R + Q_{max}(10) = -3 + 10 = 7$
$Q(4,7) = R + Q_{max}(7) = -3 + 7 = 4$
$Q(-6,4) = R + Q_{max}(4) = -10 + 4 = -6$
$Q(4,-6) = R + Q_{max}(-6) = 4 - 6 = -2$
$Q(7,-6) = R + Q_{max}(-6) = 4 - 6 = -2$

In the end, we just have to follow the path that maximises the Q value.

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 alliviated 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 alliviated 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 alliviated 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.

2018-11-27

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 alliviated 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 alliviated 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 alliviated 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.

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.

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.

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,
- Assemble all the primitives to perform Reinforcement Learning.

Challenge

You can continue working in the challenge after finishing TP4. You can now integrate reinforcement learning in your solution.

2018-11-27

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,
- Assemble all the primitives to perform Reinforcement Learning.
Challenge
You can continue working in the challenge after finishing TP4. You can now integrate reinforcement learning in your solution.