Reinforcement Learning I
Introduction on Reinforcement Learning

Antoine SYLVAIN

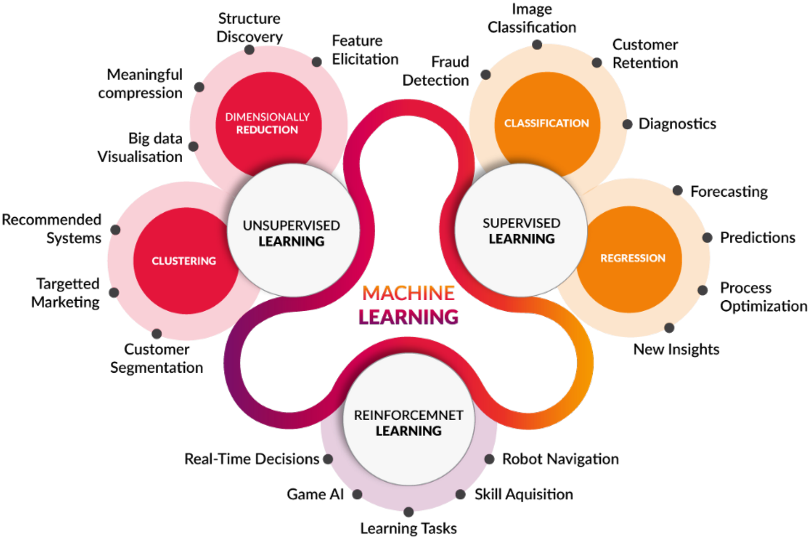
EPITA

2021

1. What is Reinforcement Learning ?
2. Markov Decision Processes
3. Bellman Equation
4. Explore or exploit ?

Contents

RL within Machine Learning



RL within Machine Learning

- **Supervised Learning** : Learns by using labelled data. Calculates outcomes.

RL within Machine Learning

- **Supervised Learning** : Learns by using labelled data. Calculates outcomes.
- **Unsupervised Learning** : Trained with unlabelled data, without any guidance. Discover underlying patterns.

RL within Machine Learning

- **Supervised Learning** : Learns by using labelled data. Calculates outcomes.
- **Unsupervised Learning** : Trained with unlabelled data, without any guidance. Discover underlying patterns.
- **Reinforcement Learning** : Works on interacting with the environment. Learn a series of actions.

Applications: Self-driving car

Applications: Self-driving car

- Trajectory optimization
- Dynamic pathing
- Lane changing
- Learning automatic parking policies
- Learning an overtaking policy to avoid collision

Applications: Finance and trading

Applications: Finance and trading

- Hold
- Buy
- Sell

Applications: Games

Applications: Games

- Backgammon
- Chess
- Atari Games
- Go

Other applications

- Healthcare
- Industrial robotics
- News recommendations
- Online advertising
- ...

Other applications

- Healthcare
- Industrial robotics
- News recommendations
- Online advertising
- ...

New applications are popping almost everywhere !

Jobs in RL: positions

- Academic research
- Private research (mainly big tech companies)
- Start-ups

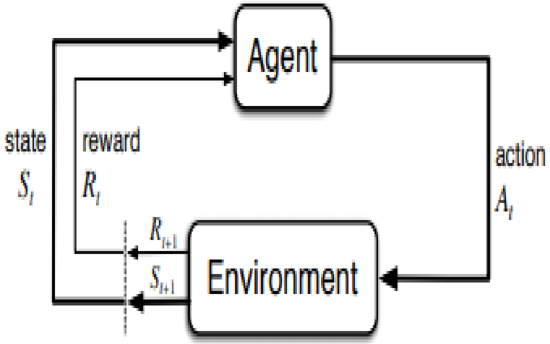
Contents

Formal definition

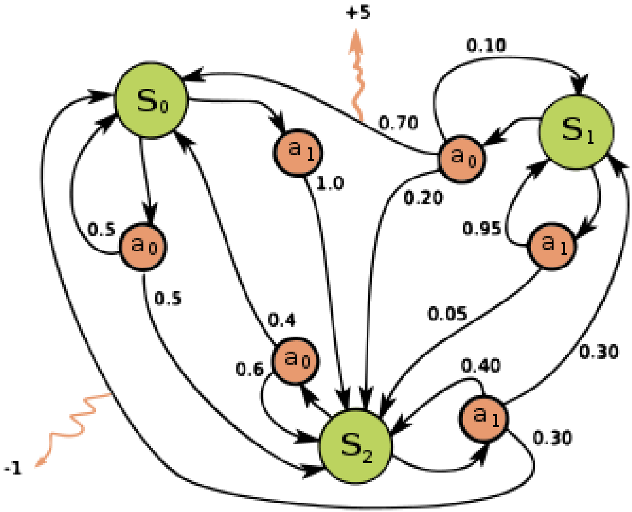
A Markov decision process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ with:

- $\mathcal{S}$: the state space
- $\mathcal{A}$: the action space
- $\mathcal{P}(s, a, s')$: the probability that action a in state s will lead to state s'
- $\mathcal{R}(s, s')$: the reward received after passing from state s to state s'

Agent-Environment Interaction in a MDP



Example of a Simple MDP



Finite MDP

- Here, we are talking about *finite* MDP
- It implies that set $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ of states, actions and rewards have a finite number of elements
- Particular values of these sets have a probability to occur at a given time t, given particular values of the preceding state and action

MDP Cycle

- At $t = 0$, environment samples initial state $s_0 \sim p(s_0)$

MDP Cycle

- At $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- for $t = 0 \rightarrow$ done:

MDP Cycle

- At $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- for $t = 0 \rightarrow$ done:
  - Agent selects an action a_t

MDP Cycle

- At $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- for $t = 0 \rightarrow$ done:
  - Agent selects an action a_t
  - Environment samples a reward $r_t \sim \mathcal{R}(.|s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim \mathcal{P}(.|s_t, a_t)$

MDP Cycle

- At $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- for $t = 0 \rightarrow$ done:
    - Agent selects an action a_t
    - Environment samples a reward $r_t \sim \mathcal{R}(.|s_t, a_t)$
    - Environment samples next state $s_{t+1} \sim \mathcal{P}(.|s_t, a_t)$
    - Agent receives the reward r_t and next state s_{t+1}

MDP Cycle

- At $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- for $t = 0 \to$ done:
  - Agent selects an action a_t
  - Environment samples a reward $r_t \sim \mathcal{R}(.|s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim \mathcal{P}(.|s_t, a_t)$
  - Agent receives the reward r_t and next state s_{t+1}
- The objective is to find a policy that maximizes cumulative discounted reward $\sum_{t \geq 0} \gamma^t r_t$

Elements of Reinforcement Learning

Between the agent and its environment, we distinguish four main elements in an RL system:

Elements of Reinforcement Learning

Between the agent and its environment, we distinguish four main elements in an RL system:

- **Policy** : agent's way of behaving at a given time

Elements of Reinforcement Learning

Between the agent and its environment, we distinguish four main elements in an RL system:

- **Policy** : agent's way of behaving at a given time
- **Reward signal** : numerical signal defining if the events are good or bad for the agent

Elements of Reinforcement Learning

Between the agent and its environment, we distinguish four main elements in an RL system:

- **Policy** : agent's way of behaving at a given time
- **Reward signal** : numerical signal defining if the events are good or bad for the agent
- **Value function** : long-term rewards anticipation

Elements of Reinforcement Learning

Between the agent and its environment, we distinguish four main elements in an RL system:

- **Policy** : agent's way of behaving at a given time
- **Reward signal** : numerical signal defining if the events are good or bad for the agent
- **Value function** : long-term rewards anticipation
- **Model of the environment** (optional) : emulation of the environment that allows to make inferences about how it will behave

Policy

- A **deterministic policy** associates an action to each state

Policy

- A **deterministic policy** associates an action to each state
- It can be represented by a table

Policy

- A **deterministic policy** associates an action to each state
- It can be represented by a table
-

s_0	a_1
s_1	a_0
s_2	a_1

Policy

- A **deterministic policy** associates an action to each state
- It can be represented by a table
s_0	a_1
s_1	a_0
s_2	a_1
- A **stochastic policy** associates various actions with a probability for each state

Policy

- A **deterministic policy** associates an action to each state
- It can be represented by a table
-

s_0	a_1
s_1	a_0
s_2	a_1

- A **stochastic policy** associates various actions with a probability for each state
- The sum of probabilities of the actions for a given state equals 1

Policy

- Deterministic policy:
  $\pi(s) = a$
- Stochastic policy:
  $\pi(a|s) = \mathbb{P}[\mathcal{A}_t = a|\mathcal{S}_t = s]$

Exercise

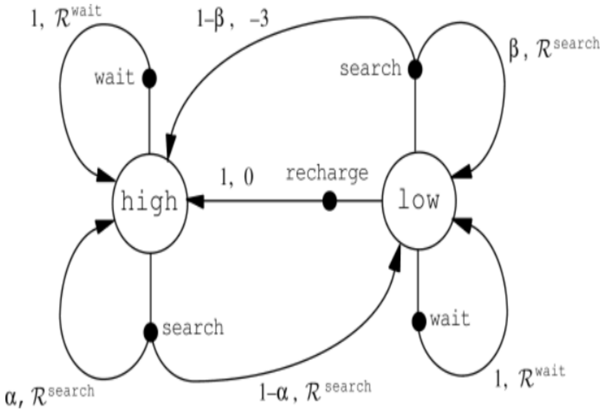
Recycling robot: the recycling robot is a robot whose mission is to gather and recycle cans. It can be in two states, high or low, depending on its battery energy level. At each time step, it has to chose between three actions: searching a can, waiting for someone to bring it a can, or recharge (only if its state is low).

- $\mathcal{S} = \{high, low\}$
- $\mathcal{A}(high) = \{search, wait\}$
- $\mathcal{A}(low) = \{search, wait, recharge\}$

Exercise

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
high	high	search	α	$\mathcal{R}^{\texttt{search}}$
high	low	search	$1 - \alpha$	$\mathcal{R}^{\texttt{search}}$
low	high	search	$1 - \beta$	-3
low	low	search	β	$\mathcal{R}^{\texttt{search}}$
high	high	wait	1	$\mathcal{R}^{\texttt{wait}}$
high	low	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	high	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	low	wait	1	$\mathcal{R}^{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

Solution


1, $\mathcal{R}^{\text{wait}}$
wait
$1-\beta$, -3
search
β, $\mathcal{R}^{\text{search}}$
high
1, 0
recharge
low
search
wait
α, $\mathcal{R}^{\text{search}}$
$1-\alpha$, $\mathcal{R}^{\text{search}}$
1, $\mathcal{R}^{\text{wait}}$

State-value function

- $v_{\pi}(s) = \mathbb{E}_{\pi}[\mathcal{R}_{t+1} + \gamma\mathcal{R}_{t+2} + \gamma^2\mathcal{R}_{t+3} + ...|\mathcal{S}_t = s]$

State-value function

- $v_\pi(s) = \mathbb{E}_\pi[\mathcal{R}_{t+1} + \gamma \mathcal{R}_{t+2} + \gamma^2 \mathcal{R}_{t+3} + ... | \mathcal{S}_t = s]$
- with $\gamma \in [0, 1]$ the discount factor
- $\gamma = 0 \rightarrow$ only the first reward is taken into account
- $\gamma = 1 \rightarrow$ every future reward has the same weight (may not converge)

State-value function

- $v_\pi(s) \doteq \mathbb{E}_\pi[\sum_{t\geq 0} \gamma^k \mathcal{R}_{t+k+1} | \mathcal{S}_t = s]$

State-value function

- $v_\pi(s) \doteq \mathbb{E}_\pi[\sum_{t\geq 0} \gamma^k \mathcal{R}_{t+k+1} | \mathcal{S}_t = s]$
- $v_\pi(s) \doteq \mathbb{E}_\pi[\mathcal{G}_t | \mathcal{S}_t = s]$

Action-value function

- $q_\pi(s, a) \doteq \mathbb{E}_\pi[\mathcal{G}_t | \mathcal{S}_t = s, \mathcal{A}_t = a]$

Function p

- $p(s', r|s, a) \doteq \Pr\{\mathcal{S}_t = s', \mathcal{R}_t = r|\mathcal{S}_{t-1} = s, \mathcal{A}_{t-1} = a\}$

Function p

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1 \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

Contents

State-value Bellman Equation

- $v_\pi(s) \doteq \mathbb{E}[\mathcal{G}_t | \mathcal{S}_t = s]$

State-value Bellman Equation

- $v_\pi(s) \doteq \mathbb{E}[\mathcal{G}_t | \mathcal{S}_t = s]$
- $v_\pi(s) = \mathbb{E}[\mathcal{R}_{t+1} + \gamma \mathcal{G}_{t+1} | \mathcal{S}_t = s]$

State-value Bellman Equation

- $v_\pi(s) \doteq \mathbb{E}[\mathcal{G}_t | \mathcal{S}_t = s]$
- $v_\pi(s) = \mathbb{E}[\mathcal{R}_{t+1} + \gamma \mathcal{G}_{t+1} | \mathcal{S}_t = s]$
- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma \mathbb{E}[\mathcal{G}_{t+1} | \mathcal{S}_{t+1} = s']]$

State-value Bellman Equation

- $v_\pi(s) \doteq \mathbb{E}[\mathcal{G}_t | \mathcal{S}_t = s]$
- $v_\pi(s) = \mathbb{E}[\mathcal{R}_{t+1} + \gamma \mathcal{G}_{t+1} | \mathcal{S}_t = s]$
- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}[\mathcal{G}_{t+1} | \mathcal{S}_{t+1} = s']]$
- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$

State-value Bellman Equation

The current time-step's state values can be written recursively in terms of future values

Action-value Bellman Equation

- $q_\pi(s, a) \doteq \mathbb{E}[\mathcal{G}_t | \mathcal{S}_t = s, \mathcal{A}_t = a]$

Action-value Bellman Equation

- $q_\pi(s, a) \doteq \mathbb{E}[\mathcal{G}_t | \mathcal{S}_t = s, \mathcal{A}_t = a]$
- $q_\pi(s, a) = \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma \mathbb{E}[\mathcal{G}_{t+1} | \mathcal{S}_{t+1} = s']]$

Action-value Bellman Equation

- $q_\pi(s,a) \doteq \mathbb{E}[\mathcal{G}_t|\mathcal{S}_t = s, \mathcal{A}_t = a]$
- $q_\pi(s,a) = \sum_{s'} \sum_r p(s', r|s,a)[r + \gamma \mathbb{E}[\mathcal{G}_{t+1}|\mathcal{S}_{t+1} = s']]$
- $q_\pi(s,a) = \sum_{s'} \sum_r p(s', r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') \mathbb{E}[\mathcal{G}_{t+1}|\mathcal{S}_{t+1} = s', \mathcal{A}_{t+1} = a']$

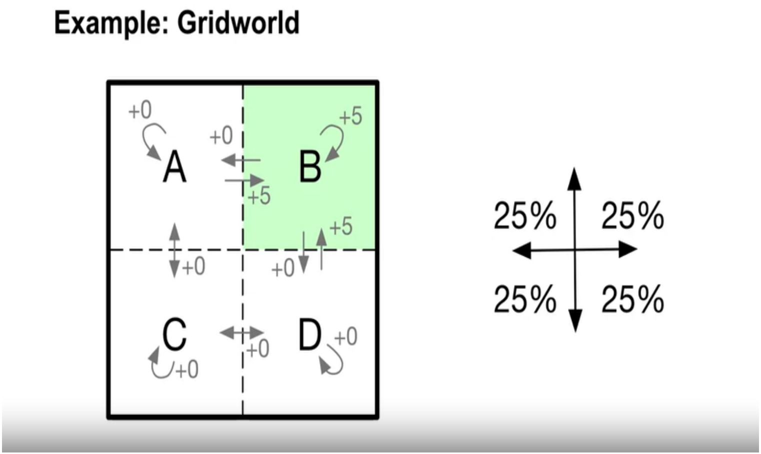
Action-value Bellman Equation

- $q_\pi(s,a) \doteq \mathbb{E}[\mathcal{G}_t|\mathcal{S}_t = s, \mathcal{A}_t = a]$
- $q_\pi(s,a) = \sum_{s'}\sum_r p(s',r|s,a)[r + \gamma\mathbb{E}[\mathcal{G}_{t+1}|\mathcal{S}_{t+1} = s']]$
- $q_\pi(s,a) = \sum_{s'}\sum_r p(s',r|s,a)[r + \gamma\sum_{a'}\pi(a'|s')\mathbb{E}[\mathcal{G}_{t+1}|\mathcal{S}_{t+1} = s', \mathcal{A}_{t+1} = a']$
- $q_\pi(s,a) = \sum_{s'}\sum_r p(s',r|s,a)[r + \gamma\sum_{a'}\pi(a'|s')q_\pi(s',a')]$

Action-value Bellman Equation

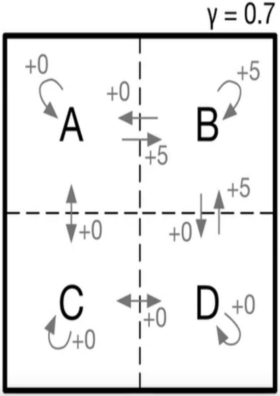
The current time-step's action values can be written recursively in terms of future values

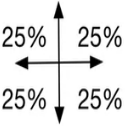
Gridworld example



Gridworld example

Example: Gridworld





$$V_\pi(A) \doteq \mathbb{E}_\pi[G_t \mid S_t = A]$$

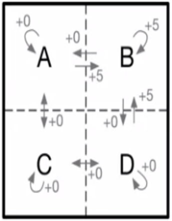
$$V_\pi(B) \doteq \mathbb{E}_\pi[G_t \mid S_t = B]$$

$$V_\pi(C) \doteq \mathbb{E}_\pi[G_t \mid S_t = C]$$

$$V_\pi(D) \doteq \mathbb{E}_\pi[G_t \mid S_t = D]$$

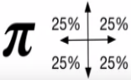
Gridworld example

Example: Gridworld



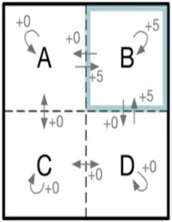
$$V_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(s',r|s,a)[r + \gamma V_\pi(s')]$$

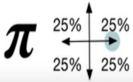
$$V_\pi(A) = \sum_a \pi(a|A)\left(r + 0.7V_\pi(s')\right)$$



Gridworld example

Example: Gridworld





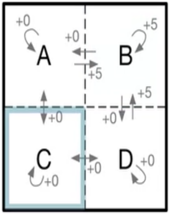
$$V_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(s', r | s, a)[r + \gamma V_\pi(s')]$$

$$V_\pi(A) = \sum_a \pi(a|A)\left(r + 0.7 V_\pi(s')\right)$$

$$V_\pi(A) = \frac{1}{4}(5 + 0.7 V_\pi(B))$$

Gridworld example

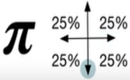
Example: Gridworld



$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_r \sum_{s'} p(s', r \mid s, a)[r + \gamma V_\pi(s')]$$

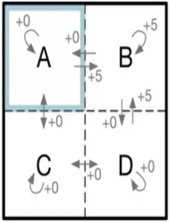
$$V_\pi(A) = \sum_a \pi(a \mid A)\left(r + 0.7 V_\pi(s')\right)$$

$$V_\pi(A) = \frac{1}{4}(5 + 0.7 V_\pi(B)) + \frac{1}{4} 0.7 V_\pi(C)$$



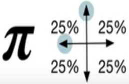
Gridworld example

Example: Gridworld



$$V_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(s',r|s,a)[r+\gamma V_\pi(s')]$$

$$V_\pi(A) = \sum_a \pi(a|A)\left(r + 0.7V_\pi(s')\right)$$

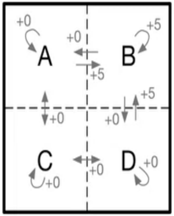


$$V_\pi(A) = \frac{1}{4}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(A)$$

Gridworld example

Example: Gridworld

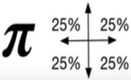
$$V_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(s', r | s, a)[r + \gamma V_\pi(s')]$$



$$V_\pi(A) = \frac{1}{4}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(A)$$

$$V_\pi(B) = \frac{1}{2}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D)$$

$$V_\pi(C) = \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) + \frac{1}{2}0.7V_\pi(C)$$

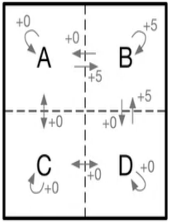


$$V_\pi(D) = \frac{1}{4}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(D)$$

Gridworld example

Example: Gridworld

$$V_\pi(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(s', r | s, a)[r + \gamma V_\pi(s')]$$

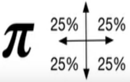


$V_\pi(A) = 4.2$

$V_\pi(B) = 6.1$

$V_\pi(C) = 2.2$

$V_\pi(D) = 4.2$



Conclusion on Bellman Equation

- Bellman Equation reduces an infinite sum over possible future to a simple linear algebra problem

Conclusion on Bellman Equation

- Bellman Equation reduces an infinite sum over possible future to a simple linear algebra problem
- It provides a powerful general relationship for MDPs

Conclusion on Bellman Equation

- Bellman Equation reduces an infinite sum over possible future to a simple linear algebra problem
- It provides a powerful general relationship for MDPs
- But...

Conclusion on Bellman Equation

- Bellman Equation reduces an infinite sum over possible future to a simple linear algebra problem
- It provides a powerful general relationship for MDPs
- But...
- Chess, for example, has over 10^{45} possible states, and Go, over 10^{170}

Conclusion on Bellman Equation

- Bellman Equation reduces an infinite sum over possible future to a simple linear algebra problem
- It provides a powerful general relationship for MDPs
- But...
- Chess, for example, has over 10^{45} possible states, and Go, over 10^{170}
- We cannot evaluate Bellman Equation for every state value

Conclusion on Bellman Equation

- Bellman Equation reduces an infinite sum over possible future to a simple linear algebra problem
- It provides a powerful general relationship for MDPs
- But...
- Chess, for example, has over 10^{45} possible states, and Go, over 10^{170}
- We cannot evaluate Bellman Equation for every state value
- Nonetheless, humans can learn to play chess or go pretty well

Contents

Exploration vs. Exploitation

- Reinforcement learning is like a trial-and-error learning

Exploration vs. Exploitation

- Reinforcement learning is like a trial-and-error learning
- Agent should discover a good policy from its experience of the environment

Exploration vs. Exploitation

- Reinforcement learning is like a trial-and-error learning
- Agent should discover a good policy from its experience of the environment
- It does not want to lose much reward along the way

Exploration vs. Exploitation

- How to discover high-reward strategies requiring a temporally-extended sequence of complex behaviors ?

Exploration vs. Exploitation

- How to discover high-reward strategies requiring a temporally-extended sequence of complex behaviors ?
- How the agent can decide whether trying new things or keeping doing the best it knows so far ?

Exploration vs. Exploitation

- How to discover high-reward strategies requiring a temporally-extended sequence of complex behaviors ?
- How the agent can decide whether trying new things or keeping doing the best it knows so far ?
- This is the same issue...

Exploration vs. Exploitation

- How to discover high-reward strategies requiring a temporally-extended sequence of complex behaviors ?
- How the agent can decide whether trying new things or keeping doing the best it knows so far ?
- This is the same issue...
- **Exploitation** : Doing what will give the highest rewards based on current knowledge

Exploration vs. Exploitation

- How to discover high-reward strategies requiring a temporally-extended sequence of complex behaviors ?
- How the agent can decide whether trying new things or keeping doing the best it knows so far ?
- This is the same issue...
- **Exploitation** : Doing what will give the highest rewards based on current knowledge
- **Exploration** : Trying something new, with the hope of getting even higher rewards

Exploration vs. Exploitation

- Exploration finds more information about the environment

Exploration vs. Exploitation

- Exploration finds more information about the environment
- Exploitation uses known information to maximize reward

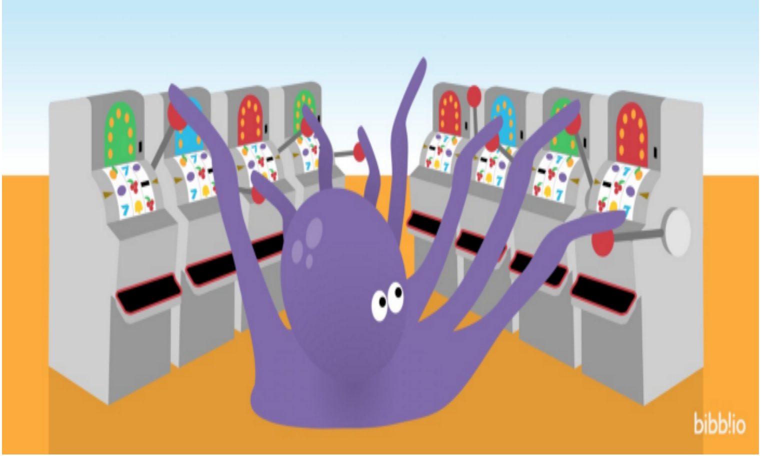
Exploration vs. Exploitation

- Exploration finds more information about the environment
- Exploitation uses known information to maximize reward
- It is usually important to explore as well as exploit

Examples

	Exploitation	**Exploration**
Restaurant	Go to your favorite restaurant	Try a new restaurant
Driving	Take the best known route	Try a new route
Ad placement	Show the most successful ad	Show a random add

k-Armed Bandit



k-Armed Bandit

- A k-armed bandit is a tuple $(\mathcal{A}, \mathcal{R})$

k-Armed Bandit

- A k-armed bandit is a tuple $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$ is a known set of k actions (or "arms")

k-Armed Bandit

- A k-armed bandit is a tuple $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$ is a known set of k actions (or "arms")
- $\mathcal{R}_a(r) = \mathbb{P}[r|a]$ is an unknown probability distribution over rewards

k-Armed Bandit

- A k-armed bandit is a tuple $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$ is a known set of k actions (or "arms")
- $\mathcal{R}_a(r) = \mathbb{P}[r|a]$ is an unknown probability distribution over rewards
- At each time step t, the agent selects an action $a_t \in \mathcal{A}$

k-Armed Bandit

- A k-armed bandit is a tuple $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$ is a known set of k actions (or "arms")
- $\mathcal{R}_a(r) = \mathbb{P}[r|a]$ is an unknown probability distribution over rewards
- At each time step t, the agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}_a^t$

k-Armed Bandit

- A k-armed bandit is a tuple $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$ is a known set of k actions (or "arms")
- $\mathcal{R}_a(r) = \mathbb{P}[r|a]$ is an unknown probability distribution over rewards
- At each time step t, the agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}_a^t$
- The aim is to maximise cumulative reward $\sum_{\tau=1}^{t} r_\tau$

Regret

- Action value: $q(a) = \mathbb{E}[r|a]$

Regret

- Action value: $q(a) = \mathbb{E}[r|a]$
- Optimal value: $V^* = q(a^*) = \max q(a)$

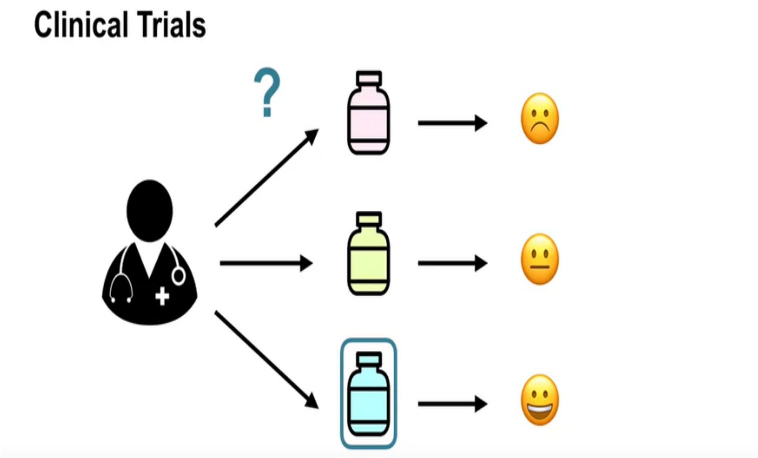
Regret

- Action value: $q(a) = \mathbb{E}[r|a]$
- Optimal value: $V^* = q(a^*) = \max q(a)$
- Regret, the opportunity lost for one step: $l_t = \mathbb{E}[V^* - q(a_t)]$

Regret

- Action value: $q(a) = \mathbb{E}[r|a]$
- Optimal value: $V^* = q(a^*) = \max q(a)$
- Regret, the opportunity lost for one step: $l_t = \mathbb{E}[V^* - q(a_t)]$
- Total regret, the total opportunity loss: $L_t = \mathbb{E}[\sum_{\tau=1}^{t} V^* - q(a_\tau)]$

Example of k-armed bandit problem



Bibliography

- Reinforcement Learning: an Introduction, R.Sutton, A.Barto, 2018
- https://deepmind.com/learning-resources

Thank you for your attention