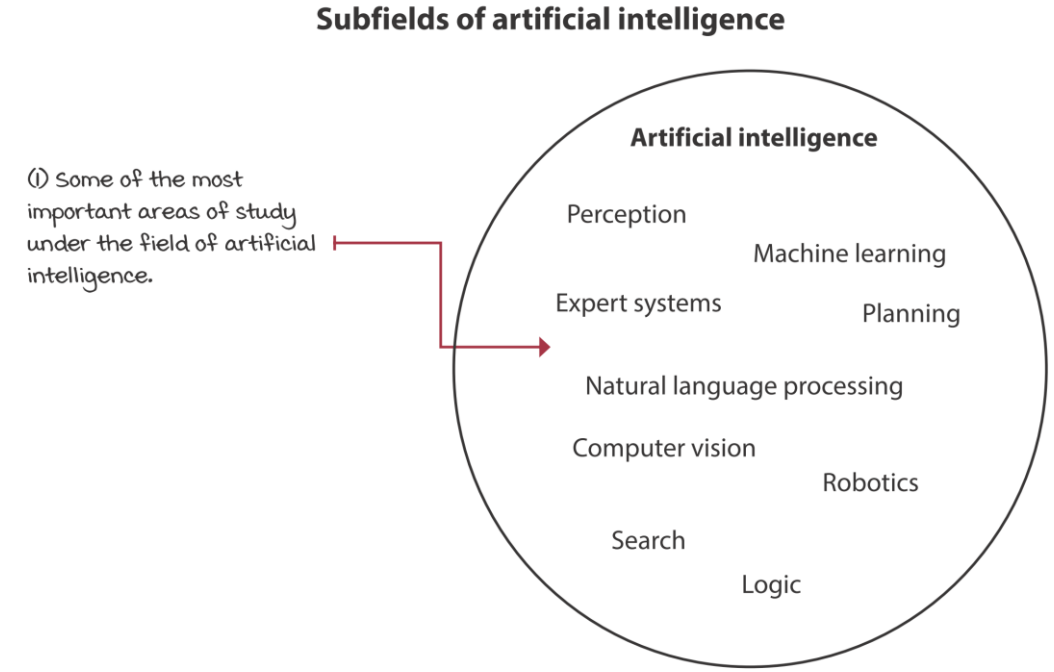
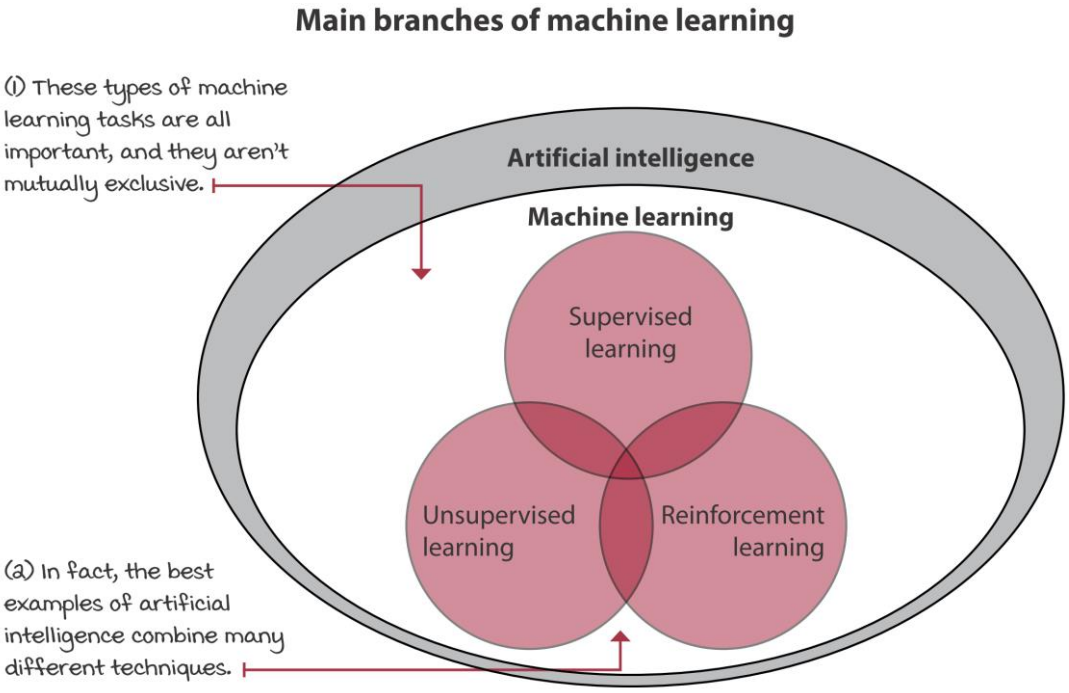
Reinforcement learning and strategies mining

Plan

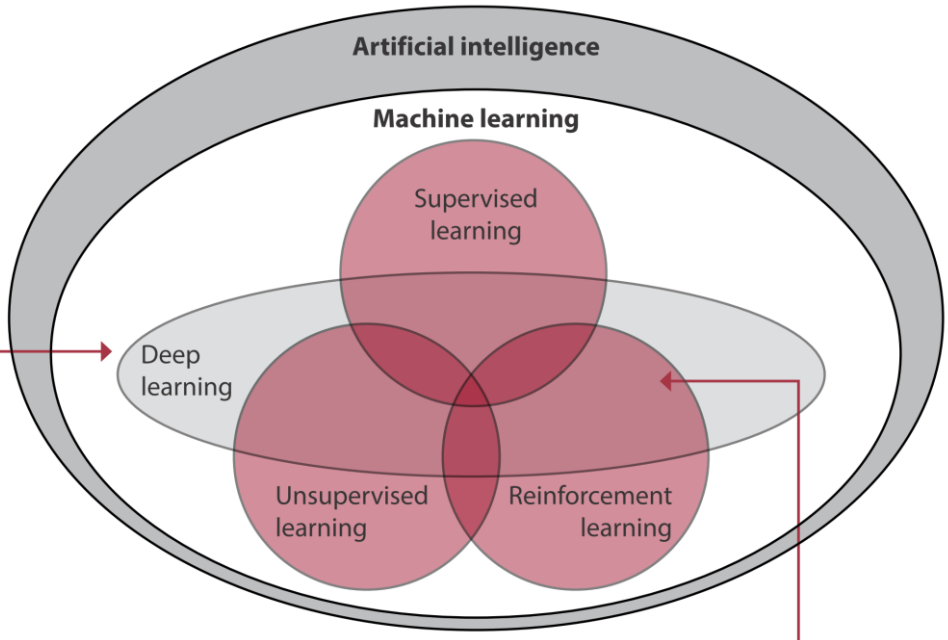
1 Introduction





Deep learning is a powerful toolbox

(1) The important thing here is deep learning is a toolbox, and any advancement in the field of deep learning is felt in all of machine learning.



(2) Deep reinforcement learning is the intersection of reinforcement learning and deep learning.

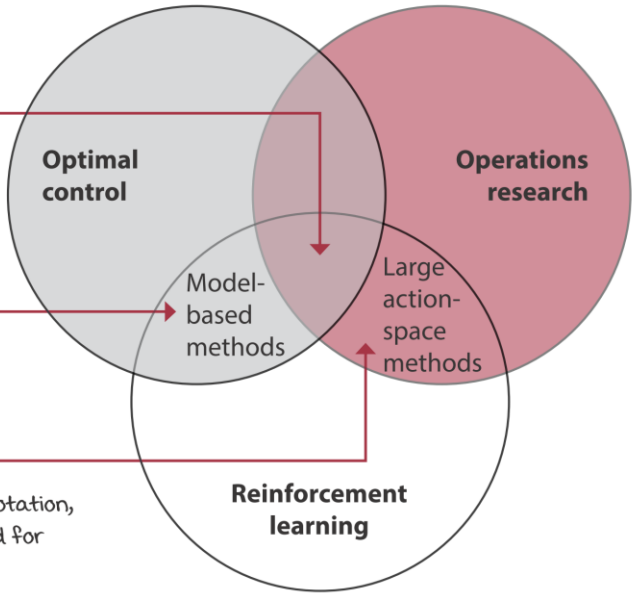
The synergy between similar fields

(1) All of these fields (and many more) study complex sequential decision-making under uncertainty.

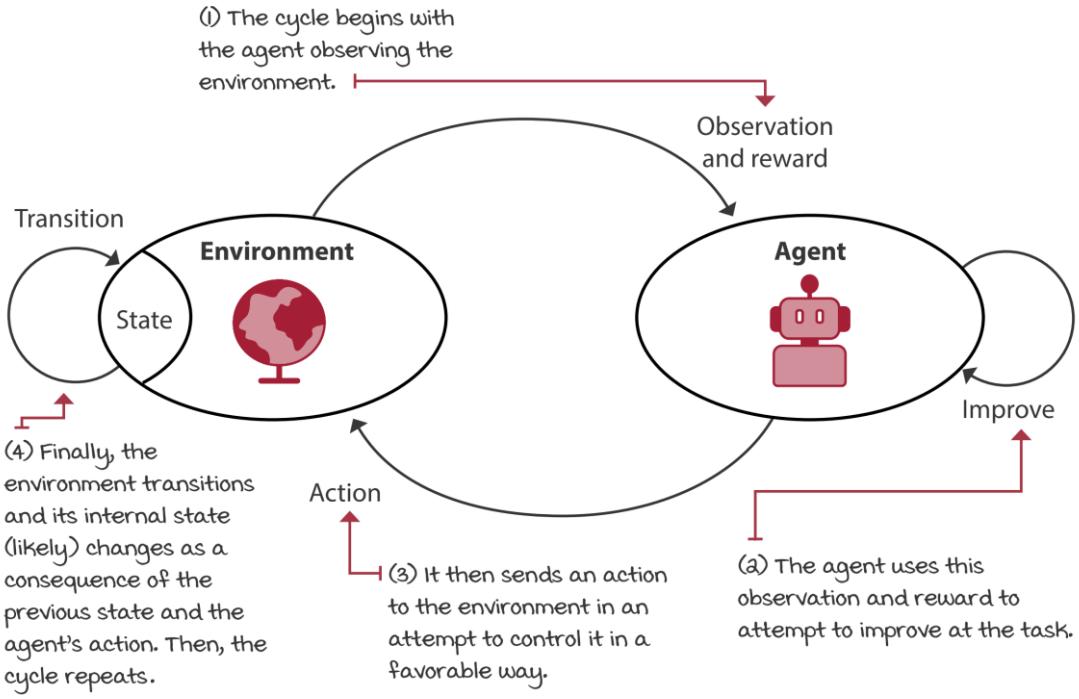
(2) As a result, there's a synergy between these fields. For instance, reinforcement learning and optimal control both contribute to the research of model-based methods.

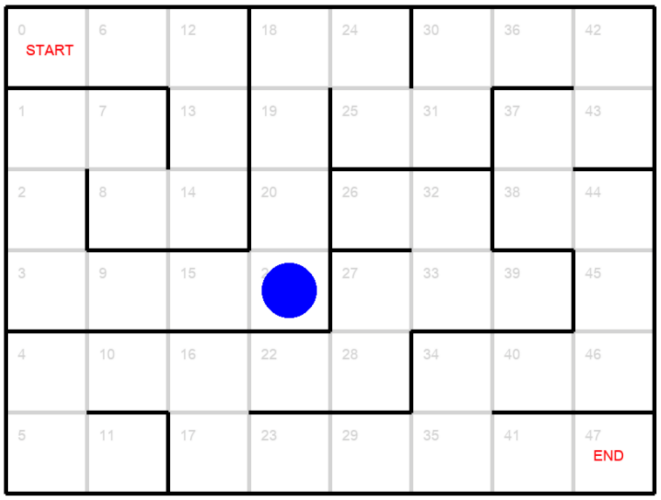
(3) Or reinforcement learning and operations research both contribute to the study of problems with large action spaces.

(4) The downside is an inconsistency in notation, definitions, and so on, that makes it hard for newcomers to find their way around.

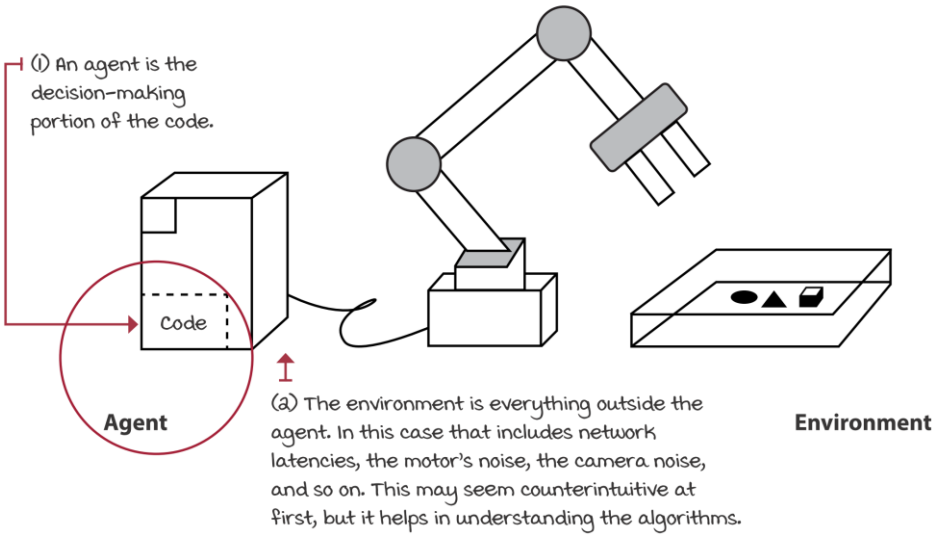


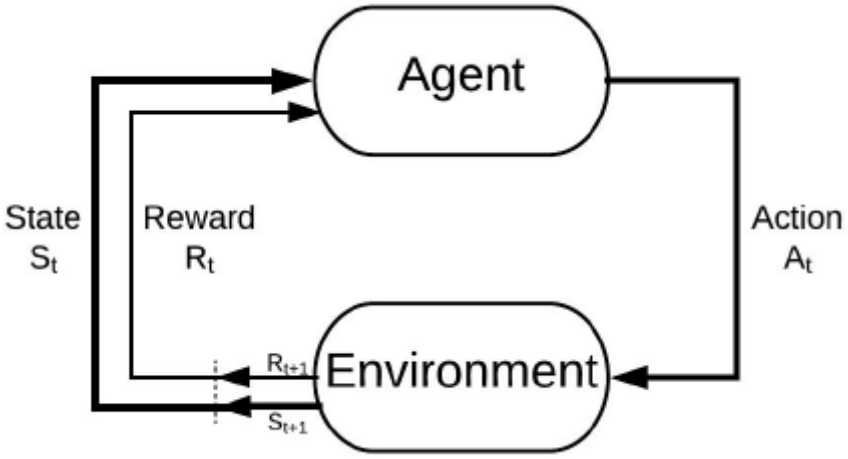
The reinforcement learning cycle





Boundary between agent and environment





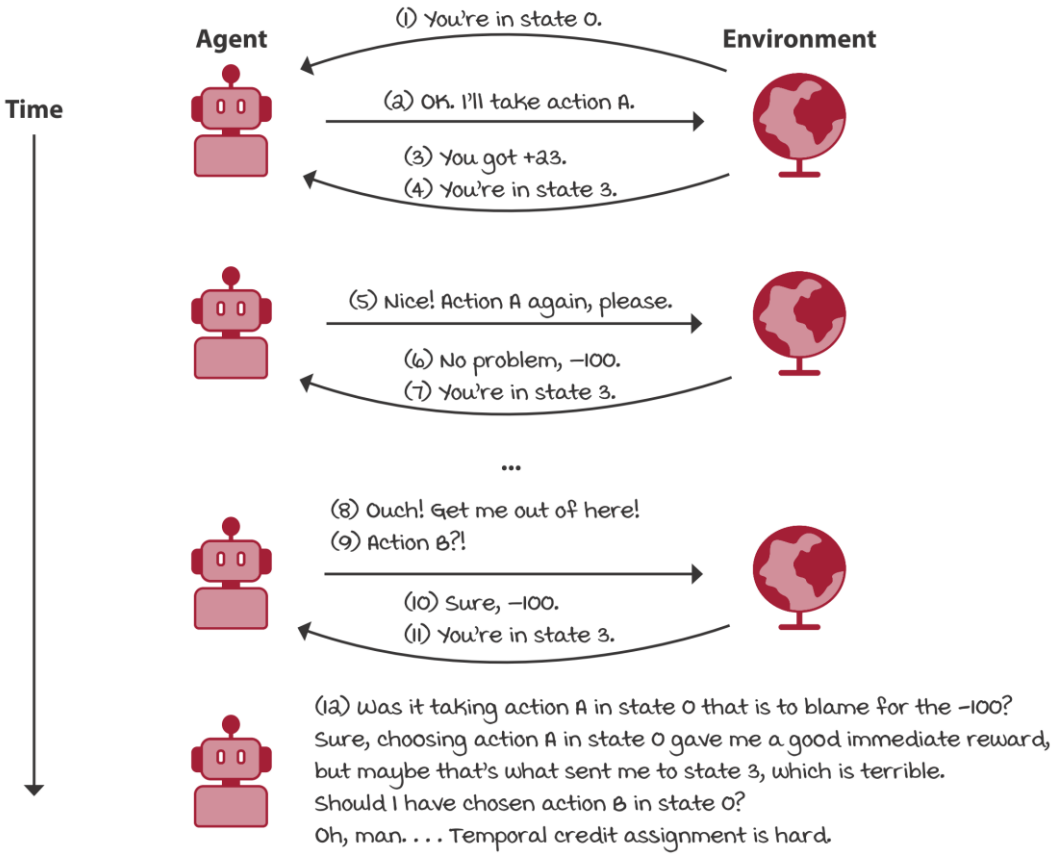
Experience tuples

Agent	Environment	Time step
Action *a*	State *s* Reward *r*	*t*
Action *a′*	State *s′* Reward *r′*	*t+1*
Action *a″*	State *s″* Reward *r″*	*t+2*
...	State *s‴* ...	*t+3*

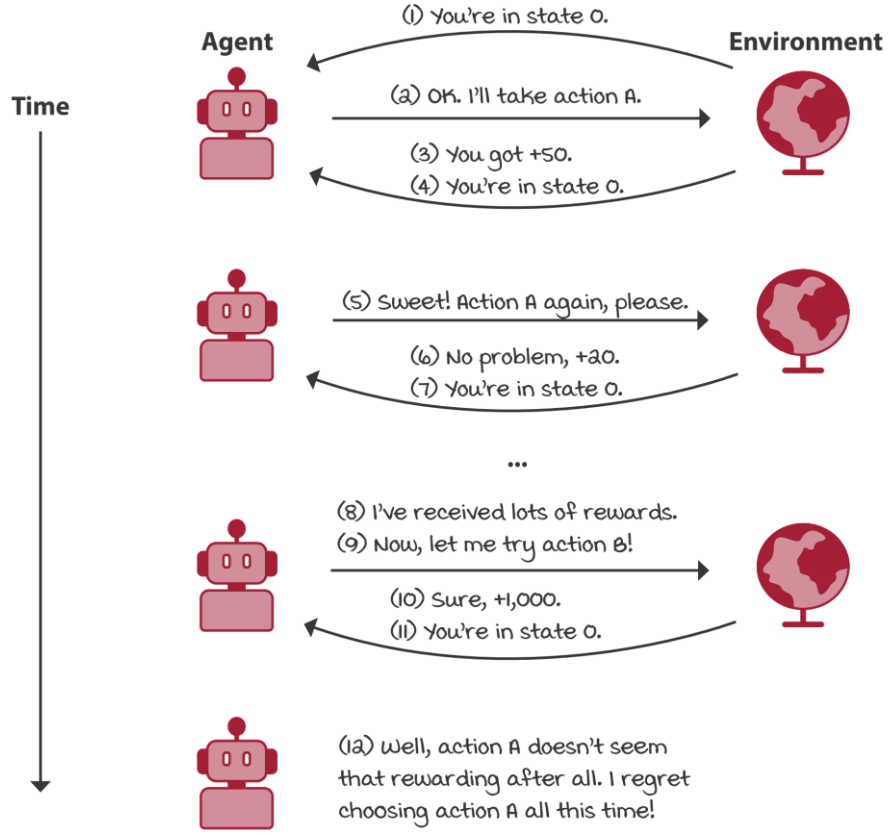
Experiences:
t, (s, a, r′, s′)
t+1, (s′, a′, r″, s″)
t+2, (s″, a″, r‴, s‴)
...

...

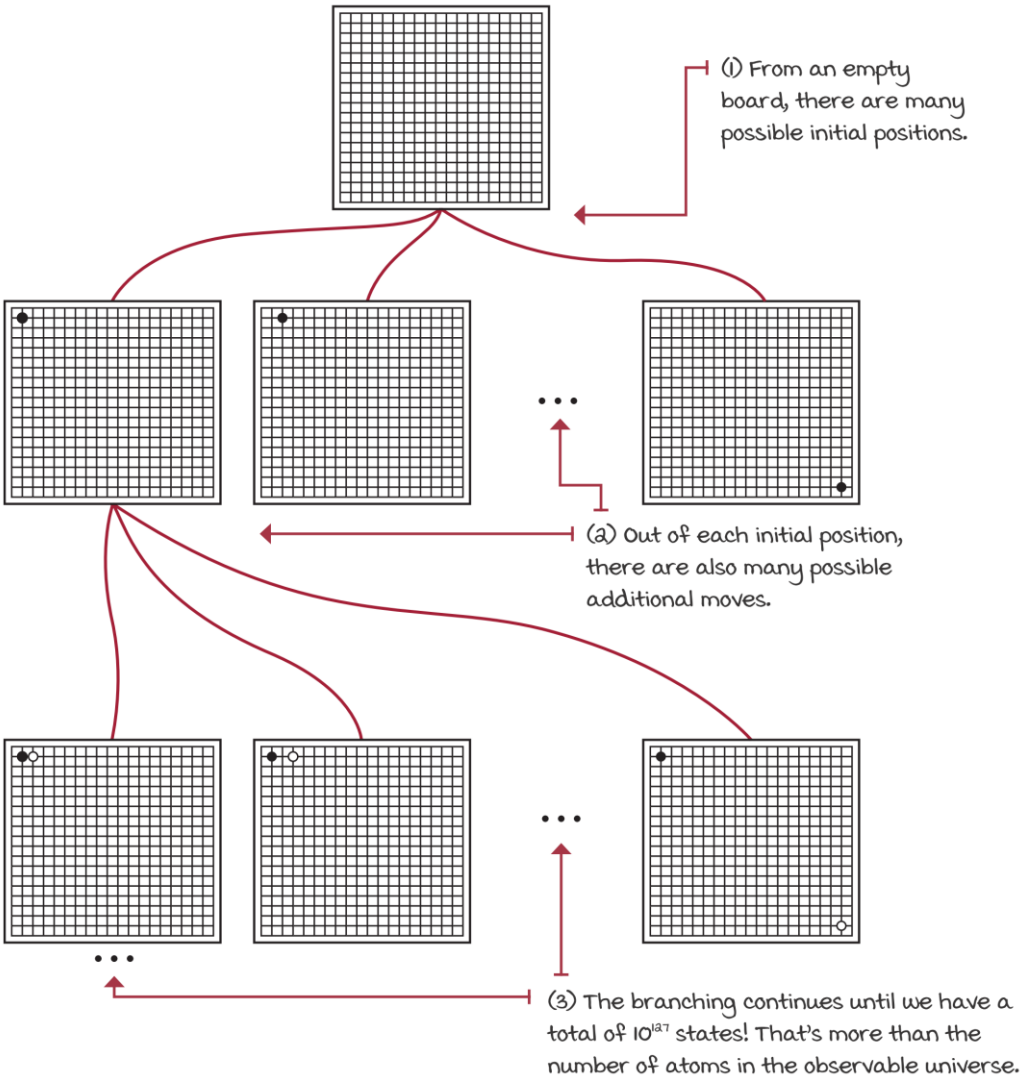
The difficulty of the temporal credit assignment problem



The difficulty of the exploration vs. exploitation trade-off

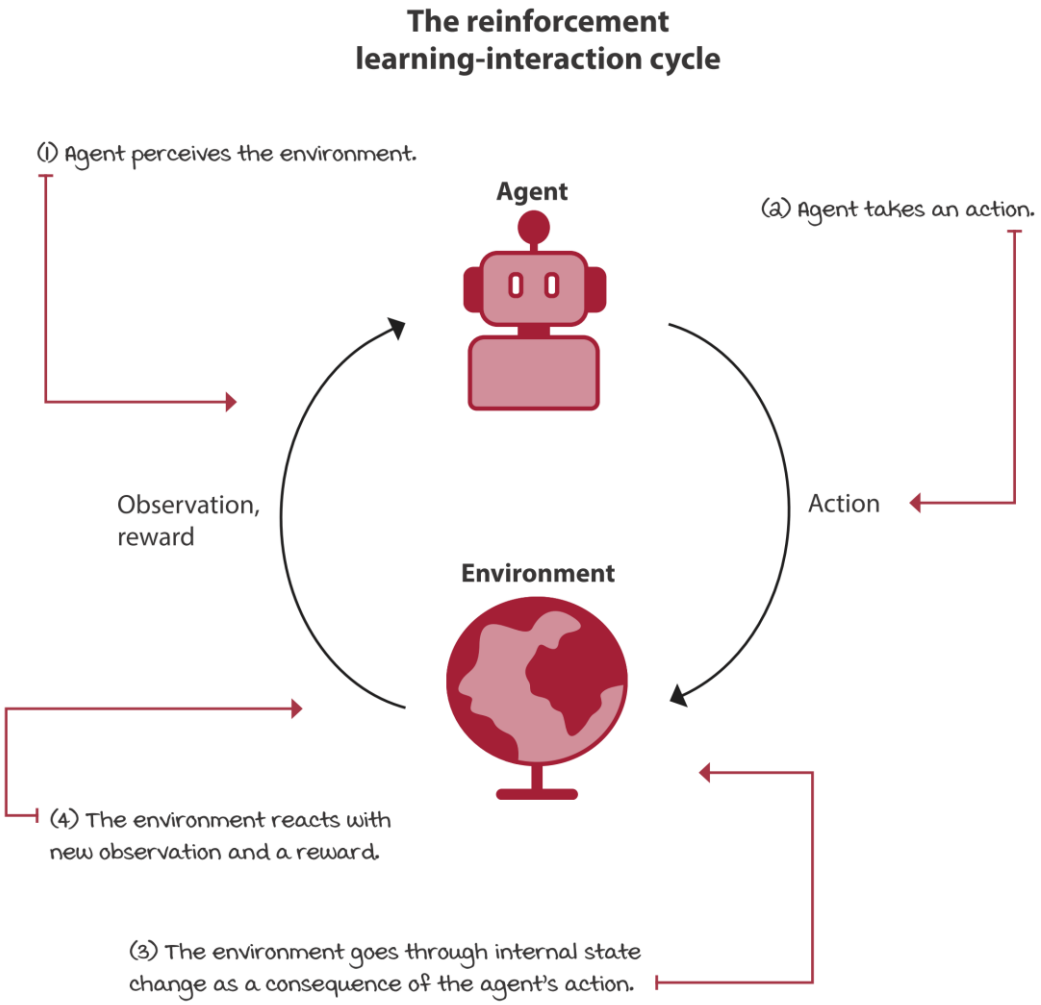


Game of Go: enormous branching factor

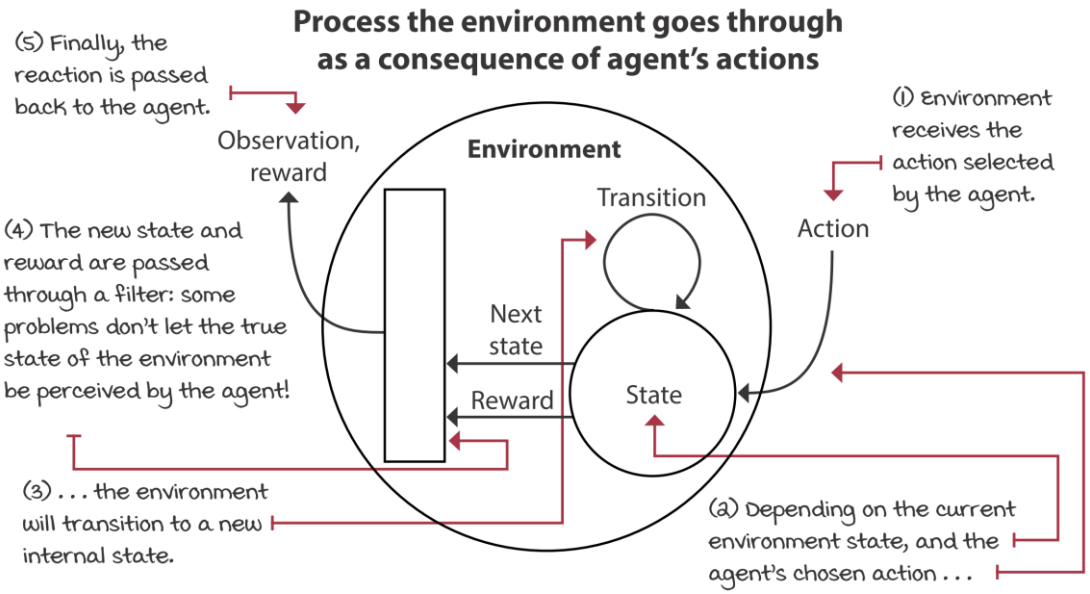


2 Basics

2.1 Components of reinforcement learning

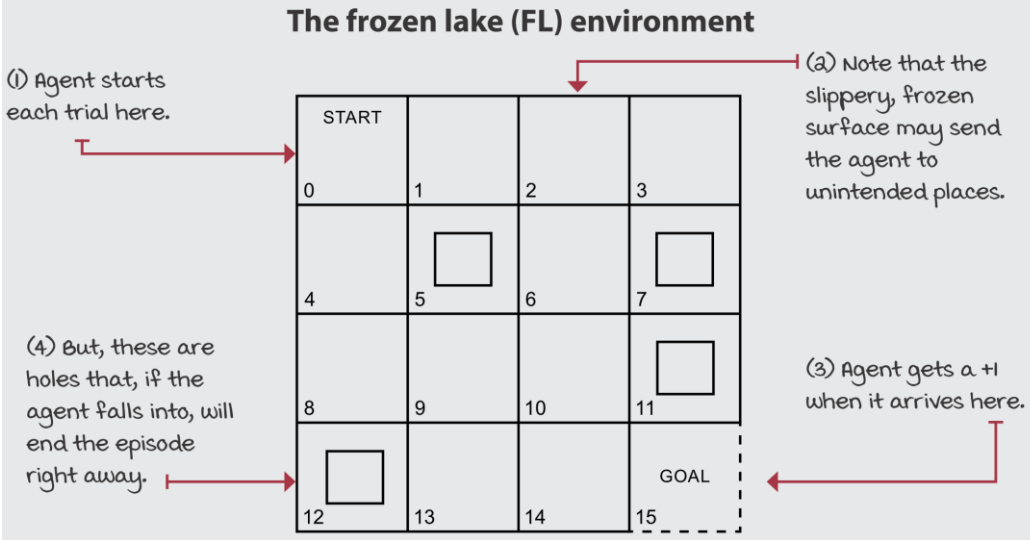


2.2 The environment



2.3 MDPs: The engine of the environment

The task in the FL environment is similar to the task in the BW and BSW environments: to go from a start location to a goal location while avoiding falling into holes. The challenge is similar to the BSW, in that the surface of the FL environment is slippery, it's a frozen lake after all. But the environment itself is larger. Let's look at a depiction of the FL.



The FL is a 4 × 4 grid (it has 16 cells, ids 0–15). The agent shows up in the START cell every new episode. Reaching the GOAL cell gives a +1 reward; anything else is a 0. Because the surface are slippery, the agent moves only a third of the time as intended. The other two-thirds are split evenly in orthogonal directions. For example, if the agent chooses to move down, there's a 33.3% chance it moves down, 33.3% chance it moves left, and 33.3% chance it moves right. There's a fence around the lake, so if the agent tries to move out of the grid world, it will bounce back to the cell from which it tried to move. There are four holes in the lake. If the agent falls into one of these holes, it's game over.

Are you ready to start building a representation of these dynamics? We need a Python dictionary representing the MDP as described here. Let's start building the MDP.

States: Specific configurations of the environment

State space: A set of sets

FL state space

```
[ [0], [1], [2], [3],
  [4], [5], [6], [7],
  [8], [9], [10], [11],
  [12], [13], [14], [15] ]
```

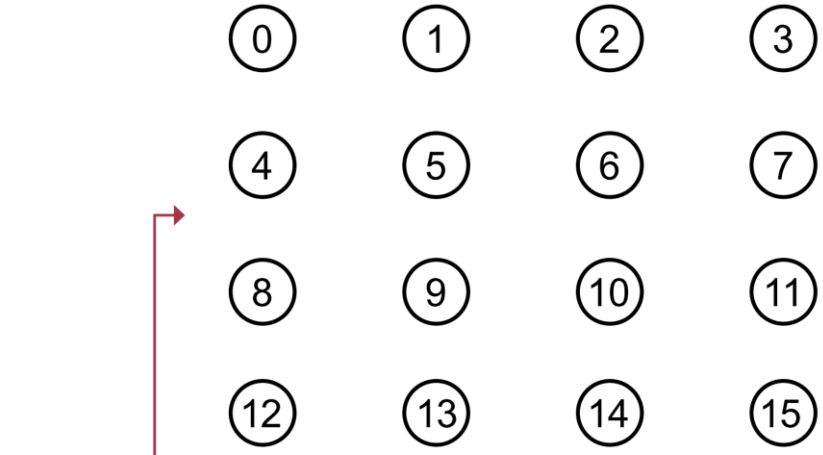
Some other state space

```
[ [0.12,     -1.24, 0, -1, 1.44],
  [0.121,    -1.24, 0, -1, 1.44],
  [0.1211,   -1.24, 0, -1, 1.44],
  ...                           ]
```

(1) The inner set (the number of variables that compose the states) must be finite. The size of the inner set must be a positive integer.

(2) But the outer set may be infinite: if any of the inner sets' elements is continuous, for instance.

States in the FL contain a single variable indicating the id of the cell in which the agent is at any given time step



(1) It's just a 4 × 4 grid!

Show Me the Math

The Markov property

(1) The probability of the next state . . . (3) . . . will be the same . . .

$$P(S_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \ldots)$$

(2) . . . given the current state and current action . . .

(4) . . . as if you give it the entire history of interactions.

Un Processus de Décision Markovien MDP (Markov Decision Process) consiste en un quadruplet : états, actions, fonction de transition entre états, et finalement une fonction récompense (reward). Nous détaillons chacun ces 4 éléments :

Etats L'ensemble des états environnementaux S est défini comme un ensemble fini d'états

$$S = \{s^1, s^2, ..., s^N\}$$

où la taille de l'espace des états est N. Un état est une caractérisation unique de tout ce que l'on pourrait qualifier un état de l'environnement. Par exemple, dans le jeu des échecs, un état peut être le contenu de chacune des 8x8 cases, comme ça pourrait être une image de l'échiquier. Il est ainsi nécessaire de distinguer des états possibles (légaux) des états irréalisables (illégaux). Nous prenons l'hypothèse que S contient uniquement des états possibles identifiés par des clés symboliques.

Actions L'ensemble des actions A est défini par l'ensemble

$$A = \{a^1, a^2, ..., a^K\}$$

où K est la taille de l'espace des actions. Les actions permettent de contrôler les états. Un ensemble restreint d'actions peuvent être effectuées à partir d'un état $s \in S$, dénoté $A(s)$ où $A(s) \subseteq A$. En supposant que toutes les actions sont possibles à partir d'un état, nous adoptons une fonction précondition $pre : S \times A \to \{true, false\}$ précisant si la transition est permis ou non.

Fonction de transition En appliquant une action $a \in A$ à partir d'un état $s \in S$, le système effectue une transition vers un nouvel état $s' \in S$, en

se basant sur une distribution de probabilité par rapport à l'ensemble des transitions possibles. La fonction de transition T est définie

$$T : S \times A \times S \to [0, 1]$$

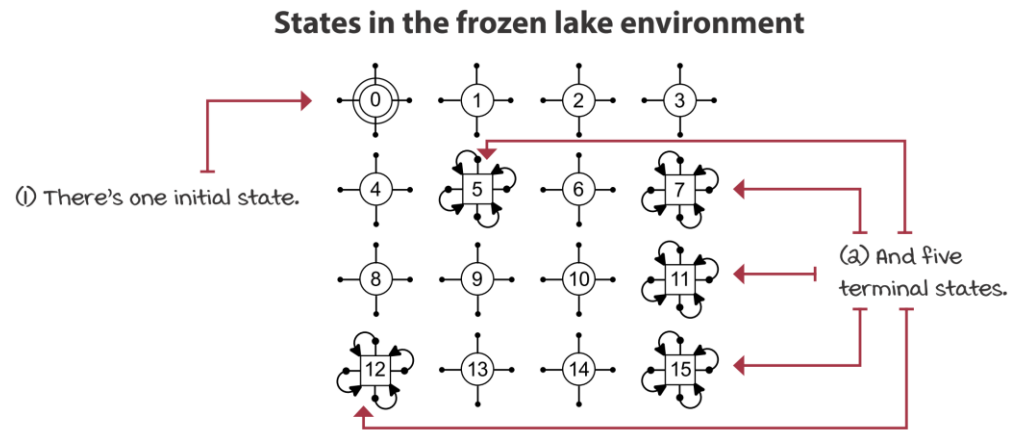
i.e., la probabilité pour atteindre l'état s' après exécution d'une action a au niveau d'un état s, dénotée $0 \leq T(s, a, s') \leq 1$. Nous avons nécessairement, $\forall s \in S, a \in A, \sum_{s' \in S} T(s, a, s') = 1$. Ainsi, au lieu d'utiliser une fonction précondition, nous pouvons simplement de supposer $T(s, a, s') = 0$ pour tous les états $s' \in S$ si a n'est pas possible à partir de s.

Pour exprimer un ordre dans lequel les actions sont possibles, nous supposons l'existence d'une horloge globale, $t = 1, 2, ...$ On utilisera la notation s_t dénotant l'état à l'instant t, et s_{t+1} à l'instance $t+1$, permettant de comparer les différents états (et donc les actions aussi) au cours du temps.

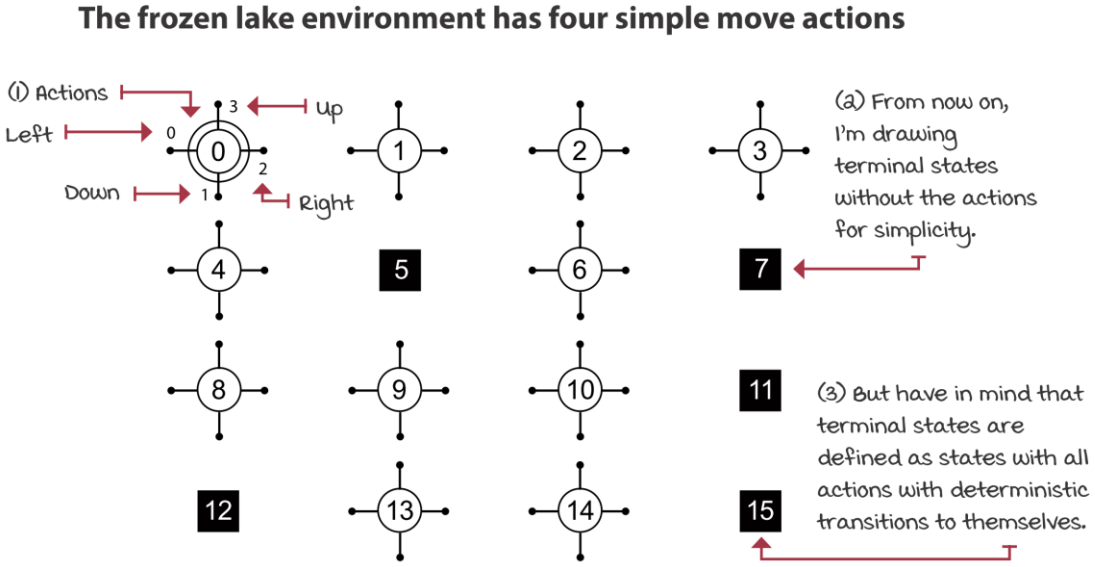
On dit que le système est Markovien si toute action ne dépend pas des états précédent, mais uniquement de l'état courant :

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}|s_t, a_t) = T(s_t, a_t, s_{t+1}) \tag{4.1}$$

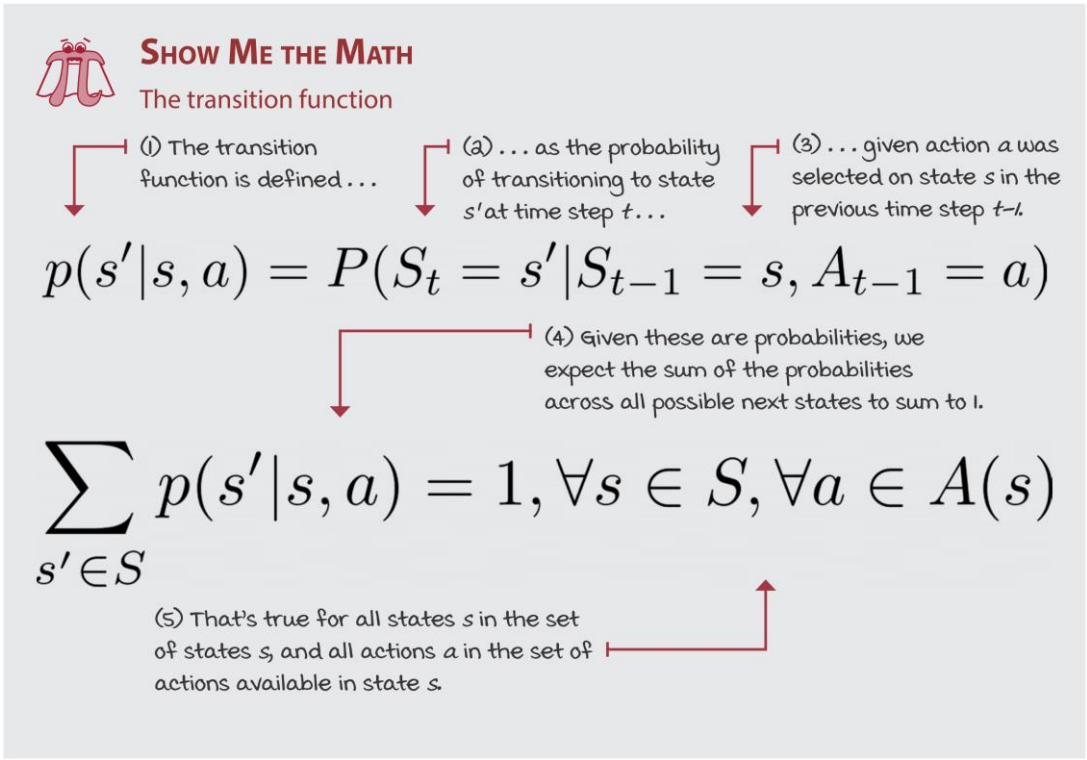
L'idée de la dynamique Markovienne est que l'état courant dispose de suffisamment d'information pour prendre la meilleure décision. Des modèles plus généraux peuvent être caractérisés comme étant k-Markov, i.e. le dernier état dépend des k états précédents, bien que un processus k-Markov peut toujours être transformé en un processus 1-Markov par regroupement d'états. Parmi ces processus généraux, on peut noter les MDPs partiellement observables POMDPs (Partially Observable MDP).

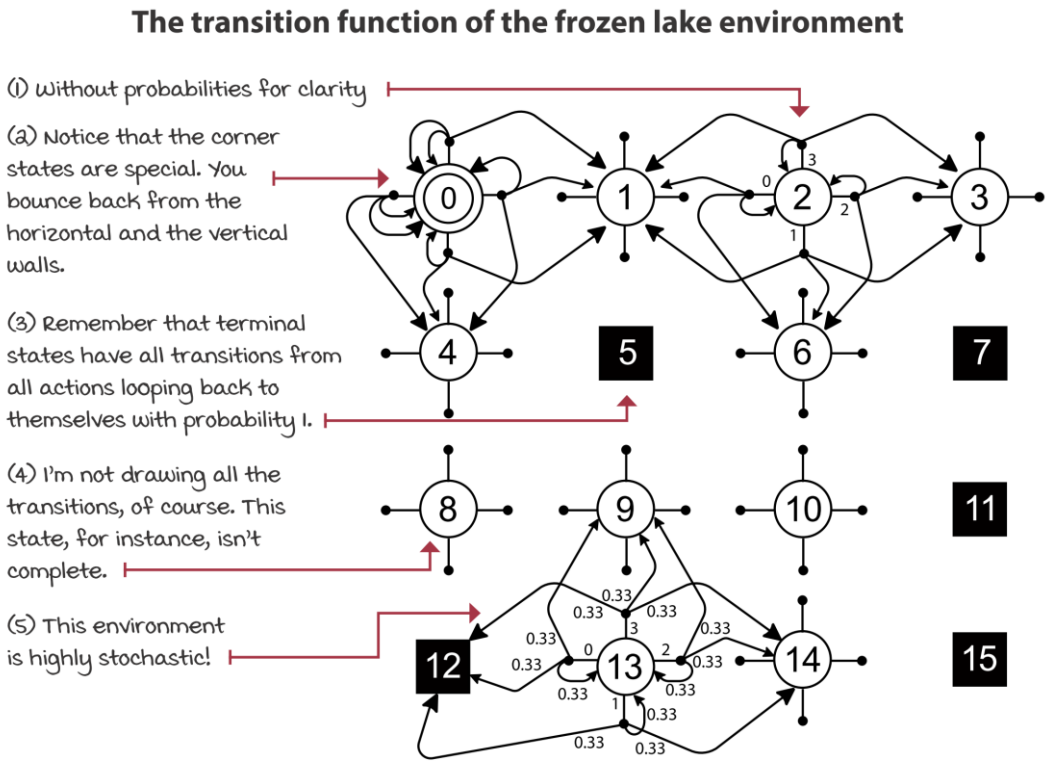


2.4 Actions: A mechanism to influence the environment



2.5 Transition function: Consequences of agent actions





2.6 Reward signal: Carrots and sticks

Fonction de récompense (Reward) La fonction Reward spécifie la récompense au niveau d'un état donné ou exécutant une action donnée. La fonction Reward est identifiée

$$R : S \to \mathbb{R}$$

et spécifie la récompense obtenue dans les états. Deux autres notations sont aussi adoptées

$$R : S \times A \to \mathbb{R}$$

ou encore

$$R : S \times A \times S \to \mathbb{R}.$$

La première caractérise la récompense d'un état s, la deuxième l'action a de transition à partir d'un état s, et la troisième l'action a de transition d'un état s vers un autre état s'. Nous utilisant le plus souvent $R(s, a, s')$.

La fonction de transition est une partie importante d'un MDP, car elle spécifie implicitement la fonction objectif de la problématique de décision. Par exemple, dans un jeu (e.g., échec, tic-tac-toe, ...), on peut mettre un Reward positif sur les états gagnants et négatif sur les états perdants.

Show Me the Math

The reward function

(1) The reward function can be defined as follows.

(3) And, it's the expectation of reward at time step t, given the state-action pair in the previous time step.

(2) It can be defined as a function that takes in a state-action pair.

$$r(s,a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$$

(4) But, it can also be defined as a function that takes a full transition tuple s, a, s!

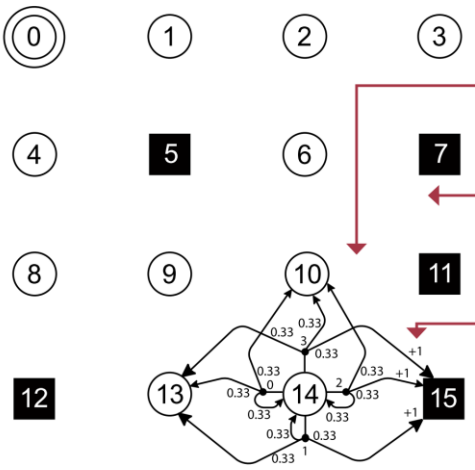
(5) And it's also defined as the expectation, but now given that transition tuple.

$$r(s,a,s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']$$

$$R_t \in \mathcal{R} \subset \mathbb{R}$$

(6) The reward at time step t comes from a set of all rewards R, which is a subset of all real numbers.

Reward signal for states with non-zero reward transitions



(1) State 14's actions transition function and reward signal

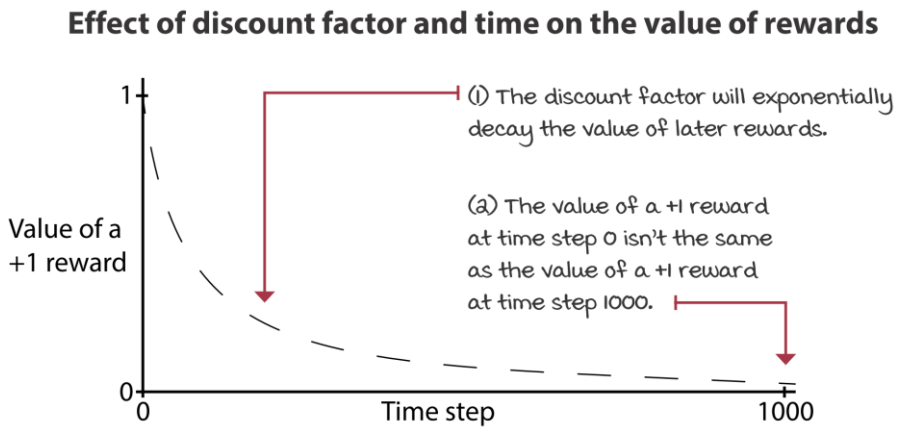
(2) Every other reward in this environment is zero, so I'm omitting all except state 14's.

(3) Notice how I'm using the most explicit form, the full transition R(s,a,s').

State	Action	Next state	Transition probability	Reward signal
0	Left	0	0.33	0
0	Left	0	0.33	0
0	Left	4	0.33	0
0	Down	0	0.33	0
0	Down	4	0.33	0
0	Down	1	0.33	0
0	Right	4	0.33	0
0	Right	1	0.33	0
0	Right	0	0.33	0

State	Action	Next state	Transition probability	Reward signal
0	Up	1	0.33	0
0	Up	0	0.33	0
0	Up	0	0.33	0
1	Left	1	0.33	0
1	Left	0	0.33	0
1	Left	5	0.33	0
1	Down	0	0.33	0
1	Down	5	0.33	0
1	Down	2	0.33	0
1	Right	5	0.33	0
1	Right	2	0.33	0
1	Right	1	0.33	0
2	Left	1	0.33	0
2	Left	2	0.33	0
2	Left	6	0.33	0
2	Down	1	0.33	0
...	...	...	...	...
14	Down	14	0.33	0
14	Down	15	0.33	1
14	Right	14	0.33	0
14	Right	15	0.33	1
14	Right	10	0.33	0
14	Up	15	0.33	1
14	Up	10	0.33	0
...	...	...	...	...
15	Left	15	1.0	0
15	Down	15	1.0	0
15	Right	15	1.0	0
15	Up	15	1.0	0

2.7 Discount: The future is uncertain, value it less



SHOW ME THE MATH

The discount factor (gamma)

(1) The sum of all rewards obtained during the course of an episode is referred to as the return.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

(2) But we can also use the discount factor this way and obtain the discounted return. The discounted return will downweight rewards that occur later during the episode.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T$$

(3) We can simplify the equation and have a more general equation, such as this one.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

(4) Finally, take a look a this interesting recursive definition. In the next chapter, we spend time exploiting this form.

$$G_t = R_{t+1} + \gamma G_{t+1}$$

Discounted return in the slippery walk five environment

$$G_0 = 1 * 0 + 0.99 * 0 + 0.9801 * 0 + 0.9702 * 0 + 0.9605 * 0 + 0.9509 * 1$$

(1) Calculating the return at time step *t=0*

(2) This is the reward obtained at time step *t+1* (0) discounted by gamma (0.99°).

(3) Reward at *t+2*, discounted by gamma raised to the power 1

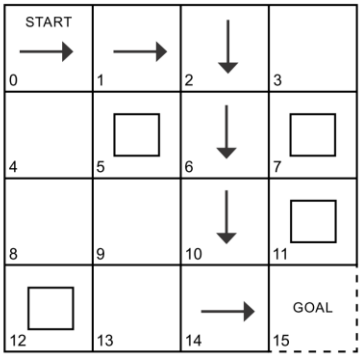
(4) Discounted reward at *t+3*

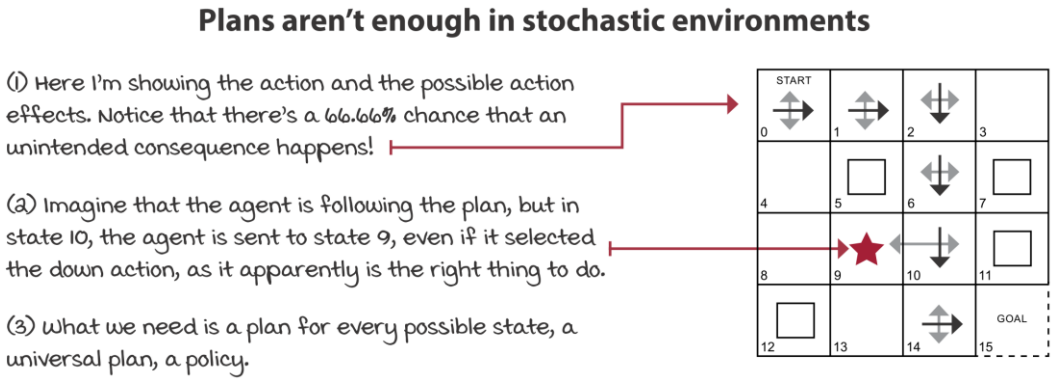
(5) And soon . . .

(6) . . . this is the discounted reward at time step T (final step).

A solid plan in the FL environment

(1) This is a solid plan. But, in a stochastic environment, even the best of plans fail. Remember that in the FL environment, unintended action effects have even higher probability: 66.66% vs. 33.33%! You need to plan for the unexpected.





2.8 State-value function: What to expect from here?

Etant donné un MDP $\langle S, A, T, R\rangle$, une stratégie est une fonction calculable fournissant pour chaque état $s \in S$, une action $a \in A$. Formellement une stratégie π est une fonction

$$\pi : S \to A.$$

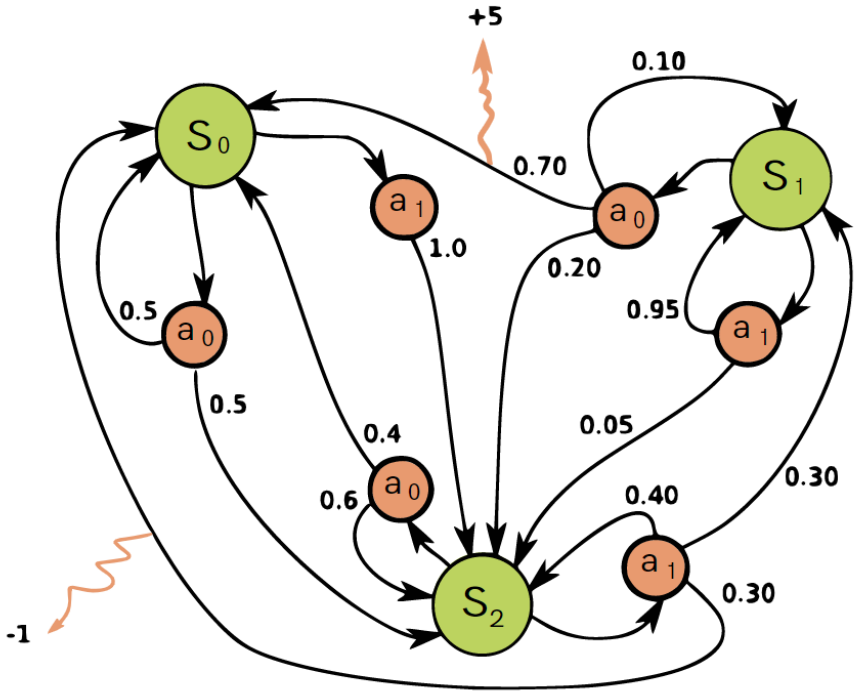
On peut aussi définir une stratégie stochastique

$$\pi : S \times A \to [0, 1]$$

tel que $\sum_{a \in A} \pi(s, a) = 1$.

L'application d'une stratégie à un MDP est faite comme suit. En premier, un état initial s_0 à partir des états initiaux possibles I est généré. Ainsi, la stratégie π suggère l'action $a_0 = \pi(s_0)$, et cette action est exécutée. En exploitant la fonction de transition T et la fonction Reward R, la transition est exécutée vers l'état s_1 avec la probabilité $T(s_0, a_0, s_1)$ et une récompense $r_0 = R(s_0, a_0, s_1)$. Ce processus continue produisant la séquence

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_0 2 r_2, ...$$



On peut démontrer que la solution optimale $V^* = V^\pi$ satisfait l'équation :

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) \tag{4.5}$$

Cette dernière équation est dite l'équation d'optimalité de Bellman. Elle stipule que dans une stratégie optimale, la valeur d'un état est égale à celle de la meilleure action. Pour faire valoir la meilleure action, on peut écrire :

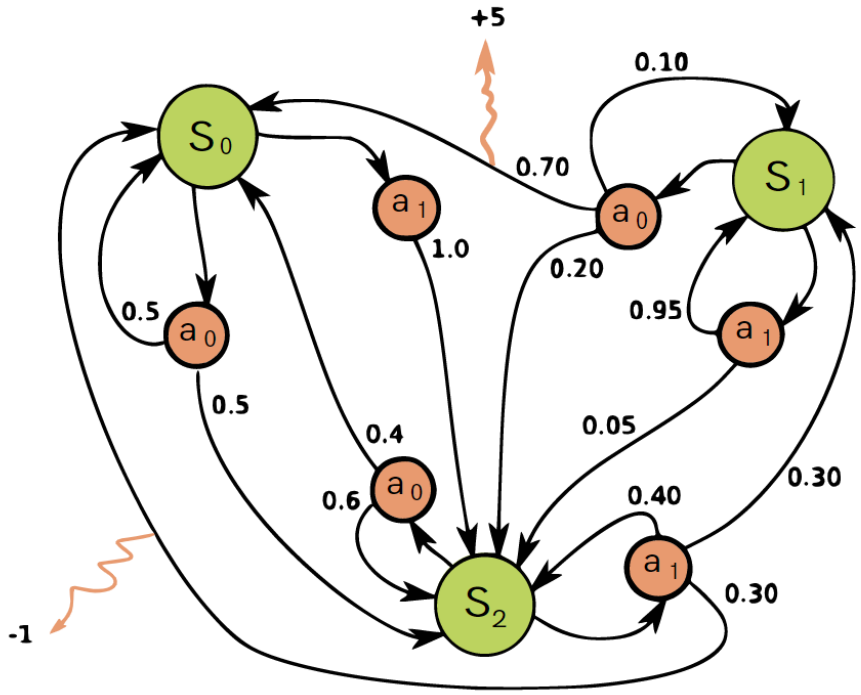
$$V^*(s) = argmax_a \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) \tag{4.6}$$

On appelle cette stratégie, la stratégie gloutonne, dénotée $\pi_{glouton}(V)$. On peut aussi la réécrite en utilisant la formulation sur les actions :

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma max_{a'} Q^*(s', a')) \tag{4.7}$$

Les fonctions Q sont utiles, car elles évitent la somme pondérée utilisant la fonction de transition. Et donc :

$$\begin{array}{lcl} Q^*(s, a) & = & \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) \\ V^*(s) & = & \max_a Q^*(s, a) \end{array} \tag{4.8}$$



Require: discount factor γ, learning parameter α
  initialize Q arbitrarily (e.g. $Q(s,a) = 0, \forall s \in S, \forall a \in A$)
  for each episode **do**
    s is initialized as the *starting* state
    repeat
      choose an action $a \in A(s)$ based on Q and an exploration strate
      perform action a
      observe the new state s' and received reward r
      $Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \cdot \max_{a' \in A(s')} Q(s',a') - Q(s,a) \right)$
      $s := s'$
    until s' is a goal state

L'approche TD consiste à apprendre l'estimation des valeurs en se basant sur les anciennes estimations. C'est un apprentissage à partir de l'expérience.

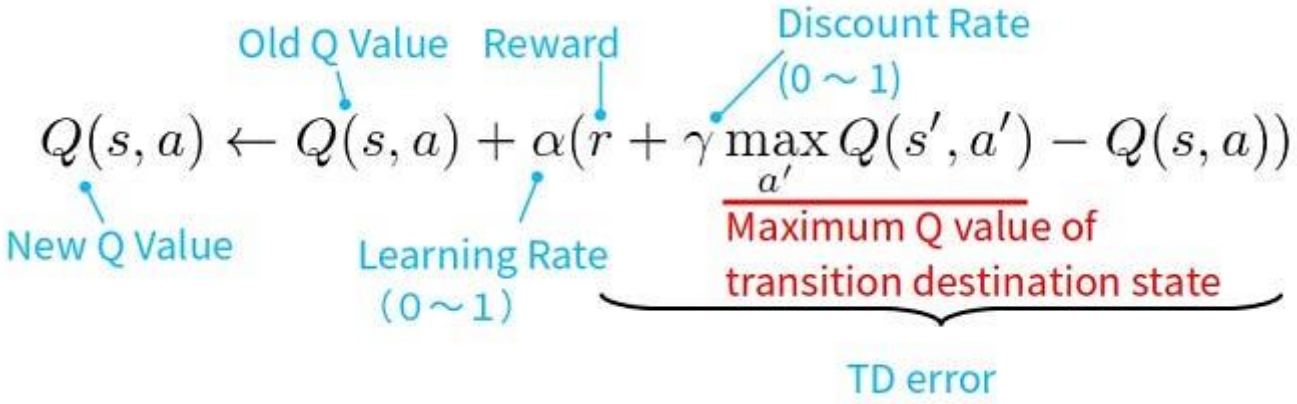
TD(0) est une famille des algorithmes TD, dont la règle de mise à jour de V est la suivante :

$$V_{k+1}(s) \leftarrow V_{k+1}(s) + \alpha(r + \gamma V_k(s') - V_k(s)) \tag{4.9}$$

où $\alpha \in [0, 1]$ est le tôt d'apprentissage.

Par la suite, l'algorithme par Q-learning a été proposé qui met à jour la fonction Q au lieu de V (voir la Figure 4.2), via la formule :

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha(r_t + \gamma max_a Q_t(s_{t+1}, a) - Q_k(s_t, a_t)) \tag{4.10}$$



L'algorithme de Q-learning est utilisé dans un environnement sans stratégie au préalable (off-policy). Justement, quand une stratégie existe, la règle de SARSA a été proposé qui consiste à remplacer la règle (4.10) par :

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \tag{4.11}$$

Cette règle est utilisée quand l'action suivante est préalablement choisie par une stratégie donnée.

2.10 Example

Notebook FrozenLake …

```
!pip install -q gym==0.15.3
```

https://towardsdatascience.com/q-learning-for-beginners-2837b777741

3 References

3.1 Complete course

http://cs330.stanford.edu/fall2021/material/CS330_RLTutorial.pdf

https://awjuliani.medium.com/super-simple-reinforcement-learning-tutorial-part-1-fd544fab149

https://www.davidsilver.uk/teaching/

https://www.youtube.com/watch?v=2pWv7GOvuf0

3.2 Top illustrated course

https://huggingface.co/learn/deep-rl-course/unit2/introduction

3.3 Ressources

https://www.kaggle.com/competitions/connectx/discussion/124421

https://github.com/tsmatz/reinforcement-learning-tutorials

https://neptune.ai/blog/best-reinforcement-learning-tutorials-examples-projects-and-courses

3.4 Youtube

https://www.youtube.com/watch?v=FgzM3zpZ55o&list=PLoROMvodv4rOSOPzutgyCTapiGlY2Nd8u

3.5 Taxi et RL

https://gymnasium.farama.org/environments/toy_text/taxi/

https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

https://www.kaggle.com/code/karthikcs1/reinforcement-learning-taxi-v3-openai

https://www.kaggle.com/code/charel/learn-by-example-reinforcement-learning-with-gym

Echecs
https://www.kaggle.com/code/arjanso/reinforcement-learning-chess-1-policy-iteration/notebook

3.6 Others

https://www.kaggle.com/code/phunghieu/connectx-with-q-learning/notebook

https://github.com/dennybritz/reinforcement-learning

https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial

https://www.javatpoint.com/reinforcement-learning

https://www.toptal.com/deep-learning/pytorch-reinforcement-learning-tutorial

https://blog.paperspace.com/getting-started-with-reinforcement-learning/

https://nbviewer.org/github/DavidSanwald/ai-notebook/blob/master/index.ipynb

https://github.com/DavidSanwald/ai-notebook

https://calvinfeng.gitbook.io/machine-learning-notebook/unsupervised-learning/reinforcement-learning/reinforcement_learning