

IFT 615 – Intelligence Artificielle

Apprentissage par renforcement

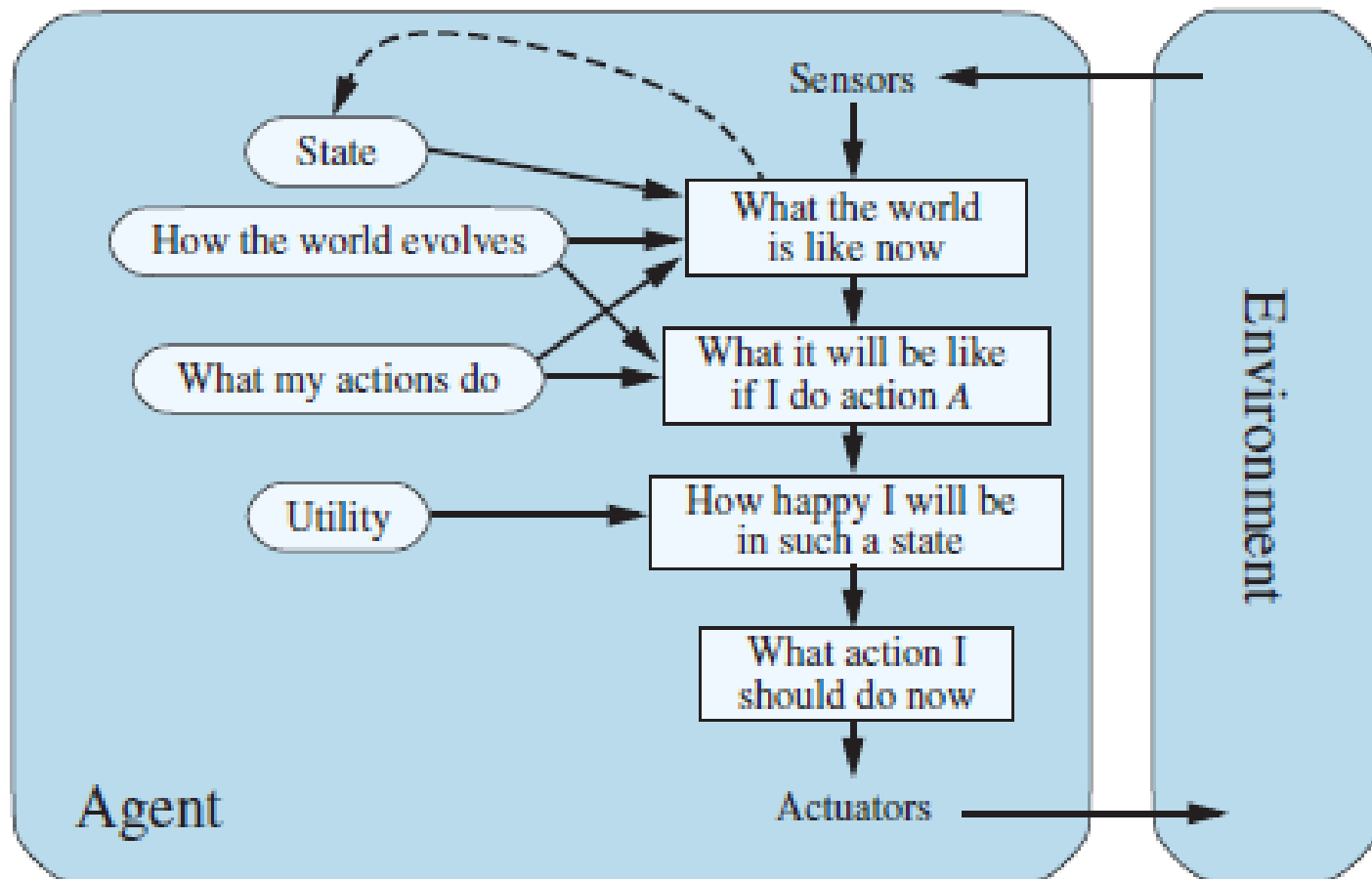
Professeur: Froduald Kabanza

Assistants: D'Jeff Nkashama

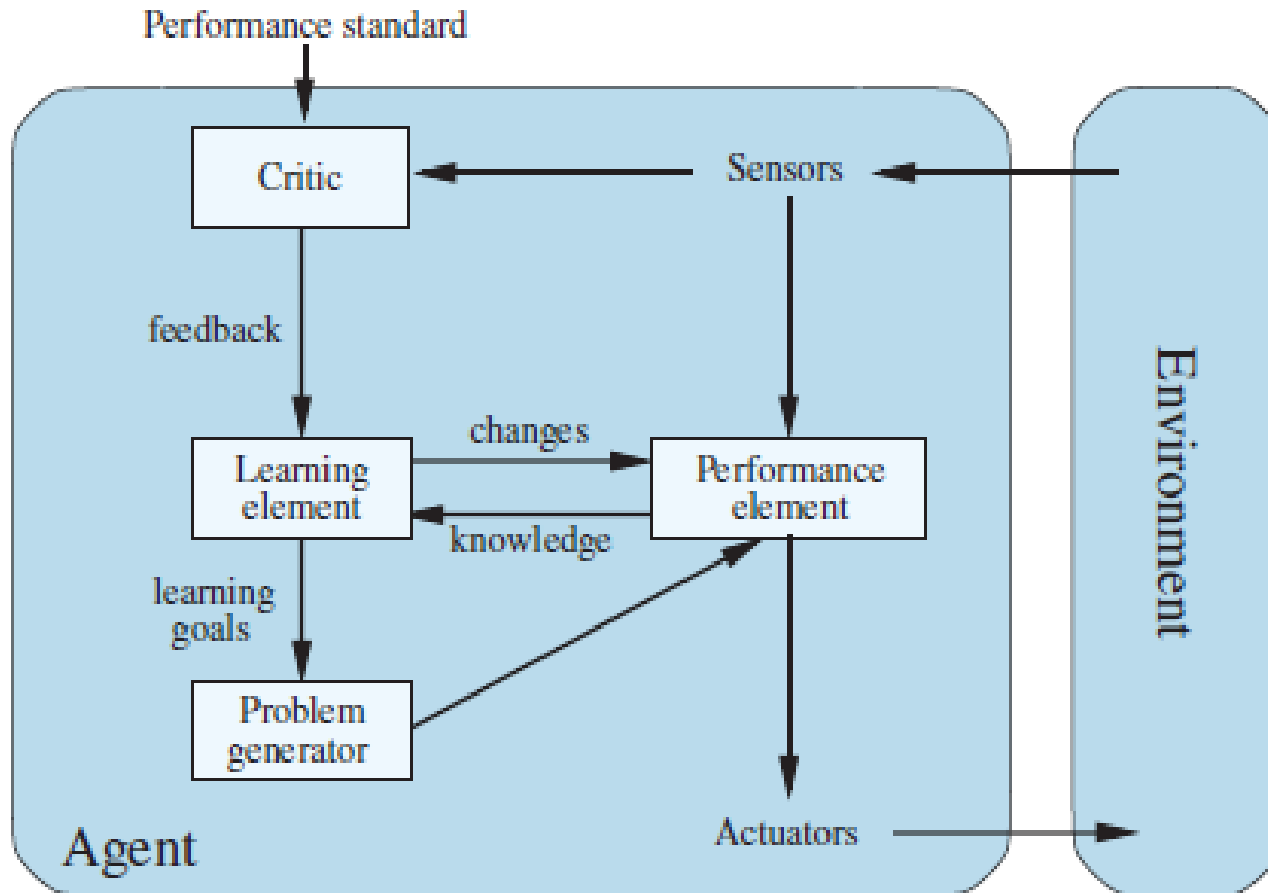
Sujets couverts

- Apprentissage par renforcement passif
 - ◆ méthode par estimation directe
 - ◆ méthode par programmation dynamique adaptative (PDA)
 - ◆ méthode par différence temporelle (TD)
- Apprentissage par renforcement actif
 - ◆ méthode PDA active
 - ◆ méthode TD active
 - ◆ méthode *Q-learning*

Agent performant (utility-based)



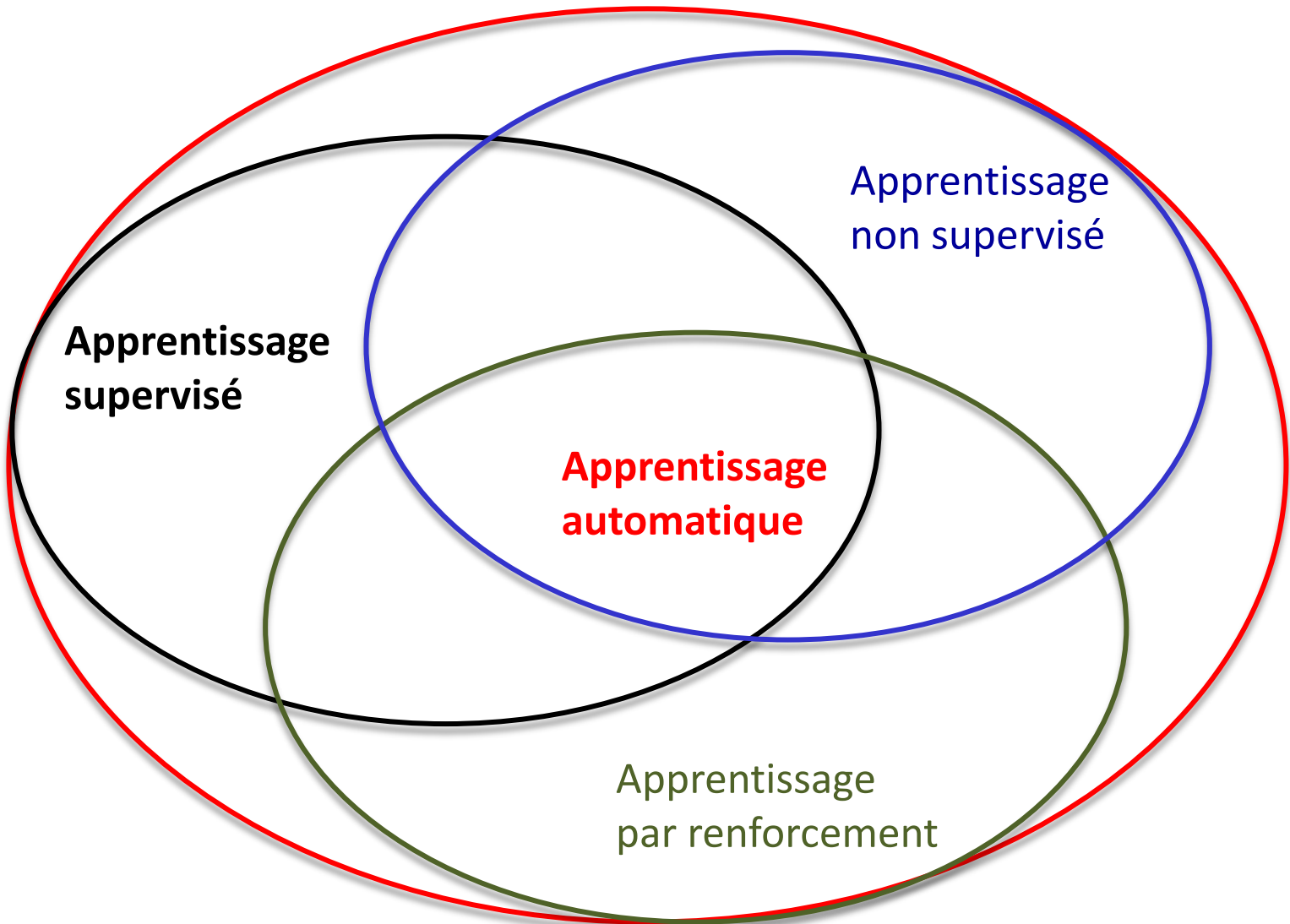
Agent qui apprend



Apprentissage dans un agent

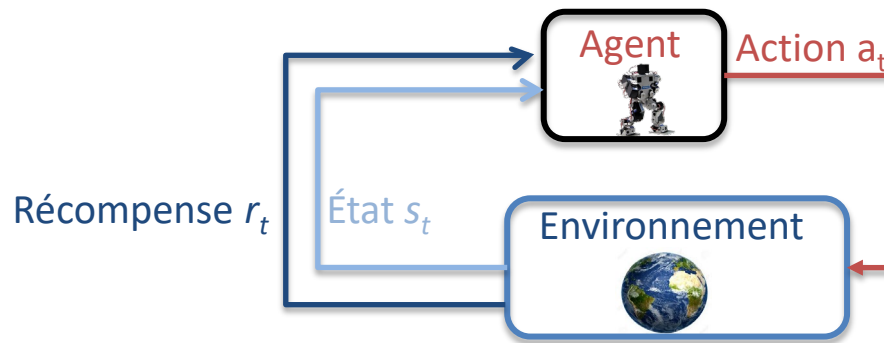
- Un agent **apprend** s'il améliore sa performance sur des tâches futures avec l'expérience
- Pourquoi programmer des programmes qui apprennent:
 - ◆ Un agent peut être dans un environnement inconnu ou difficile à modéliser (il faut qu'il le découvre)
 - ◆ Même quand l'environnement est connu au départ, il peut évoluer dans le temps de façon imprévisible
 - ◆ Parfois, on a aucune idée comment programmer la fonction de performance de l'agent

Types de problèmes d'apprentissage



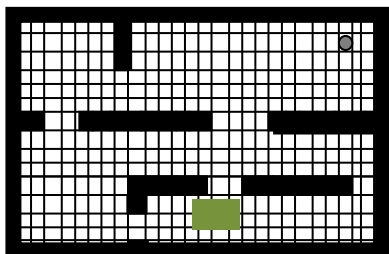
Cadre général de l'apprentissage par renforcement

- L'**apprentissage par renforcement** s'intéresse au cas où l'agent doit apprendre à agir seulement à **partir des récompenses** ou **renforcements**



- Données du problème d'apprentissage: $s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$
 - ◆ L'agent **agit** sur son environnement
 - ◆ Reçoit une retro-action sous-forme de **récompense (renforcement)**
 - ◆ Son **but** est de maximiser la somme des récompenses espérés
- Objectif: **Apprendre** à maximiser la somme des récompenses

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ avec } 0 \leq \gamma \leq 1 \text{ et } r_i < R_{\max}$$



Cadre général

Maximiser $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$, avec $0 \leq \gamma \leq 1$

Politique
 $\pi(s,a)$

Agent



Action a

$a_1 \rightarrow a_2 \rightarrow \dots a_t$

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

$$U(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

$$Q(s,a) = \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} Q(s,a) = \operatorname{argmax}_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

Récompense r

$r_1 \rightarrow r_2 \rightarrow \dots r_t$

Environnement

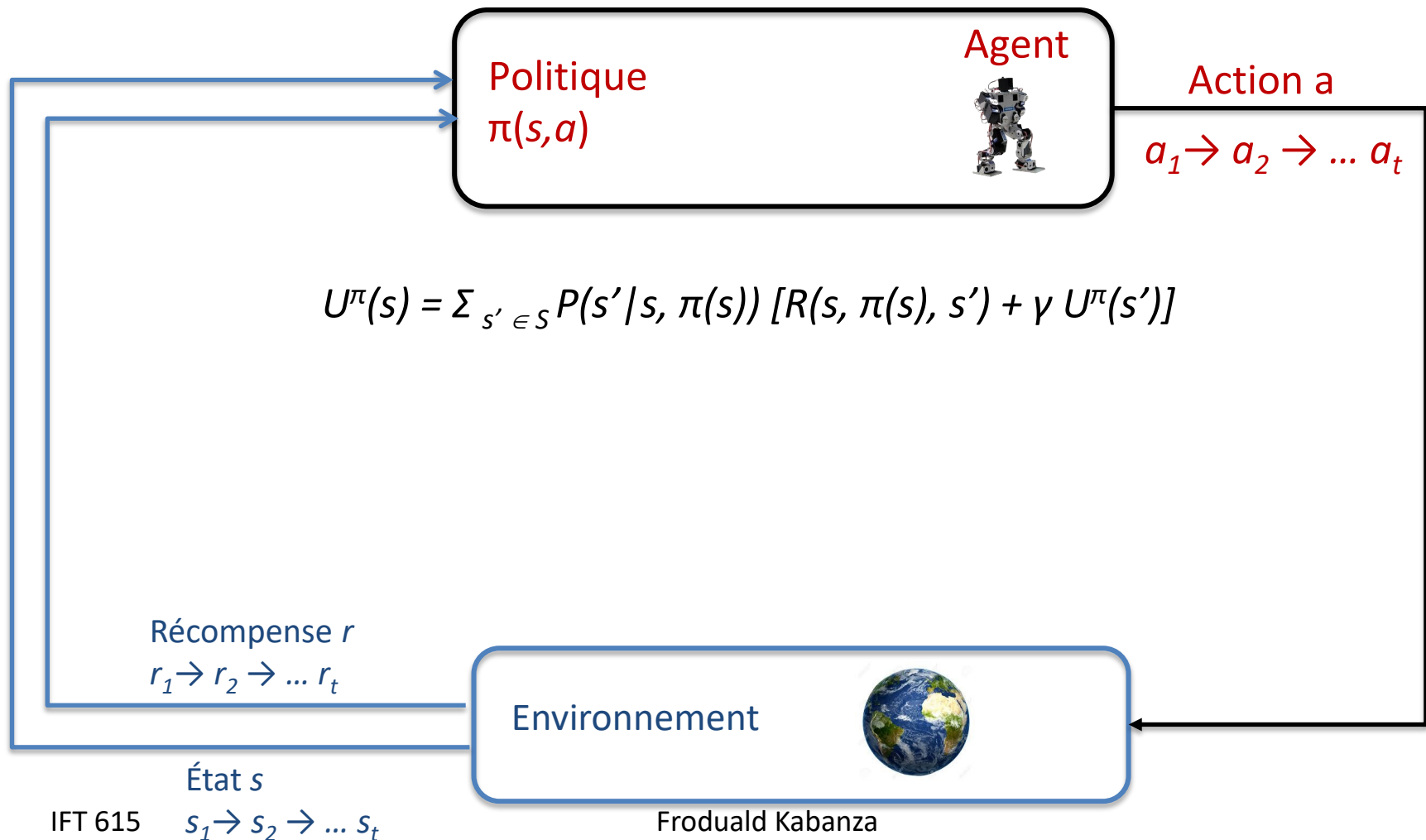


État s

$s_1 \rightarrow s_2 \rightarrow \dots s_t$

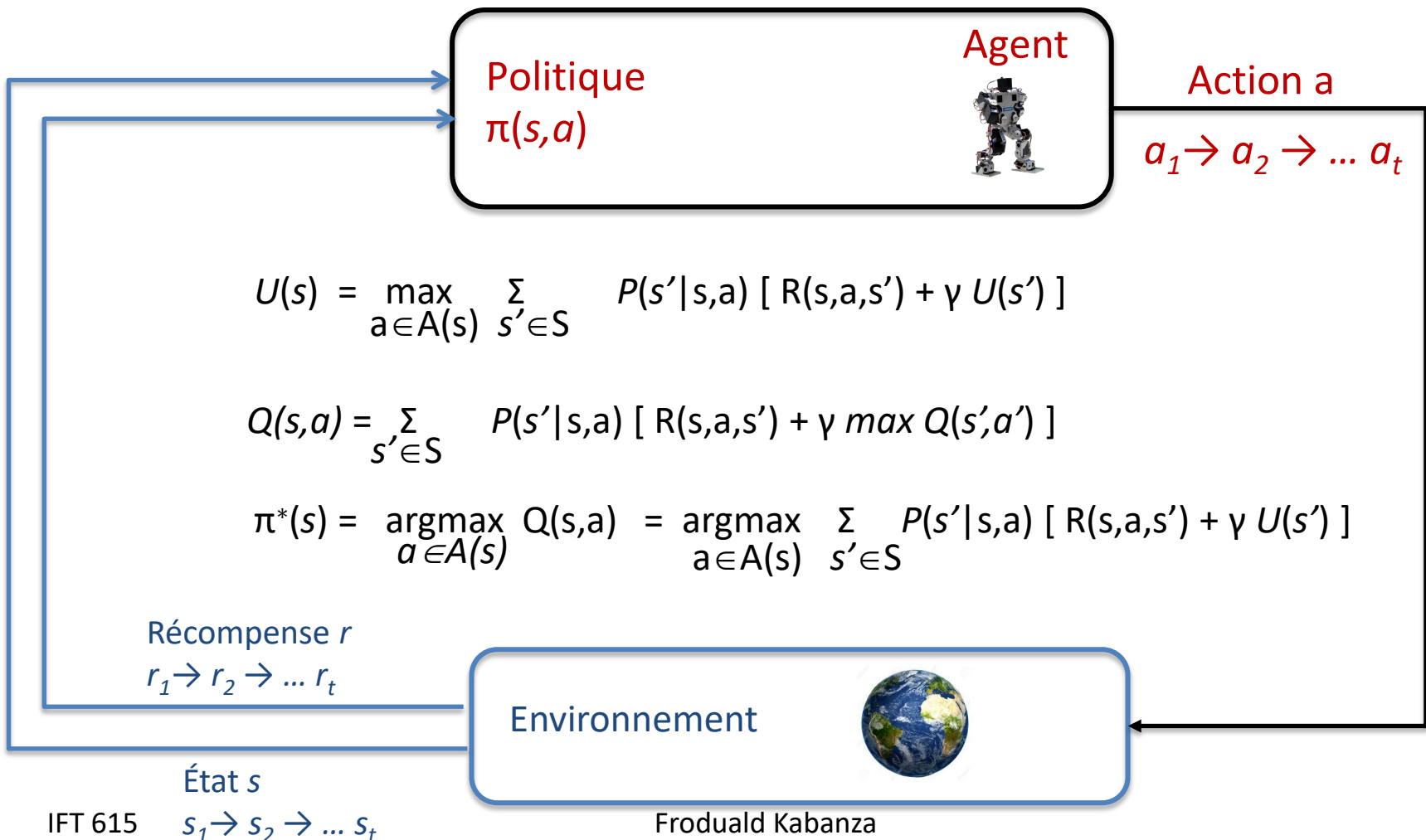
Évaluer une politique

- Étant donné π , on peut calculer $U^\pi(s)$ par *résolution du système d'équations*, *modified policy iteration*, ou *asynchronous policy iteration*.



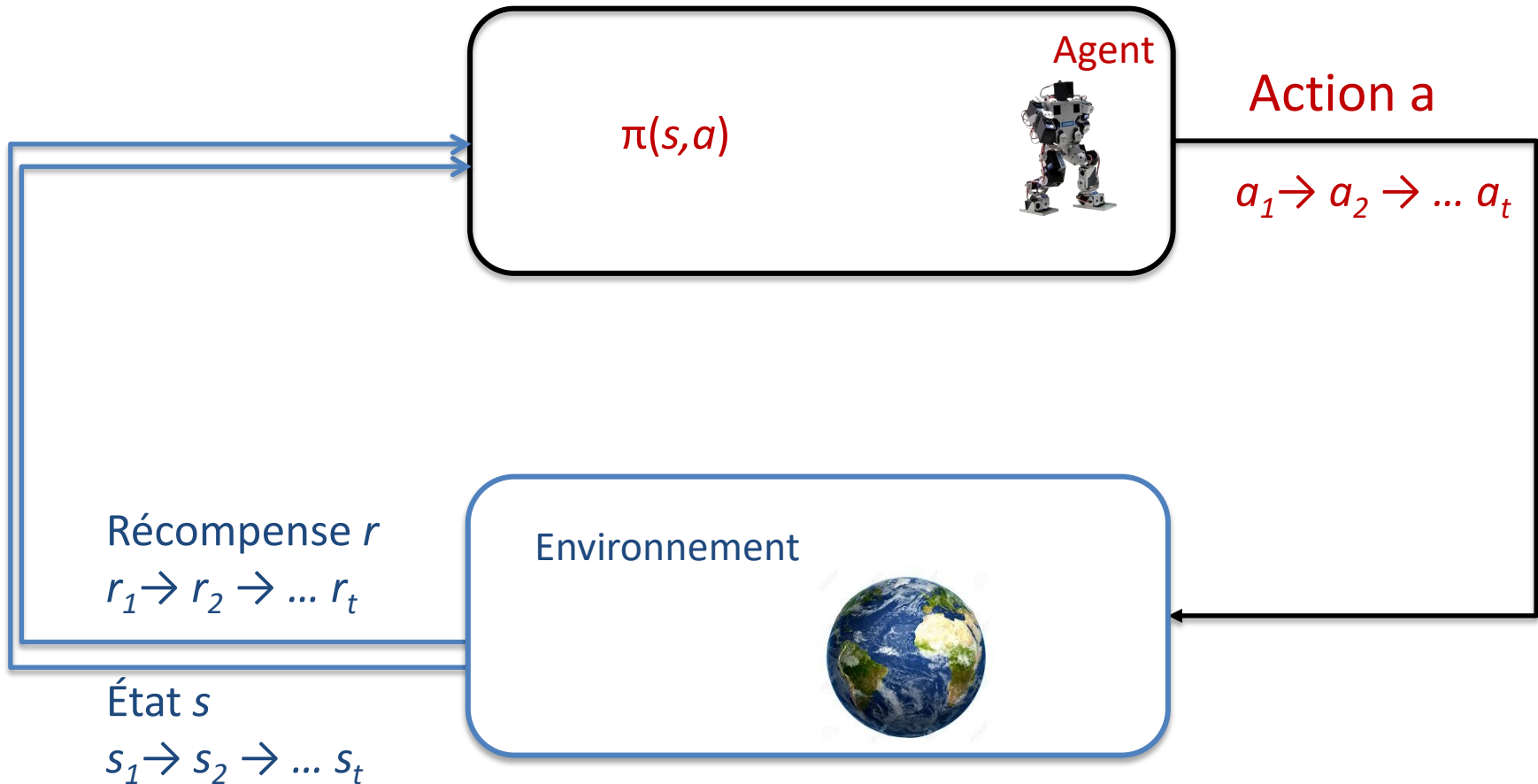
Calculer une politique optimale

- Étant donné $P(s'|s, a)$ et $R(s)$, on peut calculer une politique optimale π par *value-iteration* ou *policy iteration*.



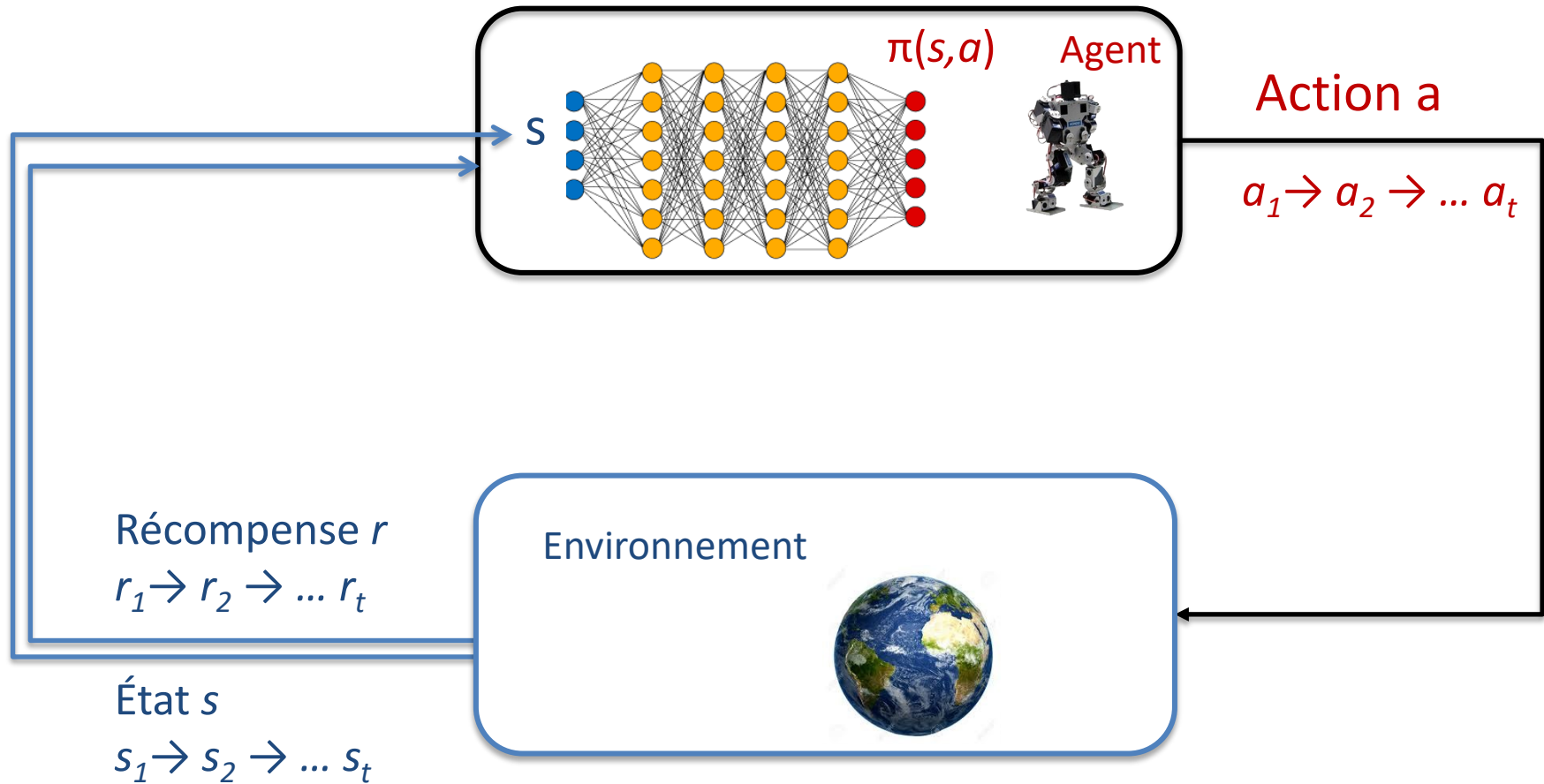
Apprentissage par renforcement

- Sans $P(s'|s, a)$ ou $R(s)$, on peut apprendre une politique optimale π par *l'apprentissage par renforcement*. Dans ce cours, on voit *Q-Learning et SARSA*



Apprentissage par renforcement profond

- Avec l'apprentissage par renforcement profond, la politique est représentée par un réseau de neurones.



Exemple – AlphaGo Zero



Example – ChatGPT

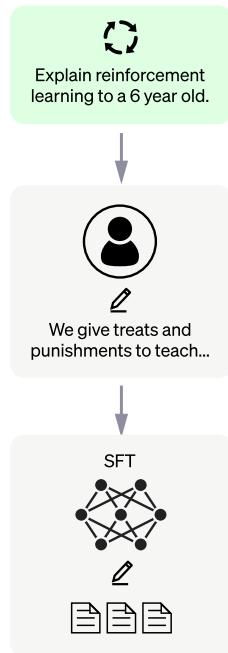
Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.



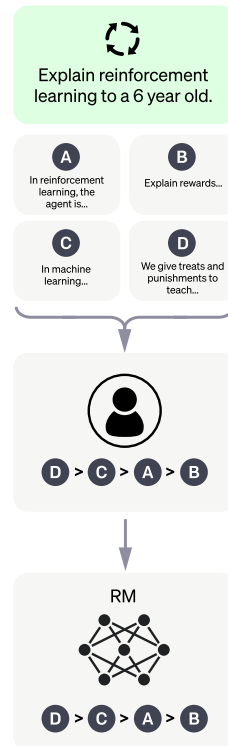
Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

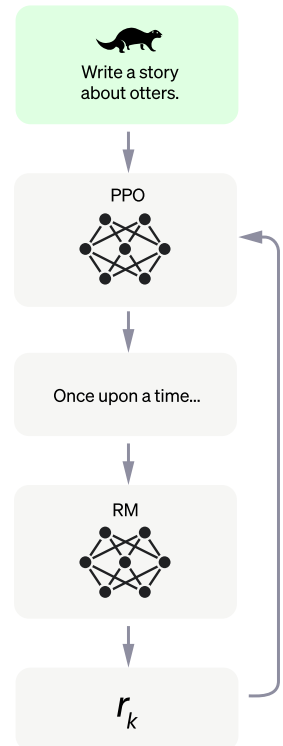
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

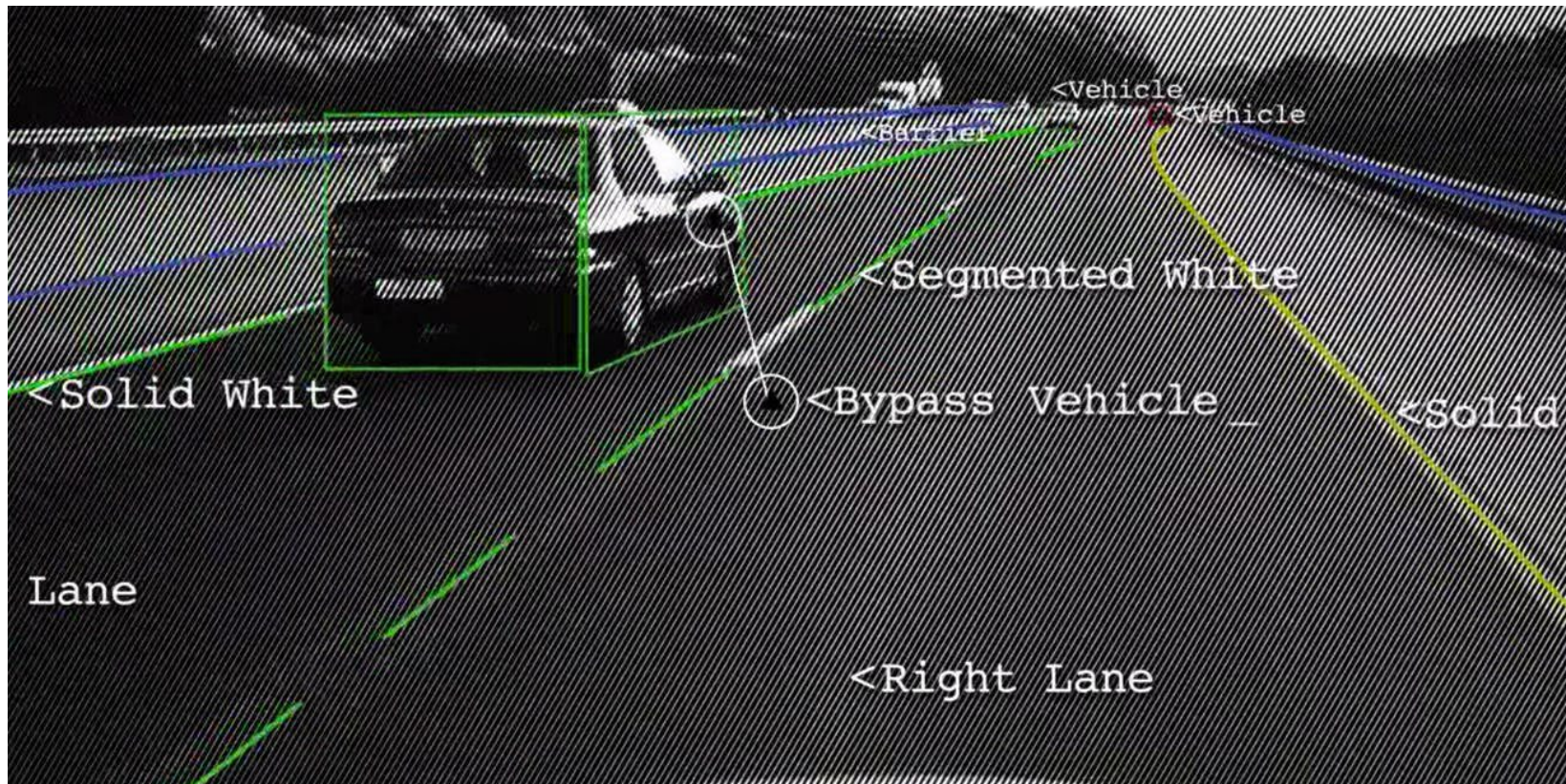
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Source: <https://openai.com/blog/chatgpt/>

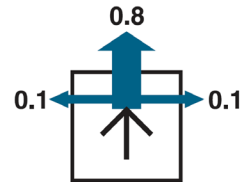
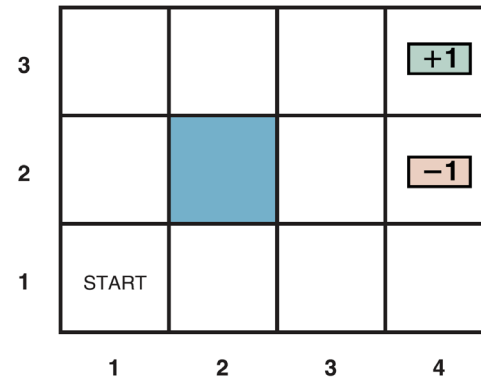
Exemple – Voitures autonomes



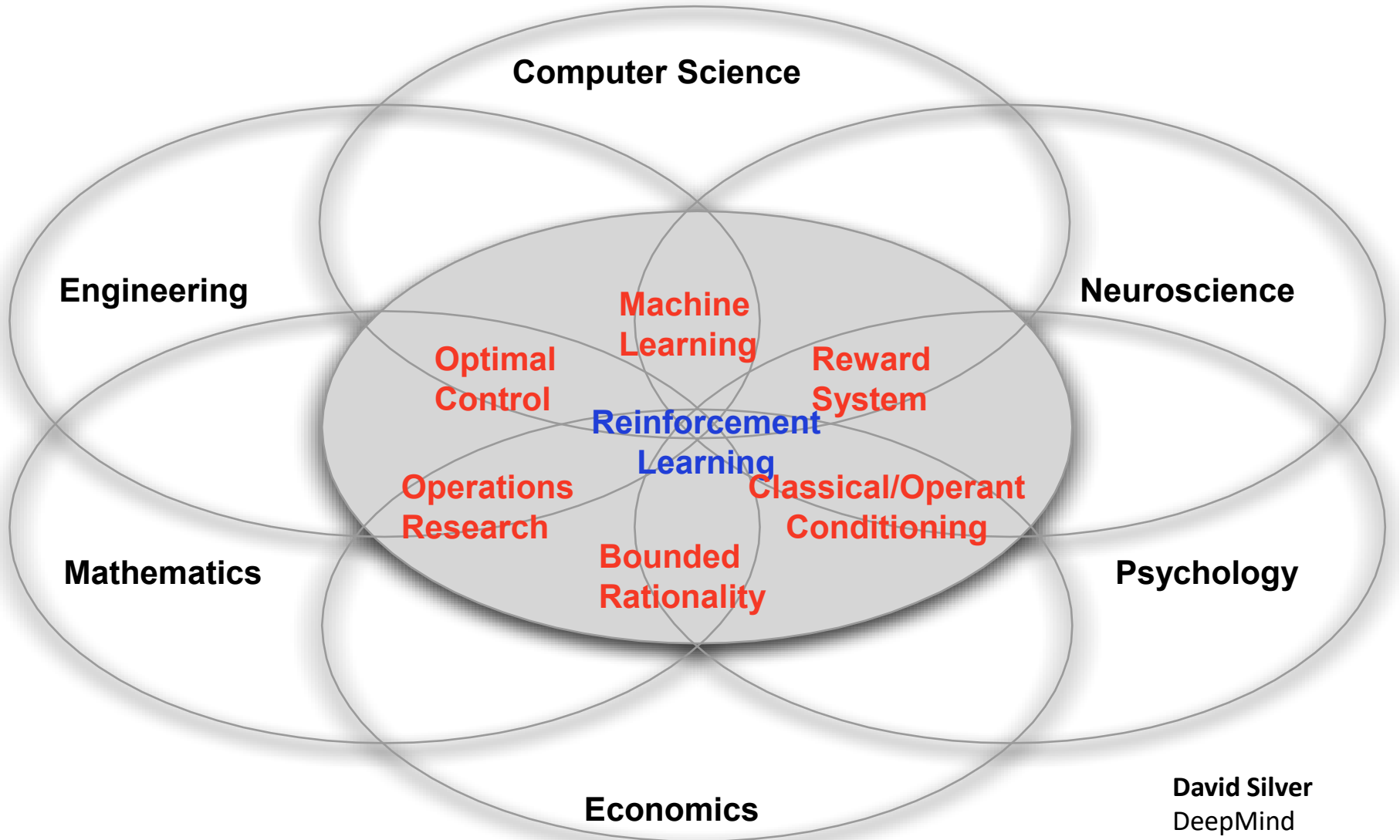
Inverse Skunkworks

Exemple académique

- L'agent vit dans un environnement représenté par une grille 3 x 4
- Des murs bloquent le chemin de l'agent
- L'environnement est stochastique
- L'agent reçoit des récompenses:
 - 0.04 partout, sauf dans (4,3) où il reçoit +1 et dans (4,2) où il reçoit -1
- L'agent ne connaît pas à priori le modèle de l'environnement $P(s'|s,a)$
- Ni la fonction de récompense $R(s)$
- Son but est de maximiser la somme des récompenses



Problème lié à plusieurs domaines



David Silver
DeepMind

Caractéristiques principales

Qu'est-ce qui distingue l'apprentissage par renforcement d'autres paradigmes d'apprentissage automatique:

- ◆ Il n'y a pas de superviseur, mais plutôt seulement un signal de récompense
- ◆ La récompense peut être retardée; elle n'est pas forcément instantanée
- ◆ Le temps est important : données séquentielles par oppositions aux données indépendantes et identiquement distribuées (*i.i.d. data*)
- ◆ Les actions de l'agent influencent les données futurs que l'agent va recevoir (observer)

Apprentissage par renforcement

- Nous avons encore un processus de décision de Markov
 - ◆ Nous cherchons encore une politique π qui maximise (approximativement) la somme des récompenses espérée.
- Mais nous ne connaissons pas le modèle d'action $P(s'|s, a)$ ou la fonction de récompense $R(s)$
 - ◆ **Nous ne savons pas quelles actions choisir**
 - ◆ **Nous ne savons pas quels états sont bons**
- En fait, il faudra tâtonner (essayer des actions) pour apprendre
- En résumé, l'apprentissage par renforcement vise aussi à **trouver un plan optimal, mais sans connaître le modèle de transition de l'environnement**
- Certains diront que c'est la forme la plus pure d'apprentissage en IA
 - ◆ c'est aussi une des plus difficiles à réussir...

Dans ce cours...

- On va se concentrer sur le cas
 - ◆ d'un environnement parfaitement observable
 - ◆ d'un environnement stochastique (le résultat d'une action n'est pas déterministe)
 - ◆ où on ne connaît pas l'environnement (c.-à-d. c'est un MDP inconnu)
 - ◆ où on peut ne pas connaître la fonction de renforcement $R(s)$
- Après ce cours, vous connaîtrez les notions de bases sur
 - ◆ apprentissage par renforcement passif
 - ◆ apprentissage par renforcement actif
 - ◆ dilemme exploration vs. exploitation

Deux cas considérés

- Apprentissage passif
 - ◆ L'agent a une ***politique fixe***
 - ◆ Essaie d'***apprendre l'utilité des états*** en observant l'occurrence des états
 - ◆ Similaire à l'évaluation d'une politique
 - » Sert souvent de composante à l'apprentissage active
 - » Inspire souvent des algorithmes d'apprentissage active
- Apprentissage actif
 - ◆ L'agent ***essaie de trouver une bonne politique*** en agissant dans l'environnement
 - ◆ Similaire à résoudre le Processus de Décision de Markov (PDM) sous-jacent à cet environnement
 - » Mais sans avoir le modèle correspondant

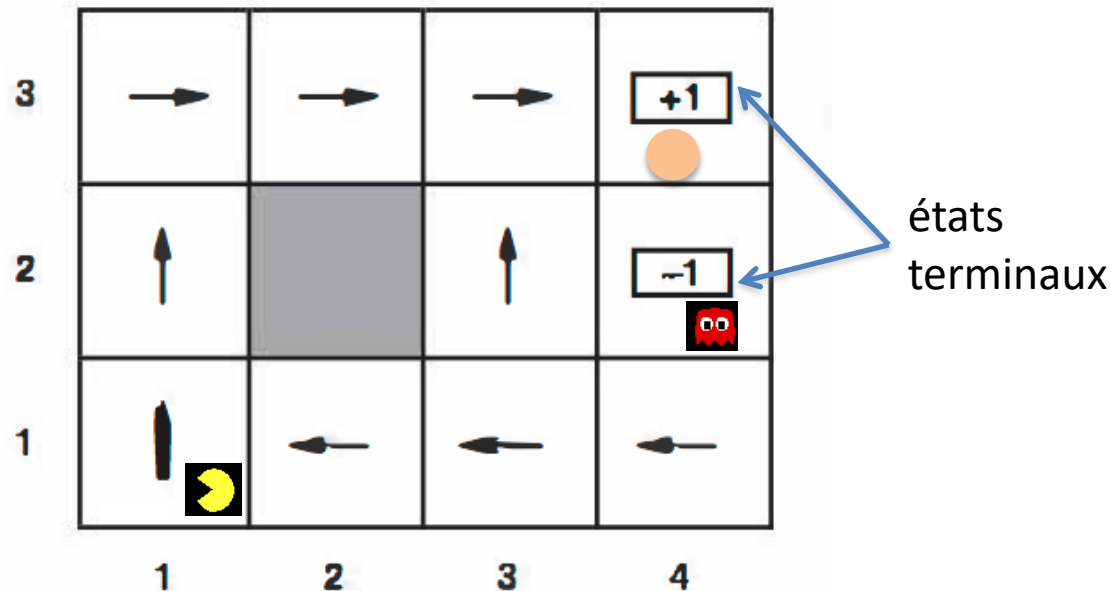
Apprentissage passif

- Apprentissage passif
 - ◆ L'agent a une *politique fixe*
 - ◆ Essaie d'*apprendre l'utilité des états* en observant l'occurrence des états
 - ◆ Similaire à l'évaluation d'une politique

Apprentissage passif

- **Définition:** soit un plan π donné, apprendre la fonction de valeur sans connaître $P(s'|s, a)$
- **Exemple illustratif:** déplacement sur une grille 3 x 4

- ◆ plan π illustré par les flèches
- ◆ $R(s) = -0.04$ partout sauf aux états terminaux
- ◆ l'environnement est stochastique
- ◆ l'agent arrête aux états terminaux
- ◆ on suppose $\gamma=1$

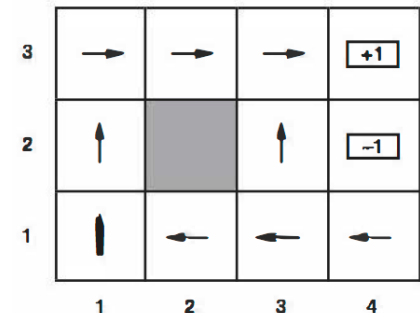


Apprentissage passif

- **Définition:** soit un plan π donné, apprendre la fonction de valeur sans connaître $P(s'|s, a)$
- Puisqu'on ne connaît pas $P(s'|s, a)$ on doit apprendre à partir d'**essais**

1. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
2. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{-.04} (3,2) \xrightarrow[\text{Up}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
3. $(1,1) \xrightarrow[\text{Up}]{-.04} (2,1) \xrightarrow[\text{Left}]{-.04} (3,1) \xrightarrow[\text{Up}]{-.04} (3,2) \xrightarrow[\text{Right}]{-1} (4,2)$

- Comment estimer la fonction de valeurs $U(s)$ à partir de ces essais?
- Trois approches:
 - ◆ *Estimation directe*
 - ◆ *Programmation dynamique adaptative*
 - ◆ *Différence temporelle*



Approche par estimation directe

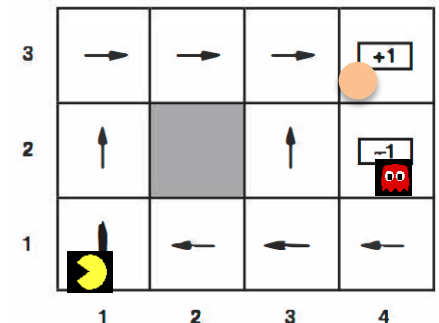
- Approche la plus simple: **calculer la moyenne de ce qui est observé** dans les essais

- **Essais**

1. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
2. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{-.04} (3,2) \xrightarrow[\text{Up}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
3. $(1,1) \xrightarrow[\text{Up}]{-.04} (2,1) \xrightarrow[\text{Left}]{-.04} (3,1) \xrightarrow[\text{Up}]{-.04} (3,2) \xrightarrow[\text{Right}]{-1} (4,2)$

- Estimation de $U^\pi((1,1))$

- ◆ dans l'essai 1, la somme des récompenses à partir de (1,1) est 0.76
- ◆ dans l'essai 2, on obtient également 0.76
- ◆ dans l'essai 3, on obtient plutôt -1.12
- ◆ l'estimation directe de $U((1,1))$ est donc $(0.76+0.76-1.12)/3 \approx 0.133$



Approche par estimation directe

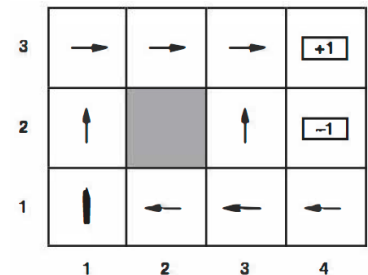
- Approche la plus simple: **calculer la moyenne de ce qui est observé** dans les essais

- **Essais**

1. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
2. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{-.04} (3,2) \xrightarrow[\text{Up}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
3. $(1,1) \xrightarrow[\text{Up}]{-.04} (2,1) \xrightarrow[\text{Left}]{-.04} (3,1) \xrightarrow[\text{Up}]{-.04} (3,2) \xrightarrow[\text{Right}]{-1} (4,2)$

- Estimation de $U^\pi(1,2)$

- ◆ dans l'essai 1, l'état (1,2) est visité deux fois, avec des sommes de récompenses à partir de (1,2) de 0.8 et 0.88
- ◆ dans l'essai 2, on observe 0.8
- ◆ l'essai 3 ne visite pas (1,2)
- ◆ l'estimation directe de $U^\pi(1,2)$ est donc $(0.8+0.88+0.8)/3 \approx 0.826$



Approche par estimation directe

- Essais

1. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-.04} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
2. $(1,1) \xrightarrow[\text{Up}]{-.04} (1,2) \xrightarrow[\text{Up}]{-.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (2,3) \xrightarrow[\text{Right}]{-.04} (3,3) \xrightarrow[\text{Right}]{-.04} (3,2) \xrightarrow[\text{Up}]{-.04} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$
3. $(1,1) \xrightarrow[\text{Up}]{-.04} (2,1) \xrightarrow[\text{Left}]{-.04} (3,1) \xrightarrow[\text{Up}]{-.04} (3,2) \xrightarrow[\text{Right}]{-1} (4,2)$

- L'estimation directe **ignore les liens entre les états**: elle manque d'occasion d'apprentissage

- ◆ Par exemple, bien que l'essai 1 ne dit rien sur (3,2), elle nous apprend que (3,3) a une valeur élevée
- ◆ On pourrait également déduire que (3,2) a une valeur élevée, puisque (3,2) est adjacent à (3,3)

- Autrement dit, on ignore les contraintes entre les utilités des états, telles que données par l'équation

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Approche par programmation dynamique adaptative

- Idée derrière la **programmation dynamique adaptative (PDA)**
 - ◆ tirer profit des équations de la fonction de valeur pour estimer $U(s)$
- L'approche par PDA **n'apprend pas directement $U(s)$** , mais apprend plutôt le modèle de transition $P(s' | s, a)$
 - ◆ étant donnée une estimation de $P(s' | s, a)$, on peut alors calculer $U^\pi(s)$
 - ◆ on obtient alors notre estimation de $U(s)$
- On peut estimer $P(s' | s, a)$ à partir des fréquences des transitions observées:

$$P(s' | s, a) = \frac{\sum_{\text{essais}} N_{s'|sa}}{\sum_{\text{essais}} N_{sa}}$$

somme sur tous les essais

nb. de transitions de (s, a, s') dans l'essai

nb. de fois que l'action a est choisi dans l'état s dans l'essai

Approche par programmation dynamique adaptative

function PASSIVE-ADP-LEARNER(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r

persistent: π , a fixed policy

mdp, an MDP with model P , rewards R , actions A , discount γ

U , a table of utilities for states, initially empty

$N_{s'|s,a}$, a table of outcome count vectors indexed by state and action, initially zero

s, a , the previous state and action, initially null

if s' is new **then** $U[s'] \leftarrow 0$

if s is not null **then**

increment $N_{s'|s,a}[s, a][s']$

$R[s, a, s'] \leftarrow r$

add a to $A[s]$

$\mathbf{P}(\cdot \mid s, a) \leftarrow \text{NORMALIZE}(N_{s'|s,a}[s, a])$  $\mathbf{P}(t \mid s, a) = N_{s'|s,a}[t, s, a] / N_{sa}[s, a]$

$U \leftarrow \text{POLICYEVALUATION}(\pi, U, mdp)$

$s, a \leftarrow s', \pi[s']$

return a

 Résoudre

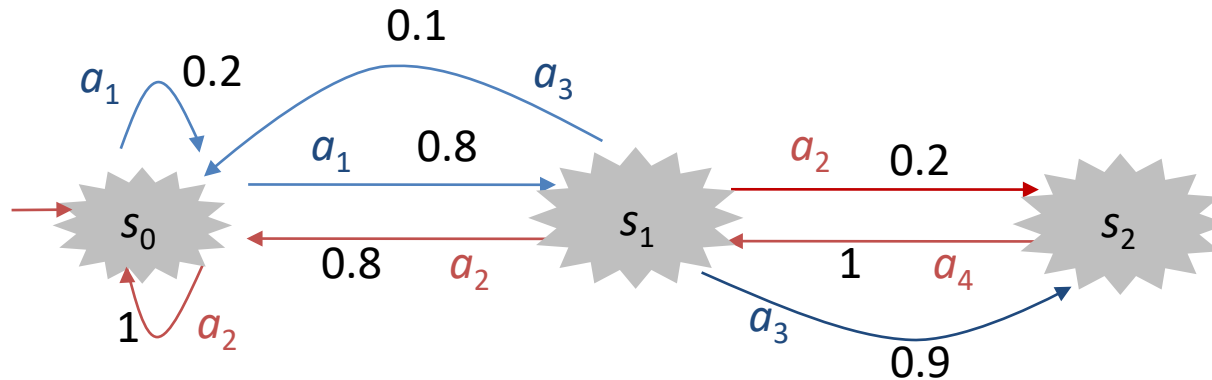
$$U^\pi(s) = \sum_{s' \in S} P(s' \mid s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Ou *modified-policy-iteration*

Ou *asynchronous policy-iteration*

Approche par programmation dynamique adaptative

- Exemple (avec état terminal)



- MDP à 3 états: $S = \{s_0, s_1, s_2\}$
- Fonction de récompense: $R(s_0) = -0.1, R(s_1) = -0.1, R(s_2) = 1$
 - ◆ $R(s, a, s') = R(s')$
- Le facteur d'escompte est $\gamma = 0.5$
- s_2 est un état terminal, s_0 est l'état initial
- Plan suivi: $\pi(s_0) = a_1, \pi(s_1) = a_3$

Approche par programmation dynamique adaptative



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

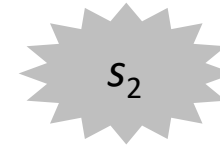
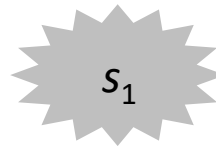
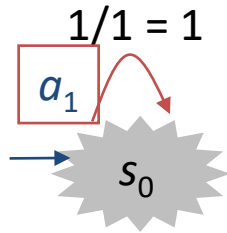
- Initialement, on suppose aucune connection entre les états

$$U(s_0) = 0$$

$$U(s_1) = 0$$

$$U(s_2) = 0$$

Approche par programmation dynamique adaptative



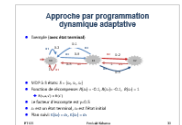
Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0)$

$$U(s_0) = 1 * [-0.1 + 1 * U(s_0)]$$

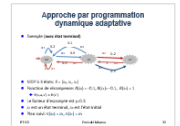
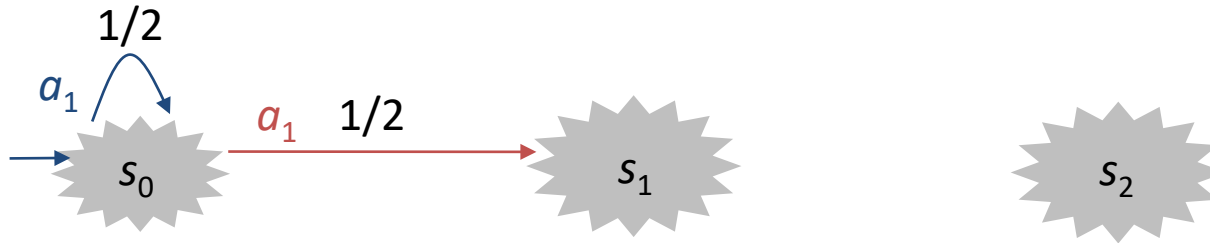
$$U(s_1) = 0$$

$$U(s_2) = 0$$



$$U^\pi(s) = \sum_{s' \in S} P(s' | s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Approche par programmation dynamique adaptative



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1)$

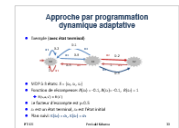
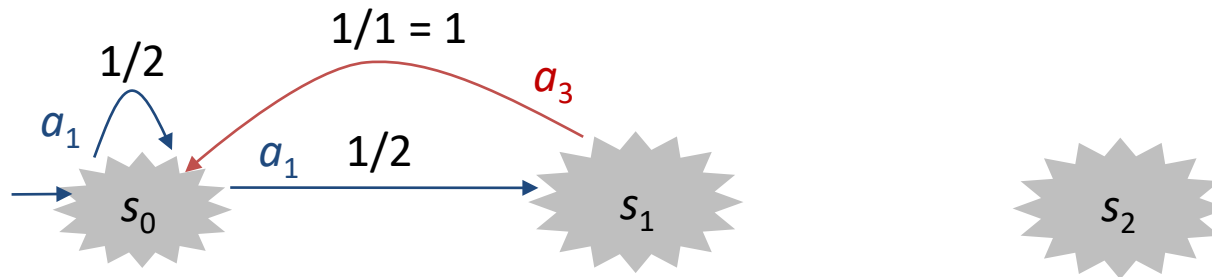
$$U(s_0) = 0.5 [-0.1 + 0.5 U(s_0)] + 0.5 [-0.1 + U(s_1)]$$

$$U(s_1) = 0$$

$$U(s_2) = 0$$

$$U^\pi(s) = \sum_{s' \in \mathcal{S}} P(s' | s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Approche par programmation dynamique adaptative

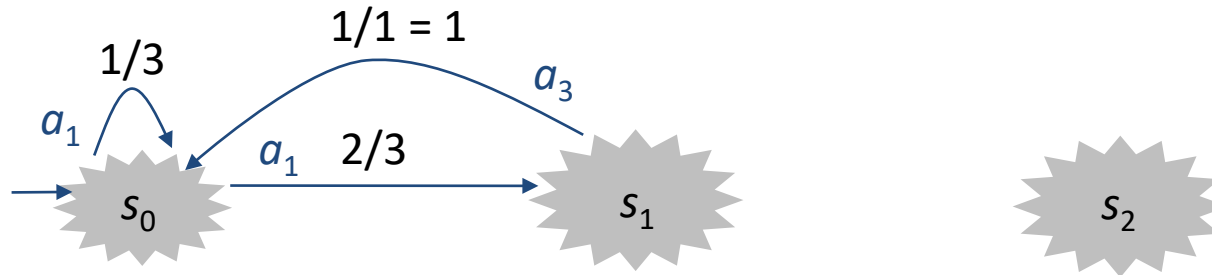


Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_0)$

$$U^\pi(s) = \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

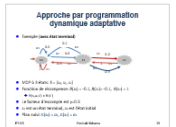
Approche par programmation dynamique adaptative



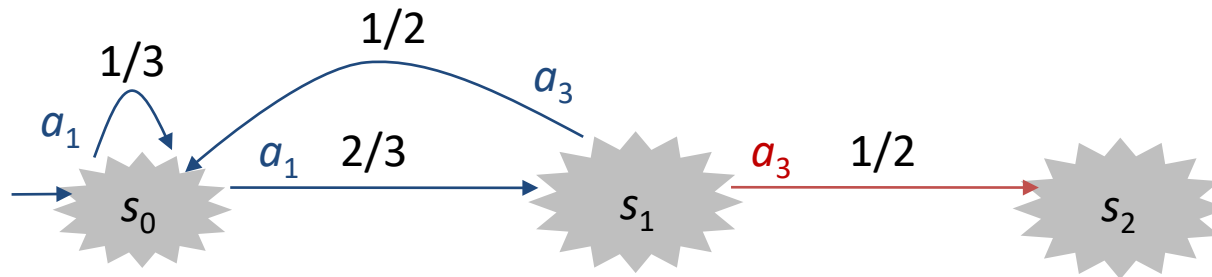
Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_0) \rightarrow (s_1)$

$$U^\pi(s) = \sum_{s' \in \mathcal{S}} P(s' | s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

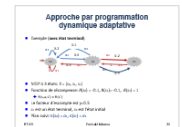


Approche par programmation dynamique adaptative



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

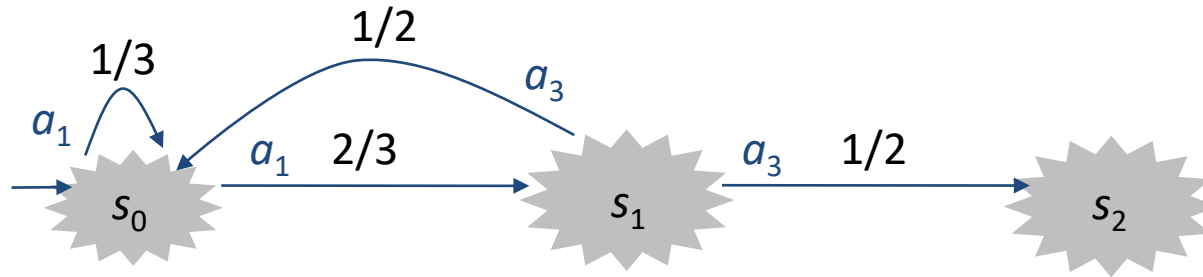
● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_2)$



fin de
l'essai

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Approche par programmation dynamique adaptative

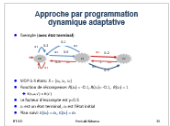


Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_2)$

À tout moment,
on peut calculer
les $U(s)$

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$



fin de
l'essai

Approche par programmation dynamique adaptative

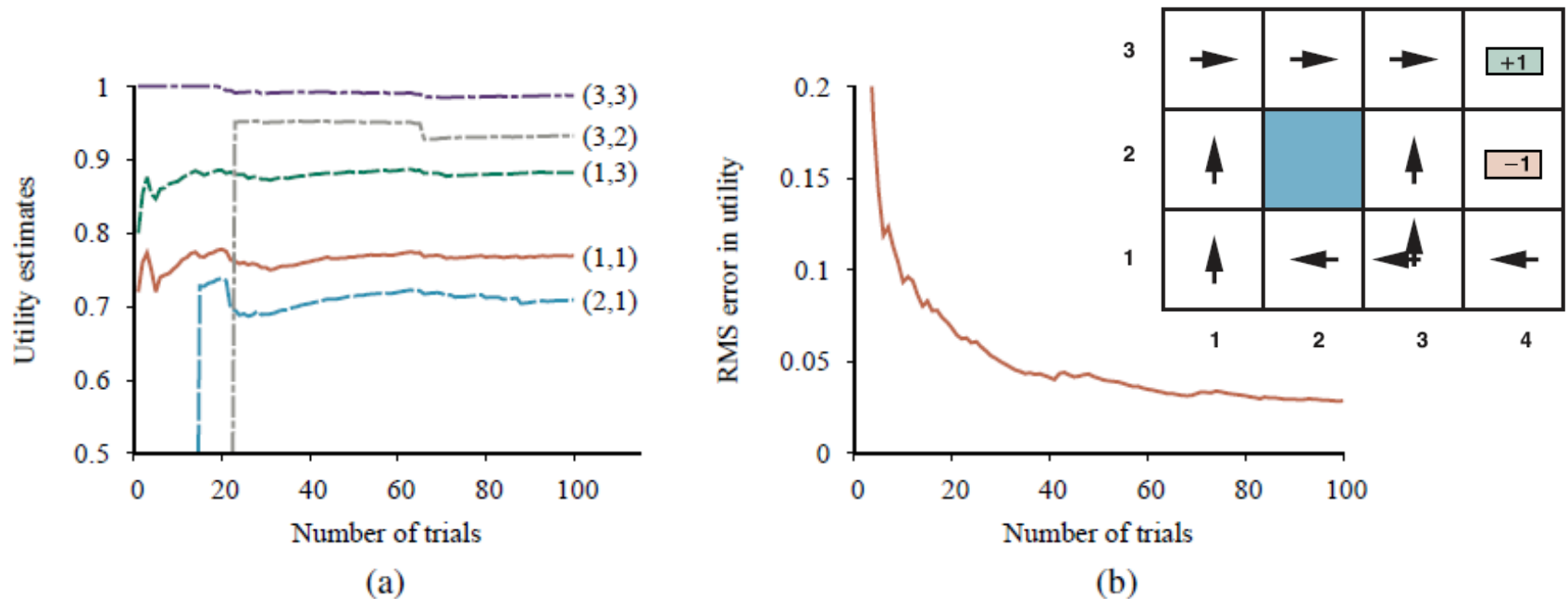


Figure 22.3 The passive ADP learning curves for the 4×3 world, given the optimal policy shown in Figure ?? . (a) The utility estimates for a selected subset of states, as a function of the number of trials. Notice that it takes 14 and 23 trials respectively before the rarely visited states (2,1) and (3,2) “discover” that they connect to the +1 exit state at (4,3). (b) The root-mean-square error (see Appendix A) in the estimate for $U(1, 1)$, averaged over 50 runs of 100 trials each.

Approche par programmation dynamique adaptative

- Contrairement à l'estimation directe, l'approche par PDA peut apprendre après chaque observation, c.-à-d. après chaque transition d'un essai
 - ◆ pas besoin de compléter un essai pour obtenir une nouvelle estimation de $U(s)$
- Parfois, la fonction de récompense n'est pas connue
 - ◆ l'agent ne fait qu'observer la récompense à chaque état, et n'a pas accès directement à la fonction $R(s)$
 - ◆ par contre on a besoin de $R(s)$ dans les équations de la fonction de valeur
 - ◆ dans ce cas, on initialise notre estimation $R(s)$ à 0, et on la met à jour lorsqu'on atteint l'état s pour la première fois

Approche par programmation dynamique adaptative

- Un problème avec l'approche par PDA est qu'on **doit mettre à jour toutes les valeurs de $U(s)$, pour tout s , après chaque observation**
 - ◆ très coûteux en pratique si le nombre d'états est grand (ex.: exponentiel)
 - ◆ inutile pour un état s' qui n'est pas atteignable via l'état de la nouvelle observation
- On doit résoudre les équations de $U(s)$ parce qu'on estime $U(s)$ seulement indirectement, via notre estimation de $P(s'|s, a)$
- Serait-il possible d'estimer directement $U(s)$ et tenir compte des interactions entre les valeurs, sans avoir à passer par $P(s'|s, a)$?
- Oui:
 - ◆ L'idée est d'utiliser les transitions pour ajuster les utilité des états observés, afin qu'ils soient en accord avec les équations exprimant l'utilité d'une politique.

Apprentissage par différence temporelle

- Considérons les essais :

$$\begin{aligned}
 1. \quad & (1,1) \xrightarrow[\text{Up}]{-0.04} (1,2) \xrightarrow[\text{Up}]{-0.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-0.04} (1,3) \xrightarrow[\text{Right}]{-0.04} (2,3) \xrightarrow[\text{Right}]{-0.4} (3,3) \xrightarrow[\text{Right}]{+1} (4,3) \\
 2. \quad & (1,1) \xrightarrow[\text{Up}]{-0.04} (1,2) \xrightarrow[\text{Up}]{-0.04} (1,3) \xrightarrow[\text{Right}]{-0.4} (2,3) \xrightarrow[\text{Right}]{-0.04} (3,3) \xrightarrow[\text{Right}]{-0.04} (3,2) \xrightarrow[\text{Up}]{-0.4} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)
 \end{aligned}$$

- Considérons la transition $(1,3) \rightarrow (2,3)$ dans le deuxième essai. On a $U((1,3)) = 0.84$
- Par contre, à l'issue du 1er essai : $U((1,3))=0.92$ et $U((2,3))=0.96$.

Si cette transition avait lieu tout le temps (avec $\gamma=1$), on aurait

$$U((1,3)) = -0.04 + 1 * U((2,3)) = -0.04 + 1 * .96 = 0.92$$

$$\begin{aligned}
 \text{Cela vient de l'équation } U^\pi(s) &= \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')] \\
 &= R(s, \pi(s), s') + \gamma U(s')
 \end{aligned}$$

- Cela suggère que la valeur de 0.84 à l'issue du deuxième essai serait donc un peu trop basse et il conviendrait de l'augmenter pour la rapprocher de 0.92

Apprentissage par différence temporelle

- Si la transition s à s' avait lieu tout le temps, on s'attendrait à ce que

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')] \\ = R(s, \pi(s), s') + \gamma U(s')$$

- À chaque observation, on peut effectuer la mise à jour suivante pour rapprocher $U(s)$ de $R(s, \pi(s), s') + \gamma U(s')$:

◆ $U^\pi(s) \leftarrow (1-\alpha) U(s) + \alpha [R(s, \pi(s), s') + \gamma U^\pi(s')]$
où α est un **taux d'apprentissage**, entre 0 et 1

Rapprochement de différentes règles d'apprentissage

- Règle d'apprentissage par renforcement passif avec la différence temporelle:
pour chaque transition (s, s') , on a des observations
 $(R) \leftarrow (R) + \alpha [R + \gamma U(s') - U(s)]$
- Algorithme du Perceptron: pour chaque paire $(x_i, y_i) \in D$
a. calculer $h_{w_i}(x_i) = \text{Perceptron}(w_i, x_i)$
b. $w_i \leftarrow w_i + \alpha (y_i - h_{w_i}(x_i)) x_i$ (pour i pour des paires et tout)
- Descente du gradient:
 $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(y_i, h_{w_i}(x_i))$
Nouvelle valeur Ancienne valeur Gradient de l'erreur

IFT615 Froduald Kabanza 47

- On obtient ainsi la règle d'apprentissage **par différence temporelle** (*temporal difference*) ou **TD**

$$U(s) \leftarrow U(s) + \alpha [R + \gamma U(s') - U(s)]$$

Apprentissage par différence temporelle

function PASSIVE-TD-LEARNER(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r

persistent: π , a fixed policy

s , the previous state, initially null

U , a table of utilities for states, initially empty

N_s , a table of frequencies for states, initially zero

if s' is new **then** $U[s'] \leftarrow 0$

if s is not null **then**

increment $N_s[s]$

$U[s] \leftarrow U[s] + \alpha(N_s[s]) \times (r + \gamma U[s'] - U[s])$

$s \leftarrow s'$

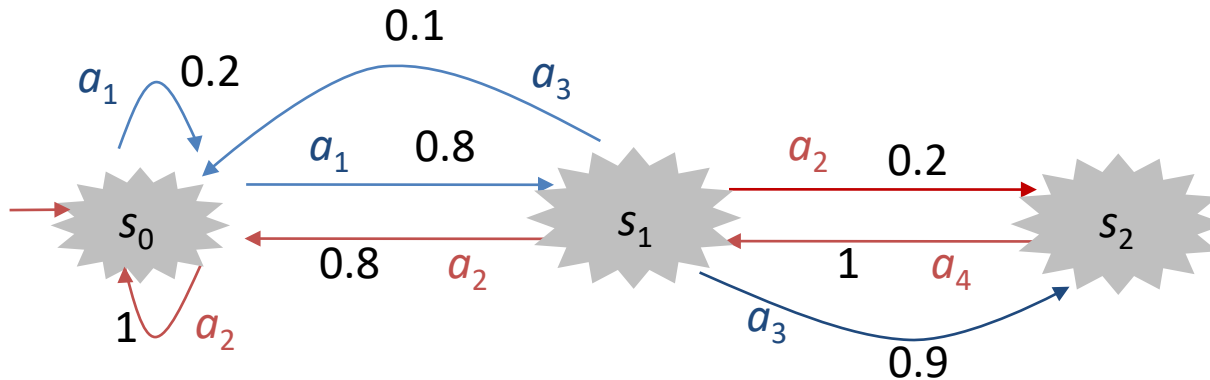
return $\pi[s']$

Nécessaire pour varier le taux d'apprentissage



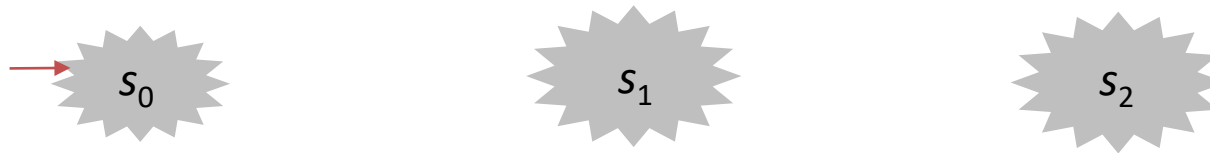
Approche par différence temporelle

- Exemple (avec état terminal)



- MDP à 3 états: $S = \{s_0, s_1, s_2\}$
- Fonction de récompense: $R(s_0) = -0.1, R(s_1) = -0.1, R(s_2) = 1$
 - ◆ $R = R(s, a, s') = R(s)$
- Le facteur d'escompte est $\gamma = 0.5$
- s_2 est un état terminal, s_0 est l'état initial
- Plan suivi: $\pi(s_0) = a_1, \pi(s_1) = a_3$

Apprentissage par différence temporelle



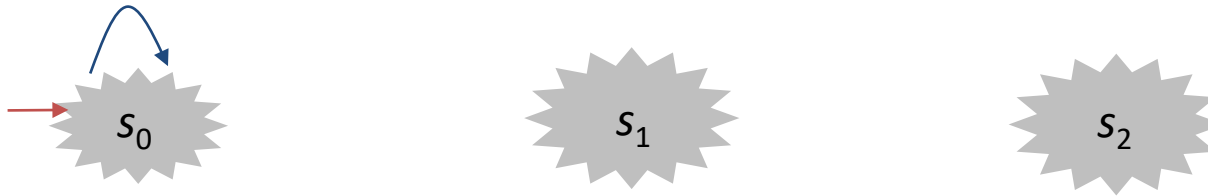
Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R = R(s, a, s') = R(s')$

- Initialisation

$U(s_0) \leftarrow 0$ - Nouvel état

- On va utiliser $\alpha = 0.1$

Apprentissage par différence temporelle



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R = R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0)$

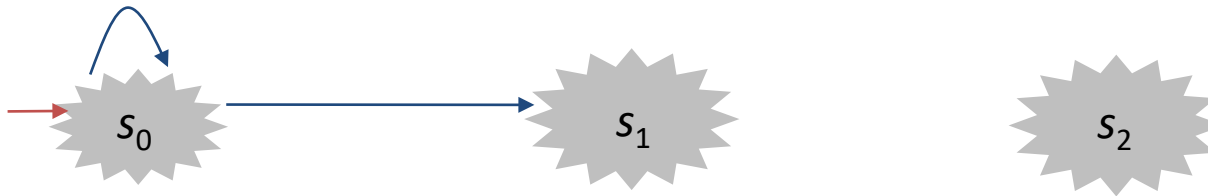
$$\begin{aligned} U(s_0) &\leftarrow U(s_0) + 0.1 (R + 0.5 U(s_0) - U(s_0)) \\ &= 0 + 0.1 (-0.1 - 0.05 \cdot 0 - 0) \\ &= -0.01 \end{aligned}$$

$$U(s_1) = 0$$

$$U(s_2) = 0$$

$$U(s) \leftarrow U(s) + \alpha (R + \gamma U(s') - U(s))$$

Apprentissage par différence temporelle



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R = R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1)$

$U(s_1) \leftarrow 0$ ← parce que s_1 est visité pour la première fois

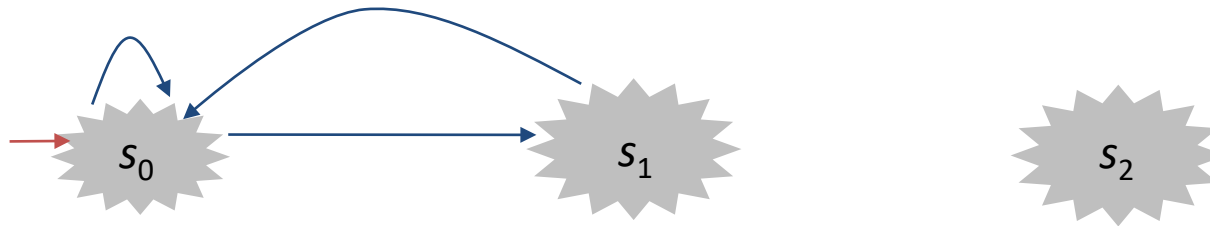
$$\begin{aligned} U(s_0) &\leftarrow U(s_0) + 0.1 (R + 0.5 U(s_1) - U(s_0)) \\ &= -0.01 + 0.1 (-0.1 + 0 + 0.01) \\ &= -0.02 \end{aligned}$$

$$U(s_1) = 0$$

$$U(s_2) = 0$$

$$U(s) \leftarrow U(s) + \alpha (R + \gamma U(s') - U(s))$$

Apprentissage par différence temporelle



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R = R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_0)$

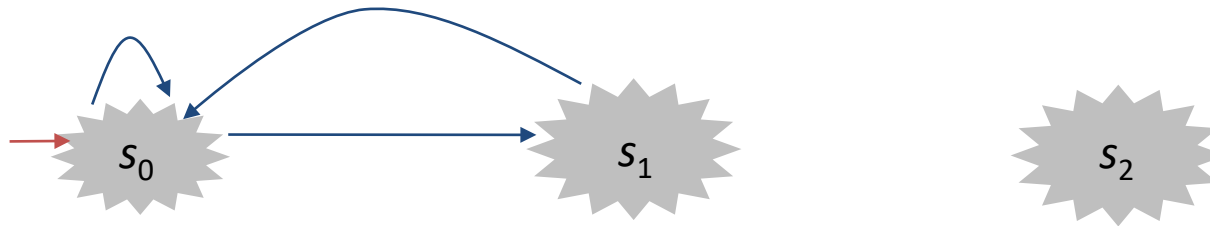
$$U(s_0) = -0.02$$

$$\begin{aligned} U(s_1) &\leftarrow U(s_1) + 0.1 (R + 0.5 U(s_0) - U(s_1)) \\ &= 0 + 0.1 (-0.1 - 0.0095 - 0) \\ &= -0.009 \end{aligned}$$

$$U(s_2) = 0$$

$$U(s) \leftarrow U(s) + \alpha (R + \gamma U(s') - U(s))$$

Apprentissage par différence temporelle



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_0) \rightarrow (s_1)$

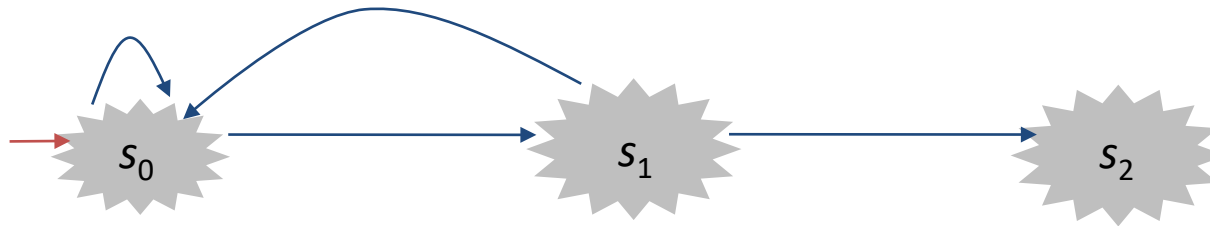
$$\begin{aligned} U(s_0) &\leftarrow U(s_0) + 0.1 (R(s_0) + 0.5 U(s_1) - U(s_0)) \\ &= -0.02 + 0.1 (-0.1 - 0.0045 + 0.02) \\ &= -0.02845 \end{aligned}$$

$$U(s_1) = -0.009$$

$$U(s_2) = 0$$

$$U(s) \leftarrow U(s) + \alpha (R + \gamma U(s') - U(s))$$

Apprentissage par différence temporelle



Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
 $R = R(s, a, s') = R(s')$

● Observations: $(s_0) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_0) \rightarrow (s_1) \rightarrow (s_2)$

fin de
l'essai

$U(s_2) \leftarrow 0$ ← parce que s_2 est visité pour la première fois

$$U(s_0) = -0.02845$$

$$\begin{aligned} U(s_1) &\leftarrow U(s_1) + 0.1 (R + 0.5 U(s_2) - U(s_1)) \\ &= -0.009 + 0.1 (1 + 0 + 0.009) \\ &= 0.0919 \end{aligned}$$

$$U(s_2) = 0$$

$$U(s) \leftarrow U(s) + \alpha (R + \gamma U(s') - U(s))$$

Apprentissage par différence temporelle

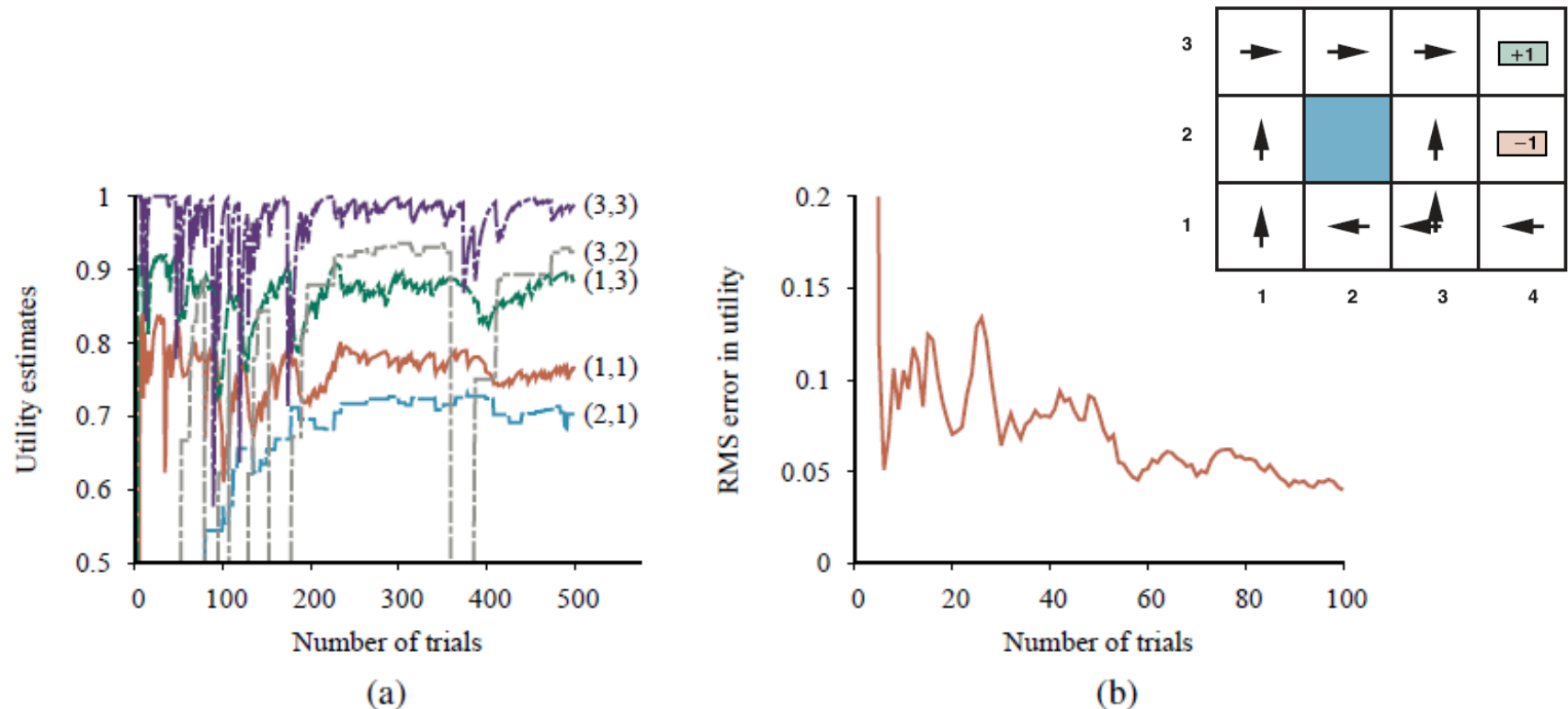
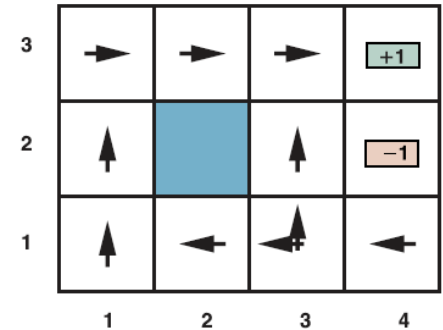
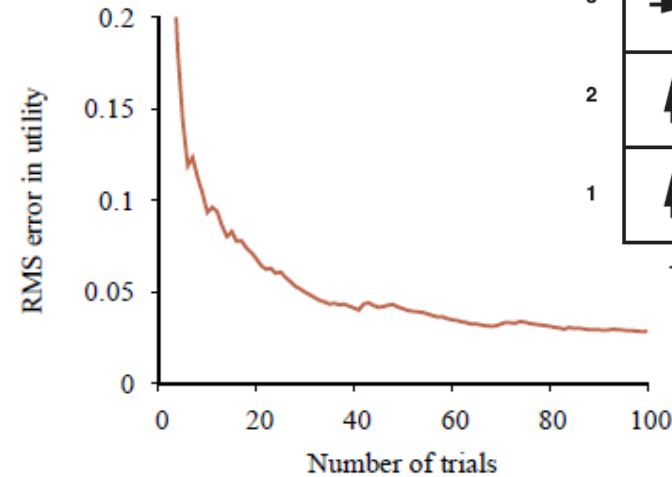
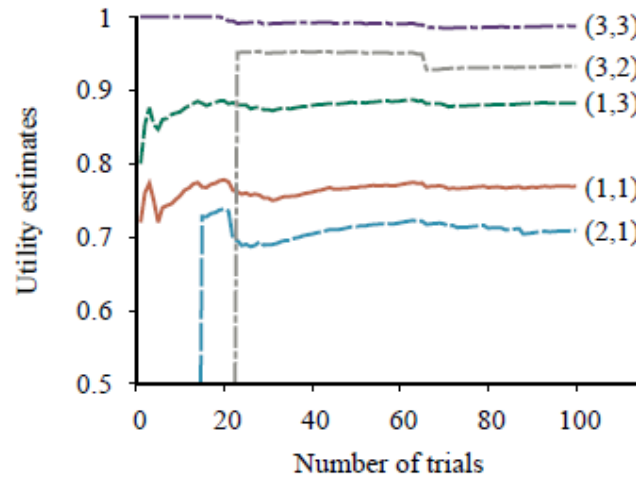


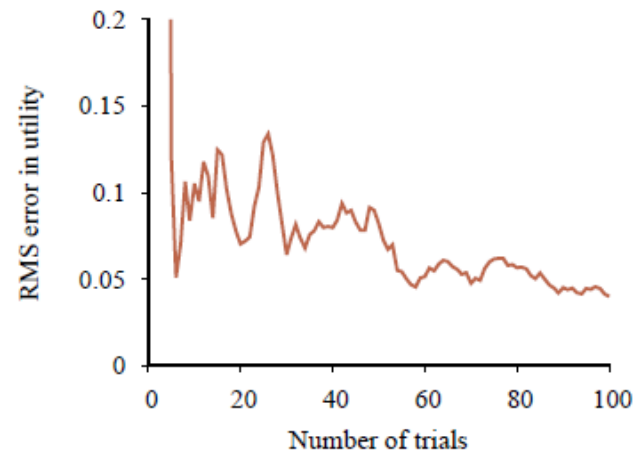
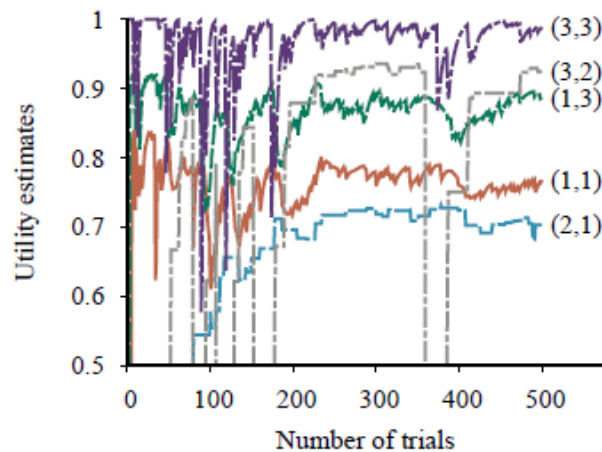
Figure 22.5 The TD learning curves for the 4×3 world. (a) The utility estimates for a selected subset of states, as a function of the number of trials, for a single run of 500 trials. Compare with the run of 100 trials in Figure ??(a). (b) The root-mean-square error in the estimate for $U(1, 1)$, averaged over 50 runs of 100 trials each.

Programmation dynamique adaptative vs différence temporelle

ADP



TD



Apprentissage par renforcement actif

- Dans le cas **passif**, l'agent a un plan (politique) fixe qui détermine son comportement.
 - ◆ Il doit apprendre la fonction d'utilité associée au plan
- Dans le cas **actif**, l'agent doit décider quelles actions effectuer
 - ◆ L'agent doit apprendre un modèle complet avec les probabilités des effets de ses actions au lieu du modèle associé à une politique fixe.
 - ◆ **$U(s)$ est maintenant une estimation de la fonction d'utilité du plan optimal**
- Un algorithme d'apprentissage naïf peut être obtenu de l'algorithme PDA-Passif par deux changements simples:
 - ◆ On applique **value iteration** au MDP estimé (afin de résoudre les équations pour la **politique optimale**)
 - ◆ L'action choisie par l'agent devient

$$\pi(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) U(s')$$



Apprentissage actif avec PDA vorace

ACTIVE-GLOUTON

function ~~PASSIVE~~-ADP-LEARNER(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r

persistent: π , a fixed policy

mdp, an MDP with model P , rewards R , actions A , discount γ

U , a table of utilities for states, initially empty

$N_{s'|s,a}$, a table of outcome count vectors indexed by state and action, initially zero

s, a , the previous state and action, initially null

if s' is new **then** $U[s'] \leftarrow 0$

if s is not null **then**

increment $N_{s'|s,a}[s, a][s']$

$R[s, a, s'] \leftarrow r$

add a to $A[s]$

$P(\cdot | s, a) \leftarrow \text{NORMALIZE}(N_{s'|s,a}[s, a])$

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$ Optimal-Policy(U, mdp)
(value-iteration ou policy-iteration)

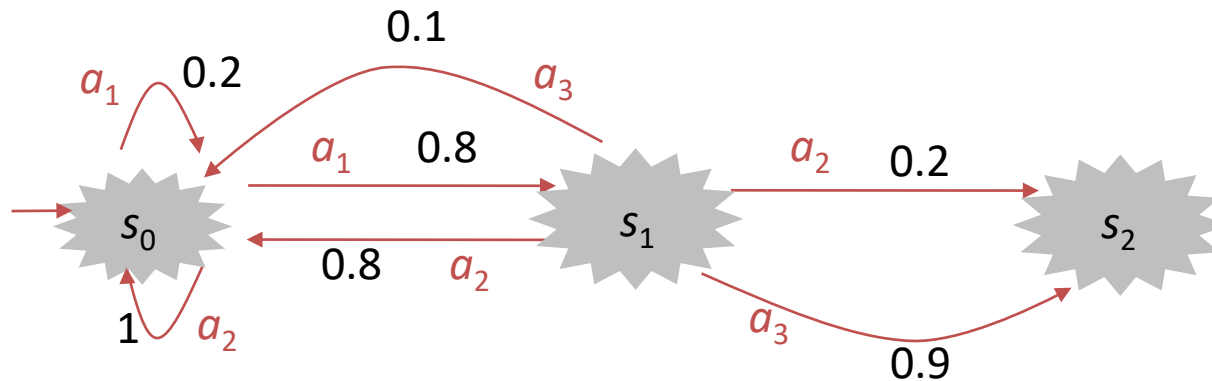
$s, a \leftarrow s', \pi[s']$

return a

$$\underset{a \in A(s)}{\operatorname{argmax}} \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

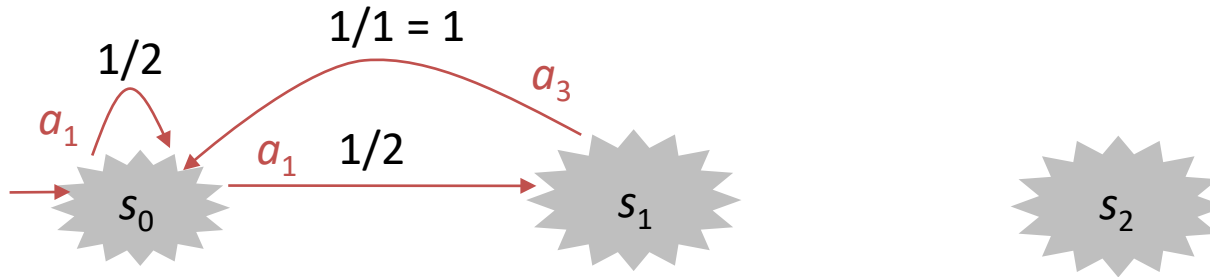
Apprentissage actif avec PDA vorace

- Rappel de l'exemple




- On a des actions possibles différentes, pour chaque état
 - ◆ $A(s_0) = \{a_1, a_2\}$
 - ◆ $A(s_1) = \{a_2, a_3\}$
 - ◆ $A(s_2) = \{\}$

Approche par programmation dynamique adaptative vorace



- Observations: $(s_0) \xrightarrow[a_1]{-0.2} (s_0) \xrightarrow[a_1]{-0.1} (s_1) \xrightarrow[a_3]{-0.1} (s_0)$

value iteration  $U(s_0)=-0.1$
 $U(s_1)=-0.1$
 $U(s_2)=0$

!! Calculs à verifier

- Pour choisir quelle action prendre dans S_0 , on compare
 - ◆ $\sum_{s' \in S} P(s|s', a_2) [.] = 0$ (puisque $P(s|s', a_2)$ pas apprise encore pour a_2)
 - ◆ $\sum_{s' \in S} P(s|s', a_1) [R(s_0, a, s) + 0.5 U(s')] = -0.1$
- L'action choisie par l'agent est donc a_2

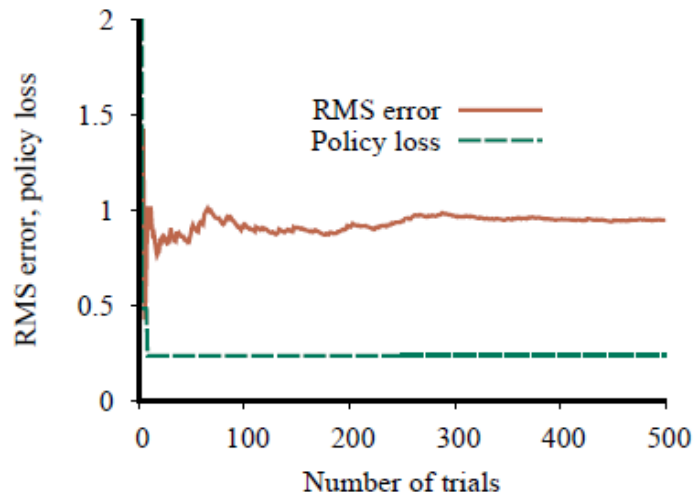
Limite du PDA vorace

- L'approche précédente est dite **vorace (gloutonne)**
 - ◆ elle choisit l'action avec la meilleure utilité espérée étant donné le modèle appris jusqu'à maintenant.
 - ◆ en d'autres mots, **elle exploite le plus possible** l'information recueillie jusqu'à maintenant
- Les approches voraces trouvent rarement le plan optimal
 - ◆ elles ne tiennent pas compte du fait que l'**information accumulée jusqu'à maintenant est partielle**
 - ◆ en d'autres mots, elles ne considèrent pas la possibilité d'**explorer l'environnement** plus longuement, pour amasser plus d'information sur celui-ci

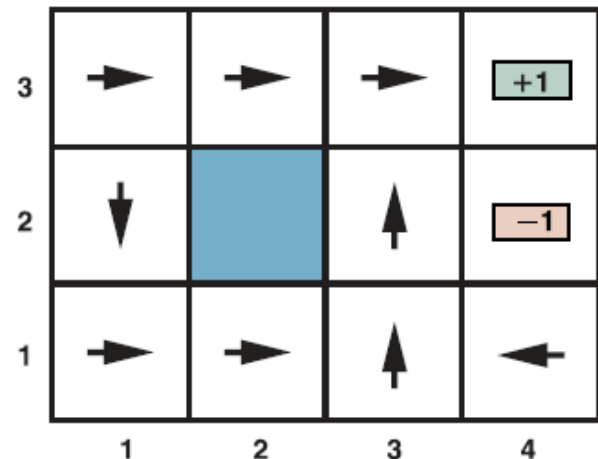
Limite du PDA vorace

- L'algorithme PDA vorace ne garantit pas d'apprendre le plan optimal
- Dans cet exemple (Fig. 21.6 du livre):
 - ◆ Au bout du 3^e essai, une politique menant à la récompense +1 est apprise via $(2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (4,1)$
 - ◆ Au bout du 8^e essai, le plan ci-après est appris.

Courbe d'apprentissage



Plan découvert



Dilemme exploration vs. exploitation

- **Trop exploiter** mène à des plans non optimaux
- **Trop explorer** ralentit l'apprentissage inutilement
- Trouver la balance optimale entre l'exploration et l'exploitation est un problème ouvert en général
- Des stratégies d'exploration/exploitation optimales existent seulement dans des cas très simples
 - ◆ Voir l'algorithme UCB1 pour le **problème de machines à sous (*bandits*) à n leviers** dans le livre, Section 17.3, p. 581

Dilemme exploration vs. exploitation

- On se contente donc d'heuristiques en pratique.
- Exemple: introduction d'une **fonction d'exploration** $f(u,n)$
 - ◆ cette fonction augmente artificiellement la récompense future d'actions inexplorées.
- L'approche par PDA basée sur *value iteration* ferait les mises à jour

$$U^+(s) \leftarrow \max_a f \left(\sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U^+(s')] , N(s,a) \right)$$

où $N(s,a)$ est le nombre de fois que l'action a a été choisie à l'état s et

$$f(u,n) = \begin{cases} R^+ & \text{si } n < N_e \\ u & \text{sinon} \end{cases}$$

Estimation optimiste de récompenses futures (hyper-paramètre)

- Garantit que a sera choisie dans s au moins N_e fois durant l'apprentissage.

Apprentissage actif avec PDA vorace

ACTIVE-GLOUTON

function ~~PASSIVE~~-ADP-LEARNER(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r

persistent: π , a fixed policy

mdp, an MDP with model P , rewards R , actions A , discount γ

U , a table of utilities for states, initially empty

$N_{s'|s,a}$, a table of outcome count vectors indexed by state and action, initially zero

s, a , the previous state and action, initially null

if s' is new **then** $U[s'] \leftarrow 0$

if s is not null **then**

increment $N_{s'|s,a}[s, a][s']$

$R[s, a, s'] \leftarrow r$

add a to $A[s]$

$P(\cdot \mid s, a) \leftarrow \text{NORMALIZE}(N_{s'|s,a}[s, a])$

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$ Optimal-Policy(U, mdp)
(value-iteration ou policy-iteration)

$s, a \leftarrow s', \pi[s']$

return a

$$\underset{a \in A(s)}{\operatorname{argmax}} f\left(\sum_{s' \in S} P(s|s',a) [R(s,a,s') + \gamma U^+(s')] , N(s,a) \right)$$

Exemple revisité

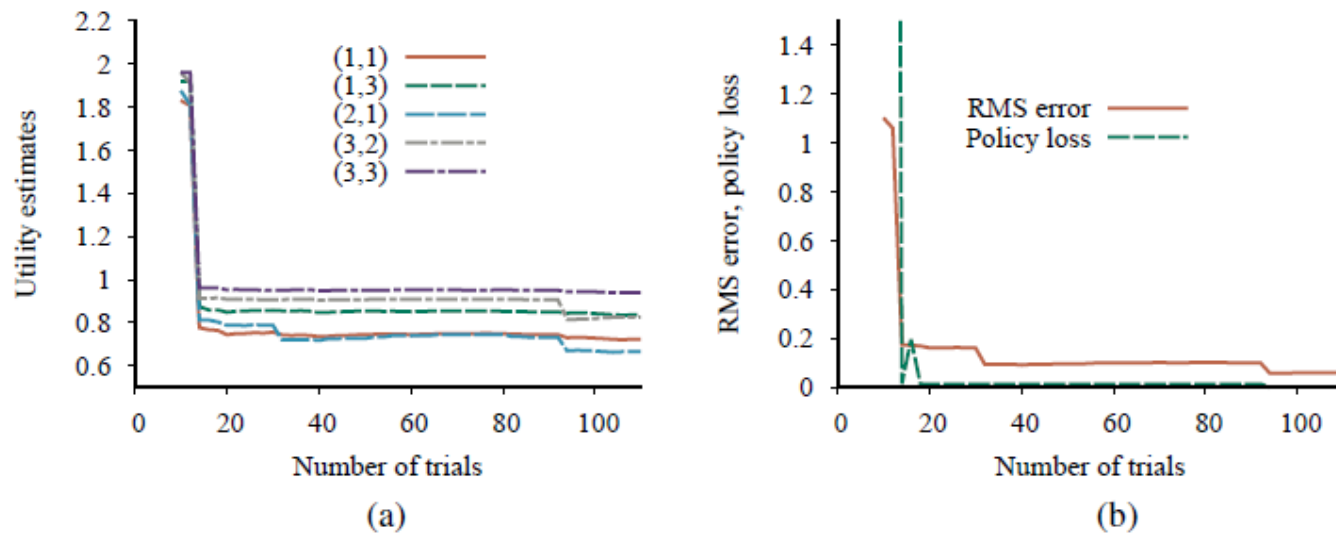


Figure 22.7 Performance of the exploratory ADP agent using $R^+ = 2$ and $N_e = 5$. (a) Utility estimates for selected states over time. (b) The RMS error in utility values and the associated policy loss.

Apprentissage actif par différence temporelle

- Dans le cas de l'apprentissage actif TD, la règle de mise à jour demeure la même que dans le cas de l'apprentissage passif

$$U(s) \leftarrow U(s) + \alpha (R(s) + \gamma U(s') - U(s))$$

- La différence entre TD passif et TD actif est au niveau du choix de l'action.
 - ◆ Pour choisir l'action, l'agent TD devra apprendre le modèle $P(s' | s, a)$.
 - ◆ L'apprentissage du modèle se fait comme pour l'agent PDA actif.

Apprentissage actif avec Q-learning

- Il existe une variante de la méthode TD, nommée ***Q-learning***, qui apprend la **fonction action-valeur** $Q(s,a)$
 - ◆ on n'apprend plus $U(s)$, soit la somme espérée des renforcements à partir de s jusqu'à la fin pour la politique optimale
 - ◆ on apprend plutôt $Q(s,a)$, soit la somme espérée des renforcements à partir de s **et l'exécution de a** , jusqu'à la fin pour la politique optimale
 - ◆ le lien entre $Q(s,a)$ et $U(s)$ est que $U(s) = \max_a Q(s,a)$
- Le plan de l'agent est alors $\pi(s) = \operatorname{argmax}_a Q(s,a)$
- L'avantage par rapport à TD actif est que pour le choix de l'action, on n'a plus besoin d'apprendre le modèle!

Apprentissage actif avec Q-learning

- Selon la définition de $Q(s,a)$, on a

$$Q(s,a) = \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

- Comme pour l'approche TD, on traduit cette équation en la mise à jour de Q-Learning (*off-policy learning*)

$$Q(s,a) \leftarrow Q(s,a) + \alpha (R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

- On voit la similarité avec l'approche TD initiale

$$U(s) \leftarrow U(s) + \alpha (R(s) + \gamma U(s') - U(s))$$

- Q-Learning a un “cousin” appelé SARSA (*on-policy learning*)

$$Q(s,a) \leftarrow Q(s,a) + \alpha (R(s,a,s') + \gamma Q(s',a') - Q(s,a))$$

Apprentissage actif avec Q-learning

function Q-LEARNING-AGENT(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r

persistent: Q , a table of action values indexed by state and action, initially zero

N_{sa} , a table of frequencies for state–action pairs, initially zero

s, a , the previous state and action, initially null

if s is not null **then**

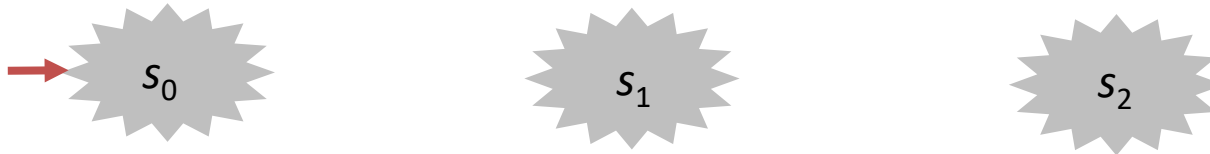
 increment $N_{sa}[s, a]$

$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

$s, a \leftarrow s', \operatorname{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$

return a

Apprentissage actif avec Q-learning



- On va utiliser $\alpha = 0.5$, $\gamma = 0.5$

- Initialisation:

$$Q(s_0, a_1) = 0 \quad Q(s_0, a_2) = 0$$

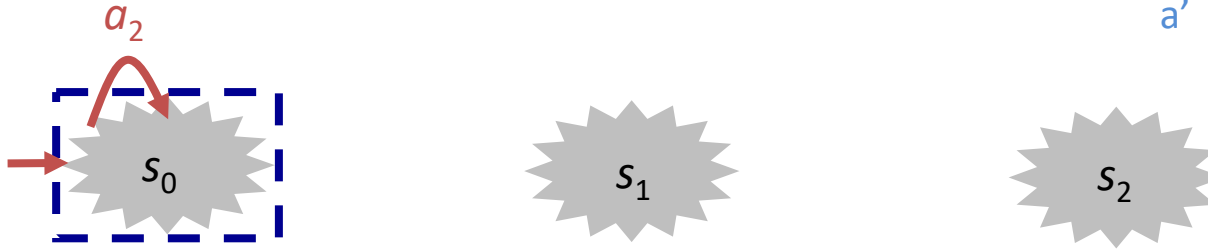
$$Q(s_1, a_2) = 0 \quad Q(s_1, a_3) = 0$$

$$Q(s_2, \text{None}) = 0$$

- Action à prendre $\pi(s_0)$ $= \operatorname{argmax}\{ Q(s_0, a_1), Q(s_0, a_2) \}$
 $= \operatorname{argmax}\{ 0, 0 \}$
 $= a_2$ (arbitraire, ça aurait aussi pu être a_1)

Apprentissage actif avec Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha (R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



- Observations: $(s_0) \xrightarrow[\bar{a}_2]{-0.1} (s_0)$

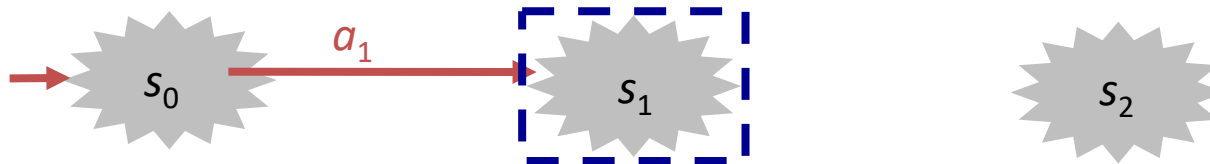
$$\begin{aligned} Q(s_0, a_2) &\leftarrow Q(s_0, a_2) + \alpha (R(s_0, a_2, s_0) + \gamma \max\{ Q(s_0, a_1), Q(s_0, a_2) \} - Q(s_0, a_2)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{ 0, 0 \} - 0) \\ &= -0.05 \end{aligned}$$

- Action à prendre $\pi(s_0)$

$$\begin{aligned} &= \operatorname{argmax}\{ Q(s_0, a_1), Q(s_0, a_2) \} \\ &= \operatorname{argmax}\{ 0, -0,05 \} \\ &= a_1 \quad \textbf{(changement de politique!)} \end{aligned}$$

Apprentissage actif avec Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha (R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

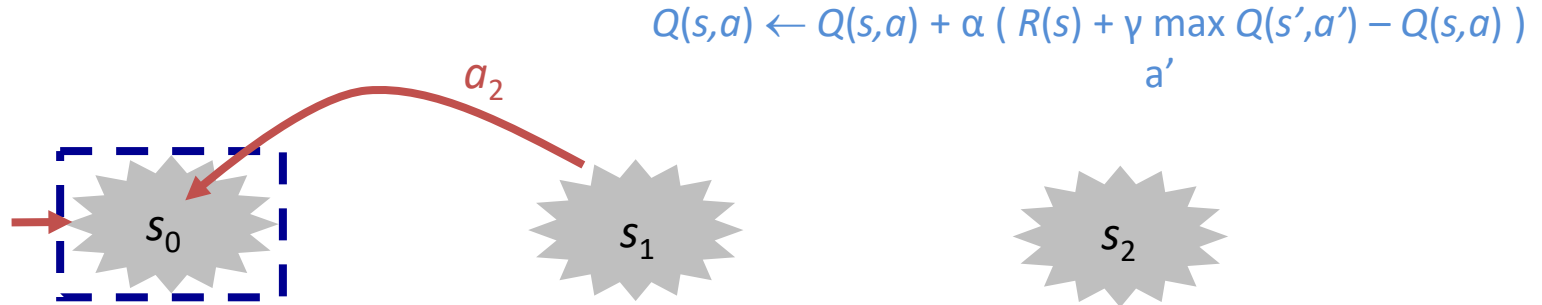


- Observations: $(s_0) \xrightarrow[a_2]{-0.1} (s_0) \xrightarrow[a_1]{-0.1} (s_1)$

$$\begin{aligned} Q(s_0, a_1) &\leftarrow Q(s_0, a_1) + \alpha (R(s_0, a_1, s_1) + \gamma \max\{ Q(s_1, a_2), Q(s_1, a_3) \} - Q(s_0, a_1)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{ 0, 0 \} - 0) \\ &= -0.05 \end{aligned}$$

- Action à prendre $\pi(s_1)$ $= \operatorname{argmax}\{ Q(s_1, a_2), Q(s_1, a_3) \}$
 $= \operatorname{argmax}\{ 0, 0 \}$
 $= a_2$ (arbitraire, ça aurait aussi pu être a_3)

Apprentissage actif avec Q-learning

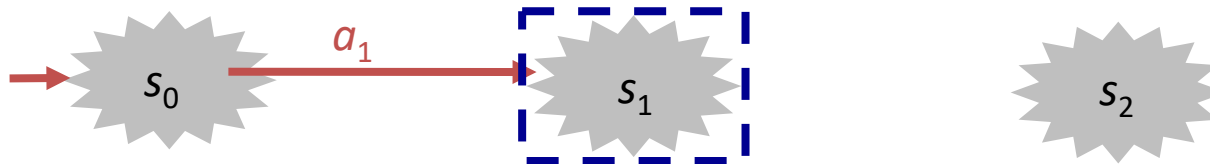


- Observations: $(s_0) \xrightarrow[a_2]{-0.1} (s_0) \xrightarrow[a_1]{-0.1} (s_1) \xrightarrow[a_2]{-0.1} (s_0)$

$$\begin{aligned} Q(s_1, a_2) &\leftarrow Q(s_1, a_2) + \alpha (R(s_1) + \gamma \max\{ Q(s_0, a_1), Q(s_0, a_2) \} - Q(s_1, a_2)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{ -0.05, -0.05 \} + 0) \\ &= -0.0625 \end{aligned}$$

- Action à prendre $\pi(s_0) = \operatorname{argmax}\{ Q(s_0, a_1), Q(s_0, a_1) \}$
 $= \operatorname{argmax}\{ -0.05, -0.05 \}$
 $= a_1$ (arbitraire, ça aurait aussi pu être a_2)

Apprentissage actif avec Q-learning



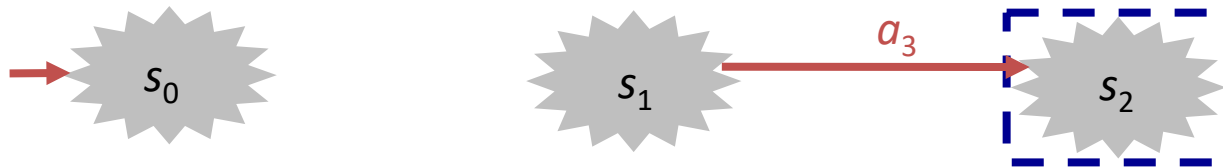
● Observations: $(s_0) \xrightarrow[a_2]{-0.1} (s_0) \xrightarrow[a_1]{-0.1} (s_1) \xrightarrow[a_3]{-0.1} (s_0) \xrightarrow[a_1]{-0.1} (s_1)$

$$\begin{aligned} Q(s_0, a_1) &\leftarrow Q(s_0, a_1) + \alpha (R(s_0) + \gamma \max\{ Q(s_1, a_2), Q(s_1, a_3) \} - Q(s_0, a_1)) \\ &= -0.05 + 0.5 (-0.1 + 0.5 \max\{ -0.0625, 0 \} + 0.05) \\ &= -0.075 \end{aligned}$$

● Action à prendre $\pi(s_1)$ = $\operatorname{argmax}\{ Q(s_1, a_2), Q(s_1, a_3) \}$
 = $\operatorname{argmax}\{ -0.075, 0 \}$
 = a_3 **(changement de politique!)**

Apprentissage actif avec Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha (R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



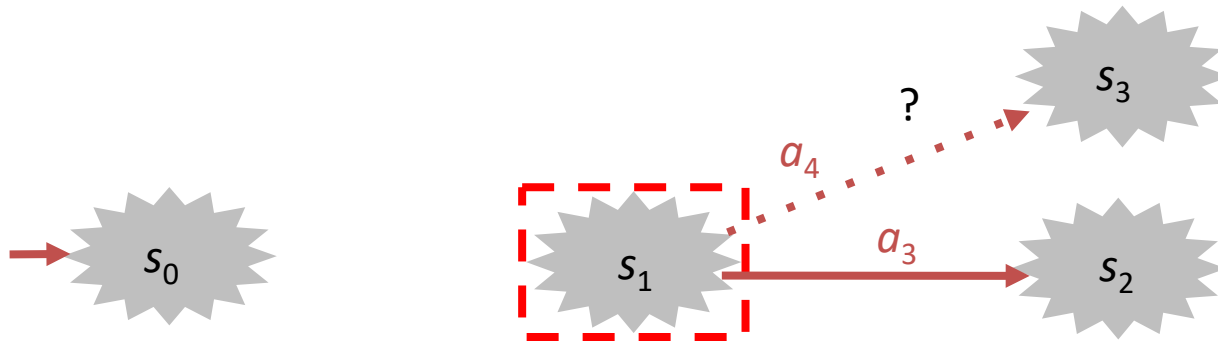
● Observations: $(s_0) \xrightarrow[a_2]{-0.1} (s_0) \xrightarrow[a_1]{-0.1} (s_1) \xrightarrow[a_3]{-0.1} (s_0) \xrightarrow[a_1]{-0.1} (s_1) \xrightarrow[a_3]{-0.1} (s_2) \quad 1$

État terminal: $Q(s_2, \text{None}) = 1$

$$\begin{aligned} Q(s_1, a_3) &\leftarrow Q(s_1, a_3) + \alpha (R(s_1) + \gamma \max\{ Q(s_2, \text{None}) \} - Q(s_1, a_3)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{ 1 \} + 0) \\ &= 0.55 \end{aligned}$$

● On recommence un nouvel essai...

Apprentissage actif avec Q-learning



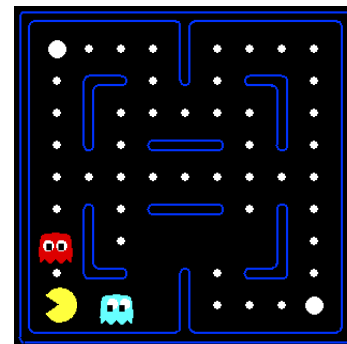
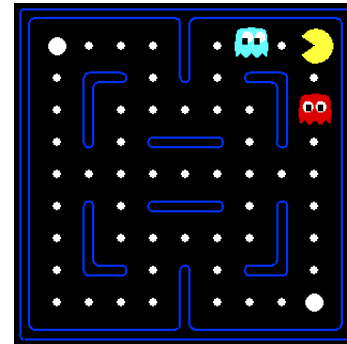
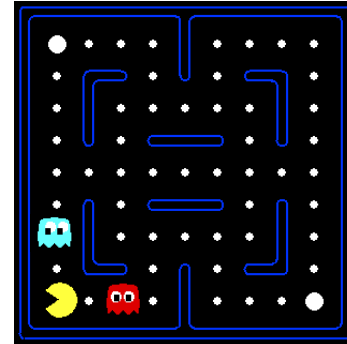
- Supposons qu'on puisse aussi faire l'action a_4 à l'état s_1 , pour mener à s_3 avec $R(s_1, a_4, s_3) = 1000$
- Puisque $Q(s_1, a_4) = 0$ à l'initialisation, et que $Q(s_1, a_3) = 0.55 > 0$ après un essai, une approche vorace n'explorerait jamais s_3 !

Généralisation par approximation de fonctions

- Dans des applications réelles, on ne peut pas visiter tous les états
 - ◆ Trop d'états à visiter dans les essais d'apprentissage
 - ◆ Mémoire limitée pour enregistrer les valeurs $Q(s,a)$ correspondant
- Par contre, on peut généraliser en utilisant les caractéristiques (*features*) des états plutôt que les états eux-mêmes:
 - ◆ Apprendre les caractéristiques des états à partir des échantillons d'entraînement.
 - ◆ Généraliser aux états similaires
- Le principe s'applique autant à l'apprentissage passif qu'à l'apprentissage actif

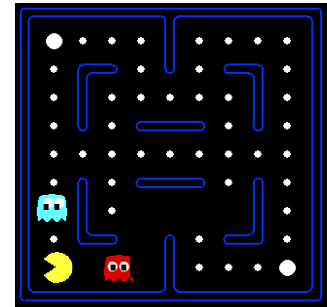
Exemple: Pacman

- Supposons que nous apprenons durant les essais que cet état est mauvais ($U(s)$ trop bas).
- Avec l'approche Q-Learning naïve telle que décrite, on ne peut rien déduire de cet état ou de sa valeur Q :
- Pas plus que pour celui-ci!



Approximation de fonction

- Représenter un état en utilisant un vecteur de caractéristiques (*features*)
 - ◆ Les caractéristiques sont des fonctions réelles (souvent 0/1) qui capturent les propriétés saillants des états
 - ◆ La taille de l'espace d'états représentés par leurs caractéristiques est beaucoup plus petit que l'espace d'états d'origine – c'est une approximation
 - ◆ Exemples de caractéristiques (*features*):
 - » Distance au fantôme le plus proche
 - » Distance à la nourriture la plus proche
 - » Nombre de fantômes
 - » $1 / (\text{distance à la nourriture})^2$
 - » Pacman proche d'un fantôme ? (0/1)
 - » etc.
 - ◆ On peut aussi décrire la fonction de qualité $Q(s, a)$ avec des caractéristiques (ex. l'action permettant à Pacman d'être proche de la nourriture)



Approximation de fonction

- Étant donné un vecteur $f = [f_1, \dots, f_n]$ de *features* d'état et un vecteur de poids correspondant $w = [w_1, \dots, w_n]$, on peut approximer U par une fonction linéaire

$$\widehat{U}_w = \sum_{i=1}^n w_i f_i(s)$$

- De façon similaire, avec un vecteur de *features* Q et des poids correspondants:

$$\widehat{Q}_w(s,a) = \sum_{i=1}^n w_i f_i(s,a)$$

- On peut alors utiliser l'apprentissage superviser pour apprendre \widehat{U}_w et \widehat{Q}_w
- On peut aussi utiliser des fonctions d'approximation non linéaire

$$\widehat{U}_w = f(s)$$

$$\widehat{Q}_w(s,a) = f(s)$$

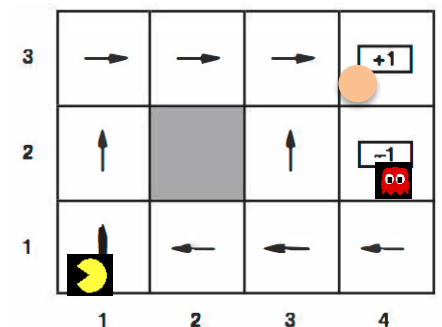


- En particulier, on pourrait utiliser un réseau de neurones – ce qu'on appelle apprentissage par renforcement profond.
- Voyons cela plus en détail.

Approximer l'estimation directe de l'utilité

- La méthode d'apprentissage passif par estimation directe de l'utilité simule des trajectoires d'exécution dans l'espace d'états (x,y) et calcule, pour chaque état, la somme des récompenses cumulées jusqu'à l'état terminal, $U(x,y)$.
- Chaque occurrence d'état (x,y) est une donnée, étiquetée par son utilité correspondante $U(x,y)$, et l'ensemble de toutes les occurrence constitue un ensemble de données d'entraînement pour un algorithme **d'apprentissage supervisé** qui peut apprendre U . On pourrait utiliser n'importe quel algorithme d'apprentissage supervisé.
- Par exemple, avec le même exemple utilisé pour la méthode d'apprentissage passif par estimation directe, approximations l'utilité par une fonction linéaire de *features* – les *features* étant simplement les positions x et y .

$$\widehat{U}_w(x,y) = w_0 + w_1x + w_2y$$



Approximer l'estimation directe de l'utilité

- Supposons qu'on approxime U par

$$\widehat{U}_w(x,y) = w_0 + w_1x + w_2y$$

- Notons $u_j(s)$ la somme des récompenses à partir de l'état dans l'échantillon j

- On peut utiliser l'erreur quadratique pour la perte

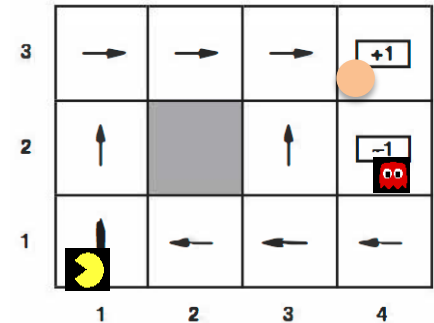
$$Loss_j(s) = (\widehat{U}_w(s) - u_j(s))^2 / 2$$

- Il en résulte la règle d'apprentissage suivante pour mettre à jour les poids

$$w_i \leftarrow w_i - \alpha \frac{\partial E_j(s)}{\partial w_i} = w_i + \alpha [u_j(s) - \widehat{U}_w(s)] \frac{\partial \widehat{U}_w(s)}{\partial w_i}$$

- C.-à-d.,

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha [u_j(s) - \widehat{U}_w(s)] \\ w_1 &\leftarrow w_1 + \alpha [u_j(s) - \widehat{U}_w(s)]x \\ w_2 &\leftarrow w_2 + \alpha [u_j(s) - \widehat{U}_w(s)]y \end{aligned}$$



Approximer l'apprentissage par différence temporelle

- On peut appliquer la même idée à l'apprentissage par différence temporelle autant passif que actif.
- La règle d'apprentissage des poids de la fonction d'approximation linéaire pour l'apprentissage par différence temporelle devient

$$w_i \leftarrow w_i + \alpha [R(s, a, s') + \gamma \widehat{U}_w(s') - \widehat{U}_w(s)] \frac{\partial \widehat{U}_w(s)}{\partial w_i}$$

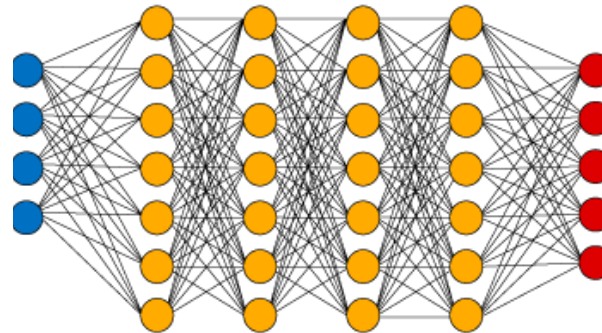
- Celle pour Q-Learning devient

$$w_i \leftarrow w_i + \alpha [R(s, a, s') + \gamma \max_{a'} \widehat{Q}_w(s', a') - \widehat{Q}_w(s, a)] \frac{\partial \widehat{U}_w(s)}{\partial w_i}$$

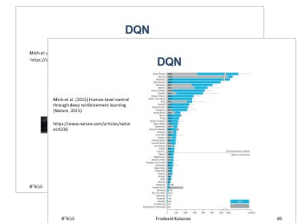
Apprentissage par renforcement profond

- Le principe pour l'apprentissage par renforcement profond est le même, mais la fonction d'approximation est un réseau de neurones, entraîné par la rétropropagation du gradient, avec une fonction de perte appropriée.
- Il y a plusieurs algorithmes. En particulier:
 - ◆ Deep Q-Learning Network (DQN) apprend $Q(s,a)$ directement
 - ◆ Deep Policy Network apprend la politique $\pi(s,a)$ directement

State
features



$\pi(s,a)$ ou $Q(s,a)$



- Dans TP #4, vous pourrez pratiquer DQN qui apprend $Q(s,a)$.
- Dans le cours IFT608 on approfondit l'apprentissage par renforcement.

Ressources intéressantes

- Gym : <https://gym.openai.com/>

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from [walking](#) to playing games like [Pong](#) or [Pinball](#).

- AWS DeepRacer



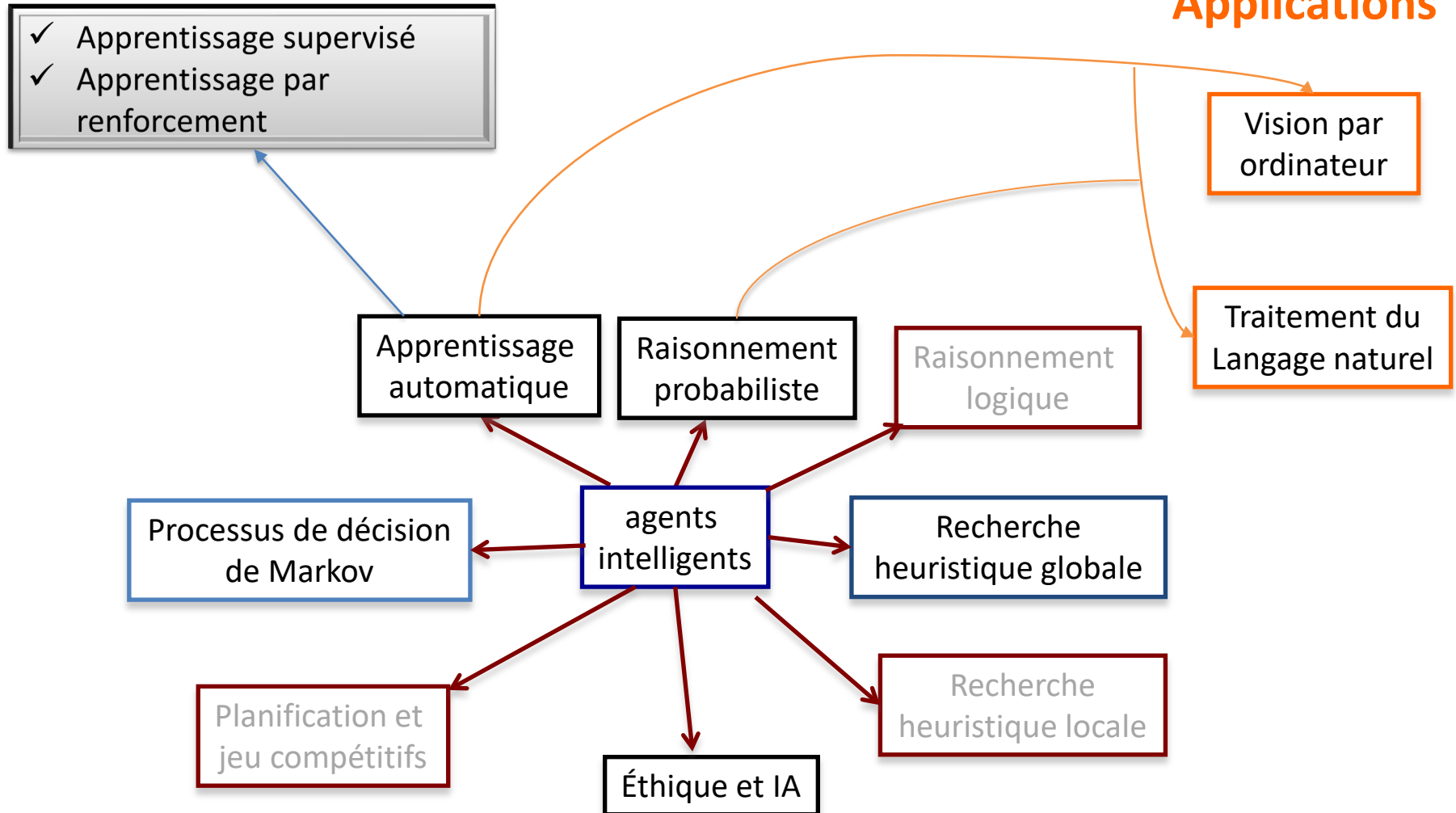
Conclusion

- L'apprentissage par renforcement permet d'apprendre la prise décision séquentielle
- C'est un domaine de recherche très actif
 - ◆ Il y a de plus en plus d'applications, en robotique, voitures autonomes, et d'autres domaines.
- L'apprentissage par renforcement est plus difficile lorsque la récompense est lointaine (ex.: à la toute fin d'une partie)
 - ◆ **problème d'assignation de crédit** (*credit assignment*)
 - » est-ce que ma victoire est due à mon premier coup? mon dernier? les deux?
 - ◆ on ajoute parfois des renforcements positifs intermédiaires appropriés (*reward shaping*)
 - » inconvénient: demande de l'expertise

Sujets couverts par le cours

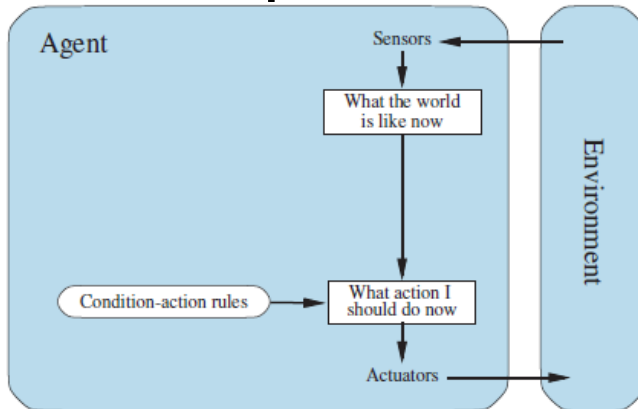
Concepts et algorithmes

Applications

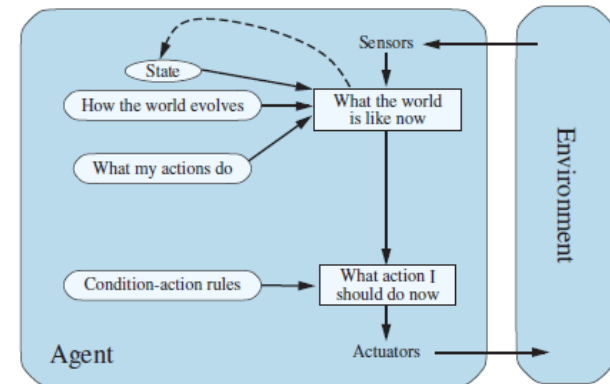


Apprentissage par renforcement pour quel type d'agents?

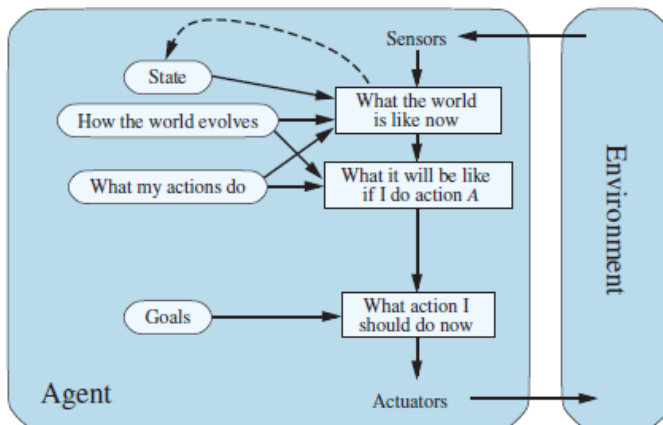
Simple reflex



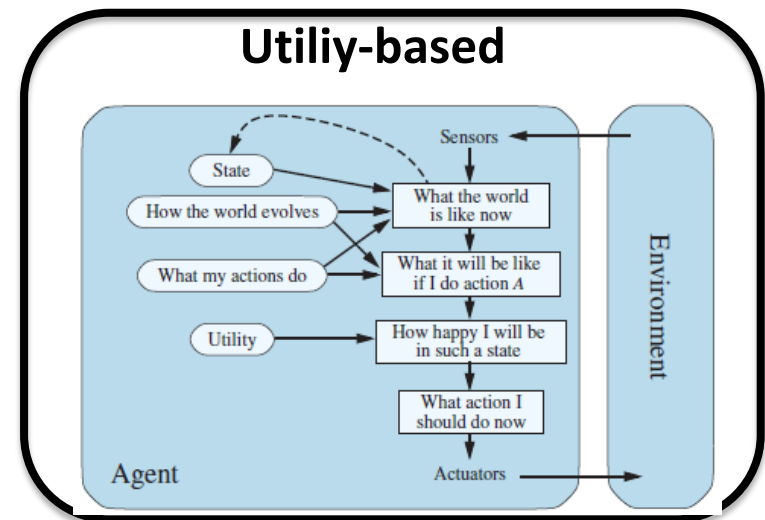
Model-based reflex



Goal-based



Utility-based



Vous devriez être capable de...

- Distinguer l'apprentissage par renforcement **passif** vs. **actif**
- Algorithmes passifs:
 - ◆ savoir simuler l'estimation directe
 - ◆ savoir simuler la programmation dynamique adaptative (PDA)
 - ◆ savoir simuler la différence temporelle (TD)
- Algorithmes actifs
 - ◆ savoir simuler PDA actif
 - ◆ savoir simuler *Q-learning*
 - ◆ *Savoir expliquer Q-Learning (On-policy) vs SARSA (off-policy)*
- Savoir comment on traite le dilemme exploration vs. exploitation
- Savoir comment on traite l'approximation de fonction

Rapprochement de différentes règles d'apprentissage

- **Règle d'apprentissage par renforcement passive avec la différence temporelle:**
pour chaque transition $(s, \pi(s), s')$ des échantillons

$$U(s) \leftarrow U(s) + \alpha [R + \gamma U(s') - U(s)]$$

- **Algorithme du Perceptron:** pour chaque paire $(\mathbf{x}_t, y_t) \in D$
 - a. calculer $h_{\mathbf{w}}(\mathbf{x}_t) = \text{Threshold}(\mathbf{w} \cdot \mathbf{x}_t)$
 - b. $w_i \leftarrow w_i + \alpha(y_t - h_{\mathbf{w}}(\mathbf{x}_t))x_{t,i} \quad \forall i$ (mise à jour des poids et biais)

- **Descente du gradient :**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i$$

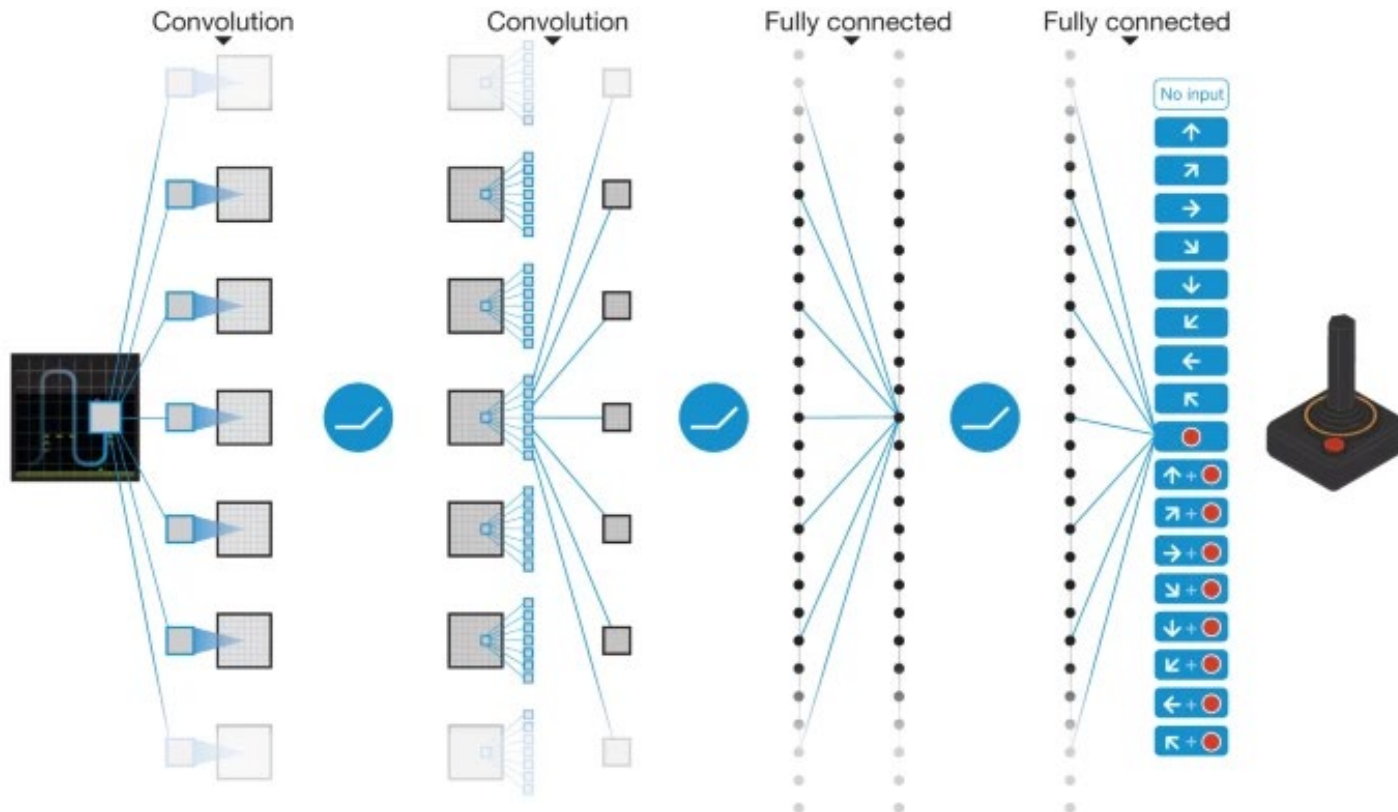
Nouvelle valeur Ancienne valeur Gradient de l'erreur

The diagram illustrates the components of the gradient descent update rule. A blue arrow points from the text 'Nouvelle valeur' to the new weight w_i on the left. Another blue arrow points from 'Ancienne valeur' to the old weight w_i in the subtraction. A third blue arrow points from 'Gradient de l'erreur' to the term $\frac{\partial}{\partial w_i} \text{Loss}(y_t, h_{\mathbf{w}}(\mathbf{x}_t))$, which is underlined in the equation.

DQN

Mnih et al. (2015) Human-level control through deep reinforcement learning (*Nature*, 2015)

<https://www.nature.com/articles/nature14236>



DQN

Minh et al. (2015) Human-level control through deep reinforcement learning (*Nature*, 2015)

<https://www.nature.com/articles/nature14236>

