

IFT 615 – Intelligence artificielle

Apprentissage par renforcement

Hugo Larochelle

Département d'informatique

Université de Sherbrooke

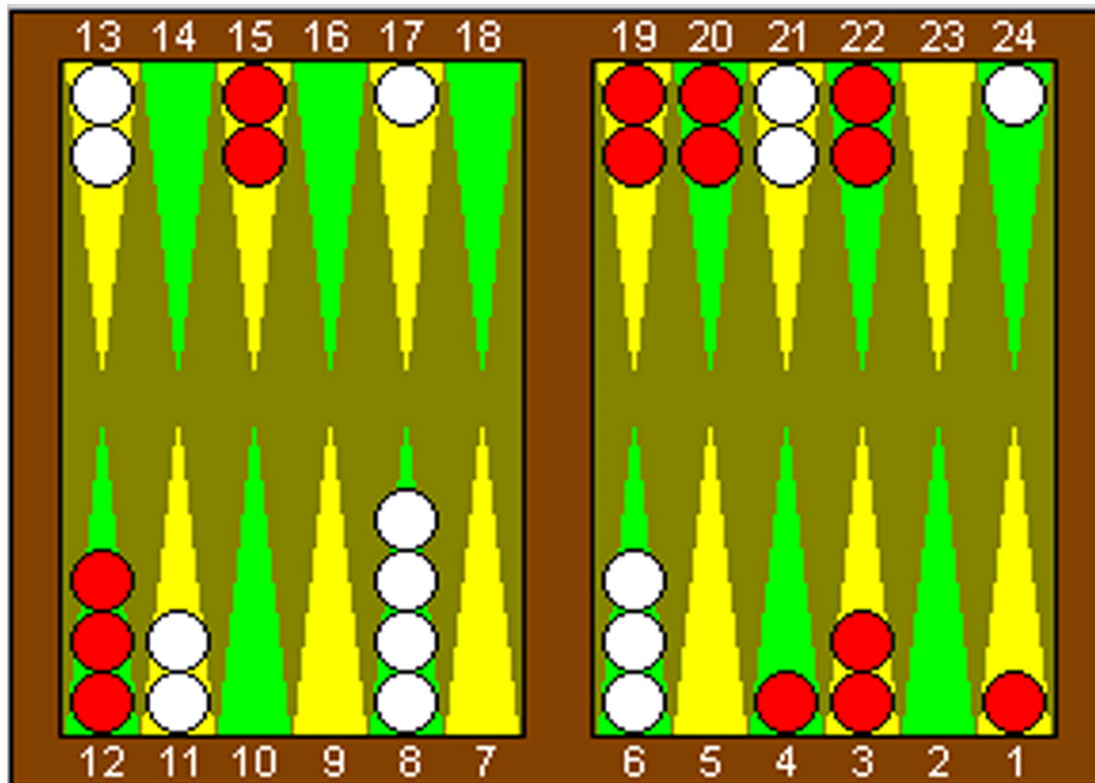
<http://www.dmi.usherb.ca/~larocheh/cours/ift615.html>

Sujets couverts

- Retour sur les processus de décision de Markov (MDP)
- 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*
 - ◆ méthode par recherche de plan/politique (*policy-gradient*)
- Dilemme exploration vs. exploitation
- Généralisation en apprentissage par renforcement

Mise en situation

- Comment développer une intelligence qui apprend elle-même un jeu?



Mise en situation

- Comment apprendre un contrôleur d'hélicoptère?



Pourquoi l'apprentissage par renforcement?

- On a vu que l'**apprentissage automatique supervisé** permet de modéliser une expertise à **partir de données étiquetées**
- Pour obtenir un agent intelligent qui joue bien aux échecs, il faudrait amasser des paires (état du jeu, mouvement à jouer) d'un joueur expert
 - ◆ amasser de telles données peut être fastidieux ou trop coûteux
- On préférerait que l'agent apprenne seulement à partir du résultat de parties qu'il joue
 - ◆ si l'agent a gagné, c'est que son plan (sa politique) de jeu était bon
 - ◆ si l'agent perd, c'est qu'il y a une faiblesse derrière sa façon de jouer

Pourquoi l'apprentissage par renforcement?

- L'**apprentissage par renforcement** s'intéresse au cas où l'agent doit apprendre seulement à **partir de telles récompenses** ou **renforcements**
- L'apprentissage se fait à l'image d'un animal qui perçoit des récompenses négatives (douleur, faim) et positives (plaisir, manger)
 - ◆ l'animal veut maximiser les récompenses positives et éviter les négatives

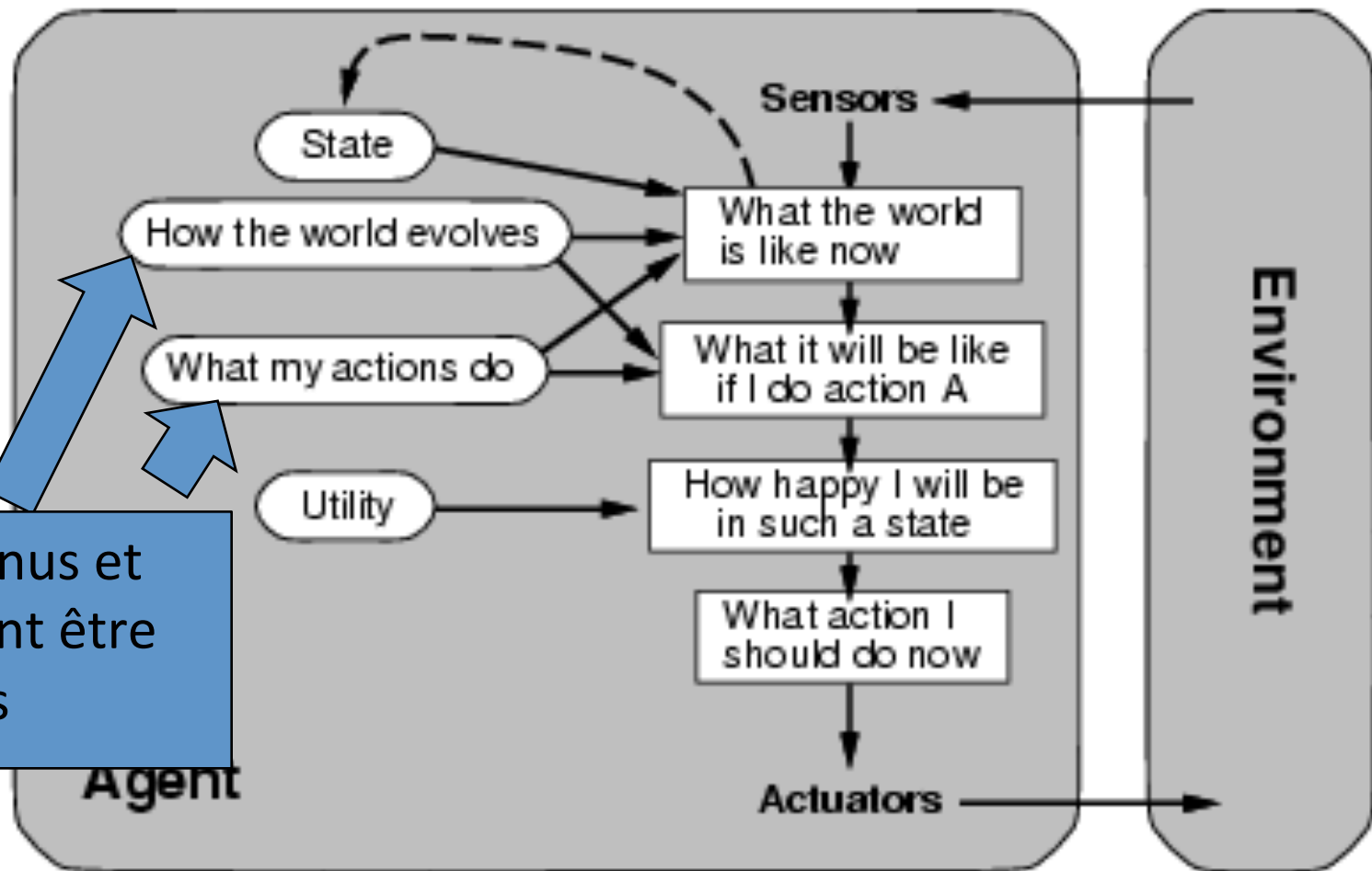
Pourquoi l'apprentissage par renforcement?

- On a vu des algorithmes pour les processus de décisions markovien (MDP) qui trouvent le plan optimal qui maximise la récompense espérée
 - ◆ Quels sont ces algorithmes?
- Ces algorithmes nécessitent une connaissance totale du modèle de transition $P(s'|s, a)$ et de la fonction de renforcement $R(s)$

Pourquoi l'apprentissage par renforcement?

- Dans un environnement réel, on ne connaît pas $P(s'|s, a)$
 - ◆ ex.: robot aspirateur placé dans une nouvelle pièce
 - ◆ ex.: agent qui contrôle un hélicoptère
 - ◆ ex.: agent qui joue à un jeu pour lequel $P(s'|s, a)$ est très complexe, avec un très grand espace d'état (Super Mario)
- Donc, 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...

Rappel: Utility-based agents



Inconnus et
doivent être
appris

Dans ce cours...

- On va se concentrer sur le cas
 - ◆ d'un environnement parfaitement observé
 - ◆ 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
 - ◆ généralisation en apprentissage par renforcement

Rappel: processus de décision markovien

- Un **processus de décision markovien** (*Markov decision process*, ou **MDP**) est défini par:
 - ◆ un **ensemble d'états** S (incluant un état initial s_0)
 - ◆ un **ensemble d'actions** possibles $Actions(s)$ (ou $A(s)$) lorsque je me trouve à l'état s
 - ◆ un **modèle de transition** $P(s'|s, a)$, où $a \in A(s)$
 - ◆ une **fonction de récompense** $R(s)$ (utilité d'être dans l'état s)
- Un **plan (politique)** π est un ensemble de décisions qui associe un état s à une action $a = \pi(s)$

Rappel: processus de décision markovien

- La **fonction de valeur** $V(s)$ d'un plan est donnée par les équations

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(s')$$

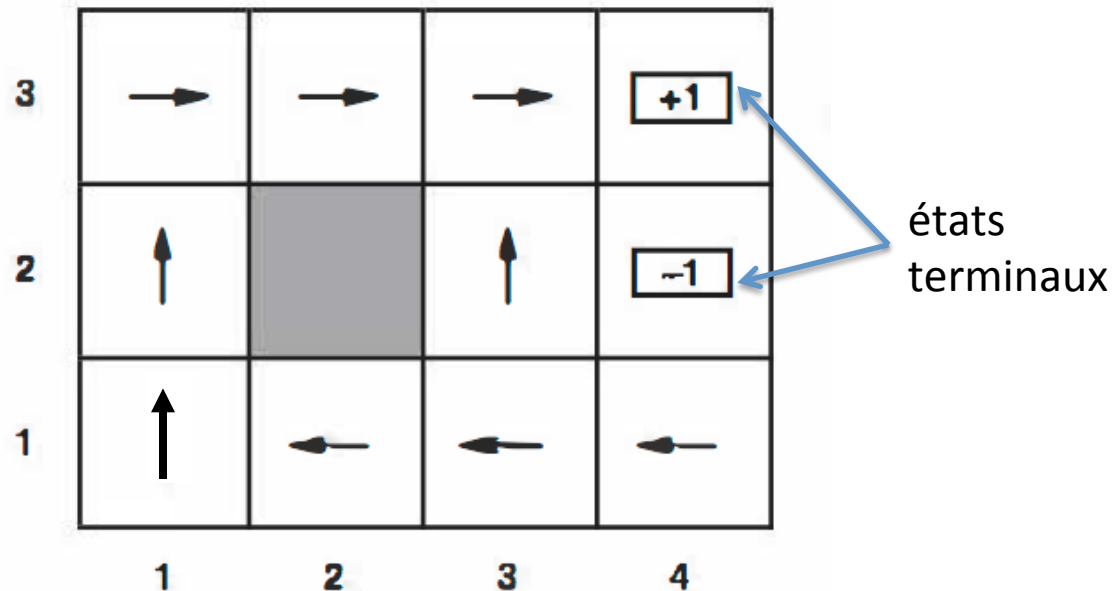
où γ est un facteur d'escompte donné

- Plutôt que $V(s)$, on note parfois $V(\pi, s)$, ou $U^\pi(s)$ dans le livre
- **NOUVEAU:** on va supposer l'existence d'**états terminaux**
 - ◆ lorsque l'agent atteint cet état, la simulation est arrêtée
 - ◆ on s'intéresse à la somme des récompenses jusqu'à l'atteinte d'un état terminal
 - » ex.: au tic-tac-toe, l'état terminal est une grille de fin de partie (c.-à-d. une grille complète ou une grille où un des joueurs a gagné)

Apprentissage par renforcement 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 par renforcement 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** (*trials*)
 1. $(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$
 2. $(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$
 3. $(1,1)_{-.04} \rightarrow (2,1)_{-.04} \rightarrow (3,1)_{-.04} \rightarrow (3,2)_{-.04} \rightarrow (4,2)_{-1}$
- Comment estimer la fonction de valeurs $V(s)$ à partir de ces essais?

Approche par estimation directe

- Approche la plus simple: **calculer la moyenne de ce qui est observé** dans les essais
- Estimation de $V(1,1)$
 - ◆ dans l'essai 1, la somme des récompenses à partir de (1,1) est 0.72
 - ◆ dans l'essai 2, on observe également 0.72
 - ◆ dans l'essai 3, on observe plutôt -1.16
 - ◆ l'estimation directe de $V(1,1)$ est donc $(0.72+0.72-1.16)/3 = 0.09333$
- Estimation de $V(1,2)$
 - ◆ dans l'essai 1, l'état (1,2) est visité deux fois, avec des sommes de récompenses à partir de (1,2) de 0.76 et 0.84
 - ◆ dans l'essai 2, on observe 0.76
 - ◆ l'essai 3 ne visite pas (1,2)
 - ◆ l'estimation directe de $V(1,2)$ est donc $(0.76+0.84+0.76)/3 = 0.78666$

Approche par estimation directe

- Cette approche est similaire à l'apprentissage supervisé
 - ◆ on apprend à partir de paires
(état visité s , somme pondérée des récompenses à partir de s)
- L'estimation **ignore la relation récursive entre les valeurs**, décrite par les équations de la fonction de valeur

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

- Par exemple, bien que l'essai 1 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)

Approche par programmation dynamique adaptative

- C'est l'idée derrière la **programmation dynamique adaptative (PDA)**
 - ◆ tirer profit des équations de la fonction de valeur pour estimer $V(s)$
- L'approche par PDA **n'apprend pas directement $V(s)$** , mais apprend plutôt le modèle de transition $P(s' | s, a)$
 - ◆ étant donnée une estimation de $P(s' | s, a)$, on peut résoudre $V(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, \pi(s)) V(s')$
 - ◆ on obtient alors notre estimation de $V(s)$
- On peut estimer $P(s' | s, a)$ à partir des fréquences des transitions observées:

$$P(s' | s, \pi(s)) = \frac{\sum_{\text{essais}} \text{freq}(s', s)}{\sum_{\text{essais}} \text{freq}(s)}$$

somme sur tous les essais

nb. de transitions de s à s' dans l'essai

nb. de fois que s est visité dans l'essai

Approche par programmation dynamique adaptative

function PASSIVE-ADP-AGENT(*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 , discount γ

U , a table of utilities, initially empty

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

$N_{s'|sa}$, a table of outcome frequencies given state-action pairs, initially zero

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

if s is not null **then**

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

for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**

$P(t | s, a) \leftarrow N_{s'|sa}[t, s, a] / N_{sa}[s, a]$

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$

if $s'.\text{TERMINAL?}$ **then** $s, a \leftarrow \text{null}$ **else** $s, a \leftarrow s', \pi[s']$

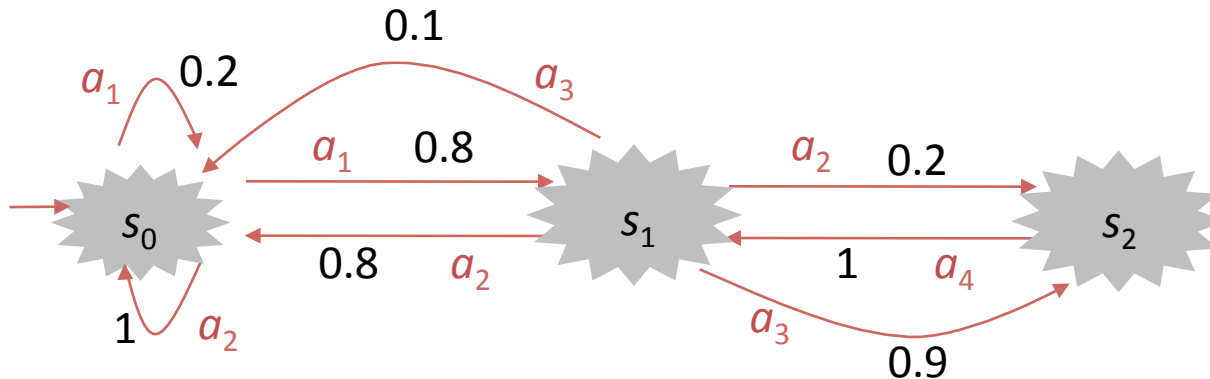
return a

Fonction qui résout

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s' | s, \pi(s)) V(s')$$

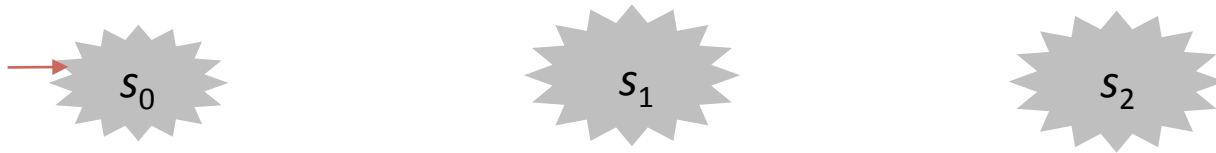
Approche par programmation dynamique adaptative

- Exemple (avec état terminal)



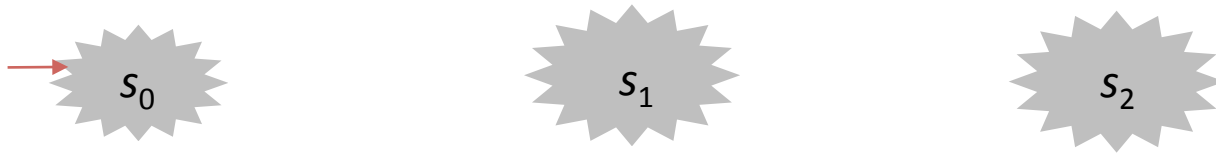
- MDP à 3 états: $S = \{s_0, s_1, s_2\}$
- Fonction de récompense: $R(s_0) = -0.1$, $R(s_1) = -0.1$, $R(s_2) = 1$
- Le facteur d'escompte est $\gamma=0.5$
- s_2 est un état terminal, s_0 est l'état initial
- Plan suivi: $\pi(s_0) = a_1$, $\pi(s_1) = a_3$

Approche par programmation dynamique adaptative



- Initialement, on suppose aucune connection entre les états

Approche par programmation dynamique adaptative



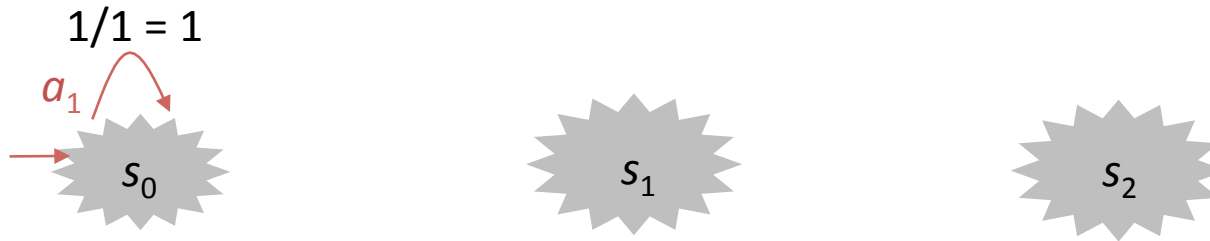
- Observations: $(s_0) -0.1$

$$V(s_0) = -0.1$$

$$V(s_1) = -0.1$$

$$V(s_2) = 1$$

Approche par programmation dynamique adaptative



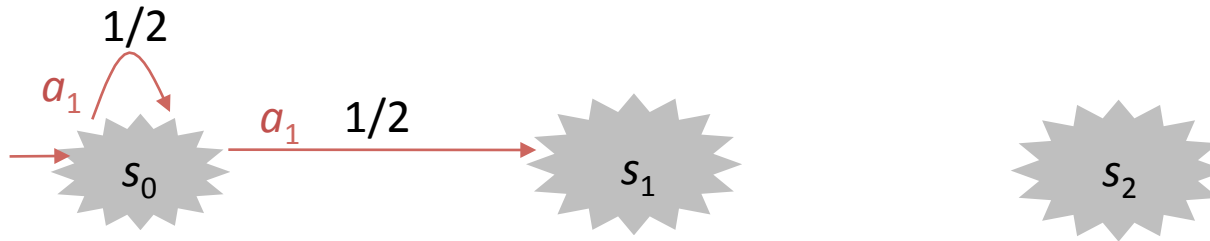
- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1}$

$$V(s_0) = -0.1 + 0.5 V(s_0)$$

$$V(s_1) = -0.1$$

$$V(s_2) = 1$$

Approche par programmation dynamique adaptative



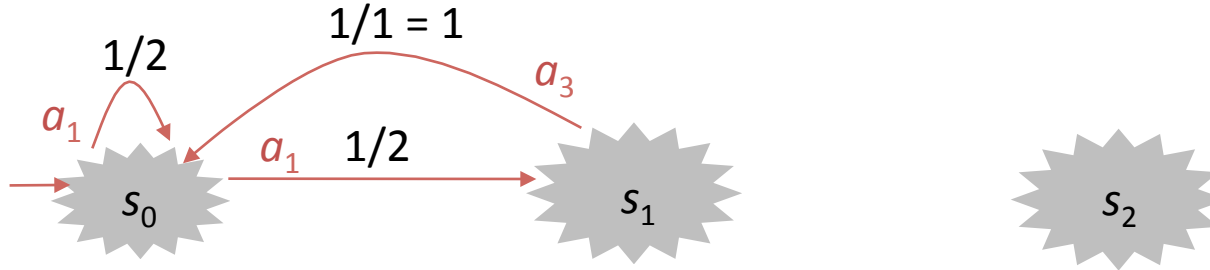
- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1}$

$$V(s_0) = -0.1 + 0.5 (0.5 V(s_0) + 0.5 V(s_1))$$

$$V(s_1) = -0.1$$

$$V(s_2) = 1$$

Approche par programmation dynamique adaptative



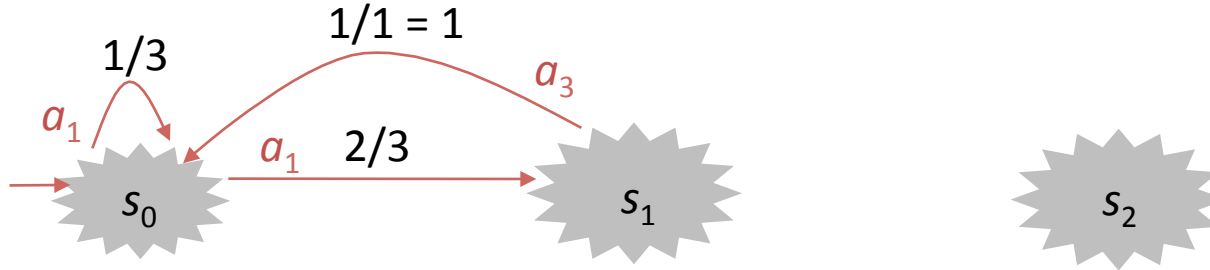
- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1}$

$$V(s_0) = -0.1 + 0.5 (0.5 V(s_0) + 0.5 V(s_1))$$

$$V(s_1) = -0.1 + 0.5 V(s_0)$$

$$V(s_2) = 1$$

Approche par programmation dynamique adaptative



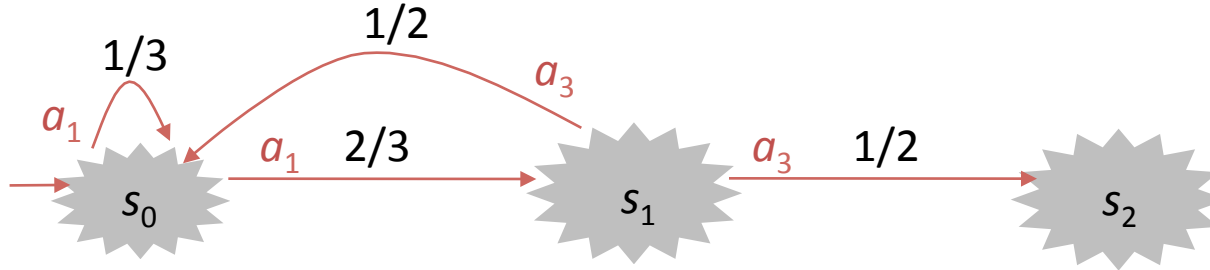
- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1}$

$$V(s_0) = -0.1 + 0.5 \left(\frac{1}{3} V(s_0) + \frac{2}{3} V(s_1) \right)$$

$$V(s_1) = -0.1 + 0.5 V(s_0)$$

$$V(s_2) = 1$$

Approche par programmation dynamique adaptative



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_2)_1$

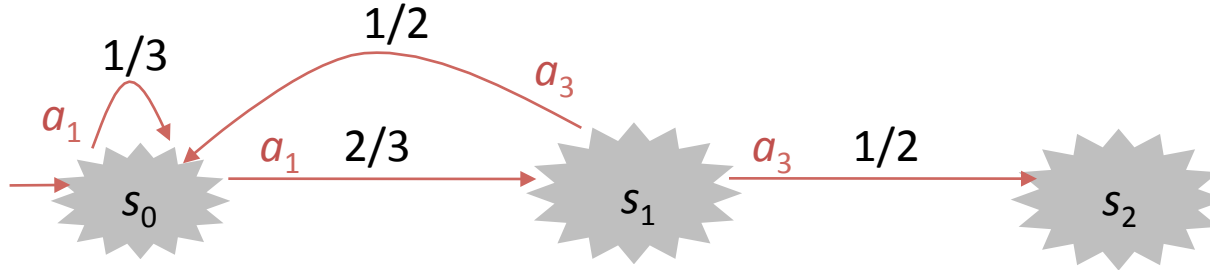
fin de
l'essai

$$V(s_0) = -0.1 + 0.5 \left(\frac{1}{3} V(s_0) + \frac{2}{3} V(s_1) \right)$$

$$V(s_1) = -0.1 + 0.5 \left(0.5 V(s_0) + 0.5 V(s_2) \right)$$

$$V(s_2) = 1$$

Approche par programmation dynamique adaptative



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_2)_1$

fin de
l'essai

$$V(s_0) = -0.1 + 0.5 \left(\frac{1}{3} V(s_0) + \frac{2}{3} V(s_1) \right)$$

$$V(s_1) = -0.1 + 0.5 \left(0.5 V(s_0) + 0.5 V(s_2) \right)$$

$$V(s_2) = 1$$

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

Approche par programmation dynamique adaptative

- On a vu comment résoudre le système d'équations des $V(s)$ dans le cours sur les MDP
 - ◆ on peut écrire le système sous forme $b = A x$, et calculer $x = A^{-1} b$
- Cette opération peut être coûteuse à répéter après chaque observation
 - ◆ inverser la matrice A est dans $O(|S|^3)$

Approche par programmation dynamique adaptative

- Approche alternative: méthode itérative, similaire à *value iteration*

1. Répéter (jusqu'à ce que le changement en V soit négligeable)

- I. pour chaque état s calculer:

$$V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(s')$$

sans le max
p/r à l'action

- II. si $|V - V'| \leq \text{tolérance}$, quitter

- III. $V \leftarrow V'$

- Plutôt qu'initialiser $V(s)$ à 0, on peut l'**initialiser à sa valeur précédente, avant la nouvelle observation**
 - ◆ une seule observation peut avoir un impact minime sur la nouvelle valeur de $V(s)$ qui en résulte
 - ◆ l'approche itérative + initialisation à la valeur précédente devrait donc converger rapidement

Approche par programmation dynamique adaptative

- Approche alternative: méthode itérative, similaire à *value iteration*
 1. Répéter (jusqu'à ce que le changement en V soit négligeable).
 - I. pour chaque état s calculer:
$$V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(s')$$
 - II. si $|V - V'| \leq \text{tolérance}$, quitter
 - III. $V \leftarrow V'$
- Autres accélérations
 - ◆ borner le nombre d'itérations (c.-à-d. ne pas attendre d'atteindre le seuil)
 - ◆ balayage hiérarchisé (*prioritized sweeping*)
 - » on garde un historique des changements $|V(s) - V'(s)|$
 - » on priorise les états s avec une grande valeur précédente de $|V(s) - V'(s)|$
- Permet de résoudre des problèmes où $|S|$ est beaucoup plus grands

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 $V(s)$
- 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

```
function PASSIVE-ADP-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state  $s'$  and reward signal  $r'$ 
  persistent:  $\pi$ , a fixed policy
             mdp, an MDP with model  $P$ , rewards  $R$ , discount  $\gamma$ 
              $U$ , a table of utilities, initially empty
              $N_{sa}$ , a table of frequencies for state-action pairs, initially zero
              $N_{s'|sa}$ , a table of outcome frequencies given state-action pairs, initially zero
              $s, a$ , the previous state and action, initially null

  if  $s$  is not null then
    increment  $N_{sa}[s, a]$  and  $N_{s'|sa}[s', s, a]$ 
    for each  $t$  such that  $N_{s'|sa}[t, s, a]$  is nonzero do
       $P(t | s, a) \leftarrow N_{s'|sa}[t, s, a] / N_{sa}[s, a]$ 
     $U \leftarrow \text{POLICY-EVALUATION}(\pi, U, \textit{mdp})$ 
  if  $s'.\text{TERMINAL?}$  then  $s, a \leftarrow \text{null}$  else  $s, a \leftarrow s', \pi[s']$ 
  return  $a$ 
```


Approche par programmation dynamique adaptative

- Un problème avec l'approche par PDA est qu'on **doit mettre à jour toutes les valeurs de $V(s)$, 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 $V(s)$ parce qu'on estime $V(s)$ seulement indirectement, via notre estimation de $P(s'|s, a)$
- Serait-il possible d'estimer directement $V(s)$ et tenir compte des interactions entre les valeurs, sans avoir à passer par $P(s'|s, a)$?

Apprentissage par différence temporelle

- Observation: si la transition de s vers s' a une probabilité de 1, on a que

$$\begin{aligned} V(s) &= R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(s') \\ &= R(s) + \gamma V(s') \end{aligned}$$

- Plutôt que d'attendre la fin de l'essai pour mettre à jour notre estimation de $V(s)$, on pourrait la rapprocher de $R(s) + \gamma V(s')$:
 - ◆ $V(s) \leftarrow (1-\alpha) V(s) + \alpha (R(s) + \gamma V(s'))$
où α est un **taux d'apprentissage**, entre 0 et 1
- On obtient la règle d'apprentissage **par différence temporelle** (*temporal difference*) ou **TD**

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

Apprentissage par différence temporelle

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

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

persistent: π , a fixed policy

U , a table of utilities, initially empty

N_s , a table of frequencies for states, initially zero

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

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

if s is not null **then**

increment $N_s[s]$

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

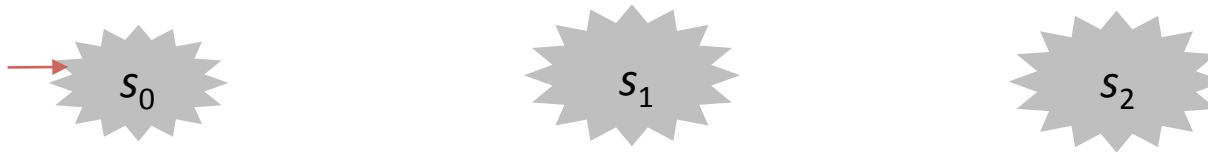
if $s'.\text{TERMINAL?}$ **then** $s, a, r \leftarrow \text{null}$ **else** $s, a, r \leftarrow s', \pi[s'], r'$

return a

Utile si on veut varier le taux d'apprentissage



Apprentissage par différence temporelle

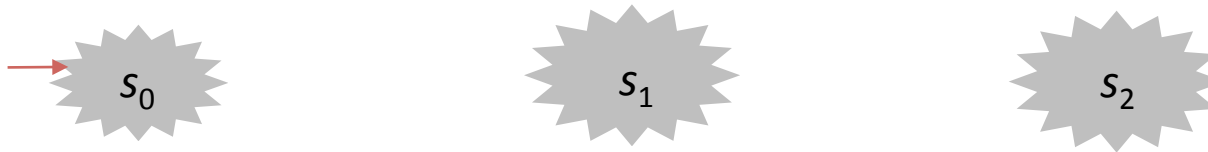


- Initialisation

$$\left. \begin{array}{l} V(s_0) = 0 \\ V(s_1) = 0 \\ V(s_2) = 0 \end{array} \right\} \text{ si on connaît } R(s), \text{ on peut tout initialiser } V(s) \text{ à } R(s)$$

- On va utiliser $\alpha = 0.1$

Apprentissage par différence temporelle



- Observations: $(s_0)_{-0.1}$

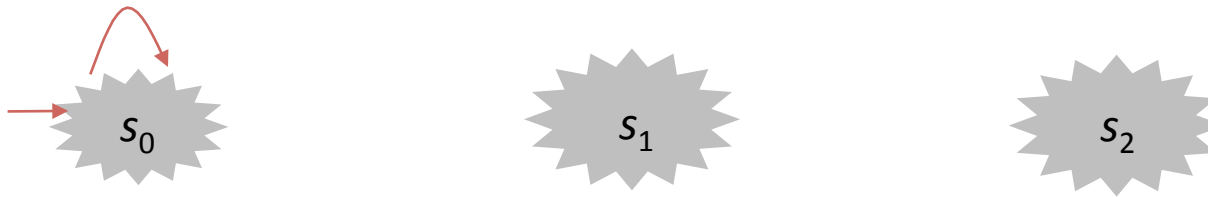
$V(s_0) \leftarrow -0.1$ ← parce que s_0 est visité pour la première fois

$$V(s_1) = 0$$

$$V(s_2) = 0$$

- On va utiliser $\alpha = 0.1$

Apprentissage par différence temporelle



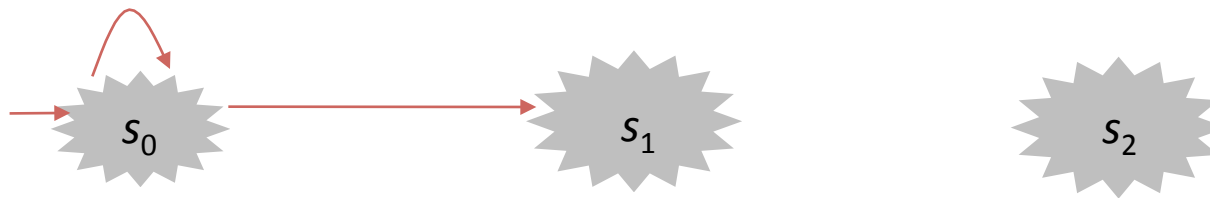
- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1}$

$$\begin{aligned} V(s_0) &\leftarrow V(s_0) + 0.1 (R(s_0) + 0.5 V(s_0) - V(s_0)) \\ &= -0.1 + 0.1 (-0.1 - 0.05 + 0.1) \\ &= -0.105 \end{aligned}$$

$$V(s_1) = 0$$

$$V(s_2) = 0$$

Apprentissage par différence temporelle



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1}$

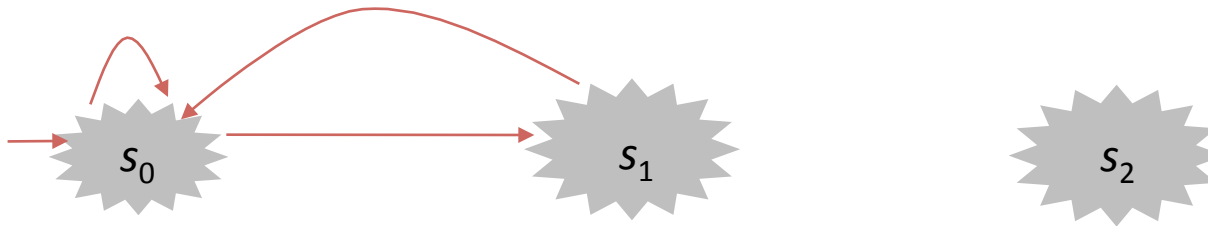
$V(s_1) \leftarrow -0.1$ ← parce que s_1 est visité pour la première fois

$$\begin{aligned} V(s_0) &\leftarrow V(s_0) + 0.1 (R(s_0) + 0.5 V(s_1) - V(s_0)) \\ &= -0.105 + 0.1 (-0.1 - 0.05 + 0.105) \\ &= -0.1095 \end{aligned}$$

$$V(s_1) = -0.1$$

$$V(s_2) = 0$$

Apprentissage par différence temporelle



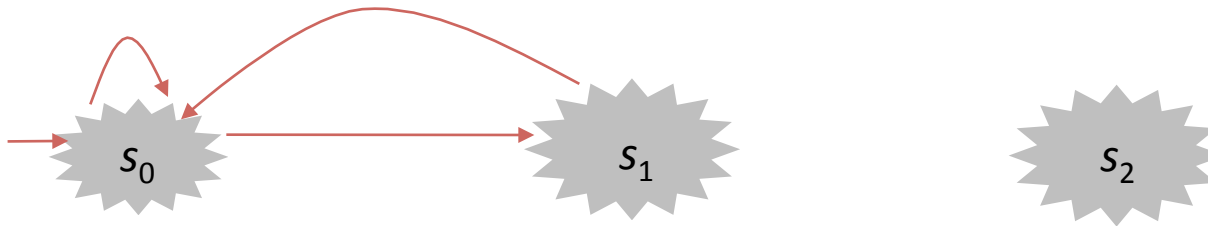
- Observations: $(s_0) \xrightarrow{-0.1} (s_0) \xrightarrow{-0.1} (s_1) \xrightarrow{-0.1} (s_0) \xrightarrow{-0.1}$

$$V(s_0) = -0.1095$$

$$\begin{aligned} V(s_1) &\leftarrow V(s_1) + 0.1 (R(s_1) + 0.5 V(s_0) - V(s_1)) \\ &= -0.1 + 0.1 (-0.1 - 0.05475 + 0.1) \\ &= -0.105475 \end{aligned}$$

$$V(s_2) = 0$$

Apprentissage par différence temporelle



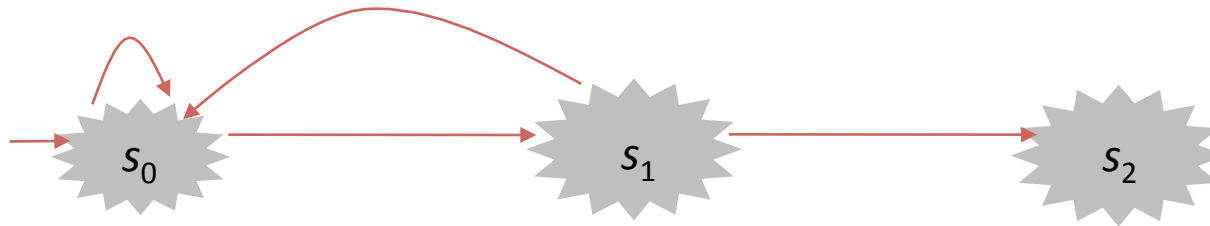
- Observations: $(s_0) \xrightarrow{-0.1} (s_0) \xrightarrow{-0.1} (s_1) \xrightarrow{-0.1} (s_0) \xrightarrow{-0.1} (s_1) \xrightarrow{-0.1}$

$$\begin{aligned} V(s_0) &\leftarrow V(s_0) + 0.1 (R(s_0) + 0.5 V(s_1) - V(s_0)) \\ &= -0.1095 + 0.1 (-0.1 - 0.0527375 + 0.1095) \\ &= -0.11382375 \end{aligned}$$

$$V(s_1) = -0.105475$$

$$V(s_2) = 0$$

Apprentissage par différence temporelle



- Observations: $(s_0)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_0)_{-0.1} \rightarrow (s_1)_{-0.1} \rightarrow (s_2)_1$

fin de
l'essai

$V(s_2) \leftarrow 1$ ← parce que s_2 est visité pour la première fois

$$V(s_0) = -0.11382375$$

$$\begin{aligned} V(s_1) &\leftarrow V(s_1) + 0.1 (R(s_1) + 0.5 V(s_2) - V(s_1)) \\ &= -0.105475 + 0.1 (1 + 0.5 + 0.105475) \\ &= 0.0550725 \end{aligned}$$

$$V(s_2) = 1$$

Apprentissage par renforcement actif

- Dans le cas **passif**, le plan à suivre est pré-déterminé
 - ◆ peu utile si on ne connaît pas le plan optimal à suivre
- Dans le cas **actif**, l'agent doit aussi chercher le plan optimal
 - ◆ l'agent doit **simultanément** chercher le plan optimal et sa fonction de valeur
 - ◆ **$V(s)$ est maintenant une estimation de la fonction de valeur du plan optimal**
- Dans le cas PDA, deux changements sont à faire
 - ◆ on applique **value iteration** au MDP estimé (c.-à-d. on résout les équations pour la **politique optimale**)
 - ◆ l'action choisie par l'agent devient

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

Apprentissage actif avec PDA

ACTIVE

function ~~PASSIVE~~-ADP-AGENT(*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 , discount γ

U , a table of utilities, initially empty

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

$N_{s'|sa}$, a table of outcome frequencies given state-action pairs, initially zero

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

if s' is new then $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$

if s is not null then

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

for each t such that $N_{s'|sa}[t, s, a]$ is nonzero do

$P(t | s, a) \leftarrow N_{s'|sa}[t, s, a] / N_{sa}[s, a]$

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$

if $s'.\text{TERMINAL?}$ then $s, a \leftarrow \text{null}$ else $s, a \leftarrow s', \pi[s']$

return a

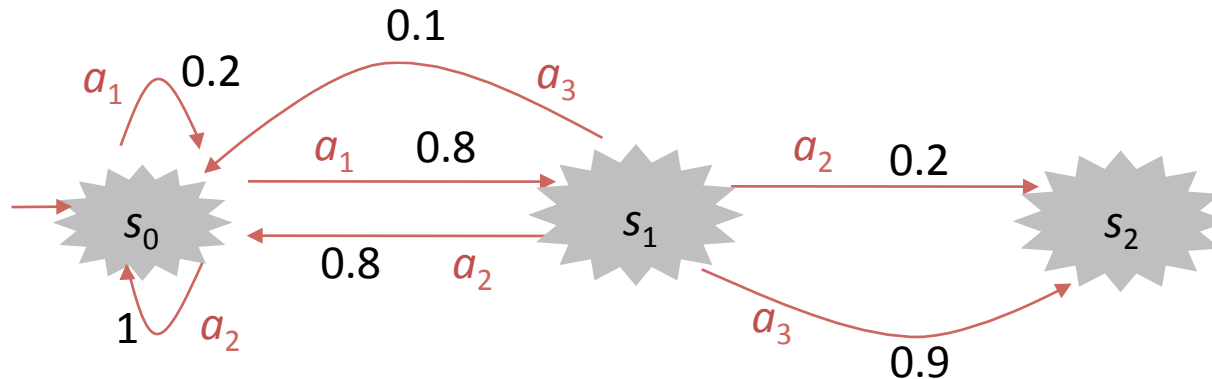
Value iteration

$$V(s) = R(s) + \max_a \gamma \sum_{s' \in S} P(s'|s, a) V(s')$$

$$\leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s \in S} P(s|s', a) V(s)$$

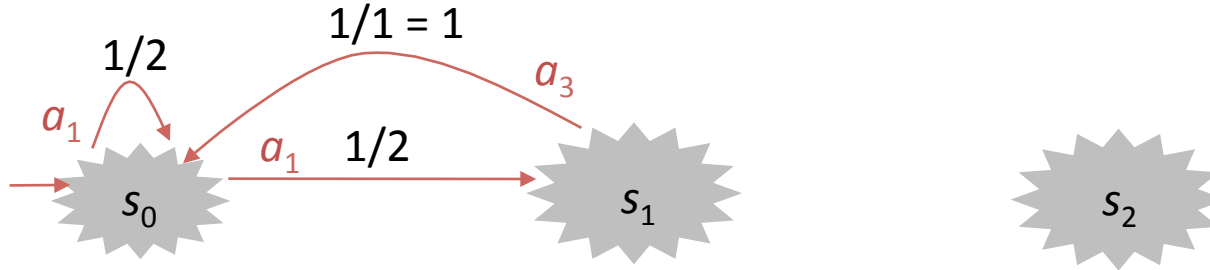
Apprentissage actif avec PDA

- 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



- Observations: $(s_0) \xrightarrow{a_1, -0.1} (s_0) \xrightarrow{a_1, -0.1} (s_1) \xrightarrow{a_3, -0.1} (s_0) \xrightarrow{-0.1}$

$V(s_0) = -0.1 + 0.5 \max\{ 0.5 V(s_0) + 0.5 V(s_1), 0 \}$ $V(s_1) = -0.1 + 0.5 \max\{ 0, V(s_0) \}$ $V(s_2) = 1$	<i>value iteration</i> 	$V(s_0) = -0.1$ $V(s_1) = -0.1$ $V(s_2) = 1$
---	----------------------------	--

- Pour choisir quelle action prendre, on compare
 - ◆ $\sum_{s \in S} P(s|s', a_2) V(s) = 0$ (puisque $P(s|s', a_2)$ pas appris encore pour a_2)
 - ◆ $\sum_{s \in S} P(s|s', a_1) V(s) = 0.5 V(s_0) + 0.5 V(s_1) = -0.1$
- L'action choisie par l'agent est donc a_2**

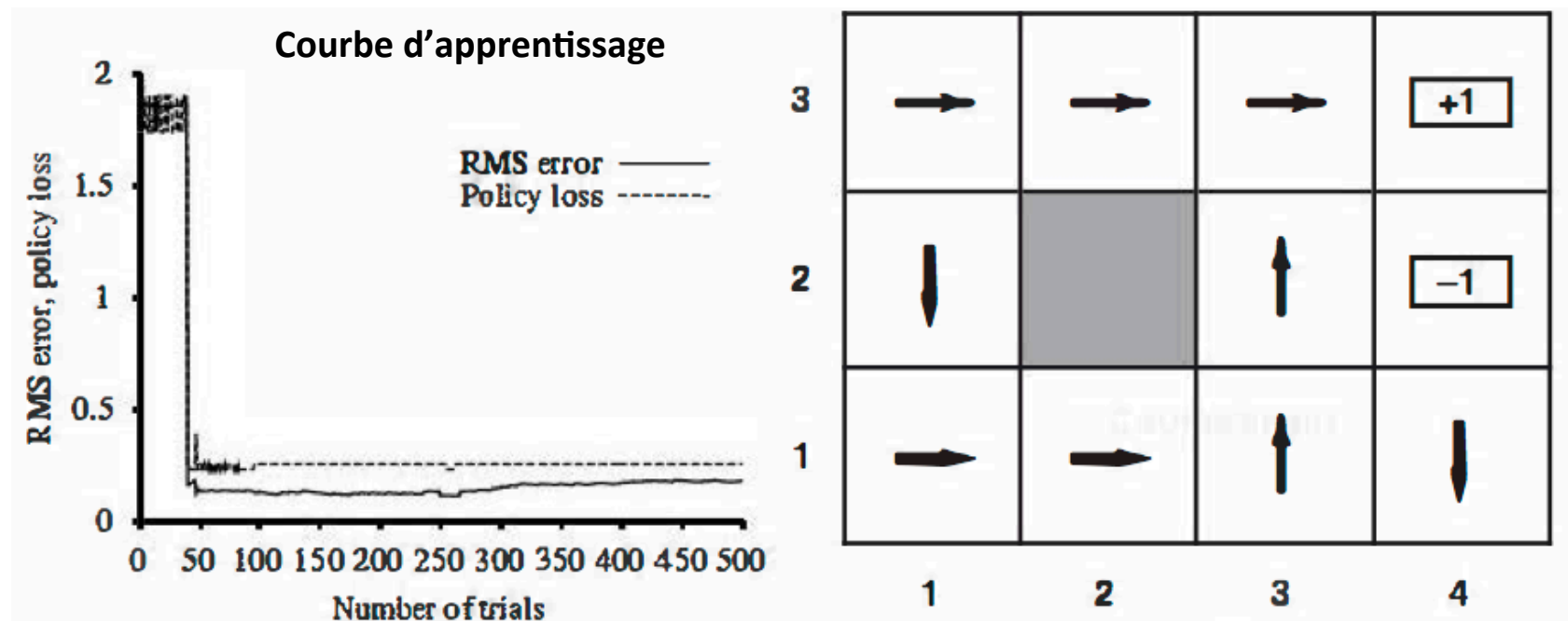
Dilemme exploration vs. exploitation

- L'approche précédente est dite **vorace (gloutonne)**
 - ◆ elle met à jour le plan suivi par celui qui est optimal **maintenant**
 - ◆ en d'autres mots, **elle exploite le plus possible** l'information recueilli 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
- Un parallèle similaire existe entre le *hill-climbing* et le *simulated annealing* en recherche locale

Dilemme exploration vs. exploitation

- Exemple: cas où l'action « \uparrow » n'a jamais été exécutée à l'état (1,2)
- L'agent ne sait pas que ça mène à (1,3), qui mène à un chemin plus court!

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 le cas du *n-armed bandit* dans le livre, p. 841

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

$$V'(s) = R(s) + \max_a \gamma f(\sum_{s' \in S} P(s'|s,a) V(s'), N(s,a))$$

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
récompense future
(hyper-paramètre)

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

Apprentissage actif par différence temporelle

- Dans le cas de l'apprentissage TD, l'approche active vorace devrait aussi utiliser

$$\pi(s) = \operatorname{argmax}_a \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s')$$

- Par contre, l'approche TD passive n'estime pas $P(s'|s,a)$
- En mode actif, on pourrait apprendre $P(s'|s,a)$ en plus de $V(s')$

Apprentissage actif avec Q-learning

- Peut-on éviter l'apprentissage de $P(s'|s,a)$?
- Une alternative est d'apprendre une **fonction action-valeur** $Q(s,a)$
 - ◆ on n'apprend plus $V(s)$, 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 $V(s)$ est que $V(s) = \max_a Q(s,a)$
- Le plan de l'agent est alors $\pi(s) = \operatorname{argmax} Q(s,a)$
 - ◆ plus besoin d'estimer $P(s'|s,a)$ et $V(s)$ séparément
- On appelle cette approche ***Q-learning***

Apprentissage actif avec Q-learning

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

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

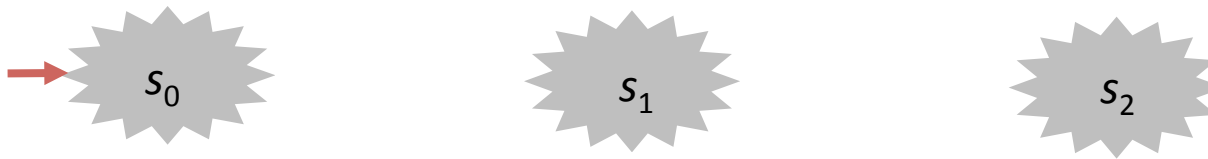
- Comme pour l'approche TD, on traduit cette équation en la mise à jour

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

- On voit la similarité avec l'approche TD initiale

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

Apprentissage actif avec Q-learning



- Initialisation:

$$\begin{aligned} Q(s_0, a_1) &= 0 & Q(s_0, a_2) &= 0 \\ Q(s_1, a_2) &= 0 & Q(s_1, a_3) &= 0 \\ Q(s_2, \text{None}) &= 0 \end{aligned}$$

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

Apprentissage actif avec Q-learning

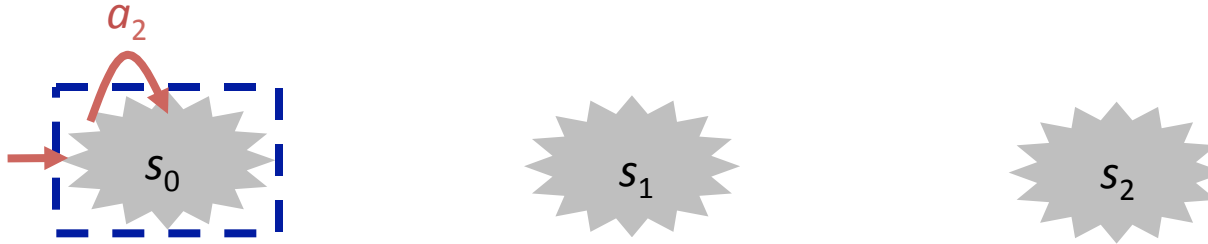


- Observations: $(s_0)_{-0.1}$

On ne fait rien (on a besoin d'un triplet (s, a, s'))

- 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

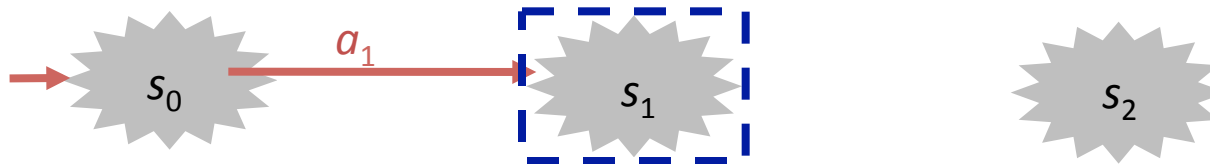


- Observations: $(s_0)_{-0.1} \xrightarrow{a_2} (s_0)_{-0.1}$

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

- Action à prendre $\pi(s_0) = \operatorname{argmax}\{ Q(s_0, a_1), Q(s_0, a_2) \}$
 $= \operatorname{argmax}\{ 0, -0,05 \}$
 $= a_1$ **(changement de politique!)**

Apprentissage actif avec Q-learning

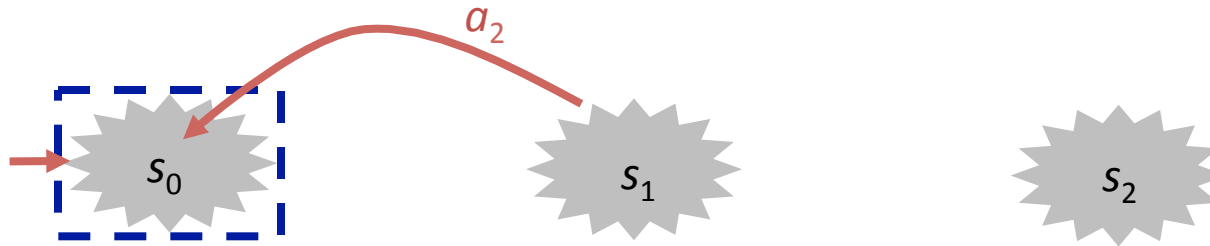


- Observations: $(s_0) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1}$

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

- 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

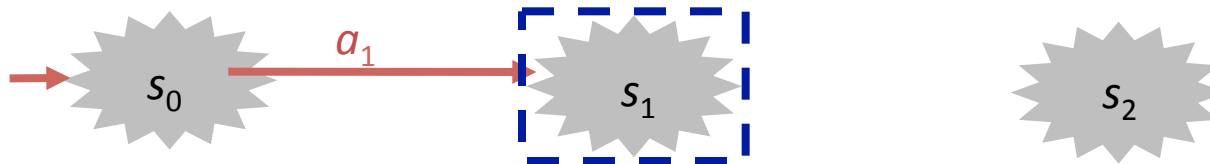


- Observations: $(s_0) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1}$

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

- 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

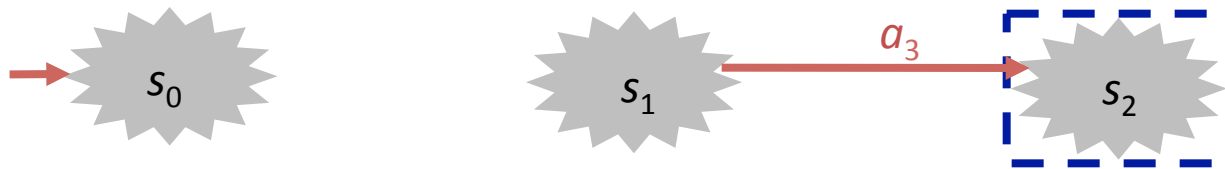


- Observations: $(s_0) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_3} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_1}$

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

- Action à prendre $\pi(s_1) = \operatorname{argmax}\{ Q(s_1, a_2), Q(s_1, a_3) \}$
 $= \operatorname{argmax}\{ -0.0625, 0 \}$
 $= a_3$ **(changement de politique!)**

Apprentissage actif avec Q-learning



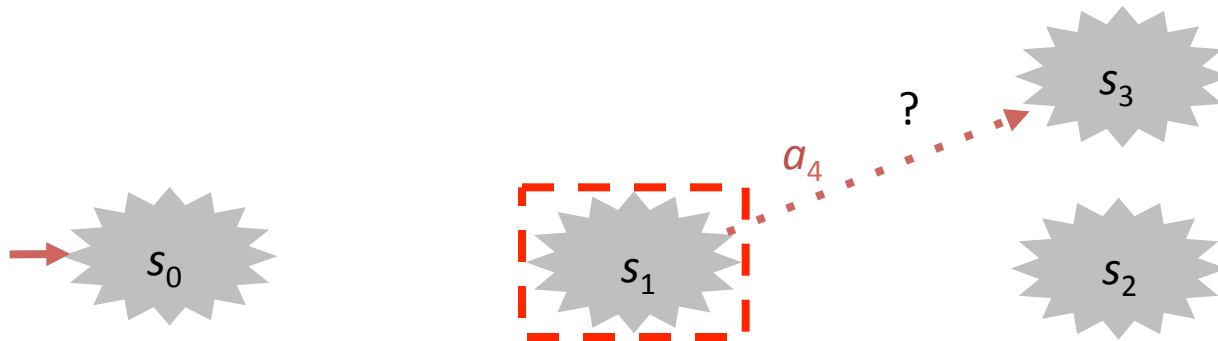
- Observations: $(s_0) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_3} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_3} (s_2)_1$

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

$$\begin{aligned} Q(s_1, a_3) &\leftarrow Q(s_1, a_3) + \alpha (R(s_1) + \gamma \max\{ Q(s_2, \text{None}) \} - Q(s_1, a_3)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{ 1 \} + 0) \\ &= 0.2 \end{aligned}$$

- 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 tel que $R(s_3) = 1000$
- Puisque $Q(s_1, a_4) = 0$ à l'initialisation, et que $Q(s_1, a_3) > 0$ après un essai, une approche vorace n'explorera jamais s_3 !
- Demo: <http://www.cs.ubc.ca/~poole/demos/rl/q.html>

Apprentissage actif avec Q-learning

- On peut également contrôler la balance entre l'exploration et l'exploitation dans *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
                $s, a, r$ , the previous state, action, and reward, initially null

  if TERMINAL?( $s'$ ) then  $Q[s', \text{None}] \leftarrow r'$ 
  if  $s$  is not null then
    increment  $N_{sa}[s, a]$ 
     $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$ 
     $s, a, r \leftarrow s', \arg\max_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$ 
  return  $a$ 
  
```

La fonction d'exploration f contrôle la balance via N_e

Apprentissage actif par la méthode de recherche de plan/politique

- Toutes les méthodes vues jusqu'à maintenant dérivent leur plan/politique à partir d'une fonction de valeur ou action-valeur
 - ◆ les méthodes diffèrent seulement dans la façon d'estimer ces fonctions
- Pourquoi ne pas optimiser directement par rapport au plan π de l'agent
 - ◆ soit s_0 l'état initial
 - ◆ **problème d'optimisation à résoudre**: trouver π dont la valeur $V(s_0)$ est la plus grande
 - ◆ on ne connaît pas $P(s'|s, \pi(s))$, donc on ne peut pas calculer $V(s_0)$ directement
 - ◆ par contre, chaque essai donne une estimation stochastique $V(s_0)$
- C'est ce qu'on appelle la **recherche de plan/politique** (*policy search*)

Apprentissage actif par la méthode de recherche de plan/politique

- Exemple de la grille 3 x 4 ($\gamma = 1$)

- ◆ soit l'essai (simulation) obtenu en suivant π

$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$

- ◆ alors on sait que $V(s_0) \approx 7 \times -0.04 + 1 = 0.72$

- Approche de recherche de politique par *hill-climbing*

- répéter durant T itérations

- 1. $V(s_0) \leftarrow$ résultat de l'essai obtenu en suivant π

- 2. pour chaque politique π' voisine de π (successeur)

- a. $v \leftarrow$ résultat de l'essai (simulation) obtenu en suivant π'

- b. si $v > V(s_0)$

- i. $V(s_0), \pi \leftarrow v, \pi'$

Apprentissage actif par la méthode de recherche de plan/politique

- Pourrait générer les politiques successeurs π' en considérant tous les changements possibles d'une seule action, pour un seul état

◆ ex.:

$$\pi(s_0) = a_1, \pi(s_1) = a_3 \xrightarrow{\text{successeurs}} \left\{ \begin{array}{l} \pi'(s_0) = a_2, \pi'(s_1) = a_3 \\ \text{ou} \\ \pi'(s_0) = a_1, \pi'(s_1) = a_2 \end{array} \right.$$

- Peut bien fonctionner seulement si l'espace des états et d'actions est petit
- L'apprentissage sera lent s'il y a beaucoup de variations stochastiques possibles
 - ◆ ex.: nos chances de gagner au Poker dépendent beaucoup des cartes que l'on pige

Apprentissage actif par la méthode de recherche de plan/politique

- Si on peut **contrôler les variations d'une simulation à l'autre**, on peut accélérer l'apprentissage
- Si on a accès au générateur de nombres aléatoires du simulateur, on peut l'initialiser au même état avant chaque simulation
- Dans le cas d'un jeu de carte, les mêmes cartes seraient pigées dans le même ordre
- C'est l'idée derrière l'algorithme **PEGASUS**, utilisé pour apprendre des manoeuvres acrobatiques à l'aide d'un hélicoptère
 - ◆ voir <http://heli.stanford.edu/> pour des vidéos de démonstration

Généralisation en apprentissage par renforcement

- Jusqu'à maintenant, on a supposé que les fonctions de valeur ou action-valeur étaient représentées sous forme de tableau

s	$V(s)$
s_0	-0.2
s_1	-0.1
s_2	3
...	...

- Cela pose deux problèmes
 - ◆ pas pratique lorsque l'espace d'états S est immense
 - ◆ impossible de **généraliser** à un état s_i jamais visité jusqu'à maintenant

Généralisation en apprentissage par renforcement

- Exemple: poker

s	$V(s)$
« quatre As, un roi de pique »	10.1
« quatre As, un trois de coeur »	9.9
« quatre As, un dix de carreau »	?
...	...

- Le fait qu'on ait quatre As est le facteur déterminant de la valeur de sa main
 - ◆ on voudrait généraliser cette observation à toutes mains avec quatre As, quelque soit la cinquième carte

Généralisation à l'aide d'approximateur de fonction

- Idée: remplacer la table $V(s)$ par une fonction linéaire $V_\theta(s)$ avec
 - ◆ des **caractéristiques** $f_i(s)$ décrivant l'état s
 - ◆ des **paramètres** θ_i à apprendre

$$V_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \theta_3 f_3(s) + \dots + \theta_n f_n(s)$$

- Exemple de caractéristique pour le poker:

$$f_i(s) = \begin{cases} 1 & \text{si } s \text{ est une main avec quatre As} \\ 0 & \text{sinon} \end{cases}$$

- L'apprentissage vise à **trouver la valeur des paramètres θ_i tels que $V_\theta(s)$ est proche de $V(s)$** , pour tout état s

Généralisation à l'aide d'approximateur de fonction

- Application à l'apprentissage TD

$$V(s) \leftarrow V(s) + \alpha (\underbrace{R(s) + \gamma V(s') - V(s)}_{\text{peut être vu comme}})$$

peut être vu comme $-\frac{\partial \text{Loss}}{\partial V(s)}$

- On dérive une règle d'apprentissage comme avec la règle de chaînage

$$\theta_i \leftarrow \theta_i + \alpha (\underbrace{R(s) + \gamma V_\theta(s') - V_\theta(s)}_{-\frac{\partial \text{Loss}}{\partial V_\theta(s)}}) \underbrace{f_i(s)}_{\frac{\partial V_\theta(s)}{\partial \theta_i}}$$

Généralisation à l'aide d'approximateur de fonction

- Application au Q-learning

- ◆ on utilise des caractéristiques de paires (état, action)

$$Q_{\theta}(s,a) = \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \theta_3 f_3(s,a) + \dots + \theta_n f_n(s,a)$$

↑ ex.: $f_1(s,a) = 1$ si main avec 4 As et action a est « *all in* »

- On dérive la règle d'apprentissage similairement

$$\theta_i \leftarrow \theta_i + \alpha \left(\underbrace{R(s) + \gamma \max_{a'} Q_{\theta}(s',a') - Q_{\theta}(s,a)}_{-\frac{\partial \text{Loss}}{\partial Q_{\theta}(s,a)}} \right) \underbrace{f_i(s)}_{\frac{\partial Q_{\theta}(s,a)}{\partial \theta_i}}$$

Généralisation à l'aide d'approximateur de fonction

- **Application à l'approche PDA**

- ◆ on utilise une fonction pour modéliser les probabilités $P(s'|s,a)$

$$P(s'|s,a) \propto \exp(\theta_1 f_1(s',s,a) + \theta_2 f_2(s',s,a) + \theta_3 f_3(s',s,a) + \dots + \theta_n f_n(s',s,a))$$

- ◆ possible de définir un coût approprié et d'en dériver un gradient
 - » c'est un problème d'apprentissage supervisé
 - » on doit prédire la cible s' étant donnée l'entrée constituée de s et de a

- Possible d'appliquer à la recherche de plan/politique

- ◆ voir section 21.5 du livre

Généralisation à l'aide d'approximateur de fonction

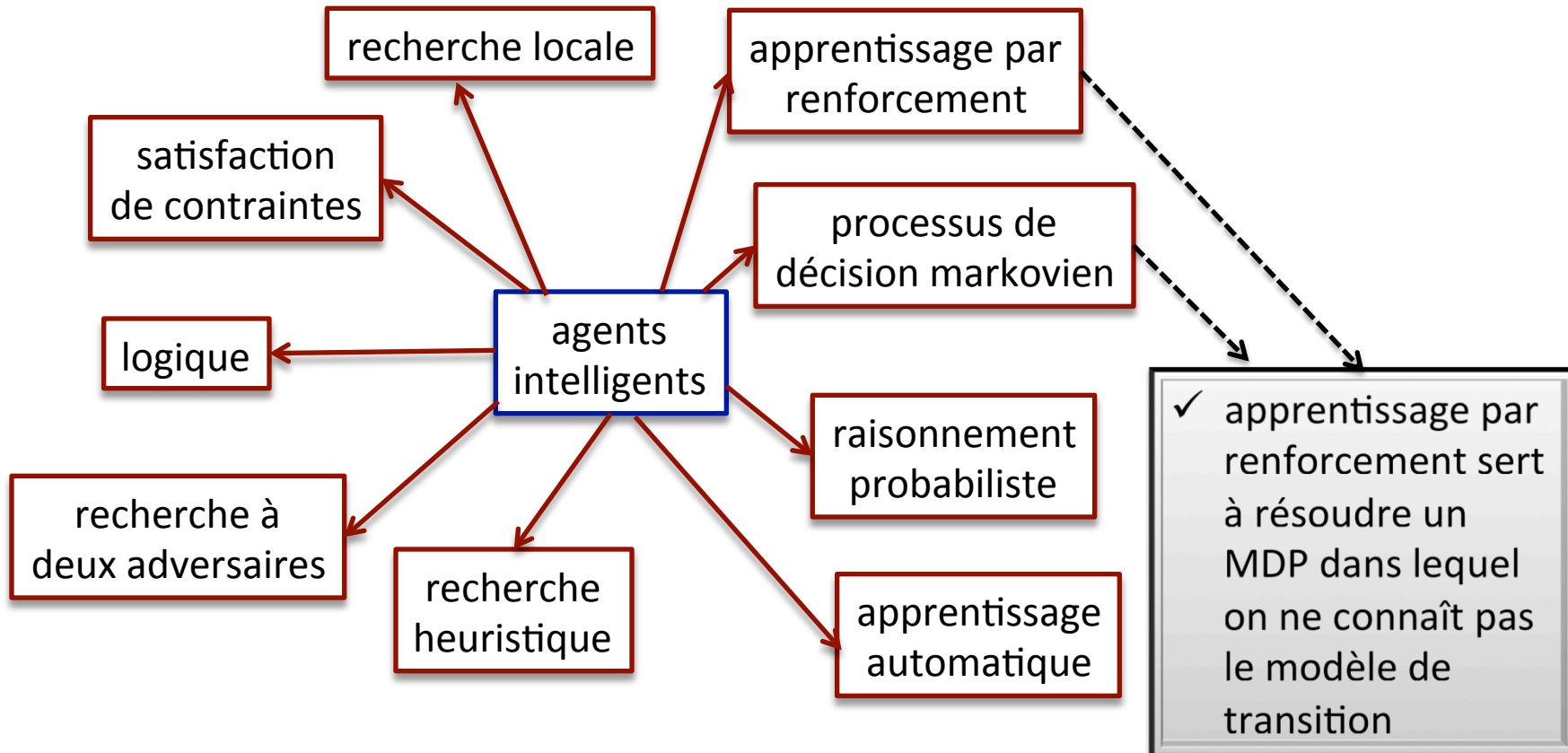
- Exemple d'application: **IA pour le backgammon** (*TD-Gammon*)
 - ◆ idée: modéliser la fonction de (action-)valeur à l'aide d'un réseau de neurones
 - » plus puissant qu'une fonction linéaire
 - ◆ première approche: demander à des experts de fournir les cibles à prédire
 - » approche d'apprentissage supervisée, et non par renforcement
 - » **problème**: difficile pour un expert de déterminer des cibles
 - » **résultat**: n'est pas compétitif contre un humain
 - ◆ deuxième approche: apprentissage TD actif avec réseau de neurones
 - » le réseau de neurones **joue contre lui-même** pendant plusieurs jours (300 000 parties!)
 - » **résultat**: compétitif par rapport aux trois meilleurs joueurs du monde!

Conclusion

- L'apprentissage par renforcement permet d'apprendre dans un environnement séquentiel
 - ◆ l'apprentissage supervisé souvent limité aux environnements périodiques
- C'est un domaine de recherche très actif
 - ◆ il y a de plus en plus d'applications impressionnantes
 - ◆ mais nous sommes loin d'une IA réelle
- 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
 - » désavantage: demande de l'expertise

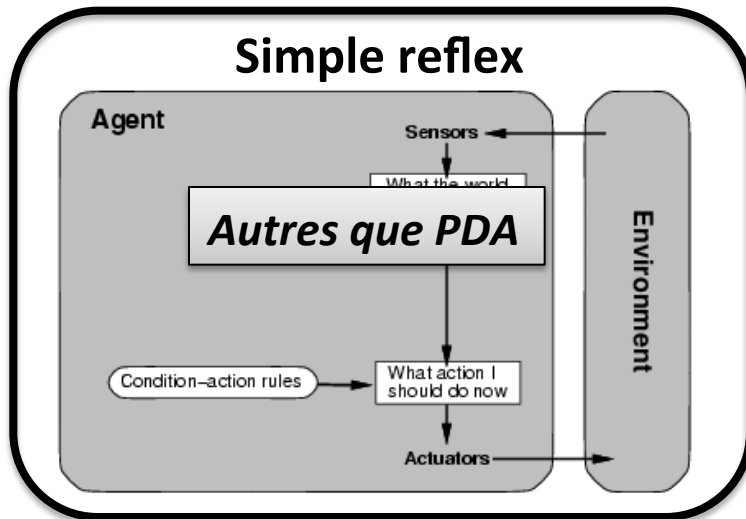
Objectifs du cours

Algorithmes et concepts

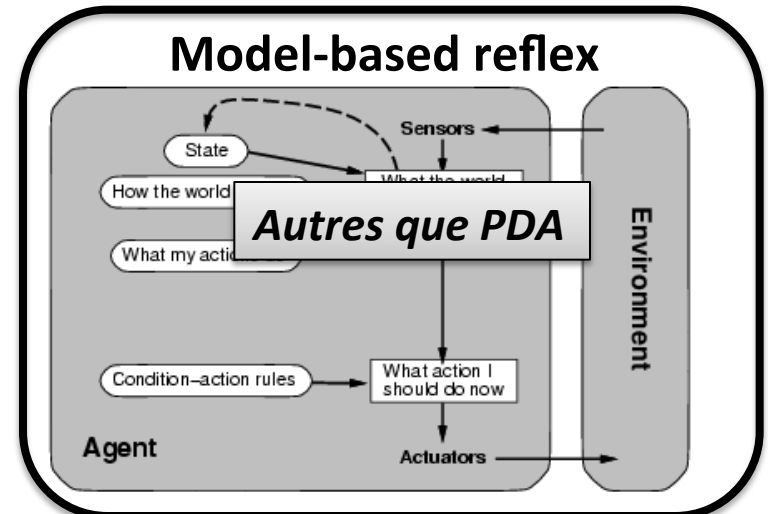


Apprentissage par renforcement : pour quel type d'agent?

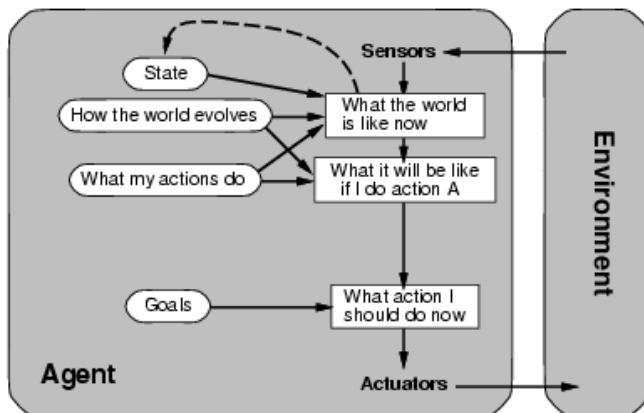
Simple reflex



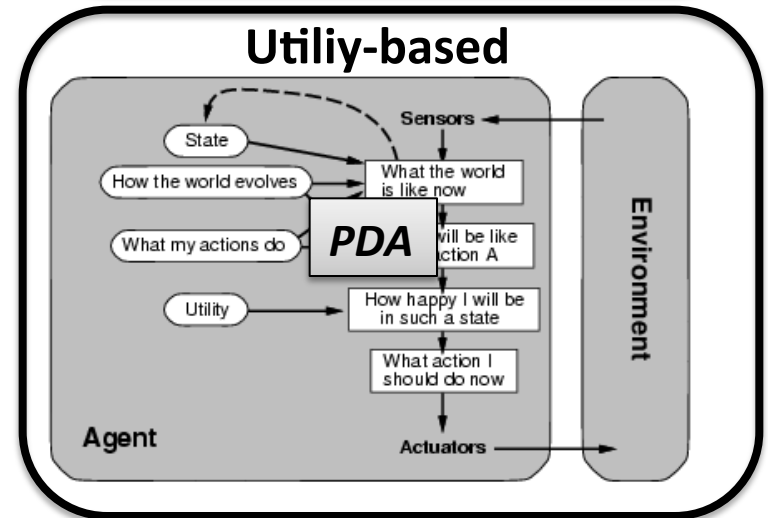
Model-based reflex



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 décrire la recherche de plan/politique
- Savoir comment on traite le dilemme exploration vs. exploitation
- Savoir ce qu'est l'approximation de fonction et à quoi ça sert