

Matière 4 : Contrôle et apprentissage

Chapitre 1 : Introduction to Reinforcement Learning (RL)

L'environnement peut être inconnu, non linéaire, stochastique, complexe.

Les agents cherchent à apprendre, et l'objectif est de maximiser la récompense.

Branches du machine learning : Supervised learning, unsupervised learning, reinforcement learning.

Caractéristiques de RL:

- Pas de supervision
- Retour avec délais
- Le temps utile
- Les actions de l'agent affectent les données reçues

1 - Rewards R_t :

Permet de savoir à quel point l'agent fait un bon travail, il faut chercher à le maximiser.

Il peut être meilleur de sacrifier le gain instantané au profit du long terme.

Reward renvoyer par l'environnement à l'instant $t-1$.

State S_t : L'état renvoyer par l'environnement a l'instant $t-1$.

On note **H_t l'histoire**, tels que : $H_t = S_1, \dots, S_t$

Un état est Markov si : $P(S_{t+1}|S_t) = P(S_{t+1}|H_t)$; Cad si, le futur ne dépend que du présent.

Action A_t : L'action exécutée par l'agent à l'instant t .

2 - Set of policies :

Markov Decision Process (MDP) :

- Récompense et état observable ;
- Distribution de récompense et probabilité de transition connues ;
- Les actions dépendent de l'état et des actions actuelles .

Partially Observable Markov Decision Process (POMDP) :

- Etat partiellement observable, on connaît z_t et $P(s_t=s|z)$;
- Récompenses observables ;
- Distribution de récompense et probabilité de transition connues.

RL :

- Récompense et état observable ;

Known model
MDP, POMDP

Unknown model
Reinforcement learning

Absence of model
Adversarial

- Distribution de récompense et probabilité de transition inconnues.

Adversarial problems :

- Récompense et état observable ;
- Etat de transition et récompense sont arbitraires et varient en fonction du temps.

3 - Major components of an RL Agent:

- Policy : une map d'états à actions. $\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$
- Value function : Prédiction de futures récompenses ;

$$v_{\pi}(s) = \mathbb{E}_{\pi}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s)$$
- Un modèle qui prédit ce que fera l'environnement ensuite

$$p(s'|s, a) = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a) \qquad r(s, a) = \mathbb{E}(R_{t+1} | S_t = s, A_t = a)$$

4 - Prediction and Control

Prediction : Evaluer le futur en fonction de la Policy

Contrôle : Trouver la meilleure Policy

Chapitre 2 : Markov

Etat Markovien : Un état S_t est dit Markovien ssi $P(S_{t+1} | S_t) = P(S_{t+1} | H_t)$ où H_t est l'historique des états passés tel que $H_t = S_1, \dots, S_t$.

La **probabilité de transition d'état** est définie par $p(s, s') = P(S_{t+1} = s' | S_t = s)$. Ainsi on peut former la **matrice de transition d'état** telle que :

$$p = \begin{pmatrix} p(1,1) & \cdots & p(1,n) \\ \vdots & \ddots & \vdots \\ p(n,1) & \cdots & p(n,n) \end{pmatrix}$$

Un **processus de Markov** est une séquence d'états aléatoires S_1, S_2, \dots , tous Markoviens. Telle que :

- Une chaîne de Markov est un tuple $\langle S, P \rangle$

où S est un ensemble fini d'états et P une matrice de transition d'état

Un **processus de Markov à récompense** est une chaîne de Markov comprenant des valeurs :

C'est un tuple $\langle S, P, R, \gamma \rangle$ où

- S est un ensemble fini d'états
- P est une matrice de transition d'état
- R est la fonction de récompense telle que $r(s) = E(R_{t+1} | S_t = s)$
- $\gamma \in [0, 1]$ est un facteur de remise permettant la convergence de la suite des récompenses
-

Le **retour** G_t est la somme des gains cumulés à partir de l'instant t tel que :

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

avec γ représentant la valeur actuelle des récompenses futures

La valeur des récompenses reçues R après $k + 1$ itérations est $\gamma^k R$.

Ainsi lorsque γ est proche de 0 on dit qu'il est "myope" et lorsqu'il est proche de 1, il est "hypermétrope".

La **fonction de valeur** $V(s)$ représente le retour moyen sur le long terme d'un état où
 $V(s) = E(G_t | S_t = s)$

Équation de Bellman :

$$\begin{aligned} V(s) &= E(G_t | S_t = s) = E(R_{t+1} + \gamma R_{t+2} + \dots | S_t = s) \\ &= r(s) + \gamma \sum_{s'} p(s, s') E(R_{t+2} + \gamma R_{t+3} + \dots | S_{t+1} = s') \\ &= r(s) + \gamma \sum_{s'} p(s, s') V(s') \end{aligned}$$

Forme Matricielle de l'équation de Bellman

$$\begin{pmatrix} V(1) \\ \vdots \\ V(n) \end{pmatrix} = \begin{pmatrix} r(1) \\ \vdots \\ r(n) \end{pmatrix} + \gamma \begin{pmatrix} p(1,1) & p(1,2) & \dots & p(1,n) \\ p(2,1) & p(2,2) & \dots & p(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ p(n,1) & p(n,2) & \dots & p(n,n) \end{pmatrix} \begin{pmatrix} V(1) \\ \vdots \\ V(n) \end{pmatrix}$$

Peut être résolue directement grâce : $V = (I - \gamma P)^{-1} R$ de complexité $O(n^3)$ pour n états ou grâce à de la programmation dynamique, l'évaluation de Monte-Carlo et l'apprentissage par différence temporel.

Un **processus de Markov à décision** est un processus de Markov à récompense avec décision et ainsi :

Un processus de Markov à décision est un tuple $\langle S, A, P, R, \gamma \rangle$ avec

S un ensemble fini d'états, A un ensemble fini d'actions, P une matrice de transition, R une fonction de récompense telle que $r(s, a) = E(R_{t+1} | S_t = s, A_t = a)$ et $\gamma \in [0, 1]$ un facteur de remise.

Une **politique** π est une répartition d'action : $\pi(s) = P(A_t | S_t = s)$ définissant entièrement le comportement d'un agent. Enfin, une politique peut être stationnaire ou indépendante du temps.

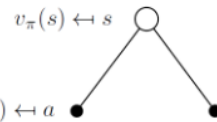
En considérant un processus de Markov à décision $\langle S, A, P, R, \gamma \rangle$ et une politique π , alors la séquence d'état S_1, S_2, \dots est un processus de Markov $\langle S, P^\pi \rangle$. Enfin, l'état et la séquence de récompense $S_1, R_1, S_2, R_2, \dots$ est un processus de Markov à récompense $\langle S, P^\pi, R^\pi, \gamma \rangle$ et on a

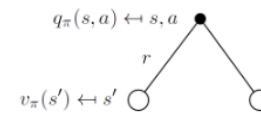
$$p^\pi(s, s') = \sum_{a \in A} \pi(a|s)p(s'|s, a) \text{ et } r^\pi(s) = \sum_{a \in A} \pi(a|s)r(s, a)$$

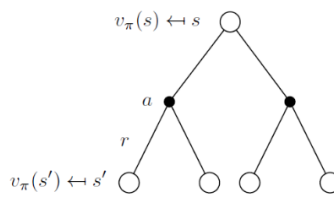
La **fonction de valeur** $V_\pi(s)$ d'un processus de Markov à décision se calcule de la façon suivante : $V_\pi(s) = E(G_t | S_t = s, A_{t:\infty} \sim \pi)$

De même, la **fonction valeur-action** vaut $q_\pi(s, a) = E(G_t | S_t = s, A_t = a, A_{t+1:\infty} \sim \pi)$

Equation des attentes de Bellman

$$V_\pi(s) = \sum_{a \in A} \pi(a|s)q_\pi(s, a)$$


$$q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)V_\pi(s')$$


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


$$q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \sum_{a' \in A} \pi(a'|s')q_\pi(s', a')$$

La **fonction de valeur optimale** notée $V_*(s)$ est la valeur maximale de la fonction de valeur quelque soit la politique d'où $V_*(s) = \max_\pi (V_\pi(s))$.

La **fonction d'action-valeur optimale** est la valeur maximale de la fonction d'action-valeur quelque soit la politique d'où $q_*(s, a) = \max_\pi (q_\pi(s, a))$.

Un processus de Markov à décision est résolu si l'on trouve $V_*(s)$ or $q_*(s, a)$.

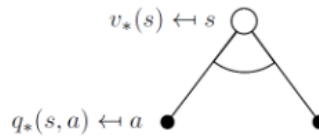
Une politique peut être classée par rapport à une autre : $\pi \geq \pi'$ si $V_\pi(s) \geq V_{\pi'}(s), \forall s$

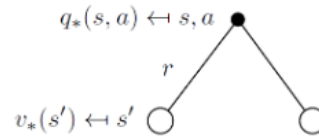
Théorème : Pour tout processus de Markov à décision :

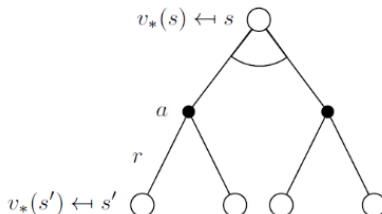
- Il existe une politique optimale π_* meilleure ou égale à toutes les autres

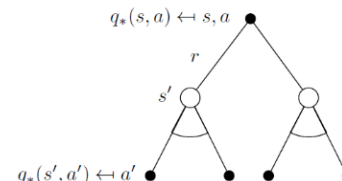
- Toutes les politiques optimales permettent d'atteindre la fonction de valeur optimale $V_{\pi^*}(s) = V_*(s)$
- Toutes les politiques optimales permettent d'atteindre la fonction d'action-valeur optimale $q_{\pi^*}(s, a) = q_*(s, a)$

Équation d'Optimalité de Bellman

$$V_*(s) = \max_{a \in A} q_*(s, a)$$


$$q_*(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V_*(s')$$


$$V_*(s) = \max_{a \in A} \left(r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V_*(s') \right)$$


$$q_*(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A} q_*(s', a')$$


Une **politique optimale** peut être trouvée en maximisant $q_*(s, a)$

$$\pi_*(a|s) = \begin{cases} 1 & \text{si } a = \operatorname{argmax}_{a \in A} q_*(s, a) \\ 0 & \text{sinon} \end{cases}$$

Processus de Markov à Décision fini

$$V_T^\pi = E(R_1 + R_2 + \dots + R_T | S_0 = i)$$

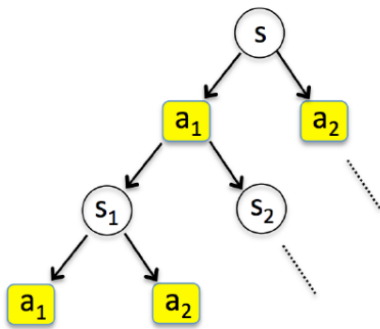
$$= E_{\pi} \left(\sum_{t=1}^T R_t \mid S_0 = i \right)$$

$$V_t(i) = \sup_{\pi} V_t^{\pi}(i)$$

Soit une action a , rapportant une récompense, $r(i, a)$, la meilleure action vaut

$$V^T(i) = \max_a \left(r(i, a) + \sum_j p(j|i, a) V^{T-1}(j) \right)$$

Idée de la clé de Bellman



Arbre de décision avec feuilles de profondeur $T: A^T S^{T+1}$

Représentant $S^2 A T$ opérations.

Chapitre 3 : Optimality

1 - Principe d'optimalité

1.1 - Définition

Soit $V_0^\pi(s) = E_\pi \left(\sum_{t=1}^T R_t | S_0 = s \right)$

Ainsi que la politique optimale $\pi^* = (\pi_1, \pi_2, \dots, \pi_T)$

Alors la queue de cette politique optimale (π_k, \dots, π_T) est optimale pour la fonction de valeur d'état prise à l'instant k :

$$V_k^\pi(s) = E_\pi \left(\sum_{t=k+1}^T R_t | S_k = s \right)$$

1.2 - Utilisation

Soit une action a dont le gain associé est $r(s, a)$

Le gain maximal que l'on peut obtenir est alors $r(s, a) + \sum_{s'} p(s'|s, a) \cdot V^{T-1}(s')$

Il vient alors $V^T(s) = \max_a [r(s, a) + \sum_{s'} p(s'|s, a) \cdot V^{T-1}(s')]$

2 - Dynamic Programming

Objectif des programmes informatiques : optimiser une politique. Cela se fait en fragmentant un problème principal en sous-problèmes.

Condition : Connaissance parfaite de la MDP

Applications :

- Pour faire de la prédiction, il faut connaître (S, A, P, R, γ) ainsi que π (on détermine alors V_π)
- Pour le contrôle, il faut connaître (S, A, P, R, γ) et on détermine alors π_*

Bellman Expectation Equation pour V_π

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

Bellman Optimality Equation pour V_π

$$V_*(s) = \max_{a \in A} [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V_*(s')]$$

Autres applications

- Algorithmes de planifications
- Algorithmes de graphes
- Modèles graphiques
- Bioinformatique

3 - Théorème de contraction (Contraction Mapping Theorem)

Définition : γ – contraction

Un opérateur T est appelé γ -contraction si

$$\|T(u) - T(v)\|_\infty \leq \gamma \|u - v\|_\infty$$

Théorème (Contraction Mapping Theorem)

Soit V un espace métrique complet, et $T(\cdot)$ une γ -contraction.

Alors :

- $T(\cdot)$ converge vers un unique point fixe
- Le taux de convergence est exponentielle en γ

Corollaire

La *Bellman Expectation Equation* est une γ – contraction

Bellman Expectation operator $T^\pi(V)$

On définit $T^\pi(V)$ comme tel : $T^\pi(V) = R^\pi + \gamma \cdot P^\pi \cdot V$

$$T^\pi(V)(s) = \sum_a \pi(a, s) \cdot [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V(s')]$$

Preuve que $T^\pi(V)$ est bien une γ – contraction

$$\text{On a } \|T(u) - T(v)\|_\infty = \|\gamma \cdot P^\pi(u - v)\|_\infty$$

$$\Rightarrow \|T(u) - T(v)\|_\infty \leq \|\gamma \cdot P^\pi\| \cdot \|u - v\|_\infty$$

Donc

$$\|T(u) - T(v)\|_{\infty} \leq \gamma \|u - v\|_{\infty}$$

4 - Évaluation itérative d'une politique (Iterative Policy Evaluation)

Objectif: évaluer une politique donnée π (= trouver V_{π})

Solution: on itère avec l'équation de Bellman en calculant $V_{k+1}(s)$ à partir de $V_k(s')$

Convergence : étant donné que l'opérateur $T^{\pi}(V)$ est une γ - contraction, il a un point fixe unique. Ce point fixe est V_{π} . Ainsi, l'itérative policy evaluation converge vers V_{π} .

$$\text{Expression: } V_{k+1}(s) = \sum_a \pi(a|s) \cdot [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V_k(s')]$$

5 - Équation d'optimalité de Bellman (Bellman Optimality Equation)

On définit V^* comme étant l'unique solution de

$$V_*(s) = \max_{a \in A} [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V_*(s')]$$

Toutes les politiques stationnaires π^* satisfaisant

$$\pi^*(s) \in \operatorname{argmax}_{a \in A} [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V_*(s')]$$

sont optimales.

5.1 - Preuve

On considère l'opérateur $T_*(V) = \max_{a \in A} [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V(s')]$

et on montre que c'est une contraction.

5.2 - Value iteration

Objectif: trouver V_* la fonction de valeur pour la politique optimale

$$V_{n+1}(s) = \max_{a \in A} [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V_n(s')]$$

Alors

$$\lim_{n \rightarrow +\infty} V_n = V^* \text{ (convergence exponentielle)}$$

On peut alors en déduire la politique optimale :

$$\pi^*(s) \in \operatorname{argmax}_{a \in A} [r(s, a) + \gamma \cdot \sum_{s' \in S} p(s'|s, a) \cdot V_*(s')]$$

5.3 - Exemple : Détermination de V^* à partir d'une politique aléatoire (Iterative policy evaluation)

La politique π est ici parfaitement aléatoire

(i.e. $\pi(a|s) = 1/4 \forall a \in \{NORD, SUD, EST, OUEST\}$)

v_k for the Random Policy

$k = 0$	<table><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr><tr><td>0.0</td><td>0.0</td><td>0.0</td><td>0.0</td></tr></table>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	$k = 1$	<table><tr><td>0.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr><tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr><tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>-1.0</td></tr><tr><td>-1.0</td><td>-1.0</td><td>-1.0</td><td>0.0</td></tr></table>	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	$k = 2$	<table><tr><td>0.0</td><td>-1.7</td><td>-2.0</td><td>-2.0</td></tr><tr><td>-1.7</td><td>-2.0</td><td>-2.0</td><td>-2.0</td></tr><tr><td>-2.0</td><td>-2.0</td><td>-2.0</td><td>-1.7</td></tr><tr><td>-2.0</td><td>-2.0</td><td>-1.7</td><td>0.0</td></tr></table>	0.0	-1.7	-2.0	-2.0	-1.7	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0	-1.7	-2.0	-2.0	-1.7	0.0
0.0	0.0	0.0	0.0																																																		
0.0	0.0	0.0	0.0																																																		
0.0	0.0	0.0	0.0																																																		
0.0	0.0	0.0	0.0																																																		
0.0	-1.0	-1.0	-1.0																																																		
-1.0	-1.0	-1.0	-1.0																																																		
-1.0	-1.0	-1.0	-1.0																																																		
-1.0	-1.0	-1.0	0.0																																																		
0.0	-1.7	-2.0	-2.0																																																		
-1.7	-2.0	-2.0	-2.0																																																		
-2.0	-2.0	-2.0	-1.7																																																		
-2.0	-2.0	-1.7	0.0																																																		
$k = 3$	<table><tr><td>0.0</td><td>-2.4</td><td>-2.9</td><td>-3.0</td></tr><tr><td>-2.4</td><td>-2.9</td><td>-3.0</td><td>-2.9</td></tr><tr><td>-2.9</td><td>-3.0</td><td>-2.9</td><td>-2.4</td></tr><tr><td>-3.0</td><td>-2.9</td><td>-2.4</td><td>0.0</td></tr></table>	0.0	-2.4	-2.9	-3.0	-2.4	-2.9	-3.0	-2.9	-2.9	-3.0	-2.9	-2.4	-3.0	-2.9	-2.4	0.0	$k = 10$	<table><tr><td>0.0</td><td>-6.1</td><td>-8.4</td><td>-9.0</td></tr><tr><td>-6.1</td><td>-7.7</td><td>-8.4</td><td>-8.4</td></tr><tr><td>-8.4</td><td>-8.4</td><td>-7.7</td><td>-6.1</td></tr><tr><td>-9.0</td><td>-8.4</td><td>-6.1</td><td>0.0</td></tr></table>	0.0	-6.1	-8.4	-9.0	-6.1	-7.7	-8.4	-8.4	-8.4	-8.4	-7.7	-6.1	-9.0	-8.4	-6.1	0.0	$k = \infty$	<table><tr><td>0.0</td><td>-14.</td><td>-20.</td><td>-22.</td></tr><tr><td>-14.</td><td>-18.</td><td>-20.</td><td>-20.</td></tr><tr><td>-20.</td><td>-20.</td><td>-18.</td><td>-14.</td></tr><tr><td>-22.</td><td>-20.</td><td>-14.</td><td>0.0</td></tr></table>	0.0	-14.	-20.	-22.	-14.	-18.	-20.	-20.	-20.	-20.	-18.	-14.	-22.	-20.	-14.	0.0
0.0	-2.4	-2.9	-3.0																																																		
-2.4	-2.9	-3.0	-2.9																																																		
-2.9	-3.0	-2.9	-2.4																																																		
-3.0	-2.9	-2.4	0.0																																																		
0.0	-6.1	-8.4	-9.0																																																		
-6.1	-7.7	-8.4	-8.4																																																		
-8.4	-8.4	-7.7	-6.1																																																		
-9.0	-8.4	-6.1	0.0																																																		
0.0	-14.	-20.	-22.																																																		
-14.	-18.	-20.	-20.																																																		
-20.	-20.	-18.	-14.																																																		
-22.	-20.	-14.	0.0																																																		

6 - Améliorer une politique : Policy Iteration

6.1 - Principe

Soit π une politique et V^π la fonction de valeurs associées.

On définit $\bar{\pi} = \text{greedy}(V^\pi)$ la politique 'greedy' par rapport à V^π telle que

$$\bar{\pi}(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^\pi(s') \right) \quad \forall s.$$

6.2 - Proposition : amélioration de la politique

La politique $\bar{\pi}(s)$ obtenue est telle que $V^{\bar{\pi}}(s) \geq V^\pi(s)$.

On a égalité si et seulement si π est optimale.

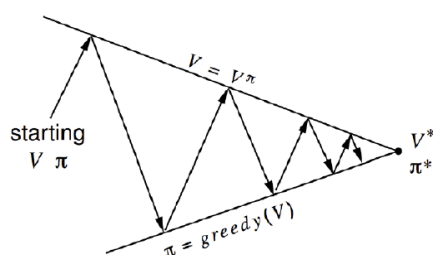
6.3 - Idée de preuve

En reprenant l'opérateur $T_{\bar{\pi}}$ précédemment défini, on peut montrer qu'il est monotone ($V_1 \leq V_2 \Rightarrow T_{\bar{\pi}}(V_1) \leq T_{\bar{\pi}}(V_2)$)

Alors on peut montrer, en l'appliquant à plusieurs reprises, que

$$V^\pi \leq T_{\bar{\pi}} V^\pi \leq (T_{\bar{\pi}})^2 V^\pi \leq \dots \leq \lim_{n \rightarrow \infty} (T_{\bar{\pi}})^n V^\pi = V^{\bar{\pi}}$$

6.4 - Algorithme Policy Iteration



Initialisation : on part d'une politique choisie arbitrairement (aléatoire par exemple) notée π_0 .

Itération k : On calcule V^{π_k} la fonction de valeur associée à la politique π sachant que

$$V_{\pi_k} = (I - \gamma \mathcal{P}^{\pi_k})^{-1} \mathcal{R}^{\pi_k}$$

Si la nouvelle fonction de valeur obtenue est égale à la précédente ($V_{\pi_k} \neq V_{\pi_{k-1}}$) on a terminé (Goto **Résultat**).

Sinon, la connaissance de V_{π_k} nous permet d'améliorer la politique, en déterminant la politique 'greedy' comme indiquée dans la section 'Principe'.

$$\pi_{k+1} = greedy(\pi_k)$$

On continue ensuite d'itérer (Goto **Itération k+1**).

Résultat : On obtient la politique optimale. $\pi_k = \pi^*$

7 - Récapitulatif des 'Synchronous Dynamic Programming Algorithm'

Pour faire de la **prédiction** (trouver V_{π}) => On utilise 'Bellman Expectation Equation' avec l'algorithme 'Iterative Policy Evaluation'

Pour faire du **contrôle** (trouver π_*) => On utilise 'Bellman Expectation Equation' + 'greedy' avec l'algorithme 'Policy Iteration'

Pour faire du **contrôle** (trouver V_*) => On utilise 'Bellman Optimality Equation' avec l'algorithme 'Value Iteration'

La complexité est en $O(mn^2)$ pour m actions et n états.

Ces algorithmes peuvent être également appliqués à la fonction de valeurs et états $q_*(s, a)$ ou $q_{\pi}(s, a)$ la complexité étant en $O(m^2n^2)$ par itération (???).

8 - Asynchronous Dynamic Programming

Sauvegarde en mode **synchronous** : tous les états sont sauvegardés en parallèle

Sauvegarde en mode **asynchronous DP**: les états sont sauvegardés les uns à la suite des autres, dans n'importe quel ordre.

Pour chaque état sélectionné, on utilise la sauvegarde correspondante.

=> Temps de calcul réduit pour le 'Asynchronous DP'

=> Convergence garantie si tous les états continus d'être sélectionné

8.1 - Approches

3 approches simples :

- Programmation dynamique 'en place'

- Balayage avec priorité ('Prioritised sweeping')
- Dynamic Programming 'temps réel'

8.1.1 - Programmation dynamique 'en place'

On enregistre seulement une copie de la fonction de valeur

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s') \right)$$

8.1.2 - Balayage avec priorité ('Prioritised sweeping')

On utilise un ordre de grandeur sur la 'Bellman error' pour orienter la sélection des états.

$$\left| \max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s') \right) - V(s) \right|$$

On sauvegarde alors les états dont la 'Bellman error' est la plus importante.

On met à jour leur 'Bellman error' après chaque sauvegarde.

Peut-être implémenté efficacement en utilisant une file à priorités.

8.1.3 - Dynamic Programming 'temps réel'

Idée:

- On se concentre sur les états qui sont pertinents pour l'agent.
- Utilisation de l'expérience de l'agent pour orienter la sélection des états
- Après chaque instant de temps, on sauvegarde l'état S_t

$$V(s_t) \leftarrow \max_{a \in \mathcal{A}} \left(r(s_t, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s_t, a) V(s') \right)$$

8.2 - Full-width backups

Utilisés par 'Dynamic Programming'.

8.2.1 - Principe

Pour chaque sauvegarde, tous les états pouvant succéder à l'état sauvegardé ainsi que les actions associées sont considérés.

On utilise la connaissance du MDP pour les fonctions de transition et de récompense.

8.2.2 - Utilisations

=> 'Dynamic Programming' est efficace pour des problèmes de taille moyenne (quelques millions d'états)

=> Pour des problèmes plus grands, DP est peu pertinent car le nombre d'états croît de manière exponentielle avec le nombre de variables d'états

=> Même une unique sauvegarde peut devenir très coûteuse

9 - Récompense moyenne MDP

Un MDP ergodique a une récompense moyenne associée par étape g^π indépendante de la date de départ.

$$g^\pi = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E} \left[\sum_{t=1}^T R_t \right]$$

La fonction de valeur est définie comme suit :

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[\sum_{k=1}^{\infty} (R_{t+k} - g^\pi) | S_t = s \right] \\ &= \mathbb{E}_\pi [(R_{t+1} - g^\pi) + V_\pi(S_{t+1}) | S_t = s] \\ &= r(s, a) - g^\pi + \sum_{s'} p(s' | s, a) V_\pi(s') \end{aligned}$$

Chapitre 4 : Performance and control

1 - Prediction

1.1 - Policy Evaluation : Monte Carlo Methods

Ici, on va fixer π et le but est d'évaluer la valeur de la fonction :

$$V(s) = E\left(\sum_{k=0}^T \gamma^k R_{t+k+1} | S_t = s\right)$$

Monte-Carlo utilise "empirical mean return" au lieu de "expected return".

On va simuler le système toujours sous la politique π :

- Au temps t_s après accès à l'état s , on ajoute le coût total jusqu'à ce qu'on atteigne

$$\text{l'objectif : } \hat{v}(s) = \sum_{t=t_s}^T R_t$$

- Après k visites à s , on obtient une séquence de coups total estimés: $\hat{v}_1(s), \dots, \hat{v}_k(s)$
- L'estimation vaut alors : $\hat{V}_k(s) = 1/k \sum_{i=1}^k \hat{v}_i(s)$
- LLN (loi des grands nombres) assure que si $k \rightarrow \infty$, $\hat{V}_k(s) \rightarrow V(s)$ a. s.

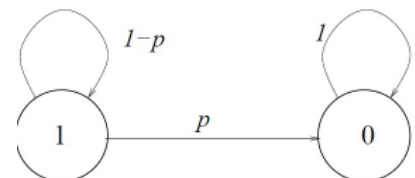
1.2 - State Counting Options

Il y a 3 options :

- Calculer $\hat{v}(s)$ à l'état initial uniquement;
- Calculer $\hat{v}(s)$ à chaque fois que l'on passe par l'état s . (donne le plus d'échantillons mais est corrélé);
- Calculer $\hat{v}(s)$ à la première visite de chaque essai.

Exemple : Comment choisir la meilleur option sur cet exemple :

$$\mathbb{E}((\hat{V} - V)^2) = \underbrace{\mathbb{E}(\hat{V}) - V}_{\text{Bias}^2} + \underbrace{\mathbb{E}((\hat{V} - \mathbb{E}(\hat{V}))^2)}_{\text{Variance}}$$



K est une R.V. géométrique de moyenne $E(K) = 1/p$

On fait MC first visit, il est non biaisé car la récompense après une trajectoire vaut K ,

$$E(K) = 1/p = V^\pi(1)$$

L'erreur au carré moyenne est : $E((K - 1/p)^2) = 1/p^2 - 1/p$

On fait maintenant MC Multiple Visits, sur une trajectoire la récompense est :

$$\frac{1}{K} \sum_{k=0}^{K-1} (K - k) = (K + 1)/2; \text{ on a alors } \mathbb{E}\left(\frac{K+1}{2}\right) = \frac{1+p}{2p} \neq V^\pi(1)$$

Mais sur plusieurs trajectoires, la moyenne empirique converge vers $1/p$, et l'erreur

quadratique vaut alors : $\mathbb{E}\left(\left(\frac{K+1}{2} - \frac{1}{p}\right)^2\right) = \frac{1}{2p^2} - \frac{3}{4p} + \frac{1}{4}$

On trouve donc que MC Multiple Visits est plus judicieux ici.

1.3 - Moyenne incrémentée

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum^k x_j \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

1.4 - Estimation d'une moyenne inconnue

Soit $X \in [0, 1]$ une variable aléatoire de moyenne inconnue notée $\mu = E(X)$

On considère $x_n \sim X$ un ensemble de variables aléatoires suivant la loi X , indépendantes et identiquement distribuées. On pose $\mu_1 = x_1$ et pour $n > 1$

$$\mu_n = (1 - \alpha_n)\mu_{n-1} + \alpha_n x_n$$

Proposition :

Si $\sum_{t=1}^{\infty} \alpha_t = \infty$ et $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$ alors $\mu_n \rightarrow \mu \text{ a.s.}$

1.5 - Monte Carlo Itératif

$$\hat{V}_k(s) = \hat{V}_{k-1}(s) + \gamma_k(\hat{V}_k(s) - \hat{V}_{k-1}(s))$$

with $\gamma_k = 1/k$ it will converge to $V(s)$ a.s.

1.6 - Temporal-Difference Learning (TD)

TD apprend des épisodes de l'expérience. On n'a pas besoin de connaître le modèle (les transitions du MDP / gains)

TD apprend aussi des épisodes non finis et met à jour en supposant la suite.

1.7 - TD(0) : Simplest temporal-difference learning algorithm

Formule : $V(S) \leftarrow V(S) + \alpha(R + \gamma V(S') - V(S))$

$R + \gamma V(S')$ est un estimateur non biaisé

$\hat{V}(S)$ est mis à jour selon la valeur de $\hat{V}(S')$
=> bootstrapping

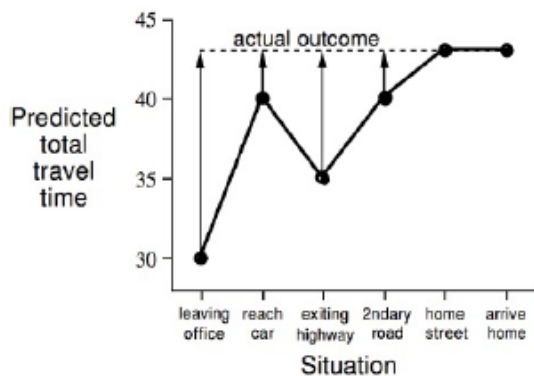
On a alors

$$\hat{V}(S) = (1 - \alpha_n)\hat{V}(S) + \alpha_n(R + \hat{V}(S'))$$

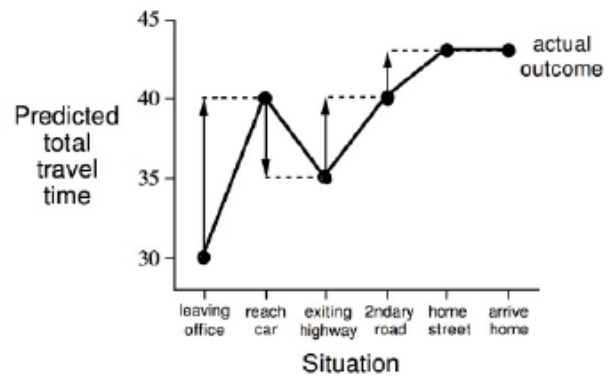
Il y a convergence si $\alpha_n \rightarrow 0$

Par exemple $\alpha_n \approx 1/\text{number of visits to } S$

Changes recommended by Monte Carlo methods ($\alpha=1$)



Changes recommended by TD methods ($\alpha=1$)



1.8 - Comparaison entre MC et TD

1.8.1 - Applicatif :

TD peut apprendre après chaque étape, avant la fin de la simulation et même si celle-ci ne se termine jamais, alors que MC doit attendre la fin de la simulation et ne peut apprendre que si la séquence en question se termine.

TD fonctionne dans un environnement continu (non terminé) alors que MC fonctionne uniquement dans un environnement épisodique (terminé).

1.8.2 - Convergence / Précision

MC:

- variance élevé, pas de biais
- Bonne propriétés de convergence (même avec des approximations de fonctions)
- Peu sensible à la valeur initiale

TD:

- variance faible, un peu biaisé
- Souvent plus efficace que MC
- TD(0) converge vers $V_{\pi}(s)$ (mais pas toujours, avec des approximations de fonctions)
- Plus sensible à la valeur initiale

2 - Contrôle :

On rappelle que la fonction d'action-valeur associée à une action 'a' et un état 's' est :

$$q_{\pi}(s, a) = \mathbb{E} \left(R_{t+1} + \alpha R_{t+2} + \alpha^2 R_3 + \dots \mid S_t = s, A_t = a, A_{t+1:\infty} \sim \pi \right)$$

2.1 - Politique optimale

Une politique π_* est optimale si $q_*(s, a) = q_{\pi_*}(s, a) = \max q_{\pi}(s, a)$

De plus, on a alors $\pi_*(s) = \operatorname{argmax}_a q_*(s, a)$

On cherche par la suite la valeur de $q_*(s, a)$

2.2 - On & off policy learning

On-policy Learning :

- On apprend sur le tas (on the go)
- On apprendre sur la politique π à partir des expériences échantillonnées de π

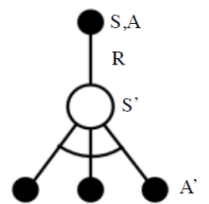
Off-policy learning:

- on repose sur quelqu'un d'autre
- On apprend la politique π à partir des expériences échantillonnées de μ

2.3 - Off-Policy learning : Q-learning

$$Q(S, A) \leftarrow Q(S, A) + \gamma \left(R + \alpha \max_{a'} Q(S', a') - Q(S, a) \right)$$

Pour une bonne valeur de γ , Q converge vers q^*



2.4 - Q-learning Algorithmme

```
Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ ;
  until  $S$  is terminal
```

2.5 - Apprendre la fonction action-valeur d'une politique donnée (SARSA)

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

2.6 - SARSA Algorithmme

```
Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A'$ ;
  until  $S$  is terminal
```

Theoreme de la convergence de SARSA :

Si $\sum_{t=1}^{\infty} \gamma_t = \infty$ et $\sum_{t=1}^{\infty} \gamma_t^2 < \infty$ alors :

On converge alors vers la fonction action-valeur $Q(s, a) \rightarrow q_{\pi}(s, a)$

2.7 - On vs Off policies



Les méthodes type On-Policy (Sarsa) sont plus performantes mais trouvent des politiques moins bonnes

2.8 - Approximation des fonctions de valeur

Problème : trop d'état / action et l'on a plus assez de mémoire pour stocker tout ça.

=> On recherche alors des solutions pour des MDPs plus grandes.

On estime la valeur des fonctions avec une fonction d'approximation : $\hat{q}(s, a, \mathbf{w}) \approx q_{\pi}(s, a)$

2.9 - Quelle approximation ?

- Combinaisons linéaires de caractéristiques
- réseau neuronal
- Arbre de décision
- Plus proche voisin
- Fourier / wavelet bases

Minimisation de l'erreur des moindres carrés entre les approximations

$$J(\mathbf{w}) = \mathbb{E}_{\pi} \left[(R_{t+1} + \alpha \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))^2 \right]$$

Descente du gradient stochastique pour trouver le minimum local

$$\Delta \mathbf{w} = \gamma (R_{t+1} + \alpha \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

2.10 - Approximateur linéaire :

On représente un état par son vecteur de caractéristiques $\mathbf{x}(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix}$

Linéaire:

$$\hat{q}(s, i, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w}$$

2.11 - Convergence :

Approximation linéaire :

- convergence pour la fonction valeur
- erreur bornée ☹ pour le contrôle

Fonction d'approximation non linéaire :

- Contre-exemples artificiels
- Travail en pratique

2.12 - Model-Based et Model-Free RL

Model-Free RL

- Pas de modèle
- Apprend la fonction de valeur et la politique par expérience

Model-Based RL:

- On apprend le modèle par l'expérience
- On s'appuie sur le modèle

$$\hat{\mathbb{P}}(S_{t+1}, R_{t+1} | S_t, A_t) \approx \mathbb{P}(S_{t+1}, R_{t+1} | S_t, A_t)$$

2.13 - Simulation-based search

On simule en utilisant le modèle (et en utilisant Model-Free RL)

