

Apprentissage par Renforcement

Rui Shibasaki

2025

1. Introduction

- L'apprentissage par renforcement est une approche tournée vers l'interaction entre la machine et son environnement
- Plus proche de l'apprentissage humain.
- Paradigme: Exploration/Exploitation (Pas la nécessité d'une quantité massive de données)
- Différent de l'apprentissage supervisé/non-supervisé où l'environnement est déjà exploré en quelque sorte. (Nécessité d'une quantité massive de données)

Apprentissage par renforcement

- Problématique: Apprendre que faire de façon à maximiser la récompense de mes actes. (Similaire au cerveau humain qui apprend pour un système de récompense)
- Éléments de l'apprentissage par renforcement:
 - ▶ L'agent
 - ▶ L'environnement
 - ▶ Les règles et associations (comment prendre les actions)
 - ▶ La récompense
 - ▶ La fonction de valeur.
- Alors que la récompense est immédiate la fonction de valeur indique ce qui est bien dans le long terme
- Une action peut générer peu de récompense mais sa valeur peut être grande si, dans le long terme, elle permettra des grande récompenses. Ex. Les études (Ou l'inverse, ex. l'addiction)

K-armed bandit

- Problématique:
 - ▶ On est constamment confronté à faire un choix parmi k .
 - ▶ Après chaque action on a une récompense donné par une distribution de probabilité qui dépend de l'action prise.
 - ▶ Objectif: Maximiser le gain total espéré après une période donnée. Ex.: Après 1000 actions

- Chaque action a possède une valeur (récompense espérée) à l'instant t :

$$q_*(a) = \mathbb{E}[R_t | A_t = a]$$

- Si les valeurs de q_* étaient connus, alors le problème est trivial: toujours choisir l'action avec la plus grande valeur. (Normalement c'est pas le cas.)
- On peut avoir des estimations $Q_t(a)$ pour chaque action a à l'instant t .
- On voudrait que Q_t soit le plus proche de q_* pour toute action a

K-armed bandit

Actions gloutonnes

Les actions qui ont les valeurs les plus élevés de Q_t

Exploitation

Choisir des actions gloutonnes

Exploration

Choisir des actions avec des valeurs Q_t plus basses que les gloutonnes.

- Exploitation VS Exploration: Il faut trouver un équilibre. Mais comment?
- Exploiter fait maximiser ma récompense.
- Mais explorer peut améliorer mes estimations et révéler une action qui donne au fait une récompense encore plus grande.

K-armed bandit - Comment estimer la valeur?

Moyenne de l'échantillon

$$Q_t(a) := \frac{\text{somme des récompenses quand } a \text{ a été choisi}}{\text{nombre de fois que } a \text{ a été choisi}}$$

Règle de sélection gloutonne:

$$A_t := \operatorname{argmax}_a Q_t(a)$$

Règle de sélection ϵ -gloutonne:

Glouton la plus part du temps mais choisir une action aléatoirement avec une probabilité de ϵ . Ex.: Considérons le cas de $k = 2$ et $\epsilon = 0.5$ alors la probabilité de choisir l'action gloutonne est de 0.75.

K-armed bandit - Algorithmes

Algorithm Algorithme du bandit manchot simple

- 1: Initialiser, pour chaque $a \in \{1, \dots, k\}$ $Q(a) = 0$ et $N(a) = 0$
 - 2: **while** objectif pas atteint **do**
 - 3: $A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{avec probabilité } 1 - \epsilon, \\ \text{une action aléatoire parmi les } k & \text{avec probabilité } \epsilon. \end{cases}$
 - 4: $R \leftarrow \text{recompense}(A)$
 - 5: $N(A) \leftarrow N(A) + 1$
 - 6: $Q(A) \leftarrow Q(a) + \frac{1}{N(A)}[R - Q(A)]$
 - 7: **end while**
-

Calcule efficace de la moyenne

$$Q_n := \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \text{ alors } Q_{n+1} = \frac{1}{n} \sum_{i=1}^n R_i$$

$$\text{On peut prouver que } Q_{n+1} := \frac{1}{n} \sum_{i=1}^n R_i = Q_n + \frac{1}{n}(R_n - Q_n)$$

K-armed bandit - Cas non-stationnaire

Jusqu'à présent nous avons considéré que les probabilités des récompenses ne changent pas avec le temps (**cas stationnaire**). Néanmoins la plus part des cas sont **non-stationnaires**.

Cas non-stationnaires

Il vaut mieux prioriser les récompenses récentes:

$$Q_{n+1} := Q_n + \alpha(R_n - Q_n)$$

le paramètre $\alpha \in (0, 1]$ étant une constante de pas. Cela résulte plutôt dans une moyenne pondérée, où les récompenses récentes ont un poid plus important:

$$Q_{n+1} := Q_n + \alpha(R_n - Q_n) = (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} R_i$$

Exercice en TD

On peut montrer que $(1 - \alpha)^n + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} = 1$

K-armed bandit - Récompenses initiales

- On peut remarquer que la valeur estimée des actions dépend de l'estimation initiale Q_1 à un certain degré. On peut dire que les valeurs présentées sont **biaisées** par leur estimations initiales.
- Pour le cas stationnaire, par la loi des grands nombres

$$\frac{1}{n} \sum_{i=1}^n R_i \rightarrow q_* \text{ quand } n \rightarrow \infty$$

donc le biais disparaît, au fur et à mesure que toutes les actions ont été choisies suffisamment de fois.

- Par contre, pour le cas non-stationnaire le biais est permanent.
- Ce biais peut être utilisé pour encourager l'exploration. Dans une approche gloutonne sur les $Q_1(a)$ super-estimés, la tendance est que les actions soient choisies au moins une fois dans les k premières itérations.

K-armed bandit - Exploration/Exploitation

- Une sélection d'actions gloutonnes ne permet pas l'exploration
- Une sélection ϵ -gloutonnes permet l'exploration mais à l'aveugle.
- Nous pouvons essayer de guider l'exploration donnant une préférence aux actions presque gloutonnes ou complètement incertaines (Upper Confidence Bound):

$$A_t := \operatorname{argmax}_a \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

K-armed bandit - Upper Confidence Bound

- On veut que $q_* \in [Q_t - \tau, Q_t + \tau]$ avec un certain degré de confiance $1 - \delta$.
- Le théorème de Hoeffding dit que pour une série de variables aléatoires i.i.d $R_t \in [0, 1]$:

$$P(|\bar{R} - \mathbb{E}[\bar{R}]| \geq \tau) \leq 2 \exp(-2n\tau^2).$$

Pour une probabilité de $\delta = 1/t^2$ on a que

$$\delta = 2 \exp(-2n\tau^2) \implies \tau = \sqrt{\frac{\ln(2/\delta)}{2n}} \implies \tau = c \sqrt{\frac{\ln t}{n}}$$

avec

$$c = \sqrt{\frac{\ln 2}{2n}}$$

- On prend alors la limite supérieure:

$$Q_t + c \sqrt{\frac{\ln t}{n}}$$

K-armed bandit - Upper Confidence Bound

- Donc, pour chaque action a on estime qu'avec un degré de confiance de $(1 - 1/t^2)$, $q_*(a) \in \left[Q_t(a) - c\sqrt{\frac{\ln t}{N_t(a)}}, Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$
- Le niveau de confiance augmente avec les itérations $(1 - 1/t^2)$, mais on peut le changer (augmenter/diminuer) ce degré en changeant juste c .
- Si Q_t est très grand, on a beaucoup de chances de le choisir. Si $N_t(a)$ est petit, le terme de la racine carrée est plus grand et augmente les chances de l'action d'être choisie.

Gradient bandit algorithme

- Jusqu'à présent on a fait une estimation de la valeur des actions pour après faire le choix des actions selon leur valeur.
- Une autre méthode: Gradient bandit algorithms.
- On établit des préférences (notées $H_t(a)$) mais qui n'ont pas de relation directe avec les récompenses. Plus grande préférence, plus grandes chances d'être choisie.
- les probabilités pour chaque action sont données par la distribution soft-max (aussi connue comme distribution de Gibbs ou de Boltzmann)

$$P(A_t = a) := \pi_t(a) = \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}}$$

- Distribution qui tombe bien: petite différence sur H_t grande différence sur e^{H_t} (exponentiel) et le terme dénominateur normalise les valeurs dans l'intervalle $[0,1]$.

Gradient bandit algorithm

- L'apprentissage se donne, à chaque itération, par la mise à jour des préférences:

$$H_{t+1}(A_t) := H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(a))$$

$$H_{t+1}(a) := H_t(a) - \alpha(R_t - \bar{R}_t)(\pi_t(a)) \quad \forall a \neq A_t$$

- α étant en paramètre de valeur strictement positive représentant une mesure de pas.
- Ce processus peut être vu comme un algorithme de descente/ascension du gradient (à voir par la suite)

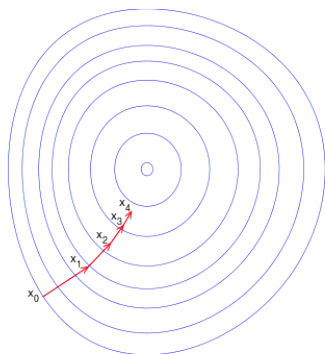
Gradient Ascent/Descent

- Le Gradient Ascent est une méthode d'optimisation/recherche utilisée pour maximiser une fonction objective. (Descent pour Minimiser)
- Soit une fonction $f(\mathbf{x})$ qu'on souhaite maximiser.
- Mise à jour des paramètres :

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla f(\mathbf{x}_t)$$

- Où :
 - ▶ \mathbf{x}_t : Position actuelle.
 - ▶ $\eta > 0$: mesure de pas.
 - ▶ $\nabla f(\mathbf{x}_t)$: Gradient de la fonction au point \mathbf{x}_t .

Gradient Ascent/Descent: Intuition graphique



- On part d'une position initiale.
- À chaque itération, on se déplace dans la direction du gradient avec un pas η . La direction du gradient donne la direction pour maximiser $f(\mathbf{x})$. (Pour la descente on va donc dans la direction contraire au gradient)

Gradient Ascent/Descent

- Le Gradient Ascent converge lorsque $\nabla f(\mathbf{x}) = 0$, ce qui correspond à un point critique de f .
- Attention : Le point trouvé peut être :
 - ▶ Un maximum global.
 - ▶ Un maximum local.
- Le choix de η est crucial :
 - ▶ η trop grand : Divergence.
 - ▶ η trop petit : Convergence lente.

Exercice (TD)

Montrer que le Gradient bandit algorithm peut être interprété comme un algorithme de Gradient Ascent.

Gradient Ascent/Descent: Exemple simple

- Maximisons la fonction $f(x) = -x^2 + 4x$.

- Gradient :

$$\nabla f(x) = \frac{d}{dx}(-x^2 + 4x) = -2x + 4$$

- Mise à jour :

$$x_{t+1} = x_t + \eta(-2x_t + 4)$$

Exercice: Exemple numérique avec $x_0 = 0$ et $\eta = 0.5$.

Récapitulation

- Jusqu'à présent nous n'avons considéré que les processus ayant un ensemble de k actions chacune ayant une valeur $q_*(a)$ et qu'à chaque prise de décision les situations ne changent pas. On se retrouve dans le même état avec les mêmes actions à choisir.
- Néanmoins, on sait que dans la plus part du temps, les actions influencent pas que les récompenses immédiates mais aussi les états futurs (les situations futures) ainsi que les récompenses futures.
- Nous appelons ce type de processus un **Processus de décision markovien (MDP)**

Processus de décision markovien (MDP)

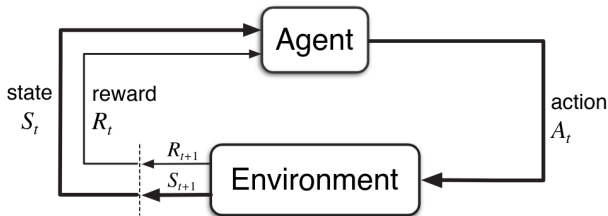


Figure: From Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

- Dans un MDP **fini** l'ensemble d'actions, états et récompenses $(\mathcal{A}, \mathcal{S}, \mathcal{R})$ sont finies. Dans ce cas, chaque $s' \in \mathcal{S}$ et $r \in \mathcal{R}$ a une probabilité d'occurrence dans l'instant t étant donné l'état actuel et l'action prise:

$$p(s', r|s, a) := \Pr(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$$

Processus de décision markovien (MDP)

- On assume la propriété de Markov:

Propriété de Markov

Les probabilités des valeurs de R_t et S_t ne dépendent que des valeurs de S_{t-1} et A_{t-1} , pas plus anciens.

$$p(s', r \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = p(s', r \mid s_t, a_t).$$

- Pour chaque état et action courant:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Processus de décision markovien (MDP)

- L'objectif reste le même: maximiser la récompense.
- Comme en économie, les valeurs des récompenses futurs peuvent être projetées dans le présent à l'aide d'un facteur de correction.
- Nous pouvons donc établir la récompense espérée donné par:

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=t+1}^T \gamma^{k-t-1} R_k = \sum_{k=0}^T \gamma^k R_{k+t+1}$$

où γ est le facteur de correction et $0 \leq \gamma \leq 1$. T est l'instant final (il peut aussi être infini, pas d'instant final)

- Remarquez que

$$G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots)$$

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

MDP: fonctions de valeur et politiques

- Les algorithmes d'apprentissage par renforcement consistent en estimer des **fonctions de valeur** d'un état ou de la paire état-action.
- Les actions à prendre seront déterminés selon une **politique** $\pi(a|s)$, normalement une probabilité de choisir l'action a étant donné l'état s .
- On peut définir la fonction de valeur d'un état sous la politique π comme

$$v_{\pi}(s) := \mathbb{E}_{\pi} [G_t \mid S_t = s]$$

- De même, on peut définir la fonction de valeur de l'action a à l'état s sous la politique π comme

$$q_{\pi}(s, a) := \mathbb{E}_{\pi} [G_t \mid S_t = s, A_t = a]$$

- Notez que $v_{\pi}(s) = \sum_a \pi(a \mid s) q_{\pi}(s, a)$

MDP: l'équation de Bellman

$$v_{\pi}(s) := \mathbb{E}_{\pi} [G_t \mid S_t = s]$$

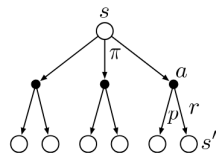
$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma G_{t+1} \mid S_t = s] \text{ et}$$

$$\mathbb{E}_{\pi} [G_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \mathbb{E}_{\pi} [G_{t+1} \mid S_{t+1} = s'] \text{ alors}$$

$$v_{\pi}(s) = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma \mathbb{E}_{\pi} [G_{t+1} \mid S_{t+1} = s']]$$

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

L'équation établit la relation entre la valeur de l'état courant et la valeur des états successeurs. Elle fait la moyenne de toutes les possibilités, pondérées par leur probabilité d'occurrence.



Backup diagram for v_{π}

MDP: l'équation de Bellman

Comme pour $v_\pi(s)$, on a une version de l'équation avec $q_\pi(s, a)$:

$$q_\pi(s, a) = \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a], \text{ comme}$$

$$\mathbb{E} [G_{t+1} \mid S_t = s, A_t = a] = \sum_{s'} p(s' \mid s, a) \mathbb{E} [G_{t+1} \mid S_{t+1} = s'] \text{ alors}$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')] \text{ ou encore}$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right].$$

MDP: Politiques optimales

- Remarquez qu'on a une équation par état.
- On a donc un système de $|\mathcal{S}|$ équations à $|\mathcal{S}|$ variables
- Résoudre ce système c'est déterminer la valeur des états $s \in \mathcal{S}$.
- Une fonction de valeur optimale sera telle que:

$$v_*(s) := \max_{\pi} v_{\pi}(s) \quad \forall s \in \mathcal{S}$$

- Une politique est meilleure qu'une autre ($\pi \geq \pi'$) si et seulement si $v_{\pi}(s) \geq v_{\pi'}(s) \quad \forall s \in \mathcal{S}$
- La politique optimale est $\operatorname{argmax}_{\pi} v_{\pi}(s) \quad \forall s \in \mathcal{S}$ (il peut y avoir plusieurs)
- De même, on a $q_*(s, a) := \max_{\pi} q_{\pi}(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

MDP: Équation de Bellman sous la politique optimale

- La première étape consiste à comprendre l'équation de Bellman pour la politique optimale π^* . Sous cette politique, la relation entre la fonction de valeur optimale d'état et la fonction de valeur optimale d'action est donnée par :

$$v_*(s) = \max_a q_*(s, a).$$

ce qui est équivalent à

$$v_*(s) = \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')].$$

- Et la relation:

$q_*(s, a) = \mathbb{E} [R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$ nous permet d'affirmer que:

$$q_*(s, a) = \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

MDP: limitations

- Résoudre le système d'équations de politique optimale de Bellman nous donnerai toutes les valeurs de $q_*(s, a)$ pour chaque $s \in \mathcal{S}$, $a \in \mathcal{A}$.
- Si on connaît toutes les valeurs de la fonction $q_*(s, a)$, alors le problème est résolu
- il suffit de suivre la politique optimale (choisir l'action qui donne $q_*(s, a)$) à chaque état.
- Limitations:
 - ▶ La propriété de Markov n'est pas toujours présente.
 - ▶ On ne connaît pas toute la dynamique de l'environnement pour définir les probabilités utilisées dans les équations.
 - ▶ Connaître les valeurs de $q_*(s, a)$ implique la résolution d'un système de équations non-linéaires potentiellement grand.

Programmation dynamique: résoudre le système d'équations de Bellman

La programmation dynamique (DP)

Consiste à:

- ➊ Décomposer un problème en sous-problèmes plus simples
- ➋ Résoudre chacun de ces sous-problèmes une seule fois
- ➌ Stocker leurs résultats pour les réutiliser par la suite
- ➍ Cela permet d'éliminer les recalculs inutiles, réduisant considérablement la charge de calcul.

Structure Optimale

Sa solution optimale peut être construite efficacement à partir des solutions optimales de ses sous-problèmes. Par exemple, le chemin le plus court dans un graphe peut être déterminé en combinant les chemins les plus courts des sous-parcours.

Programmation dynamique: résoudre le système d'équations de Bellman

- **Objectif:** résoudre les équations de Bellman et trouver la politique optimale.
- La programmation dynamique (DP) classique a une utilité limitée:
 - ▶ Considère un modèle parfait (toutes les bonnes conditions réunies - ce qui n'est rarement le cas en pratique)
 - ▶ Demande un grand effort computationnel
- Donne les fondements théoriques: toutes les autres méthodes essayent de se rapprocher de son fonctionnement.
- La plus part des algorithmes essayent de résoudre les équations de Bellman de manière approximative et ainsi approximer les valeurs de q_* .

Programmation dynamique: évaluation d'une politique π

Algorithm Evaluation itérative d'une politique

- 1: **Input:** π , la politique à être évaluée
 - 2: **Initialize:** Un tableau $V(s) = 0$, pour tout $s \in S^+$
 - 3: **repeat**
 - 4: $\Delta \leftarrow 0$
 - 5: **for all** $s \in S$ **do**
 - 6: $v \leftarrow V(s)$
 - 7: $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma V(s'))$
 - 8: $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
 - 9: **end for**
 - 10: **until** $\Delta < \theta$ (un paramètre de valeur petite et > 0)
 - 11: **Output:** $V \approx v_\pi$
-

Le principe de la programmation dynamique consiste du fait que la série $\{v_k\} \rightarrow v_\pi$ quand $k \rightarrow \infty$

Itération Général sur les Politiques

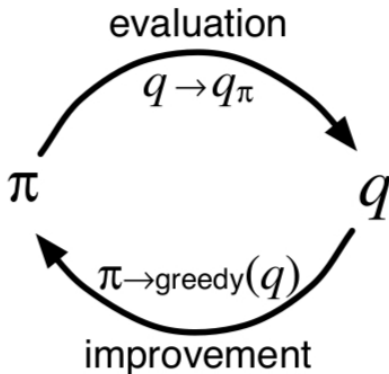


Figure: Itération Général sur les Politiques

- Evaluation (Évaluation, Estimation)
- Improvement (Amélioration, Contrôle)

Programmation dynamique: recherche d'une politique optimale π^*

Algorithm Itération sur les Politiques

- 1: Initialiser $V(s) \in \mathbb{R}$ et $\pi(s) \in A(s)$ arbitrairement pour tout $s \in S$.
 - 2: **Evaluer** politique π
 - 3: $\text{stable} \leftarrow \text{true}$
 - 4: **for all** $s \in S$ **do**
 - 5: $a \leftarrow \pi(s)$
 - 6: $\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$
 - 7: Si $a \neq \pi(s)$ alors $\text{stable} \leftarrow \text{false}$
 - 8: **end for**
 - 9: Si stable alors retourne V et π . Sinon Revenir à l'étape 2.
-

- Si la politique est stable alors ça vaut dire qu'elle est optimale.
- À chaque itération on est sûr d'avoir une politique meilleure ou de même qualité.

Programmation dynamique: recherche d'une politique optimale π^*

- L'algorithme précédent peut être lent et faire des itérations inutiles.
- L'algorithme suivant produit le même résultat plus efficacement

Algorithm Itération sur les Valeurs (Value Iteration)

- 1: Initialiser $V(s)$ arbitrairement (par exemple, $V(s) = 0$ pour tout $s \in S$).
- 2: **repeat**
- 3: $\Delta \leftarrow 0$
- 4: **for all** $s \in S$ **do**
- 5: $v \leftarrow V(s)$
- 6: $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$
- 7: $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
- 8: **end for**
- 9: **until** $\Delta < \theta$ (un petit nombre positif)
- 10: **return** une politique π telle que pour tout $s \in S$:
- 11: $\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$

Méthode de Monte Carlo

- On apprend par l'expérience. SANS considérer une connaissance complète de l'environnement. (Ex. les probabilités $p(s', r|s, a)$ ne sont pas connues)
- Le principe consiste à échantillonner et faire les moyennes des expériences vécues. Ex.: Jouer à un jeu plusieurs fois (échantillonnage) et établir une politique de jeu qui va bien en moyenne.
- Une expérience est appelé un épisode
- Pour estimer la valeur $v_{\pi}(s)$ d'un état s on fait donc la somme des récompenses obtenues divisé par le nombre d'épisodes dans lesquelles s a été visité.
- Au fur et à mesure des épisodes la valeur estimé par la moyenne converge vers la vraie valeur des $v_{\pi}(s)$, $s \in \mathcal{S}$ selon la loi des grand nombres.

Méthode de Monte Carlo

- Si on connaît un modèle de l'environnement à partir d'un état je sais quel sera l'état après une action prise. Donc il suffit d'estimer les valeurs des états.
- Par contre, s'il n'existe pas un tel modèle, je ne sais pas quel état j'obtiendrai après une action choisie. Je suis obligé donc d'estimer $q_{\pi}(s, a)$: la valeur des états-actions.
- on procède comme avec l'estimation des valeurs d'état.
- Remarquez qu'on peut avoir un grand nombre d'état-actions (s, a) . Avec une politique déterministes il y aura potentiellement beaucoup de tuples (s, a) jamais visités.
- On a le trade-off exploration X exploitation.

Méthode de Monte Carlo: Algorithme On-policy

Algorithm Estimation et contrôle Monte Carlo

```
1: Initialiser, pour tout  $s \in S$ ,  $a \in A(s)$  :  $Q(s, a) \leftarrow$  valeur arbitraire,  
    $Récompenses(s, a) \leftarrow$  liste vide,  $\pi(a | s) \leftarrow$  une politique  $\varepsilon$ -douce arbitraire  
2: repeat  
3:   Générer un épisode en utilisant  $\pi$   
4:   for all  $(s, a)$  apparaissant dans l'épisode do :  
5:      $G \leftarrow$  récompense à partir de la première occurrence de  $(s, a)$   
6:     Ajouter  $G$  à  $Récompenses(s, a)$ .  
7:      $Q(s, a) \leftarrow$  moyenne( $Récompenses(s, a)$ )  
8:   end for  
9:   for all  $s$  dans l'épisode do:  
10:     $a^* \leftarrow \arg \max_a Q(s, a)$ .  
11:    Pour tout  $a \in A(s)$  :
```

$$\pi(a | s) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s)|}, & \text{si } a = a^* \\ \frac{\varepsilon}{|A(s)|}, & \text{si } a \neq a^* \end{cases}$$

```
12:   end for  
13: until Toujours
```

Méthode de Monte Carlo: Algorithme Off-policy

- L'algorithme précédent est On-policy, on utilise la propre politique pour estimer les valeurs selon cette politique
- Une alternative puissante est d'apprendre sur une politique π en utilisant une autre politique μ : **Algorithme Off-policy**
- La condition c'est que $\pi(a|s) > 0$ implique $\mu(a|s) > 0$ et π doit être déterministe.
- Typiquement:
 - ▶ la politique cible: π est la politique gloutonne par rapport aux estimations des valeurs des états-actions.
 - ▶ la politique d'usage: μ reste exploratoire et stochastique, comme les politiques ϵ -gloutonnes.

Méthode de Monte Carlo: échantillonnage préférentiel (Importance sampling)

- L'importance sampling dit que

$$\mathbb{E}_p[f(x)] = \int p(x)f(x)dx = \int q(x) \left[\frac{p(x)}{q(x)} f(x) \right] dx = \mathbb{E}_q \left[\frac{p(x)}{q(x)} f(x) \right]$$

- Rappelez vous que $v_\pi(s)$ est égal à l'espérance de la récompense, étant donnée que s a été visité.
- En ce sens, la valeur d'un état s peut être donnée par

$$v_\pi(s) = \mathbb{E}_\pi[G_t] = \mathbb{E}_\mu \left[\frac{\pi}{\mu} G_t \right]$$

considérant t un épisode parmi l'ensemble $\tau(s)$ des épisodes dont la trajectoire inclut s

Méthode de Monte Carlo: échantillonnage préférentiel (Importance sampling)

- Nous allons donc estimer $\mathbb{E}_\mu \left[\frac{\pi}{\mu} G_t \right]$ par Monte Carlo (moyennes des récompenses obtenues à la fin des épisodes):

$$V(s) = \frac{\sum_{t \in \tau(s)} \rho_t G_t}{|\tau(s)|} \text{ ou la moyenne pondérée: } V(s) = \frac{\sum_{t \in \tau(s)} \rho_t G_t}{\sum_{t \in \tau(s)} \rho_t}$$

considérant $T(t)$ l'état terminal de l'épisode $t \in \tau(s)$ et

$$\rho_t = \frac{\prod_{k=1}^{T(t)-1} \pi(a_k|s_k) p(s_{k+1}|s_k, a_k)}{\prod_{k=1}^{T(t)-1} \mu(a_k|s_k) p(s_{k+1}|s_k, a_k)} = \frac{\prod_{k=1}^{T(t)-1} \pi(a_k|s_k)}{\prod_{k=1}^{T(t)-1} \mu(a_k|s_k)}$$

- Mise à jour de la moyenne: comme pour l'algorithme du bandit

$$V_{n+1} = V_n + \frac{\rho_n}{C_n} [G_n - V_n]$$

où $C_n = C_{n-1} + \rho_n$

Méthode de Monte Carlo: (Importance sampling)

Algorithm Monte Carlo: Importance Sampling

- 1: **Initialisation:** Pour tout $s \in S$, $a \in A(s)$: $Q(s, a) \leftarrow$ une valeur arbitraire; $C(s, a) \leftarrow 0$; $\mu(a|s) \leftarrow$ une politique d'usage; $\pi(a|s) \leftarrow$ une politique cible
- 2: **repeat**
- 3: Générer un épisode avec μ : $\{S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T\}$
- 4: $G \leftarrow 0$, $W \leftarrow 1$
- 5: **for** $t = T - 1, T - 2, \dots, 0$ **do**
- 6: $G \leftarrow \gamma G + R_{t+1}$
- 7: $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
- 8: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
- 9: $\pi(S_t) \leftarrow \arg \max_a Q(S_t, a)$
- 10: If $A_t \neq \pi(S_t)$ then **break**
- 11: $W \leftarrow W \cdot \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$
- 12: **end for**
- 13: **until** Interruption

Méthode de Monte Carlo: (Importance sampling)

- Remarquez que le dernier l'algorithme présenté considère toutes les fois où s a été visité dans un épisode.
- Des petites modifications sont nécessaires si on ne s'intéresse qu'à la première fois où un état (ou état-action) est visité.
- Remarquez aussi que les méthodes Monte Carlo doivent attendre la fin d'un épisode pour apprendre.
- Nous allons voir maintenant un dernier type de méthode:
L'apprentissage par différence temporelle (Temporal-difference learning) qui n'attend pas la fin de l'épisode.

L'apprentissage par différence temporelle (TD Learning)

- Rappelez-vous que:

$$v_{\pi}(s) = \mathbb{E}_{\pi} [G_t \mid S_t = s] \quad (1)$$

$$\begin{aligned} &= \mathbb{E}_{\pi} \left[\sum_{k=0}^{T(t)} \gamma^k R_{k+t+1} \mid S_t = s \right] \\ &= \mathbb{E}_{\pi} [R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_{\pi} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s]. \end{aligned} \quad (2)$$

- Les méthodes Monte Carlo essayent d'estimer (1), alors que les méthodes par différence temporelle essayent d'estimer (2).
- Remarquez que la programmation dynamique travaille aussi sur (2). La différence maintenant c'est que $v_{\pi}(S_{t+1})$ n'est peut-être pas connu. On travaille donc avec une estimation $V(S_{t+1})$.
- Le fait de faire des estimations sur d'autres estimations s'appelle **Bootstrap** ou **Bootstrapping**.

SARSA: On-policy TD Learning

- Le nom SARSA vient du fait de dépendre du quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$
- Comme dans toutes les méthodes sur politique (*on-policy*), nous estimons continuellement q_π pour la politique π .
- En parallèle, nous modifions π pour qu'elle devienne gloutonne (*greedy*) par rapport à q_π .
- La valeur de récompense G d'un épisode est estimée sur un certain moment intermédiaire t par $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

SARSA: On-policy TD Learning

Algorithm Apprentissage Q par différence temporelle (SARSA)

- 1: **Initialiser** : $Q(s, a)$ arbitrairement pour tout $s \in S, a \in A(s)$
 - 2: $Q(\text{état-terminal}, \cdot) \leftarrow 0$
 - 3: **repeat** Pour chaque épisode
 - 4: Initialiser S
 - 5: Choisir A avec une politique sur Q (par exemple, ε -greedy)
 - 6: **repeat** Pour chaque étape de l'épisode
 - 7: Effectuer l'action A , observer R et S'
 - 8: Choisir A' avec une politique sur Q (par exemple, ε -greedy)
 - 9: $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$
 - 10: $S \leftarrow S'$
 - 11: $A \leftarrow A'$
 - 12: **until** S est un état terminal
 - 13: **until** condition d'arrêt
-

- Faire en sorte que ε diminue à chaque étape, par exemple $\varepsilon = 1/t$

Q-Learning: Off-policy TD Learning

- Dans Q-learning on utilise une politique quelconque pour apprendre directement sur q_* les valeurs optimales.

Algorithm Apprentissage Q par différence temporelle

- 1: **Initialiser** : $Q(s, a)$ arbitrairement pour tout $s \in S, a \in A(s)$
 - 2: $Q(\text{état-terminal}, \cdot) \leftarrow 0$
 - 3: **repeat** Pour chaque épisode
 - 4: Initialiser S
 - 5: **repeat** Pour chaque étape de l'épisode
 - 6: Choisir A avec une politique sur Q (par exemple, ε -greedy)
 - 7: Effectuer l'action A , observer R et S'
 - 8: $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
 - 9: $S \leftarrow S'$
 - 10: **until** S est un état terminal
 - 11: **until** condition d'arrêt
-