Modélisation Probabiliste et Apprentissage par Renforcement

Benoît Delahaye

Nantes Université, LS2N UMR 6004

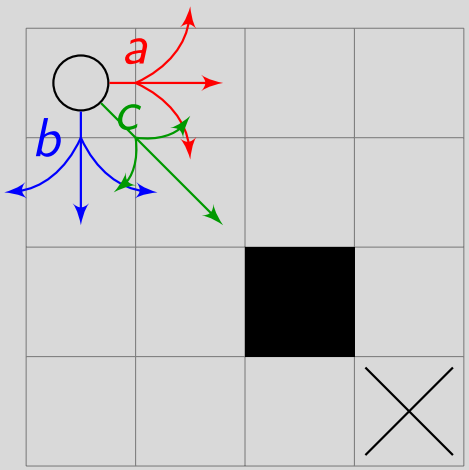
Chapitre 3 : Apprentissage par Renforcement

`https://pagesperso.ls2n.fr/~delahaye-b/MPAR/chapitre3.pdf`

Problème

Etant donné un problème décisionnel, comment trouver la politique optimale ?

Exemple 1 : Récompense terminale



Dans le cas de la bille sur le plateau, comment savoir quel coup jouer dans chaque case pour optimiser les chances de gain ?

Exemple 2 : Récompense continue

Dans le cas d'une machine à sous (Bandit manchot), comment trouver la bonne stratégie **tout en maximisant son gain** ?

Outline

Outline

MDP et récompenses

Un Processus de Décision Markovien (MDP) est un tuple $\mathcal{M} = (S, \text{Act}, P, \iota_{\text{init}}, r)$, où

- S, ι_{init}, sont les états et la distribution initiale,
- Act est un ensemble d'actions,
- $P : S \times \text{Act} \times S \to [0, 1]$ est la fonction de transition probabiliste, telle que pour tout état s et pour toute action a,

$$\sum_{s' \in S} P(s, a, s') \in \{0, 1\},$$

- $r : S \to \mathbb{R}^+$ est une fonction de récompense sur les états ou $r : S \times A \to \mathbb{R}^+$ sur les transitions.

Adversaires / Politiques et Gain

On ne considère ici que les adversaires positionnels / politiques stationnaires.

Definition (Adversaire positionnel / politique stationnaire)

Un *adversaire* positionnel de $\mathcal{M}$ est une fonction $\mathfrak{S} : S \to \text{Act}$ telle que, pour tout $s \in S$, on a $\mathfrak{S}(s) \in \text{Act}(s)$.

Definition (Gain pondéré)

Etant donnés un MDP $\mathcal{M} = (S, \text{Act}, P, \iota_{\text{init}}, r)$, une constante $\gamma \in (0, 1]$ et une trajectoire $\pi = s_0 \dots s_n s_{n+1}$, le gain pondéré associé à π est $G(\pi) = r(s_0) + \gamma r(s_1) + \dots + \gamma^n r(s_n)$.

Fonction de Valeurs

On définit une **fonction de valeur** pour chaque état. Elle dépend de l'adversaire choisi.

Definition (Fonction de Valeurs)

La fonction de valeur pour l'état s, le facteur de pondération γ et l'adversaire $\mathfrak{S}$ est définie comme suit :

$$V_\gamma^{\mathfrak{S}}(s) = \mathbb{E}_{\mathcal{M}_{\mathfrak{S}}}(G(\pi = s \ldots)).$$

C'est l'espérance du gain pondéré obtenu depuis s en suivant la politique $\mathfrak{S}$.

On peut remarquer que

$$V_\gamma^{\mathfrak{S}}(s) = r(s) + \gamma \sum_{s' \in S} \left(P(s, \mathfrak{S}(s), s') V_\gamma^{\mathfrak{S}}(s') \right).$$

Outline

Equation de Bellman

On sait qu'il existe un adversaire optimal (sous conditions) par rapport au gain. Notons cet adversaire $\mathfrak{S}^*$, et notons V^* la fonction de valeurs associée. On obtient

Definition (Equation de Bellman)

La fonction de valeur optimale V^* est l'unique solution de l'équation de Bellman :

$$\forall s \in S, \quad V(s) = \max_{a \in Act} \left(r(s) + \gamma \sum_{s' \in S} \left(P(s, a, s') V(s') \right) \right).$$

Problème : Comment calculer cette solution ?

Rappel : Programmation Dynamique

Lorsqu'on connaît le modèle en détails, on peut effectivement calculer V^* et trouver $\mathfrak{S}^*$. Pour calculer V^*, on utilise la programmation dynamique (résolution du programme linéaire) :

$$\forall s \in S_?, a \in Act(s), V^*(s) \geq r(s) + \gamma \sum_{t \in S} P(s,a,t) \cdot V^*(t),$$

$$\text{avec } \sum_{s \in S} V^*(s) \text{ minimal.}$$

Algorithme d'itération de valeurs

Initialize(V_0);
$n \leftarrow 0$;
repeat
  for $s \in S$ **do**
    $$V_{n+1}(s) = \max_{a \in A} \left\{ r(s) + \gamma \sum_{s' \in S} P(s, a, s') V_n(s') \right\}$$
  end
  $n \leftarrow n + 1$;
until $\|V_{n+1} - V_n\| < \varepsilon$;
for $s \in S$ **do**
  $$\mathfrak{S}(s) = \operatorname*{argmax}_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V_n(s') \right\}$$
end
return $V_n, \mathfrak{S}$

Algorithme d'itération de politique

Initialize($\mathfrak{S}_0$);

$n \leftarrow 0$;

repeat

  Solve

$$V_n(s) = r(s) + \gamma \sum_{s' \in S} P(s, \mathfrak{S}(s), s') V_n(s'), \quad \forall s \in S$$

  for $s \in S$ **do**

$$\mathfrak{S}_{n+1}(s) \in \operatorname*{argmax}_{a \in A} \left\{ r(s) + \gamma \sum_{s' \in S} P(s, a, s') V_n(s') \right\}$$

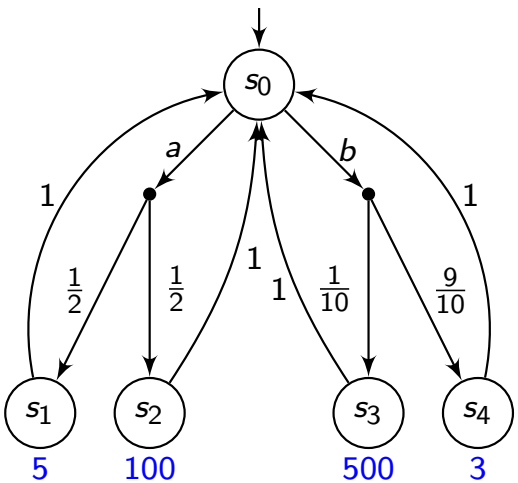
  end

  $n \leftarrow n + 1$;

until $\mathfrak{S}_{n+1} = \mathfrak{S}_n$;

return $V_n, \mathfrak{S}_n$

Exercice



Exercice 1

1. Appliquer l'algorithme d'itération de valeurs sur l'exemple ci-contre, pour $\gamma = 1$ et $\gamma = 1/2$, et $\varepsilon = 1$.
2. L'algorithme converge-t-il ? En combien d'itérations ?
3. Proposer un modèle qui convergerait pour $\gamma = 1$. Quelle condition permettrait de le garantir ?
4. Même questions pour le modèle de la bille sur le plateau.
5. Implémenter l'algorithme et le tester sur un modèle de l'exemple ci-contre, puis sur le modèle de la bille sur le plateau.

Inconvénients

Dans tous les algorithmes présentés précédemment, on doit connaître le MDP.

Que faire lorsqu'on ne le connaît pas ? ?

Outline

Pourquoi l'apprentissage par renforcement ?

Permet de calculer la politique optimale *à la volée* lorsqu'on ne connaît pas précisément le modèle ou lorsque le modèle est trop complexe pour une analyse symbolique.
Beaucoup d'autres méthodes

- Optimisation
- Algorithmes génétiques
- Recuit simulé

Apprentissage par Renforcement

Prise en compte de la structure **temporelle**.
$\Rightarrow$ L'ordre des expériences est important.

Dilemne Exploration / Exploitation

Vaut-il mieux choisir une bonne action (état) que l'on connaît bien ou explorer une nouvelle action (état) ?
Plusieurs méthodes de choix possible, non-dirigées ou dirigées

- *Constante* : Choisir N fois la meilleure actions, puis M fois une action aléatoire.
- ε-*greedy* : Choisir la meilleure action avec probability $1 - \varepsilon$, et une action aléatoire avec probabilité ε.
- *Softmax* : Tirer l'action selon une distribution de Boltzman (dépendant de la valeur des actions).
- *Recency-based* : bonus d'exploration fonction du nombre d'itérations depuis le dernier a.
- *Uncertainty estimation* : bonus d'exploration selon le nombre de fois que a a été choisie.
- ...

Algorithmes de calcul de V

On utilise des méthodes *incrémentales*. 3 classes d'algorithmes :

- Programmation dynamique : si on connaît parfaitement le modèle a priori.
- Monte Carlo : aucune connaissance a priori nécessaire, mais pas vraiment incrémentale.
- Différence temporelle : utilisent une connaissance partielle du modèle, propagent les fonctions de valeur. $\Rightarrow$ Le véritable apprentissage par renforcement.

Outline

Estimation de la valeur d'un état (adversaire fixé)

On suppose que tout chemin se termine en un état absorbant de récompense nulle.

Estimation de $V^{\mathfrak{S}}(s)$

On effectue k simulations π_i, $1 \leq i \leq k$, à partir de s, et on mesure le gain $G_i(s) = r(\pi_i)$. On note

$$V_k^{\mathfrak{S}}(s) = \frac{G_1(s) + \ldots + G_k(s)}{k}$$

$\Rightarrow$ Pour tout s, $\lim_{k\to\infty} V_k^{\mathfrak{S}}(s) = V^{\mathfrak{S}}(s)$.

Remarque :

$$V_{k+1}^{\mathfrak{S}}(s) = V_k^{\mathfrak{S}}(s) + \frac{1}{k+1}(G_{k+1}(s) - V_k^{\mathfrak{S}}(s))$$

Remarque 2 : on peut remplacer $\frac{1}{K+1}$ par n'importe quelle fonction $\alpha \to 0$.

Monte Carlo : Estimation des V (adversaire fixé)

- Effectuer k simulations π_i de $\mathcal{M}_{\mathfrak{S}}$.
  Pour chaque simulation $\pi_i = s_0 \dots s_N$,
  1. On mesure les récompenses obtenues dans chaque état $r_j = r(s_j)$
  2. On met à jour les fonctions de valeur :

$$V^{\mathfrak{S}}(s_j) \leftarrow V^{\mathfrak{S}}(s_j) + \alpha(r_j + \gamma r_{j+1} + \ldots + \gamma^{N-1-j} r_{N-1} - V^{\mathfrak{S}}(s_j))$$

- Attention : biais si plusieurs mises à jour d'un même état sur une trajectoire (*every-visit*) $\Rightarrow$ On ne met à jour que la première fois (*first-visit*).

Outline

Différence Temporelle

Pour un adversaire $\mathfrak{S}$ et une simulation $\pi = s_0 \dots s_N$, on note $r_j = r(s_j)$. En théorie, si les estimations $V^{\mathfrak{S}}$ étaient exactes, on aurait

$$V^{\mathfrak{S}}(s_j) = r_j + \gamma r_{j+1} + \ldots + \gamma^{N-1-j} r_{N-1}$$
$$V^{\mathfrak{S}}(s_{j+1}) = r_{j+1} + \gamma r_{j+2} + \ldots + \gamma^{N-2-j} r_{N-1}$$

On a donc

$$V^{\mathfrak{S}}(s_j) = r_j + \gamma V^{\mathfrak{S}}(s_{j+1})$$

L'erreur de *différence temporelle* pour l'état s_j est notée

$$\delta_j = r_j + \gamma V^{\mathfrak{S}}(s_{j+1}) - V^{\mathfrak{S}}(s_j)$$

Elle mesure l'erreur obtenue lors de l'estimation (par rapport à la théorie où $\delta_j = 0$).

Algorithme TD(0)

Permet d'estimer la fonction de valeur de tous les états *à politique* $\mathfrak{S}$ *fixée*.

Mise à jour de TD(0)

À la $k+1^{ieme}$ itération, on met à jour

$$V_{k+1}^{\mathfrak{S}}(s_j) \leftarrow V_k^{\mathfrak{S}}(s_j) + \alpha[r_j + \gamma V_k^{\mathfrak{S}}(s_{j+1}) - V_k^{\mathfrak{S}}(s_j)] = V_k^{\mathfrak{S}}(s_j) + \alpha\delta_j$$

Avantage : Peut être mis à jour à la volée, plusieurs fois le long d'une même exécution

Inconvénient : Ne permet pas (directement) de calculer la politique optimale

Outline

Fonctions d'action-valeur

On définit des fonctions d'action-valeur pour les paires état-action.

Q-functions

Etant donnée une politique $\mathfrak{S}$, un état s et une action a, on définit

$$Q^{\mathfrak{S}}(s,a) = \mathbf{r}(\mathbf{s},\mathbf{a}) + \gamma \sum_{s'} P(s,a,s') V^{\mathfrak{S}}(s')$$

Remarque : On a $V^{\mathfrak{S}}(s) = Q^{\mathfrak{S}}(s, \mathfrak{S}(s))$.

L'équation de Bellman pour la politique optimale devient alors

$$\forall s,a, \quad Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s,a,s') \max_b Q^*(s',b)$$

State-Action-Reward-State-Action (SARSA)

En utilisant l'équation de Bellman pour Q^*, on obtient :

$$\forall s, \quad V^*(s) = \max_a Q^*(s,a)$$
$$\mathfrak{S}^*(s) = \operatorname{argmax}_a Q^*(s,a)$$

Algorithme SARSA

À la $k+1^{ieme}$ itération, avec l'adversaire courant $\mathfrak{S}$ dans l'état s_j,

$$Q_{k+1}(s_j, \mathfrak{S}(s_j)) \leftarrow Q_k(s_j, \mathfrak{S}(s_j)) + \alpha[r_j + \gamma Q_k(s_{j+1}, \mathfrak{S}(s_{j+1})) - Q_k(s_j, \mathfrak{S}(s_j))]$$

Inconvénient : On doit connaître le choix effectué dans l'état suivant pour mettre à jour.

Outline

Principes du Q-Learning

Pour simplifier SARSA, et ne pas avoir besoin de connaître l'adversaire en cours, on utilise la mise à jour suivante :

Q-Learning

À la $k+1^{ieme}$ itération, si on choisit a dans l'état s_j,

$$Q_{k+1}(s_j, a) \leftarrow Q_k(s_j, a) + \alpha[r_j + \gamma \max_b Q_k(s_{j+1}, b) - Q_k(s_j, a)]$$

$\Rightarrow$ Plus simple que SARSA, convergence prouvée plus tôt. C'est l'algorithme le plus utilisé en Apprentissage par Renforcement

Inconvénient : Souvent moins efficace que SARSA en pratique

Algorithme Q-Learning

Initialize(Q_0) ;
for $t \leftarrow 0$ **to** $T_{tot} - 1$ **do**
    $s_t \leftarrow$ ChooseState ;
    $a_t \leftarrow$ ChooseAction ;
    $(s_{t+1}, r_t) \leftarrow$ Simulate(s_t, a_t) ;
    // Update de la fonction Q_t
    begin
        $Q_{t+1} \leftarrow Q_t$;
        $\delta_t \leftarrow r_t + \gamma \max_b Q_t(s_{t+1}, b) - Q_t(s_t, a_t)$;
        $Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha(s_t, a_t)\delta_t$;
    end
end
return $Q_{T_{tot}}$

Exercice

Exercice 2

1. Proposer un modèle Python permettant de simuler la machine de l'exercice précédent.
2. Implémenter l'algorithme de Q-learning dans ce cas particulier et l'utiliser pour déterminer la meilleure stratégie ($\gamma = 1/2$, α bien choisi).
3. Implémenter l'algorithme de Q-learning dans votre outil de model-checking.
4. Appliquer cet algorithme sur le modèle (MDP) de la machine à deux bras.
5. Tester aussi sur le modèle de la bille sur le plateau.

Exercice

Exercice 3

À préparer à la maison pour la séance du 17/03/23 (Questions 1-3).

1. Télécharger l'article
   `https://pagesperso.ls2n.fr/~delahaye-b/MPAR/SMC-MDP-2012.pdf`
2. Expliquer le lien avec l'apprentissage par renforcement.
3. Quelles sont les différences avec le Q-learning ?
4. Implémenter dans votre outil un algorithme permettant de faire du SMC avec des MDP (pas forcément celui de l'article).
5. Le tester sur un modèle bien choisi.

D'autres articles sont disponibles si cela vous intéresse :

- `https://pagesperso.ls2n.fr/~delahaye-b/MPAR/Planning-MDP-2012.pdf`
- `https://pagesperso.ls2n.fr/~delahaye-b/MPAR/MDP-RL-2015.pdf`
- `https://pagesperso.ls2n.fr/~delahaye-b/MPAR/PAC-SMC-2019.pdf`
- `https://pagesperso.ls2n.fr/~delahaye-b/MPAR/SMC-MDP-RL-2020.pdf`