Programmation Dynamique et Apprentissage par Renforcement

Rhouma Haythem

Avril 2025

Contents

1 Introduction à la Programmation Dynamique (DP)

La **programmation dynamique** est une technique algorithmique permettant de résoudre des problèmes complexes en les décomposant en sous-problèmes plus simples. Elle repose sur l'idée de *mémoriser* les solutions intermédiaires afin d'éviter des recalculs inutiles (principe de mémoïsation).

Approches principales

- **Itération de la politique (Policy Iteration)** : consiste à évaluer une politique actuelle, puis à l'améliorer de manière itérative jusqu'à convergence vers une politique optimale.
- **Itération de la valeur (Value Iteration)** : consiste à mettre à jour directement la valeur de chaque état, en calculant à chaque étape les meilleures décisions possibles.

2 Estimation de l'Espérance

L'espérance mathématique est la moyenne pondérée des valeurs possibles d'une variable aléatoire, pondérées par leurs probabilités respectives.

$$\mathbb{E}(X) = \sum_i x_i \cdot \mathbb{P}(X = x_i) \tag{1}$$

Exemple

Soit une variable aléatoire prenant les valeurs $\{1, 2, 3\}$ avec des probabilités respectives $\{0.2, 0.5, 0.3\}$. L'espérance est alors :

$$\mathbb{E}(X) = 1 \cdot 0.2 + 2 \cdot 0.5 + 3 \cdot 0.3 = 2.1 \tag{2}$$

L'espérance est un outil fondamental pour anticiper les résultats moyens dans le cadre de décisions séquentielles, comme dans l'apprentissage par renforcement (RL).

article amsmath amssymb graphicx hyperref

Introduction à la Méthode Monte Carlo en Apprentissage par Renforcement Rhouma Haythem Avril 2025

3 Objectif de la méthode Monte Carlo

La méthode **Monte Carlo** est une technique d'estimation utilisée en apprentissage par renforcement pour approximer les valeurs des actions et améliorer les politiques, en s'appuyant sur les résultats observés dans un environnement.

Caractéristiques principales

- **Utilisation d'épisodes complets** : La méthode nécessite d'attendre la fin de chaque épisode pour effectuer les mises à jour.
- **Méthode sans modèle** : Elle ne dépend pas d'une connaissance explicite des probabilités de transition entre états. Elle se base uniquement sur l'expérience.

4 Principe de base

Le principe fondamental de Monte Carlo repose sur l'utilisation de **moyennes d'échantillons** pour estimer la valeur d'un état ou d'une action. On observe des séquences complètes d'interactions (épisodes) et on utilise les retours totaux obtenus pour chaque occurrence d'un état ou d'une action.

5 Formule d'estimation de la valeur d'un état

Soit $V(s)$ la valeur estimée de l'état s, et G_t le retour observé à partir du moment où l'état s est visité. On estime $V(s)$ par :

$$V(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_t^{(i)} \tag{3}$$

où :

- $N(s)$ est le nombre de fois que l'état s a été visité
- $G_t^{(i)}$ est le retour total obtenu dans le i-ième épisode où s est apparu

Exemple de calcul

Supposons que l'état s ait été visité trois fois au cours de trois épisodes, avec les retours suivants : 10, 20, et 15. Alors :

$$V(s) = \frac{10 + 20 + 15}{3} = 15$$

6 Types de méthodes Monte Carlo

Il existe deux variantes principales :

1. **Méthode de la première visite (First-Visit)** : on ne prend en compte que le **premier passage** dans l'état s à chaque épisode.
2. **Méthode de chaque visite (Every-Visit)** : on considère **toutes les occurrences** de l'état s dans chaque épisode.

7 Méthodes Monte Carlo : First-Visit vs Every-Visit

1. Méthode de la première visite (First-Visit)

- Cette méthode ne prend en compte que la **première fois** où l'état s est visité dans un épisode donné.
- Toutes les visites suivantes de s dans le même épisode sont ignorées.

Exemple : Si l'état s est rencontré deux fois pendant un épisode, seul le retour obtenu à partir de la première visite sera utilisé pour mettre à jour $V(s)$.

2. Méthode de toutes les visites (Every-Visit)

- Dans cette méthode, **chaque visite** de l'état s est prise en compte.
- Si s est visité plusieurs fois dans le même épisode, toutes ces visites contribuent à la moyenne estimée de $V(s)$.

Exemple : Si l'état s est visité deux fois dans un épisode, les deux retours associés seront pris en compte pour la mise à jour de $V(s)$.

8 Mise à jour avec la formulation $(1 - \alpha)$

Il est également possible de formuler la mise à jour Monte Carlo en utilisant une pondération explicite entre l'ancienne estimation et le retour observé, via $(1 - \alpha)$.

Équation de mise à jour pondérée

$$V(s) \leftarrow (1 - \alpha)V(s) + \alpha G_t \tag{4}$$

où :

- $(1 - \alpha)V(s)$: fraction conservée de la valeur actuelle
- αG_t : contribution pondérée du nouveau retour observé

Intuition pédagogique

- $(1-\alpha)$ permet de ne pas "effacer" complètement la mémoire du passé
- α contrôle à quel point on fait confiance à la nouvelle expérience
- Une valeur faible de α implique un apprentissage lent mais plus stable
- Une valeur élevée accélère l'adaptation mais peut causer de l'instabilité

Cette formulation est cohérente avec les mises à jour en TD-Learning et en Q-Learning, ce qui unifie le cadre d'apprentissage par renforcement sous une approche commune d'estimation incrémentale.

9 Méthodes Monte Carlo : First-Visit vs Every-Visit

1. Méthode de la première visite (First-Visit)

- Cette méthode ne prend en compte que la **première fois** où l'état s est visité dans un épisode donné.
- Toutes les visites suivantes de s dans le même épisode sont ignorées.

Exemple : Si l'état s est rencontré deux fois pendant un épisode, seul le retour obtenu à partir de la première visite sera utilisé pour mettre à jour $V(s)$.

2. Méthode de toutes les visites (Every-Visit)

- Dans cette méthode, **chaque visite** de l'état s est prise en compte.
- Si s est visité plusieurs fois dans le même épisode, toutes ces visites contribuent à la moyenne estimée de $V(s)$.

Exemple : Si l'état s est visité deux fois dans un épisode, les deux retours associés seront pris en compte pour la mise à jour de $V(s)$.

10 Mise à jour avec la formulation $(1-\alpha)$

Il est également possible de formuler la mise à jour Monte Carlo en utilisant une pondération explicite entre l'ancienne estimation et le retour observé, via $(1-\alpha)$.

Équation de mise à jour pondérée

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha G_t \tag{5}$$

où :

- $(1-\alpha)V(s)$: fraction conservée de la valeur actuelle
- αG_t : contribution pondérée du nouveau retour observé

Intuition pédagogique

- $(1-\alpha)$ permet de ne pas "effacer" complètement la mémoire du passé
- α contrôle à quel point on fait confiance à la nouvelle expérience
- Une valeur faible de α implique un apprentissage lent mais plus stable
- Une valeur élevée accélère l'adaptation mais peut causer de l'instabilité

Cette formulation est cohérente avec les mises à jour en TD-Learning et en Q-Learning, ce qui unifie le cadre d'apprentissage par renforcement sous une approche commune d'estimation incrémentale.

11 Introduction à l'Apprentissage par Différence Temporelle (TD)

L'apprentissage par différence temporelle, ou TD-Learning, est une technique centrale en apprentissage par renforcement. Elle permet à un agent d'estimer la valeur d'un état tout en progressant dans l'environnement, sans avoir besoin d'attendre la fin d'un épisode (contrairement à la méthode Monte Carlo).

Objectif

Estimer la récompense cumulée attendue lorsqu'un agent se trouve dans un certain état et suit une politique donnée.

Principe

Mettre à jour la valeur d'un état à partir de la récompense obtenue immédiatement, ainsi que de la valeur estimée de l'état suivant, en utilisant une approche incrémentale.

12 Mise à jour de la valeur d'état (TD)

La mise à jour de la valeur de l'état S_t suit la règle suivante :

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right) \tag{6}$$

où :

- α : taux d'apprentissage (contrôle la vitesse de l'ajustement)
- R_{t+1} : récompense reçue après avoir quitté l'état S_t
- $V(S_{t+1})$: estimation actuelle de la valeur du prochain état
- γ : facteur d'actualisation (discount factor)

Erreur temporelle (TD error)

L'écart entre la prédiction actuelle $V(S_t)$ et la cible $R_{t+1} + \gamma V(S_{t+1})$ est appelé *erreur temporelle* :

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

La mise à jour peut donc s'écrire comme :

$$V(S_t) \leftarrow V(S_t) + \alpha \cdot \delta_t$$

13 Formulation avec $(1 - \alpha)$

Pour renforcer l'intuition, on peut reformuler la mise à jour comme une combinaison pondérée entre l'ancienne valeur et la nouvelle estimation :

$$V(S_t) \leftarrow (1 - \alpha)V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1})\right) \tag{7}$$

Cette version permet de mieux visualiser la conservation partielle de l'ancienne valeur, et le poids donné à la nouvelle estimation. Elle met en évidence le rôle de α comme un compromis entre mémoire et adaptation.

Quiz à choix multiples : TD, SARSA, Q-Learning

(a) **Qu'est-ce que l'apprentissage par différence temporelle (TD) ?**

A. Une méthode basée sur des épisodes complets

B. Une approche combinant Monte Carlo et programmation dynamique

C. Une méthode nécessitant un modèle de l'environnement

D. Une méthode de supervision directe

(b) **Lequel des éléments suivants est un avantage des méthodes TD ?**

A. Elles nécessitent un modèle de l'environnement

B. Elles attendent la fin de l'épisode pour mettre à jour les valeurs

C. Elles peuvent fonctionner avec des épisodes incomplets

D. Elles ont besoin de calculs complexes

(c) **Quelle est la règle de mise à jour TD(0) pour estimer la valeur d'un état ?**

A. $V(s) \leftarrow R_{t+1}$

B. $V(s) \leftarrow \alpha[R_{t+1} - V(s)]$

C. $V(s) \leftarrow V(s) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(s)]$

D. $V(s) \leftarrow \gamma V(s)$

(d) **Dans les méthodes TD, que représente γ ?**

A. Le taux d'apprentissage

B. La probabilité d'exploration

C. Le facteur d'actualisation

D. Le nombre d'actions possibles

(e) **Pourquoi SARSA est-il considéré comme une méthode sur politique ?**

A. Il utilise une politique différente pour le comportement et la mise à jour

B. Il suit la même politique pour la prise d'action et la mise à jour

C. Il nécessite un modèle de l'environnement

D. Il ne dépend pas d'une politique d'apprentissage

(f) **Quel est le rôle du paramètre α dans les méthodes TD ?**

A. Fixer la probabilité d'explorer

B. Ajuster la vitesse de mise à jour des estimations

C. Sélectionner la prochaine action

D. Déterminer la politique de récompense

(g) **Dans une politique ε-greedy, que se passe-t-il si ε est proche de 1 ?**

A. L'agent choisit toujours l'action optimale

B. L'agent explore souvent de nouvelles actions

C. L'agent reste bloqué dans une boucle

D. L'agent refuse d'explorer

Corrigé détaillé

(a) **Réponse correcte : B**
L'apprentissage par différence temporelle (TD) combine les idées de Monte Carlo (apprentissage par l'expérience) et de la programmation dynamique (mises à jour basées sur des estimations).

(b) **Réponse correcte : C**
Les méthodes TD peuvent fonctionner avec des épisodes incomplets, ce qui les rend plus flexibles que les méthodes Monte Carlo, qui nécessitent la fin de l'épisode.

(c) **Réponse correcte : C**
La règle de mise à jour TD(0) est :

$$V(s) \leftarrow V(s) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(s)]$$

(d) **Réponse correcte : C**
γ est le facteur d'actualisation. Il permet de pondérer les récompenses futures. Plus γ est proche de 1, plus l'agent valorise les récompenses à long terme.

(e) **Réponse correcte : B**
SARSA est *on-policy* car l'agent met à jour ses estimations en suivant exactement la politique qu'il utilise pour agir.

(f) **Réponse correcte : B**
α est le taux d'apprentissage. Il ajuste la rapidité avec laquelle les nouvelles informations influencent les valeurs estimées.

(g) **Réponse correcte : B**
Si ε est proche de 1, alors l'agent explore fréquemment de nouvelles actions au lieu d'exploiter l'action la plus rentable connue.

Pourquoi reformuler l'équation de mise à jour avec $(1 - \alpha)$?

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a')\right) \tag{8}$$

1. Mise en perspective de l'impact de l'apprentissage

Dans la forme classique de mise à jour, on voit principalement ce que l'on ajoute grâce à α, mais on ne voit pas ce qu'il advient de la valeur précédemment apprise. La reformulation explicite en $(1 - \alpha)$ montre immédiatement la part de l'information historique conservée.

- Par exemple :

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') \right)$$

- On voit que $(1-\alpha)Q(s,a)$ représente la mémoire du passé.

2. Rendre plus intuitif le rôle de α

Cette formulation aide à comprendre α comme un *facteur de confiance* :

- Si $\alpha = 0.1$, alors $1-\alpha = 0.9$: on garde 90% de l'ancienne valeur.
- Si $\alpha = 0.9$, alors $1-\alpha = 0.1$: on privilégie fortement l'information récente.

Cela permet d'ajuster l'agressivité de l'apprentissage de manière plus intuitive.

3. Mise en évidence de la pondération de l'historique

Le terme $(1-\alpha)$ joue le rôle de mémoire historique. Plus il est grand, plus l'algorithme est conservateur. Plus il est faible, plus on donne d'importance à l'expérience actuelle.

- Pour un apprentissage lent mais stable, il est préférable d'avoir un α faible.
- Pour un apprentissage rapide, mais plus instable, un α plus grand est utilisé.

4. Contrôle de la stabilité

- Si α est trop grand, l'algorithme risque de surcorriger à chaque nouvel échantillon.
- Si α est trop petit, il apprend très lentement.
- En rendant visibles les deux termes $(1-\alpha)$ et α, on comprend mieux ce compromis.

Conclusion

La reformulation de la mise à jour sous la forme :

$$\text{Nouvelle valeur} = (1-\alpha) \times \text{Ancienne valeur} + \alpha \times \text{Nouvelle estimation}$$

permet de mieux comprendre comment les algorithmes d'apprentissage par renforcement combinent passé et présent pour converger progressivement. C'est une aide précieuse pour la pédagogie, l'analyse de la stabilité, et la conception des algorithmes.

14 Introduction à la Programmation Dynamique

La **programmation dynamique** est une technique de résolution de problèmes qui consiste à diviser un problème complexe en sous-problèmes plus simples. Elle repose sur un principe fondamental : mémoriser les solutions des sous-problèmes déjà résolus pour éviter leur recalcul. Cela permet un gain significatif en temps de calcul, notamment pour les problèmes à structure récursive.

Exemple classique : suite de Fibonacci

Un exemple classique est le calcul de la suite de Fibonacci :

$$F(n) = F(n-1) + F(n-2)$$

Sans programmation dynamique, cette opération peut entraîner de nombreux recalculs redondants des mêmes valeurs de $F(n-1)$ et $F(n-2)$.

15 Principes de base

Deux concepts clés sous-tendent la programmation dynamique :

i. **Sous-problèmes qui se répètent** : Le problème peut être décomposé en sous-problèmes identiques ou très similaires.
ii. **Stockage des résultats intermédiaires** : Les résultats sont mémorisés dans une structure de données (tableau, dictionnaire, etc.), afin d'éviter de refaire les calculs.

Ce principe est appelé **mémoïsation** lorsque les résultats sont stockés de manière explicite durant l'exécution récursive.

16 Deux grandes approches

1. Approche Top-down (avec mémoïsation)

- Utilise une fonction récursive.
- Lorsqu'un sous-problème est résolu, le résultat est mémorisé.
- Si le même sous-problème se présente à nouveau, le résultat mémorisé est directement utilisé.

2. Approche Bottom-up (itérative)

- Résout d'abord les sous-problèmes les plus simples.
- Utilise généralement une boucle pour remplir un tableau de résultats.
- Permet souvent une meilleure gestion de la mémoire et un contrôle plus précis du flux de calcul.

17 Conclusion

La programmation dynamique est un outil puissant pour optimiser des algorithmes ayant une structure récursive. Elle est largement utilisée en algorithmique (chemins optimaux, alignement de séquences, optimisation de ressources, etc.) ainsi qu'en apprentissage par renforcement (notamment à travers l'équation de Bellman).

Objectif pédagogique

Ce document explique la différence entre la **valeur attendue théorique** et la **moyenne empirique** à travers l'exemple d'un lancer de dé. Il s'appuie sur une simulation informatique pour illustrer un concept fondamental des probabilités : la loi des grands nombres.

1. Valeur attendue théorique

La valeur attendue théorique d'un lancer de dé à six faces équilibré est :

$$E(X) = \frac{1}{6} \cdot (1 + 2 + 3 + 4 + 5 + 6) = \frac{21}{6} = 3.5$$

Chaque face ayant une probabilité uniforme de $\frac{1}{6}$, la moyenne de toutes les valeurs possibles converge vers 3.5. Cette valeur est fixe et ne dépend pas du hasard.

2. Moyenne empirique

La moyenne empirique est obtenue à partir d'une simulation réelle : on lance un dé un grand nombre de fois, puis on calcule la moyenne des résultats observés.
Soit $x_1, x_2, \ldots, x_n$ les résultats des n lancers. Alors :

$$\bar{x}_n = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Cette moyenne dépend du hasard, mais plus n est grand, plus $\bar{x}_n$ tend vers $E(X) = 3.5$.

3. Description du code

Le programme simule un grand nombre de lancers (par exemple 10 000) et calcule progressivement la moyenne des résultats :

- À chaque lancer, une valeur aléatoire entre 1 et 6 est générée.
- Cette valeur est ajoutée à une somme cumulative.
- La moyenne courante est enregistrée à chaque étape.

Variables principales

- `sum_of_dice_values` : somme cumulée des résultats
- `running_average` : liste contenant la moyenne après chaque lancer
- `dice_value` : valeur obtenue à chaque lancer

4. Résultat attendu

Le graphe de la moyenne empirique (en fonction du nombre de lancers) montre une courbe qui fluctue au début, mais qui se stabilise autour de la valeur 3.5 lorsque le nombre de lancers devient grand.

Cela illustre parfaitement la **loi des grands nombres** :

> Plus on répète une expérience aléatoire, plus la moyenne des résultats converge vers la valeur théorique attendue.

Conclusion

Ce type de simulation est une excellente introduction au concept de convergence en probabilité, en rendant visuellement accessible une notion mathématique importante. Il montre également l'intérêt de l'analyse empirique pour vérifier des théories abstraites.