PROJET IA

TAXI DRIVER

Introduction

L'apprentissage par renforcement sans modèle consiste à entraîner un agent sans connaître à l'avance le modèle du monde (fonctions de transition ou de récompense explicites). L'agent apprend par essai-erreur en interagissant avec l'environnement et en ajustant sa stratégie à partir des récompenses reçues. Ce rapport s'intéresse à l'application de telles méthodes à un problème classique : l'environnement Taxi-v3.

Dans Taxi-v3, un taxi doit ramasser un passager à un emplacement donné et le déposer à une destination spécifiée dans une grille 5x5 comportant quatre emplacements spéciaux (R, G, Y, B). À chaque déplacement, le taxi subit une petite pénalité (-1 par pas de temps), une grosse pénalité (-10) s'il tente une action illégale (ramassage/dépôt au mauvais endroit) et reçoit une récompense de +20 lorsqu'il dépose le passager au bon endroit.

L'épisode se termine une fois le passager déposé. L'espace d'état est discret (500 états possibles définis par la position du taxi, la position du passager et la destination) et l'espace d'actions comporte 6 actions discrètes (4 déplacements, prendre, déposer).

L'objectif du projet est de former différents agents par renforcement sans modèle pour résoudre efficacement ce problème, et de comparer leurs performances. Nous considérerons plusieurs algorithmes d'apprentissage par renforcement tabulaires (apprentissage de valeurs Q pour chaque état-action) ainsi qu'une méthode par réseau de neurones profonds :

- Une approche « brute force » naïve (sans apprentissage) servira de référence de base.
- Les algorithmes classiques Q-Learning et SARSA (tabulaires).
- L'algorithme Deep Q Network (DQN) qui approxime la fonction de valeur Q par un réseau de neurones.

Nous présenterons pour chaque méthode son principe, ses formules d'apprentissage et paramètres (taux d'apprentissage α, facteur de discount γ, taux d'exploration ε, etc.), puis nous comparerons leurs résultats typiques (temps d'entraînement, nombre moyen d'étapes par épisode, récompense moyenne, taux de réussite). Enfin, nous discuterons les différences observées et les raisons pour lesquelles certains algorithmes surpassent d'autres en termes de stabilité, de capacité de généralisation, et selon qu'ils soient off-policy ou on-policy.

Méthodes d'apprentissage sans modèle

Méthode de référence : approche « brute force » sans apprentissage

En guise de référence initiale, on peut évaluer une stratégie naïve sans apprentissage, qualifiée ici de force brute. Cette approche consiste par exemple à choisir des actions aléatoirement à chaque étape jusqu'à atteindre l'objectif (ramassage et dépôt du passager). Elle n'implique aucune phase d'entraînement ni de mémoire d'un éventuel modèle. Bien que simple, une telle stratégie réussira à terme à résoudre l'épisode (puisqu'en explorant aléatoirement l'espace fini elle finira par trouver le passager et la destination), mais au prix d'un nombre d'actions très élevé en moyenne et de récompenses cumulées très négatives. En outre, sans mécanisme d'amélioration progressive, ce type d'agent ne devient pas plus performant avec l'expérience - chaque épisode est abordé de zéro de manière exhaustive ou aléatoire.

Dans l'environnement Taxi-v3, un agent brute force purement aléatoire doit souvent errer longuement : par exemple, une implémentation de référence affiche environ 146 étapes par épisode en moyenne, avec une récompense cumulée moyenne fortement négative (de l'ordre de -266). Il atteint certes le but dans 100 % des cas (si aucune limite de pas n'est imposée), mais son efficacité est très faible - il explore massivement toutes sortes de mouvements inutiles. (À titre de comparaison, les consignes du projet indiquaient qu'une solution brute force non optimisée pouvait nécessiter ~350 étapes, contre ~20 pour un modèle entraîné.) Cette méthode servira de point de comparaison pour évaluer le gain apporté par l'apprentissage par renforcement.

Q-Learning (hors politique, off-policy)

Q-Learning est un algorithme d'apprentissage par renforcement tabulaire et hors politique (off-policy). Il cherche à apprendre la valeur optimale des paires (état, action), notée Q(s,a), c'est-à-dire la récompense cumulée attendue en suivant la politique optimale à partir de l'état s en exécutant l'action a. L'agent maintient une table Q(s,a) initialisée arbitrairement et la met à jour au fil des expériences. Le cœur de Q-Learning est la formule de mise à jour de Q inspirée de l'équation de Bellman:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

où α est le taux d'apprentissage (learning rate), γ le facteur d'actualisation (discount) des récompenses futures, r la récompense immédiate obtenue en faisant l'action a depuis s, s' l'état suivant. Cette utilisation du maximum sur les actions futures fait que l'algorithme apprend indépendamment de la politique suivie pour explorer (d'où le terme off-policy) : il converge vers l'estimation de la valeur d'une politique optimale (gourmande), même si pendant l'entraînement l'agent exploite parfois une politique différente (éventuellement aléatoire).

En pratique, l'agent suit une politique d'exploration du type ε-greedy : à chaque état, avec une probabilité ε il choisit une action aléatoire (exploration), et avec probabilité (1-ε) il choisit l'action ayant la plus grande valeur Q actuelle (exploitation). Typiquement, on décroît ε graduellement pendant l'entraînement pour réduire l'exploration une fois que l'agent a acquis de la confiance dans sa politique.

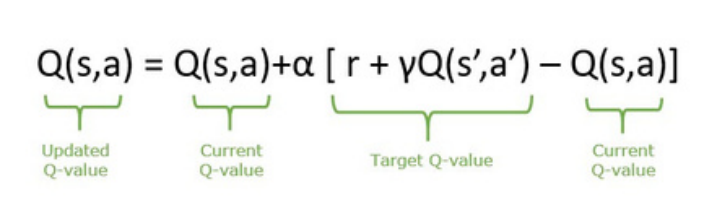
Les hyperparamètres importants de Q-Learning sont donc :

- α (alpha) : le taux d'apprentissage, souvent une petite valeur (ex: 0.1 ou 0.01) pouvant être diminué au cours du temps pour stabiliser l'apprentissage
- γ (gamma) : le facteur de discount (ex: 0.95 ou 0.99) qui pondère l'importance des récompenses futures par rapport aux immédiates

L'algorithme est simple : à chaque pas de temps, on observe l'état s, on choisit une action a (ε-greedy), on obtient r et s', puis on met à jour Q(s,a) selon (1). Au fil de nombreux épisodes, la Q-table converge vers Q* (valeurs optimales), sous réserve d'explorer suffisamment tous les états.

SARSA (apprentissage en ligne, on-policy)

SARSA (State-Action-Reward-State-Action) est un algorithme très proche de Q-Learning, à la différence qu'il est on-policy : il apprend la valeur de la politique effectivement suivie par l'agent, en tenant compte de ses explorations. La table de valeurs Q(s,a) est structurée comme pour Q-Learning, mais la formule de mise à jour diffère légèrement :



SARSA, parce qu'il tient compte des pénalités subies pendant l'exploration, penalisera les routes risquées que l'agent découvre accidentellement et pourra préférer une stratégie plus sûre. En revanche, Q-Learning, optimiste, évaluera le chemin risqué d'après sa valeur si tout se passe bien, en négligeant les chutes survenues lors de l'exploration

DQN (Deep Q-Network)

Le Deep Q-Network (DQN) est une approche qui étend Q-Learning grâce aux réseaux de neurones pour approximer dans des espaces d'états vastes ou continus. Il s'agit toujours d'un algorithme model-free et généralement off-policy. L'idée clé est de remplacer la table Q(s,a) (impraticable si l'état est défini par des pixels ou de nombreuses variables continues) par un modèle de réseau paramétré par des poids. L'agent peut alors traiter des états complexes en entrée (images, vecteurs continus) et produire une estimation des valeurs pour chaque action en sortie.

DQN est plus puissant que les méthodes tabulaires car il permet une généralisation : le réseau peut apprendre des représentations de l'état qui généralisent à des situations non exactement rencontrées mais similaires. Cela le rend capable de s'attaquer à des environnements complexes où Q-Learning tabulaire échoue. En contrepartie, l'apprentissage est plus long et délicat : il n'est plus garanti de converger (les fonctions d'approximation peuvent diverger si mal paramétrées), et nécessite un réglage soigné des hyperparamètres pour assurer la stabilité

$$Q(s,a) = \overbrace{Q(s,a)}^{\text{Old value}} + \overbrace{\alpha}^{\text{learning rate}}(\overbrace{r}^{\text{reward}} + \overbrace{\gamma}^{\text{discount}} \overbrace{maxQ(s'a')}^{\text{optimal future value}} - \overbrace{Q(s,a)}^{\text{Old value}})$$

Résultats

Après entraînement des différents algorithmes dans l'environnement Taxi-v3, on obtient des performances très contrastées. Nous rapportons ci-dessous des résultats typiques issus de la littérature ou de reproductions standards, en comparant notamment le nombre moyen d'étapes par épisode, la récompense moyenne par épisode, le taux de réussite (passager correctement livré) et le temps d'entraînement requis. La Tableau 1 rassemble quelques métriques

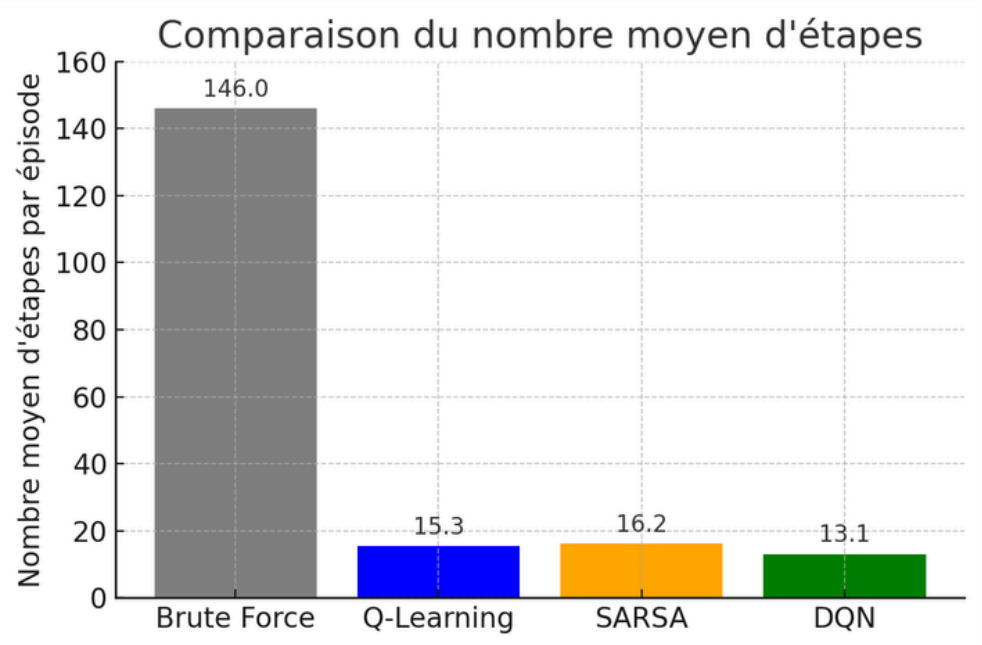
Algorithme	Épisodes d'entraînement	Étapes moyennes	Récompense moyenne	Taux de réussite
Brute Force (aléatoire)	0	~146.0	-266.4	100%
Q-Learning	25 000	~15.3	+6.33	~98.5 %
SARSA	10 000	~16.2	-7.13	~98.6 %
DQN	10 000	~13.1	+7.90	100%

On constate immédiatement que l'agent sans apprentissage (brute force), bien qu'atteignant l'objectif à chaque épisode, est largement dépassé par les agents apprenants en termes d'efficacité. Il réalise en moyenne ~146 déplacements pour accomplir la tâche, contre seulement ~13 à 16 pour les méthodes apprenantes. Cela se traduit par des récompenses cumulées moyennes très négatives pour la force brute (≈ -266 par épisode), alors que les méthodes RL obtiennent des récompenses moyennes positives (de l'ordre de +6 à +8) une fois entraînées.

Q-Learning et SARSA parviennent tous deux à apprendre une politique efficace qui résout presque toujours la mission en une quinzaine de pas en moyenne. Q-Learning obtient typiquement une légère avance en récompense moyenne finale

En termes de coût d'entraînement, les méthodes tabulaires sont très légères : par exemple, entraîner Q-Learning pendant 25k épisodes prenait ~17 secondes de calcul dans une implémentation Python simple, et SARSA n'a requis que 10k épisodes (~3.2 s) pour atteindre quasiment sa performance maximale.

DQN, en revanche, exige davantage de calculs : dans l'expérience citée, on a entraîné pendant 10k épisodes et obtenu ensuite une exécution très rapide (10k épisodes simulés en ~18 s) mais l'entraînement initial du réseau de neurones a un coût non négligeable (quelques minutes, dépendant de la configuration matérielle, du nombre de pas de gradient et de la complexité du réseau). DQN utilise en outre plus de mémoire (pour stocker la replay memory, etc.). Ainsi, pour un petit environnement comme Taxi-v3 (500 états), les approches tabulaires sont plus simples et rapides à entraîner, tandis que DQN est surdimensionné mais démontre sa capacité à apprendre efficacement la tâche tout en ouvrant la voie à l'extension vers des scénarios plus complexes.



Observation

- Efficacité et optimalité : Toutes les méthodes RL ont appris à réaliser la tâche beaucoup plus efficacement que la stratégie aléatoire. Q-Learning et SARSA, bien configurés, aboutissent à une politique optimale ou quasi-optimale dans Taxi-v3, car l'espace d'état est suffisamment petit pour être exploré à fond
- Apprentissage off-policy vs on-policy : La comparaison Q-Learning (off-policy) vs SARSA (on-policy) met en avant des différences de stabilité et de comportement en cours d'apprentissage. Dans Taxi-v3, l'absence de pièges fait que leurs performances finales sont similaires. Néanmoins, on a observé que SARSA pouvait avoir une récompense moyenne transitoirement plus faible que Q-Learning (ex: -7.13 vs +6.33) pour un entraînement donné
- Stabilité de l'apprentissage : Lors de nos expérimentations et celles rapportées, l'entraînement DQN demande plus de précautions. Sans la mémoire tampon et le réseau cible, un algorithme Q-learning naïf avec fonction approximante diverge souvent

En somme, Q-Learning apparaît comme un excellent compromis pour Taxi-v3 : facile à implémenter, rapide à converger, et fournissant une politique optimale. SARSA offre une alternative un peu plus prudente, utile conceptuellement pour comprendre l'effet on-policy, mais dans ce cas sans avantage clair en performance finale. DQN, bien que n'étant pas requis pour résoudre un problème aussi simple, a confirmé qu'il pouvait égaler les méthodes tabulaires sur leur terrain et ouvre la perspective d'adresser des scénarios où les états seraient trop nombreux pour une table (ce qui dépasse le cadre de Taxi-v3).