RL Project: Black Jack

Oscar Frelot, Antoine Guiot

April 2020

1 Description

Dans ce projet nous allons tenter d'appliquer les algorithmes de reinforcement appris en cours sur le jeu du Black Jack.

2 Regles

Avant tout faisons un point sur les règles de ce jeu : Le jeu commence avec deux cartes distribuées au croupier et au joueur. L'une des cartes du croupier est face visible et l'autre est face cachée. Si le joueur a 21 immédiatement (un as et un 10), on dit qu'il a un "Black Jack". Il gagne, sauf si le croupier a également un "Black Jack", auquel cas le jeu est nul (il y a égalité.). Si le joueur n'a pas de "Black Jack", il peut demander des cartes supplémentaires, une par une (hits), jusqu'à ce qu'il s'arrête (sticks) ou dépasse 21 (go bust). S'il dépasse 21, il perd ; s'il reste, c'est le tour du croupier. Le croupier demande des cartes supplémentaires ou s'arrête selon une stratégie fixe sans choix : il s'arrête sur toute somme de 17 ou plus, et demande un carte autrement. Si le croupier dépasse 21, le joueur gagne ; sinon, le résultat (victoire, défaite ou match nul) est déterminé par la somme finale qui est la plus proche de 21. Si le joueur détient un as il pourrait le compter comme 11.

3 Environement

Avant d'implémenter un algorithme, il faut que nous définissions le cadre dans lequel nous allons travailler. Nous avons 3 éléments :

- le state
- les actions
- les rewards

3.1 State

3 informations seront contenues dans le state:

- la valeur des cartes du joueur
- la valeur de la carte du croupier
- la présence d'un as utilisable dans les cartes du joueur

Le state sera donc un array de longueurs 3 avec ces trois informations.

3.2 Actions

2 actions sont possibles à chaque tour:

- S'arreter
- Demander une carte

On representera donc l'action "s'arreter" par 0 et l'action "demander une carte" par 1.

3.3 Rewards

Il y a trois issues possible une partie:

- victoire
- match nul
- défaite

On définit donc respectivement les rewards 1,0,-1.

4 Implementation

Dans ce projet, nous allons implémenter différents algorithmes afin de résoudre au mieux le problème du black jack. L'idée est de construire un tableau comportant toutes les combinaisons (valeur des cartes du joueur, valeur des cartes du croupier). Et ainsi pour chaque combinaison, indiquer quelle est la meilleure action à effectuer (demander une carte, s'arrêter). Pour ce faire, nous allons dans un premier temps décrire l'environnement du jeu (voir classe 'BlackJackEnv'). Ensuite, nous allons comparer plusieurs policy obtenues à l'aide de différents algorithmes:

- Random policy (qui sera notre policy de reférence pour les comparaisons)
- MonteCarlo
- Sarsa
- Q-learning

La difficulté du problème que nous étudions provient du fait que nous ne connaissons pas les transitions d'état de la chaîne de markov (model free). Les 3 algorithmes que nous allons étudier se décomposent donc en 2 phases :

- Estimation de la value function
- Amélioration de la policy qui se fait itérativement

Pour chacune des policy, notre métrique de comparaison sera le rapport entre le pourcentage de défaite et le pourcentage de victoire.

5 Random

A chaque étape le joueur choisit une action de manière aléatoire avec une probabilité 0,5 .

Les résultats obtenus sont les suivants :

On peut noter que cette policy est très défavorable au joueur qui présente un gros ratio de défaite.

	Victoire	Défaite	Egalité
Résultats	28.52%	58.5%	12.98%

Table 1: Random Policy

6 Monte Carlo

Pour l'algorithme de Monte Carlo l'estimation se fait à l'aide d'épisode. Il faut simuler l'ensemble de l'épisode avant de mettre à jour la policy.

On fixe une policy qui nous permet de générer un épisode (c'est-à-dire mener la partie à son terme) et à l'aide du résultat de l'épisode on peut estimer la value function et ensuite mettre à jour la policy.

La méthode de monte carlo nous fournit le meilleur estimateur en norme L_2 et elle converge en $O(1/\sqrt{n})$, n représentant le nombre d'épisodes. On obtient donc la state, value function suivante.

$$q_\pi(s,a) = E^\pi(G_t|S_t = s, A_t = a) \tag{1}$$

La méthode de monte carlo à un fort coût d'exploration. Il faut en effet simuler l'ensemble des couples $(state, action)$.

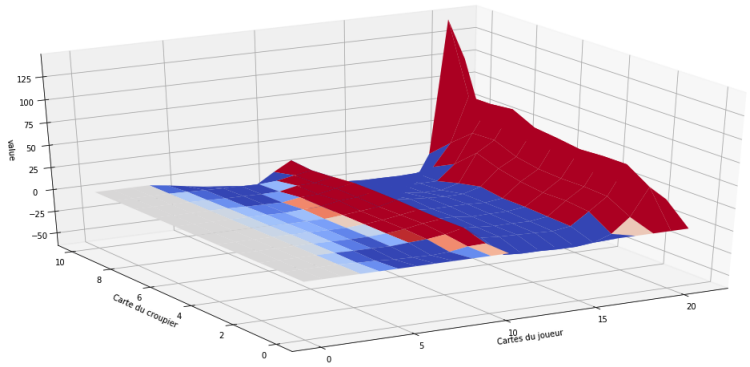

Figure 1: Value function: MC

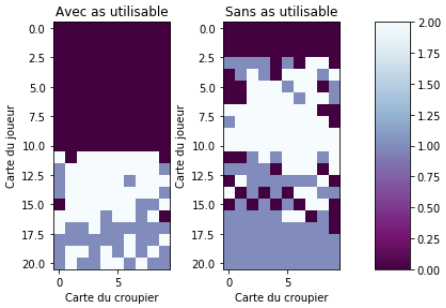

Figure 2: Policy: MC

Après 10000 épisodes, les résultats de l'approche monte-carlo sont les suivants :

	Victoire	Défaite	Egalité
Résultats	39.87%	46.04%	14.09%

Table 2: Monte Carlo

On peut noter que les résultats obtenus sont bien supérieurs à ceux obtenus à l'aide d'une policy aléatoire, cependant le ratio de défaite est toujours supérieur à celui de victoire.

7 Sarsa

La méthode SARSA comme la méthode Q-learning repose sur la différentiation temporelle (TD). Contrairement à la méthode Monte Carlo elle ne nécessite pas de simuler l'ensemble d'un épisode. La policy est mise à jours au fur et à mesure.

Dans le cas de la méthode SARSA (pour State-Action-Reward-State-Action) on met à jour la Q-function à chaque changement d'état.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2}$$

On choisit ensuite l'action suivante en se basant sur une policy ϵ-greedy. Qui permet de choisir un compromis entre exploitation et exploration.

Soit r un nombre aléatoire entre 0 et 1. Si $r < \epsilon$, on choisit a_{t+1} de la façon suivante

$$a_{t+1} = argmax(Q(s_t, a_t)) \tag{3}$$

Sinon a_{t+1} est choisit aléatoirement.

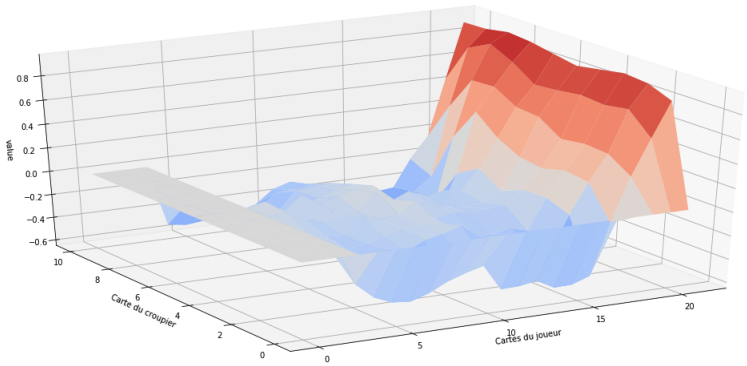

Figure 3: Value function: Sarsa

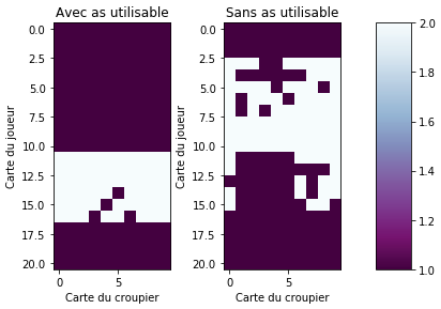

Figure 4: Policy: Sarsa

Les résultats pour la méthode SARSA sont les suivants :

	Victoire	**Défaite**	**Egalité**
Résultats	42.89%	46.13%	10.98%

Table 3: Q-learing

8 Q-learning

Pour l'algorithme de Q-learning le principe est le même que pour l'algorithme de SARSA, il repose sur la différentiation temporelle.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{4}$$

La méthode Q-learning est une "off policy" contrairement à la méthode SARSA qui est une "on policy". Cela signifie que SARSA suit la policy pour choisir l'action suivante alors que Q-learning suit une approche "greedy" (d'où le maximum dans la formule)

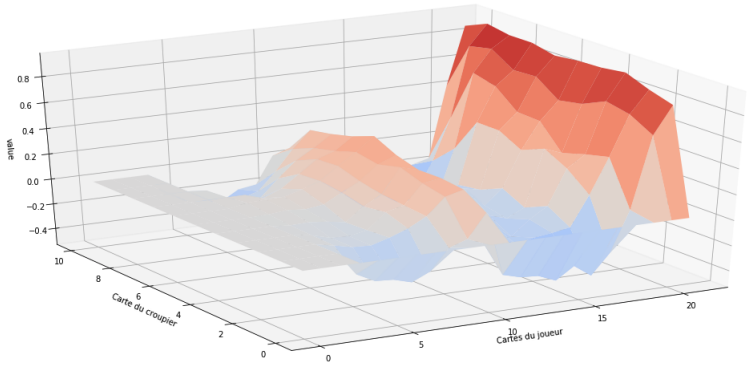

Figure 5: Value function: Q-learning

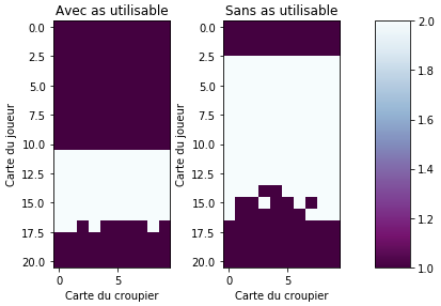

Figure 6: Policy: Q-learning

Les résultats pour la methode de Q-learning sont les suivants :

	Victoire	**Défaite**	**Egalité**
Résultats	41.95%	40.23%	17.82%

Table 4: Q-learing

9 Conclusion

Au cours de ce projet, nous avons pu mettre en pratique les différents algorithmes que nous avions étudiés en cours.

Un premier travail d'analyse a été nécessaire afin de modéliser l'environnement dans lequel nous avons travaillé. Il a fallu définir clairement ce qui allait être notre state, notre reward et nos actions possibles.

Nous avons pu ensuite implementer les méthodes de Monte Carlo et de différentiation temporelle (TD) pour SARSA et Q-learning.

Nous avons obtenu des résultats plus que satisfaisant notamment à l'aide de l'algorithme de Q-learning. En effet, celui est le seul à présenter un ratio de victoire supérieur à celui de défaite.