Reinforcement Learning

Morpion et puissance 4 grâce à des algorithmes de reinforcement learning

---

AXEL MARCHAND, RODRIGUE RILLARDON

1 Introduction

Ces dernières années, d'énormes avancées ont été faites en terme de reinforcement learning. Ainsi, de nombreux chercheurs développent des algorithmes capables de jouer à des jeux complexes comme Alpha Go et AlphaGo Zero pour le jeu de go, ou encore des algorithmes plus génériques capables d'apprendre sur des échecs ou sur le jeu de shogi. Ce genre d'algorithme sont en général capable de générer eux-mêmes des parties et d'apprendre sur les données obtenues lors de ces auto-parties.
Lors de notre projet, nous avons voulu adapté l'algorithme AlphaZero à des jeux plus simples comme le morpion et le puissance 4. Pour se faire, nous avons utilisé une architecture comprenant un algorithme basé sur des réseaux de neurones convolutifs, ainsi que sur un Monte Carlo Tree Search (MCTS).
L'objectif de ce rapport est de rendre compte du fonctionnement de ce réseau convolutif ainsi que du Monte Carlo Tree Search.

2 Fonctionnement de l'algorithme

L'objectif du réseau de neurones est d'être capable de retourner un vecteur de probabilité ainsi qu'une valeur associée, à partir d'un plateau de jeu (puissance 4 ou morpion). Le vecteur de probabilité représente la distribution de probabilité des actions du joueur. En clair, chaque élément du vecteur correspond à la probabilité de jouer sur une case en particulier (ainsi pour le puissance 4, ce vecteur est de taille 6 alors que pour le morpion il est de taille 9). La valeur associée au vecteur de probabilité représente simplement la probabilité de gagner (pour le joueur dont c'est le tour de jouer) à partir d'un plateau donné.
Ce réseau est composé d'un premier bloc convolutif, contenant une convolution, puis d'un bloc résiduel à 19 couches contenant lui-même deux convolutions, enfin d'un block de sorti permettant d'obtenir les distributions de probabilités prédites ainsi que la probabilité de gagner. Ce modèle est entraîné à partir d'une fonction de perte défini de la manière suivante : une somme entre l'erreur moyenne au carré de la probabilité de gagner et la perte de la cross-entropy de notre vecteur de probabilités.
Ce réseau est utilisé dans un MCTS afin de chercher à optimiser le choix du "joueur". On peut imaginer le jeu de morpion ou de puissance 4 (comme le go ou les échecs d'ailleurs) comme un arbre, dont la racine est l'état du plateau, et les feuilles sont toutes les états possibles de plateau que l'on peut obtenir à partir de l'état du plateau initial. Problème, pour des jeux complexes tels que le jeu de Go, évaluer tous les états de plateau possible serait très coûteux en terme de mémoire et de temps de calcul. Ce genre d'arbre permet notamment de répondre au dilemme classique de reinforcement entre exploration et exploitation, en s'appuyant notamment sur des méthodes UCB (upper confidence bound).

Le but de l'algorithme MCTS est d'explorer l'arbre des possibles, où la racine est l'état initial du jeu, les noeuds sont des états possibles et chaque enfant correspond aux configurations suivantes du plateau. L'idée est que le MCTS garde en mémoire l'arbre dans lequel il y a déjà eu des phases d'explorations (certains noeuds ont déjà été déterminé mais pas forcément tous -pour les jeux complexes notamment). Ainsi, une feuille peut prendre différentes formes ; soit être une feuille finale (un joueur gagne ou match nul), soit être une partie de l'arbre liée à un état qui n'a pas encore été exploré. Il est important de noter qu'au niveau de chaque noeud, l'arbre stock à la fois le nombre de simulation gagnantes (à partir de cet état) et le nombre de simulations total (ces valeurs sont importantes notamment pour le calcul de l'UCB).
L'arbre fonctionne de la manière suivante :

— (Phase d'initialisation) On part d'un plateau à l'état s (certaines cases peuvent déjà être jouée si on n'est pas au début de la partie)

— (Phase de selection) Puis on choisit la branche (l'état de jeu suivant) qui a le plus grand UCB jusqu'à atteindre une feuille (état qui n'a pas été encore exploré) ou un noeud final (jeu fini : gagnant ou match nul). Ce choix est donc guidé par le dilemme exploitation (choisir le noeud qui jusqu'ici a été le plus performant) et exploration (visiter un noeud qui pourrait être plus prometteur).

— (Extension) Dans le cas où notre feuille n'est pas une feuille finale, on va créer un nouvel enfant à ce noeud en évaluant les actions possibles à partir de notre réseau de neurones pour obtenir les probabilités des différentes actions en fonction de l'état du plateau. Il est à noter que les actions illégales (par exemple, dessiner un symbole sur une case où il y a déjà un symbole dans le cas du morpion) restent cachées, et que l'on rajoute du bruit si le noeud actuel est une racine afin de rajouter de l'aléatoire dans notre arbre pour diversifier la phase d'exploration.

— (Phase de simulation) On simule une exécution d'une partie à partir de cet enfant jusqu'à atteindre une feuille finale. Ceci permet au réseau d'évaluer l'issue de la branche et donc de déterminer la probabilité v de gagner à partir d'un état donné.

— (Phase de Backpropagation) On va alors utiliser cette probabilité v pour mettre à jour les probabilités v (moyennes des victoires) des noeuds parents jusqu'à la racine. Ainsi, si les noeuds précédents avaient comme valeurs 0/0, 2/4 et 3/5, dans le cas où la partie est gagnante, ils sont actualisées de la manière suivante : 0/0 devient 1/1, 2/4 devient 3/5 et 3/5 devient 4/6.

Notre algorithme va donc s'entraîner sur un certain nombre de parties permettant d'obtenir le vecteur de probabilités associés aux actions à jouer à partir de l'état initial. Le choix d'action est donc guidé par cette distribution de probabilité.