Projet : Puissance 4

Gazzo Sandro
Dellouve Théo

19 janvier 2021

Définir l'environnement

L'objectif de ce projet est d'obtenir un réseau de neurones permettant de jouer au jeu du puissance 4. Pour cela, il est nécessaire de commencer par définir les fonctions permettant de modéliser le jeu.

L'environnement du jeu contient différents types de fonctions.

Tout d'abord, il faut des fonctions permettant de définir et initialiser la grille de jeu. Dans notre cas, on obtient une grille de taille 6×7.

Pour faire fonctionner le jeu, on a également besoin d'une fonction permettant de jouer un coup et de changer de joueur.

Il faut ensuite une fonction pour vérifier si un coup est jouable ou si la colonne sélectionnée est pleine.

On définit ensuite la fin du jeu, avec une fonction vérifiant l'alignement des pions et une autre vérifiant si la grille est pleine.

Finalement, pour interagir avec un joueur, on a besoin d'une fonction permettant d'afficher la grille du jeu.

Les emplacements vides sont représenté par des 0 tandis que les pions sont représentés par de "O" et des "X" selon le joueur, auxquels on affecte la valeur ± 1 dans la partie programmation.

Définir le réseau de neurones

On définit ensuite le réseau de neurones qui correspond au jeu. On considère que chaque emplacement de la grille correspond à un neurone d'entrée du réseau, et chaque action possible correspond à un neurone de sortie. Ainsi, on obtient 42 neurone d'entrées et 7 neurones de sortie, correspondant aux choix de la colonne où le pion sera lâché.

Dans le cas d'un réseau à 1 couche cachée, on met 100 neurones sur celle-ci.

Concernant les paramètres, on choisit $\alpha = 0.5$, que l'on multiplie par 0.9999 à chaque partie. On a ensuite $\epsilon = 0.1$ permettant l'exploration et $\gamma = 0.9$. A chaque nouvelle période d'apprentissage, on remet ces paramètres à leur valeur initiale, precisée ci-dessus.

On passe donc à la phase d'apprentissage à l'aide de l'algorithme deep Q-learning. Pour cela, on considère 3 méthodes d'apprentissage différentes. Le premier cas consiste à entraîner le réseau contre une IA déjà programmée, jouant en fonction des poids de chaque cases tout en bloquant ou alignant automatiquement lorsque nécessaire. La deuxième méthode consiste à entraîner le réseau de neurones contre une version passée de lui même, que l'on met à jour toutes les 1000 parties. La dernière méthode consiste à entraîner le réseau de neurones contre une IA aléatoire.

Dans les deux cas, le joueur jouant le 1er coup est choisi aléatoirement. En effet, on sait que le 1er joueur possède un avantage et ce tirage aléatoire permet d'équilibrer les chances de victoire.

Pour définir les récompenses, on choisit de renvoyer 1 en cas de victoire, -1 en cas de défaite et 0 en cas de match nul.

Pour pouvoir observer un effet d'apprentissage, on utilise le réseau de neurones obtenu après son apprentissage contre une politique aléatoire sur 1000 parties. Le pourcentage de victoire sera ensuite comparé au résultat d'une politique aléatoire contre une autre politique aléatoire, s'élevant en moyenne à 50% de victoire, ou un nombre à peu près égal de victoires/défaites et l'obtention de matchs nuls.

Résultats obtenus

Pour comparer nos résultats, on se base sur une partie IA aléatoire contre IA aléatoire, où l'on obtient environ un taux de victoires et de défaites quasiment identiques.

Après un apprentissage sur 20 000 parties contre l'IA automatisée, on obtient des scores allant de 0 à 5% de taux de victoire. Cela est très faible et ne nous satisfait pas. On va donc essayer d'autres méthodes d'apprentissage.

De même, après un apprentissage sur 20 000 parties contre l'IA aléatoire, on obtient des scores d'environ 70 à 73% de taux de victoires contre une politique aléatoire. On a donc cette fois un bon apprentissage. Cependant, c'est une IA aléatoire, donc sans réelle stratégie et le réseau de neurones se révèle toujours très peu efficace si on la refait jouer contre une IA automatisée. Il n'y a pas de réelle stratégie qui se met en place.

Après un apprentissage sur 20 000 parties contre une version passée de lui même, on obtient des scores d'environ 50% de taux de victoires et de défaites contre une politique aléatoire, parfois un peu moins (45%) mais dans ce cas, c'est compensé par des matchs nuls. Ensuite, le réseau apprend et augmente son taux de victoire jusqu'à parfois 65% de taux de victoires jusqu'à la mise à jour de son adversaire, c'est à dire son lui-même passé. On observe alors une baisse subite du taux de victoires, autour de 30% à 40%. Puis le réseau réapprend à nouveau pour atteindre les 55 à 60%. Cela représente un taux de victoire plus important qu'une politique aléatoire, témoignant ainsi de l'évolution de la méthode de jeu du réseau de neurones.

Après avoir obtenu ce nouveau réseau de neurones, on le test sur 1000 parties contre une IA aléatoire. On obtient un taux de 76 à 80% de taux de victoires, ce qui est légèrement supérieur que le match contre l'IA aléatoire (hausse d'environ $5-6\%$).

On pourrait alors croire que les deux premiers apprentissages ne servent à rien, mais ils permettent au réseau de neurones de découvrir le jeu. On pourrait plutôt baisser le nombre de parties le temps de faire découvrir le jeu et d'entrainer le réseau de neurones.

De plus, en jouant contre le réseau de neurones après son apprentissage, on remarque que ce dernier développant une stratégie de jeu. En effet, il a tendance à empiler verticalement ses pions. Bien que la stratégie soit simple, cela permet d'observer le phénomène d'apprentissage. De plus, une telle stratégie est adaptée contre une politique aléatoire et c'est ce qui permet d'obtenir un taux de victoire élevé.

En affichant chaque partie que le réseau de neurones joue contre une version passée

de lui même, on observe une politique de jeu aléatoire au début, mais les coups joués deviennent très vite plus rapprochés avant d'obtenir des empilements de pions.

Dans tous les cas, on observe une augmentation du taux de victoire contre la politique aléatoire, témoignant de l'apprentissage du réseau de neurones.

Axes d'amélioration

On a pu voir certaines conclusions au point précédent. Nous y apportons ici une possibilité d'amélioration à effectuer. Comme nous avons un très mauvais taux de victoires contre l'IA programmée, il faudrait peut-être faire plus de parties pour que notre réseau de neurones puissent encore apprendre.

On pourrait aussi voir à faire une alternance entre apprentissage sur cette IA, enregistrer le réseau de neurones, le faire s'affronter contre lui même comme dans notre deuxième méthode en mettant à jour régulièrement et sauvegarder le réseau de neurones obtenu à la fin. On referait alors jouer ce réseau contre l'IA programmée et ainsi de suite jusqu'à commencer à obtenir des résultats corrects.

Une autre méthode serait d'attribuer des récompenses à chaque joué en fonction des cases occupées, du nombre de pions alignés et des blocages de l'adversaire.

Egalement, il serait intéressant de mettre en place une fonction qui juge quand mettre le réseau de neurones à jour lorsqu'il s'affronte lui-même : avoir un taux de victoires assez important et savoir à partir de combien de parties nous pouvons le faire. En effet, il est très probable que notre deuxième méthode puisse détériorer le réseau de neurones en le mettant à jour alors qu'il n'arrive déjà pas à battre la précédente mise à jour.

Enfin, nous nous sommes inspirés de la méthode de résolution du Backgammon et nous avons donc choisi un réseau de neurones ne possédant qu'une seule couche. Il pourrait être intéressant d'étudier les résultats du réseau de neurones à 2 couches pour voir s'il y a une amélioration.