RL – Rapport de projet

Juliette J., Matis B.

18 Novembre 2024

Table des matières

1 Description du sujet

Le projet de Deep Reinforcement Learning consiste à développer un modèle de Deep Q-Learning pour le jeu Pong de la console Atari. Pour cela, nous nous sommes inspiré du papier de recherche "Playing Atari with Deep Reinforcement Learning".

2 Expérimentation et améliorations

Un des premiers problèmes que nous avons repérés est celui de la mémoire utilisée pendant l'entraînement. En effet, le modèle utilisé dans le papier de recherche garde en mémoire le dernier million d'états de jeu, ce qui n'était pas possible sur les ordinateurs que nous utilisions. Comme chaque état de jeu contenait 4 frames consécutives du jeu, stocker ces états de façon naïve créait une duplication conséquente de données. Nous avons donc créé une nouvelle gestion de mémoire, qui permet de stocker chaque frame qu'une seule fois sans changer la sélection aléatoire d'états de jeu.

Un autre problème que nous avons rencontré a eu lieu avec le modèle décrit dans l'article. En effet, celui-ci utilisait une activation ReLU sur [10, 18] après la première couche de convolution, mais cette limite sur le ReLU semblait empêcher l'entraînement du modèle. Nous avons donc utilisé une activation ReLU simple à la place.

Pour le choix du learning rate utilisé pendant l'entraînement des modèles, nous avons effectué des petits entraînements sur une cinquantaine de jeux avec des puissances de 10 différentes en observant l'évolution de la loss des modèles. Le premier learning rate ne causant pas une grande augmentation de la loss pendant l'entraînement était de 10^{-5}, nous avons donc utilisé cette valeur pour le learning rate de tous nos modèles.

Nous avons ensuite ajouté des améliorations dans l'implémentation pour avoir de meilleurs résulats. La première amélioration a été la Double Q-Learning. Au lieu d'utiliser 1 seul réseau de neurones pour sélectionner la meilleure action (le max), nous en utilisons 2 : l'un pour sélectioner les actions optimales, et l'autre pour évaluer leur valeur. Cela évite de systématiquement prendre l'action optimale, ce qui peut introduire un biais.

Une autre amélioration que nous avons pu faire est le Dueling DQL, qui améliore les 2 réseaux de neurones du Double Q-Learning en ajoutant 2 branches après les couches convolutives. La première branche qui va calculer la valeur de l'état V(s) (va indiquer si l'état à l'instant donné est bien ou non), et la deuxième qui va calculer l'avantage des actions (qui mesure si l'action est pire ou meilleur que les autres). Cela permet d'identifier les états importants. En théorie, le Dueling DQL accélere la convergence durant notre entrainement.

La dernière amélioration que nous avons fait est la Noisy Layer. La méthode est d'ajouter des paramètres aléatoires aux poids des couches de notre réseau Dueling DQL. Cela va remplacer la méthode e-greedy, et va permettre d'explorer directement dans le modèle grâce aux données apprises au fil de l'entrainements. Normalement, cela améliore l'équilibre entre la décision d'explorer et d'exploiter.

3 Résultats

Avant notre implémentation de Double Q-Learning, Dueling DQL et du Noisy Layer, nous avions eu des difficultés à obtenir un modèle qui gagnait des jeux de pong. Cependant, suite à l'ajout de l'algorithme Double Q-Learning, nous avons obtenu un modèle qui gagnait les jeux avec 16 points d'avance en moyenne suite à un entraînement sur 2 millions de frames. Ce modèle a été utilisé pour créer la vidéo de démonstration, qui contient un jeu gagné avec 19 points d'avance. Une partie intéressante a eu lieu pendant un des entraînements de Double Q-Learning : un modèle avait trouvé une méthode contre-intuitive de gagner, en se plaçant à un endroit précis puis en restant immobile. Cela avait permit à ce modèle de gagner sans laisser l'adversaire marquer un seul point.

Quant à l'entrainement avec le Dueling DQL et Noisy Layer, faute de bonnes machines et de temps, nous avons plutôt privilégié l'entrainement avec le Double Q-Learning. Malgré tout, nos tests sur environ 300 parties montre que le modèle réussissait à gagner quelques points à chaque fois. L'implémentation est tout de même disponible sur le Github du projet.