Apprentissage par renforcement, Breakout - Novembre 2024

alexandre.devaux-riviere

Présentation

Pour ce projet, j'ai implémenté deux modèles en suivant les instructions données dans les différents papier scientifiques traitant du sujet : un Deep Q-Network (DQN) basique, et un "rainbow" qui est une approche d'apprentissage par renforcement combinant plusieurs techniques modernes pour améliorer considérablement les performances de l'agent sur les jeux Ataris et en particulier Breakout.

Choix d'implémentation

Pre-processing de l'environnement

J'utilise plusieurs techniques de prétraitement standard d'Atari, proposées dans les différents papiers que j'ai pu lire, et qui sont directement appliquées via la librairie gym :

- Redimensionnement des images à 84x84 pixels pour enregistrer uniquement la zone de jeu
- Conversion en niveaux de gris
- On saute 4 images consécutives
- Coupe des récompenses à l'intervalle [-1, 1].
- Empilement des 4 dernière observations

Architecture des modèles

- **DQN basique** : voir 'classic_dqn/classic_dqn/dqn.py'
  Cette implémentation utilise un réseau neuronal convolutionnel avec deux couches primaires qui extraient des caractéristiques de plus en plus abstraites des images du jeu. Suivi par deux layers denses pour arriver à un choix parmi les 4 actions possibles pour tout état dans le jeux.

- **DQN rainbow** : voir 'rainbow_dqn/DuelingDQN_model.py'
  Cette implémentation suit les bases données dans le DQN basique, cependant le réseau convolutionnel est un peu plus profond et est ensuite divisé en deux flux parallèles avec génération de bruit pour encourager l'exploration : un flux de valeurs qui estime la valeur globale de l'état et un flux d'avantages qui évalue les avantages des actions individuelles.

  Récapitulatif :
  - Couches convolutives initiales de profondeur croissante (32, 64, 64 canaux)
  - Taille des noyaux progressivement réduite (8x8, 4x4, 3x3)
  - Couches NoisyLinear pour encourager l'exploration (voir : arXiv:1706.10295v3)
  - Des flux de sortie distincts pour les valeurs et les avantages (voir : arXiv:1511.06581v3)

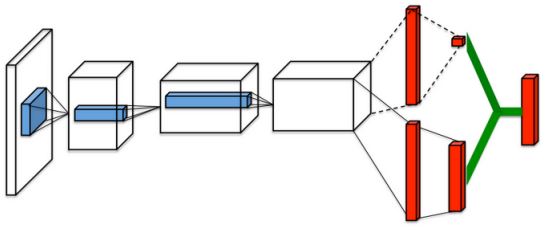

FIGURE 1 – Illustration du modèle présenté dans le papier arXiv:1511.06581v3

Sélection des hyperparamètres

Du fait de l'instabilité des modèles d'apprentissage par renforcement, un réglage minutieux des hyperparamètres a été nécessaire. En me basant sur les valeurs proposés par les papiers, et en expérimentant certains changements du fait des limitations de mes ressources de calcul (RTX 3070Ti et 16Go de RAM), j'ai abouti à ces valeurs qui permettent au modèle d'apprendre même si les performances sont instables d'une époque à l'autre pour le rainbow DQN :

Hyperparamètres	Basique DQN	Rainbow DQN
Learning rate	0.0001	0.0000625
Discount factor (γ)	0.99	0.99
Replay memory size	100,000	100,000
Batch size	32	32
Target update frequency	5,000	5,000
Frame skip	4	4
Min epsilon	0.1	N/A
Max epsilon	1.0	N/A
Epsilon decay steps	4M (steps)	N/A
Max steps	4.5M	8M
Replay start size	32	80,000
Save frequency	50,000	50,000
Noisy nets std_init	N/A	0.5
PER* alpha (α)	N/A	0.6
PER* beta start (β)	N/A	0.4
Reward clipping	[-1, 1]	[-1, 1]
Input frame stack	4	4

TABLE 1 – Comparison des hyperparamètres entre le DQN basique et rainbow

***PER** pour "prioritized experience replay" permettant de hiérarchiser l'expérience, afin de rejouer les transitions importantes plus fréquemment, et donc d'apprendre plus efficacement. Voir le papier arXiv:1511.05952v4.

Résultats

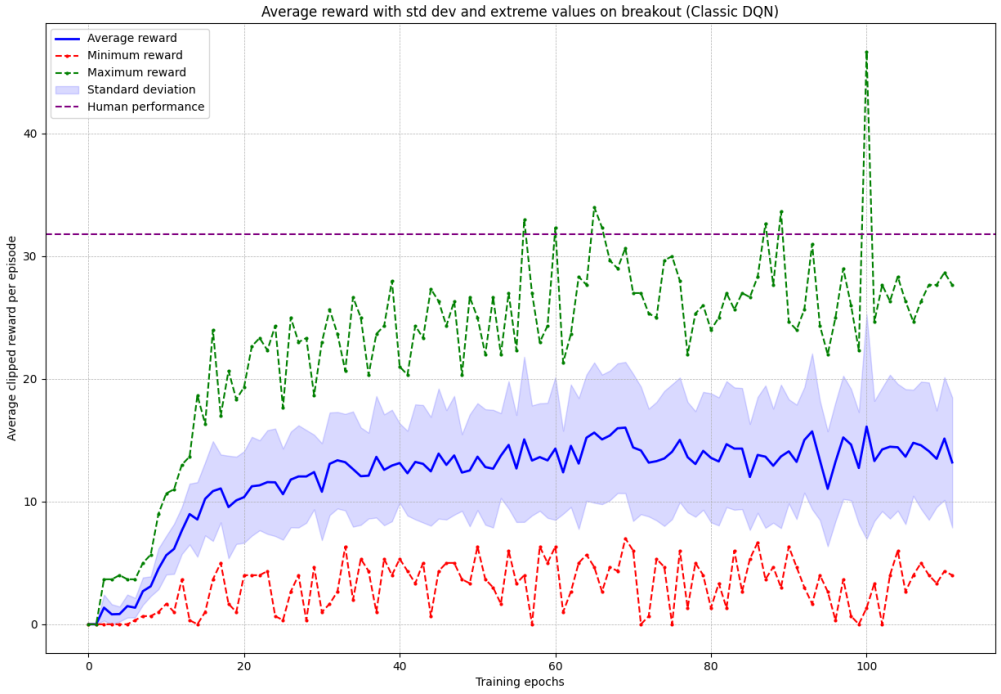

FIGURE 2 – Courbe d'apprentissage pour le DQN basique en 10h d'entrainement (chaque époque représente 50K lots)

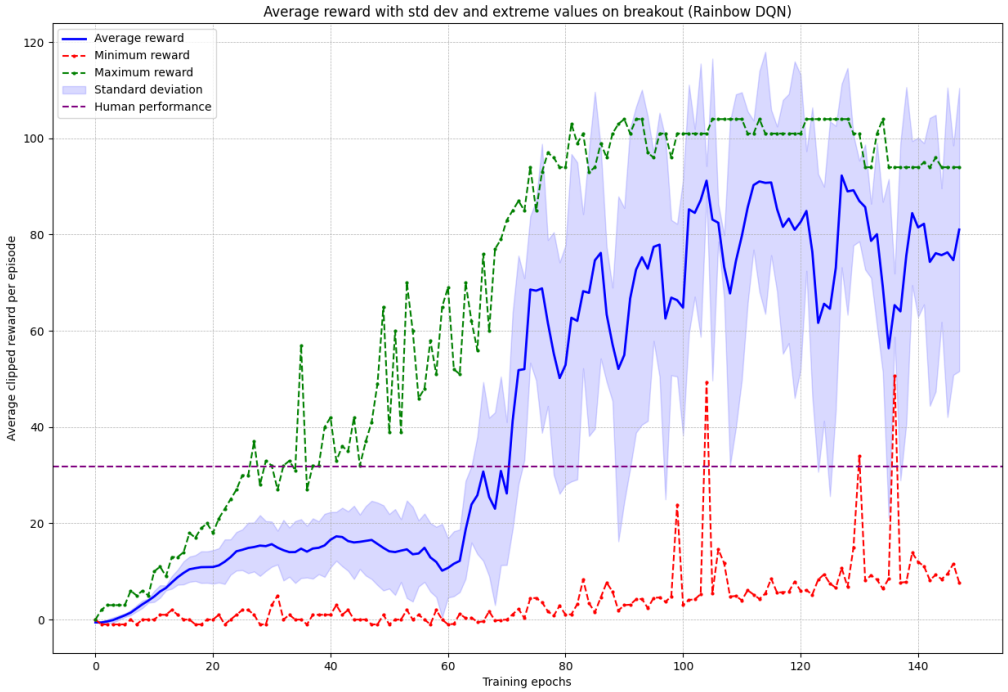

FIGURE 3 – Courbe d'apprentissage pour le DQN rainbow DQN en 14h d'entrainement (chaque époque représente 50K lots)

Agent	Score Moyen	Écart-Type	Min	Max
Agent aléatoire	1.364	**1.394**	0.0	7.0
Performance humaine	31.8	-	-	-
DQN basique	14.211	5.852	0.0	28.0
DQN rainbow	**70.242**	32.931	**8.0**	**105.0**

TABLE 2 – Comparaison des performances des agents sur Breakout (le score maximal atteignable est 6*18=108 briques détruites)

Le **DQN basique** montre une amélioration par rapport à l'agent aléatoire, atteignant un score moyen de 14.211. Cependant, son score maximum de 28.0 reste bien en dessous de la performance humaine, suggérant que ce modèle ne parvient pas encore à rivaliser avec des stratégies expertes comme casser les briques sur un coté pour amener la balle au dessus de la zone à détruire. On remarque qu'il est plutôt stable dans son apprentissage.

Le **rainbow DQN**, surpasse de loin les autres agents. Avec un score moyen de 70.242 et un score maximum de 104.0, il dépasse non seulement le DQN basique, mais également la performance humaine. Cependant, l'écart-type élevé (32.931) et la courbe d'apprentissage indiquent une forte variabilité dans ses performances, possiblement due à des cas spécifiques où le modèle échoue ou excelle.

Cela peut aussi venir au fait que je n'ai implémenté que 4 des 6 améliorations proposées dans le papier "rainbow" (voir : arXiv:1710.02298v1), à savoir :

- La combinaison de l'apprentissage par deux réseaux DQN (policy et target network dans 'rainbow_dqn/deep_q_learning_agent.py' et voir : arXiv:1509.06461v3)
- Une architecture de réseaux en duel (DuelingDQN)
- Le PER permettant de hiérarchiser l'expérience
- Et l'introduction de bruit dans les couches linéaires