Compte-rendu Pong

Clovis Lechien & Florian Ségard-Gahery

Epita - Reinforcement Learning

Novembre 2024

Clovis LECHIEN *(clovis.lechien@epita.fr)*
Florian Ségard-Gahery *(florian.segard-gahery@epita.fr)*

Table des matières

1 Code - Implémentation en Python

1.1 main_dqn.py

Ce fichier contient la boucle d'entraînement principale de notre modèle.
Nous définissons d'abord l'environnement Gym, en optant pour "PongNoFrameskip-v4" plutôt que la dernière version proposée par Gym, "ALE/Pong-v5".
Le tableau suivant présente les différences entre ces deux environnements :

Env-id	obs_type	frameskip	repeat_action_probability
PongNoFrameskip-v4	"rgb"	1	0.0
ALE/Pong-v5	"rgb"	4	0.25

Nous définissons ensuite les variables d'entraînement ainsi que l'agent, représentant le joueur de Pong.
L'entraînement démarre, et nous enregistrons les scores obtenus. Chaque fois que le score maximal est amélioré, le modèle est sauvegardé.

1.2 agent.py

Le fichier `agent.py` constitue le cœur de notre modèle.
Il gère l'interaction entre le modèle Deep Q-Network (DQN) et l'environnement Gymnasium.
Les expériences sont stockées dans un "replay buffer", et des échantillons y sont régulièrement tirés pour entraîner le modèle.
Les actions sont choisies suivant une politique "epsilon-greedy".
Le modèle est mis à jour périodiquement en fonction des décisions et des résultats obtenus.
Enfin, ce fichier permet de sauvegarder et de charger les poids du modèle.

1.3 utils.py

Ce fichier contient des wrappers pour l'environnement Gym ainsi que des fonctions utilitaires.
Le wrapper `SkipEnv` permet de sauter certaines frames et de cumuler les récompenses.
`PreprocessFrame` convertit les frames en niveaux de gris et les redimensionne pour le modèle.
`MoveImgChannel` réorganise les canaux de couleur de l'image.
La fonction `make_env` combine ces wrappers et renvoie un environnement modifié.

1.4 replay_buffer.py

Ce fichier définit la classe `ReplayBuffer`, qui stocke et gère les expériences d'entraînement de l'agent.
Ces expériences incluent l'état, l'action choisie, la récompense, l'état résultant et un indicateur de fin de partie.
Le constructeur `__init__` initialise les variables nécessaires.

La fonction `store_transition` enregistre une expérience dans le buffer.
La fonction `sample_buffer` tire un échantillon du buffer pour l'entraînement.

1.5 model_dqn.py

Ce fichier contient l'implémentation du modèle de réseau de neurones, basé sur les couches de convolution et les couches entièrement connectées, en utilisant PyTorch.
Le modèle traite les frames du jeu, extrayant des caractéristiques via les couches convolutionnelles, puis les utilise pour estimer les valeurs-Q associées à chaque action possible.
La classe `DQN`, héritant de `nn.Module`, définit la structure suivante :

- Trois couches de convolution
- Une couche dense
- Une couche de sortie

La méthode `forward` réalise la passe avant dans le réseau, tandis que `calculate_fc_input_dims` calcule la taille de la sortie de la troisième couche de convolution.
Enfin, la fonction `build_dqn` crée et initialise une instance de la classe `DQN` sur le dispositif souhaité.

1.6 animation.py

Le fichier `animation.py` est responsable de la visualisation des parties.
Il sauvegarde les frames pour former une animation, soit au format GIF, soit au format MP4.

2 Résultats

L'entraînement complet du modèle s'est déroulé sur une carte graphique RTX 3070 Laptop et a pris environ 4 heures.
Le tableau suivant montre l'évolution de l'entraînement à différents intervalles de temps :

Temps d'entraînement	Épisode	Meilleur score	Score moyen	Étapes
30 minutes	125	-19.0	-17.99	~170,000
1 heure	250	19.0	5.93	~530,000
4 heures	900	21.0	20.5	~2,000,000

Les résultats montrent une progression significative des performances de l'agent au cours de l'entraînement. Voici une interprétation détaillée :

- **Après 30 minutes** : L'agent a atteint l'épisode 125, avec un score moyen de -17.99, indiquant des performances limitées et des actions souvent inefficaces. Le meilleur score de -19.0 reste très bas, signe d'un manque de maîtrise du jeu à ce stade précoce de l'entraînement.
- **Après 1 heure** : À l'épisode 250, l'agent commence à montrer des signes d'apprentissage avec un score moyen de 5.93. Le meilleur score est passé à 19.0, ce qui indique que l'agent est capable de jouer plus efficacement et d'obtenir des scores positifs sur certaines parties.
- **Après 4 heures** : À la fin de l'entraînement, à l'épisode 900, l'agent atteint un score moyen de 20.5, proche du score maximal possible, avec un meilleur score de 21.0. Cette progression montre que l'agent a acquis une stratégie efficace pour maximiser son score de manière régulière.

L'évolution du score moyen et du meilleur score au fil des étapes témoigne de la stabilité et de l'efficacité croissante de l'agent.

Pour plus de détails, consultez le dépôt GitHub contenant les animations des performances de l'agent en formats GIF et MP4 : `https://github.com/ClovisDyArx/rl-tp4-pong`.