Rapport

Exploration de Q-Learning, DQN et PPO sur MiniGrid

1. Introduction et Organisation du Groupe

Notre groupe de 4 s'est organisé en **binômes**, chacun se concentrant sur un axe du projet :

- **Binôme 1** : implémentation et expérimentation de l'algorithme DQN (Deep Q-Network)
- **Binôme 2** : implémentation et expérimentation du Q-Learning classique

Ce partage a permis d'aborder à la fois l'approche "tableau" du Q-Learning et la version "réseau de neurones" du DQN.

En fonction de l'avancée, le groupe a ensuite travaillé sur **PPO** et testé les trois algorithmes sur des environnements MiniGrid de difficulté croissante (notamment `MiniGrid-Empty-16×16-v0`).

2. Retour sur les Concepts Théoriques — Q-Learning vs DQN

2.1 Q-Learning (tabulaire)

- **Principe** : l'agent construit une **Q-table** où il apprend pour chaque état-action la "valeur" attendue (Q(s,a)).
- **Mise à jour** :

  $Q(s, a) \leftarrow Q(s, a) + \alpha \times [r + \gamma \times \max_{a'} Q(s', a') - Q(s, a)]$
- **Limite** : Devient impraticable quand le nombre d'états est très élevé (par exemple, lorsqu'une observation est une image).

2.2 DQN (Deep Q-Network)

- **Principe** : la Q-table est remplacée par un **réseau de neurones** qui approxime Q(s, a).
- **Cible** :

  $\text{target} = r + \gamma \times \max_{a'} Q\_\text{target}(s', a')$
- **Techniques avancées** :
  - **Replay Buffer** : l'agent apprend à partir d'un échantillon d'expériences passées (plus de stabilité).
  - **Target Network** : un réseau "figé" pour calculer les cibles, mis à jour régulièrement.

- **ε-greedy** avec décroissance : exploration contrôlée, puis exploitation.

2.3 **PPO (Proximal Policy Optimization)**

- **Principe** : approche “policy gradient” : le réseau apprend directement une politique (et une estimation de la valeur).
- **Robustesse** : stabilité grâce au “clipping” de la mise à jour de la politique.
- **Utilisé** pour des environnements avec observations complexes et espace d’action large.

2.4 **Pertinence des choix**

- **Q-Learning** : rapide à implémenter, utile pour petits espaces d’états (ex : MiniGrid-Empty-8×8).
- **DQN** : indispensable dès que l’état devient une image ou une séquence complexe (ex : MiniGrid-Empty-16×16).
- **PPO** : recommandé pour aller plus loin, généraliser et s’attaquer à des environnements à forte variabilité.

---

3. **Travail Réalisé et Méthodologie**

3.1 **Découverte de l’environnement MiniGrid**

- **Installation** : via pip install minigrid, test de différents environnements pour se familiariser avec les observations et les actions.
- **Exploration** : lecture de la documentation, observation des retours d’`env.step(action)` (état, récompense, done, etc.).
- **Manipulations manuelles** : utilisation de scripts pour déplacer l’agent à la main et comprendre le système de récompense.

3.2 **Implémentation des agents**

Q-Learning

- Implémentation de l’algorithme tabulaire.
- Discrétisation de l’espace d’état si besoin.
- Tests sur `MiniGrid-Empty-8×8-v0` puis `MiniGrid-Empty-16×16-v0`.

DQN

- **Architecture :**
  - Entrée : représentation de l’état (image ou vecteur)

    - 2 couches cachées : ex : 128 → 64 neurones (ReLU)
    - Sortie : une valeur Q par action
- **Techniques utilisées** :
    - Replay buffer (mémoire de 10 000 transitions)
    - Target network (update tous les 100 pas)
    - ε-greedy décroissant (ex: ε=1→0.05)
- **Entraînement :**
    - Tests sur MiniGrid-Empty-16×16-v0
    - Visualisation des courbes de récompense et du taux de réussite

PPO

- Implémentation basée sur un modèle Actor-Critic avec deux têtes.
- Suivi des pertes (policy_loss, value_loss, entropy, total_loss) avec TensorBoard.
- Expérimentation sur des environnements de plus en plus complexes.

---

4. Évaluation et Analyse des Résultats

Mesures de performance

- Taux de réussite : proportion d'épisodes où l'agent atteint la sortie.
- Récompense cumulée : évolution de la somme des récompenses par épisode.
- Courbes de convergence : analyse de la stabilité et de la vitesse d'apprentissage.

Comparatif des modèles selon l'environnement

Algorithme	Facilité d'implémentation	Performance sur 8×8	Performance sur 16×16	Stabilité	Limites/Forces
Q-Learning	++	+++	+	++	Simple, mais pas scalable
DQN	+	++	++	+++	Plus lent, mais passe à l'échelle
PPO	-	++	+++	+++	Très robuste, généralise bien

Défis rencontrés

- Difficulté d'exploration (l'agent peut tourner en rond sans explorer de nouvelles zones).

- Problèmes de convergence sur les grandes grilles.
- Nécessité d’ajuster le taux d’exploration et les hyperparamètres.

---

5. Discussion critique et perspectives

- **Q-Learning** montre ses limites sur des environnements à grande dimension ou riches en observations.
- **DQN** permet de s’affranchir de la limitation de la Q-table, mais nécessite une bonne gestion de la mémoire et des cibles.
- **PPO** offre la meilleure stabilité, particulièrement dans les environnements où l’agent n’a qu’une vue partielle et les états sont nombreux.

Améliorations possibles

- Ajouter la visualisation des trajectoires de l’agent.
- Tester des variantes d’architectures (réseaux convolutionnels pour DQN sur observation image).
- Appliquer l’approche à des environnements à obstacles ou avec des missions plus complexes.

---

6. Conclusion

Ce TP nous a permis de comparer concrètement **Q-Learning, DQN et PPO** sur la famille MiniGrid.

Chaque approche a ses forces et faiblesses selon la taille et la richesse de l’environnement.

Le travail de groupe en binôme a facilité la progression sur plusieurs fronts et permis un partage de compétences.