Rapport de projet - Apprentissage par renforcement

Participants :

- Abdallah Abdelsadek
- Adam Tawfik

Études de cas : Environnements Simple et TicTacToe

Ce rapport décrit les choix d'implémentation et les approches utilisées pour résoudre deux études de cas dans le cadre du projet **MN(AA)** : un environnement simplifié (**Simple**) et le jeu du Morpion (**TicTacToe**) à l'aide d'algorithmes d'apprentissage par renforcement.

1. Environnement Simple

Pour cet environnement statique à un seul état, nous avons testé trois algorithmes de bandits multi-bras :

- Epsilon-Greedy
- Gradient Bandit
- UCB (Upper Confidence Bound)

Chaque algorithme a été intégré dans une architecture d'agent et a permis d'apprendre les actions optimales à long terme en maximisant la récompense moyenne. L'agent n'interagit pas avec un adversaire, ce qui permet une convergence rapide.

2. Environnement TicTacToe

Pour le jeu du Morpion, nous avons utilisé les algorithmes suivants :

- Q-Learning

Algorithm 5 Q-Learning

Require: Taux d'apprentissage $\alpha > 0$, facteur d'actualisation $\gamma \in (0, 1)$
Ensure: Fonction d'action-valeur $Q(s, a)$ convergeant vers $Q^*(s, a)$

Initialiser $Q(s, a)$ arbitrairement pour tous les états s et actions a
for chaque épisode **do**
  Observer ou **choisir** un état initial s
  while s n'est pas terminal **do**
    Choisir une action a en fonction de Q (avec une exploration ε-greedy)
    Exécuter l'action a et **observer** la récompense r et l'état suivant s'
    Mettre à jour $Q(s, a)$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

    Affecter $s \leftarrow s'$
  end while
end for
Retourner la table $Q(s, a)$

- Value Iteration

Algorithm 4 Value Iteration pour estimer π^*

Require: Seuil de convergence $\theta > 0$, facteur d'actualisation γ
Ensure: Fonction de valeur $V(s)$ pour tout $s \in S$ et politique optimale π^*

Initialiser $V(s)$ arbitrairement pour tout $s \in S^+$, avec $V(\text{terminal}) \leftarrow 0$
repeat
  $\Delta \leftarrow 0$
  for chaque état $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma V(s')\Big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
Retourner la politique déterministe π^* telle que, pour tout $s \in S$:

$$\pi^*(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma V(s')\Big].$$

- Policy Iteration

Algorithm 3 Iterative Policy Evaluation with Policy improvement

Require: S, A, $p(s', r \mid s, a)$, $\gamma \in [0, 1]$, $\theta > 0$
Ensure: Politique optimale π^* et fonction de valeur optimale V^*

Initialiser arbitrairement $V(s)$ et $\pi(s)$ pour tout $s \in S$, avec $V(\text{terminal}) \leftarrow 0$
repeat
  repeat
    $\Delta \leftarrow 0$
    for chaque état $s \in S$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma V(s')]$
      $\Delta \leftarrow \max(\Delta,\ |v - V(s)|)$
    end for
  until $\Delta < \theta$
  policy-stable $\leftarrow$ true
  for chaque état $s \in S$ **do**
    $a_{\text{old}} \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma V(s')]$
    if $a_{\text{old}} \neq \pi(s)$ **then**
      policy-stable $\leftarrow$ false
    end if
  end for
until policy-stable est vraie
return π^* et V^*

-

Nous avons défini une fonction de récompense spécifique :

+1: quand le joueur 0 gagne, **-1:** quand le joueur 1 gagne, **0:** en cas de match nul, et **-0.25:** pour tout autre coup. Cela permet d'encourager des victoires rapides et décourager les actions non optimales.

Le joueur 0 **(agent)** apprend seul dans un environnement partiellement coopératif. À la fin de l'apprentissage, il est capable de battre des adversaires aléatoires ou coopératifs. Nous avons intégré un mode de jeu **[Agent vs Human]** et **[Agent vs Agent]** .

Conclusion

Les environnements et les algorithmes ont été choisis pour illustrer des cas simples et complexes.
Le système développé est modulaire, ce qui facilite l'ajout d'autres environnements et algorithmes.
Nos choix de fonction de récompense et de stratégie d'adversaire ont fortement influencé la performance de l'apprentissage.