Introduction à l'apprentissage par renforcement

Cours 2 : Algorithmes basés sur la valeur

Zhi YAN

ENSTA

16 janvier 2025

Algorithmes basés sur la valeur

- Q-learning
- SARSA (State-Action-Reward-State-Action)
- DQN (Deep Q-Network)

Les algorithmes ci-dessus sont également appelés *Temporal Difference (TD) Learning*.

```
def update_q_table(state, action, reward, next_state):
    best_next_action = np.argmax(q_table[next_state, :])
    td_target = reward + gamma * q_table[next_state, best_next_action]
    td_error = td_target - q_table[state, action]
    q_table[state, action] += alpha * td_error
```

ENSTA
IP PARIS

TD Learning

Apprentissage basé sur un modèle

- L'agent connaît toutes les informations de l'environnement (comme les probabilités de transition d'état et les fonctions de récompense).

$\Longrightarrow$ *Dynamic Programming*

Apprentissage sans modèle

- l'agent ne connaît aucune information sur l'environnement et ne peut apprendre qu'en interagissant avec l'environnement.

$\Longrightarrow$ *TD Learning*

ENSTA
IP PARIS

Bootstrapping

- TD Learning : Apprendre la **fonction de valeur** par bootstrapping.
- Bootstrapping : Utiliser une estimation pour en améliorer une autre.
- Bootstrapping dans TD Learning : Utiliser l'estimation de la valeur de **l'état suivant** pour mettre à jour l'estimation de la valeur de **l'état actuel**.

Paramètre

TD(0) :

- Le plus simple qui ne regarde qu'une étape en avant :

$$V(s) \leftarrow V(s) + \alpha[R + \gamma V(s') - V(s)]$$

TD(λ) :

- $\lambda \in [0, 1]$
- Plus λ est grand, plus les récompenses à long terme sont prises en compte.
- $\lambda = 1$: TD devient équivalent à la méthode de Monte Carlo.

SARSA

- La différence entre SARSA et Q-learning : Mise à jour de la Q-table.
  - Pour SARSA, a_{t+1} dans $< s_t, a_t, r_t, s_{t+1}, a_{t+1} >$ indique **l'action réellement exécutée** par l'agent dans l'état suivant, dont

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

  - Pour Q-learning : a_{t+1} dans $< s_t, a_t, r_t, s_{t+1}, a_{t+1} >$ indique **toutes les actions possibles** dans l'état suivant, dont

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

SARSA

1. Initialiser $Q(s, a)$ pour tous les états s et actions a (par exemple, à 0).
2. Définir le taux d'apprentissage α ($0 < \alpha \leq 1$).
3. Définir le facteur d'actualisation γ ($0 < \gamma \leq 1$).
4. **Répéter** (pour chaque épisode) :
5. Initialiser l'état s.
6. Choisir l'action a en l'état s selon la politique ε-greedy.
7. **Répéter** (pour chaque étape de l'épisode) :
8. Prendre l'action a, obtenir la récompense r, et l'état passe à s'.
9. Choisir l'action a' en l'état s' selon la politique ε-greedy.
10. Mettre à jour $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$.
11. $s \leftarrow s'$, $a \leftarrow a'$.
12. **Jusqu'à ce que** s soit terminal.
13. **Jusqu'à ce que** la condition de résiliation soit remplie.

ENSTA
IP PARIS

SARSA

Q-learning :

- Algorithme off-policy : Apprend la politique optimale (indépendamment de la politique réellement exécutée).
- Solution optimale (processus d'apprentissage moins stable).
- IA des jeux, planification de trajectoires, etc.

$\Longrightarrow$ Performance first!

SARSA :

- Algorithme on-policy : Apprend la valeur Q sous la politique actuelle (le processus d'apprentissage est donc influencé par la politique actuelle).
- Convergence rapide (mais peut vers une solution sous-optimale).
- Contrôle robotique, conduite autonome, etc.

$\Longrightarrow$ Safety first!

ENSTA
IP PARIS

Questions

- Dans le TP de notre dernier cours, combien d'états y avait-il ?
- Quelle était la taille de votre table Q ?
- Combien de temps vous a-t-il fallu pour entraîner votre modèle ?
- Avez-vous réfléchi à ce qui se passe lorsque les états ou les actions sont continus ?

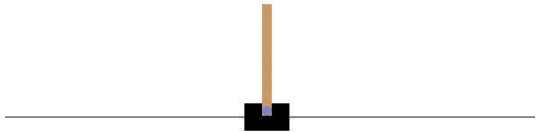
ENSTA
IP PARIS

Réflexions

- Traiter la table Q comme des données.
- Utiliser une fonction (paramétrée) pour adapter ces données.
- Une certaine perte de précision, mais peut gérer des espaces d'états de grande dimension et des états continus.

DQN

Le problème du poteau de chariot (Cart Pole) :



Pour plus d'informations : `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`

DQN

Espace d'action :

Num	Action
0	Pousser le chariot vers la gauche
1	Pousser le chariot vers la droite

Espace d'état / Espace d'observation :

Num	Observation	Min	Max
0	Position du chariot	-4.8	4.8
1	Vitesse du chariot	-Inf	Inf
2	Angle du poteau	-24°	24°
3	Vitesse angulaire du poteau	-Inf	Inf

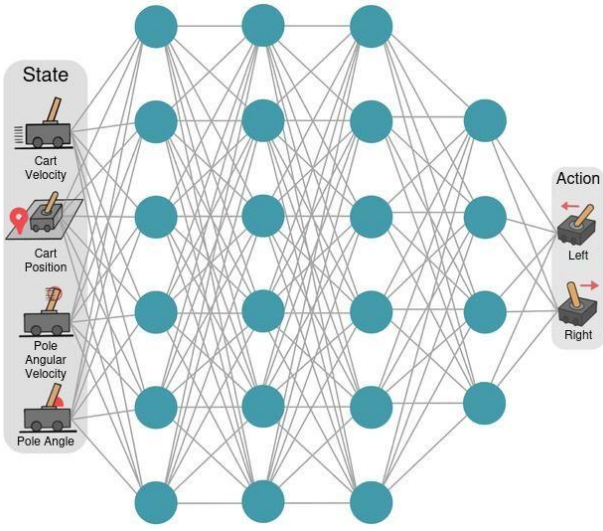
DQN

$$\hat{Q}(s, a; \theta) \approx Q(s, a)$$

Les paramètres du réseau neuronal (les poids et les biais)

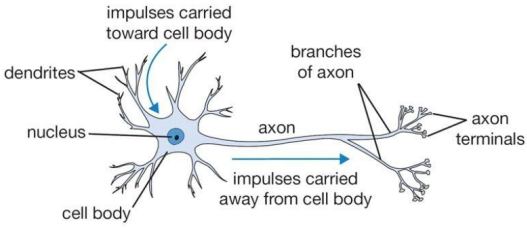
Function Approximation
(à l'aide d'un réseau neuronal)

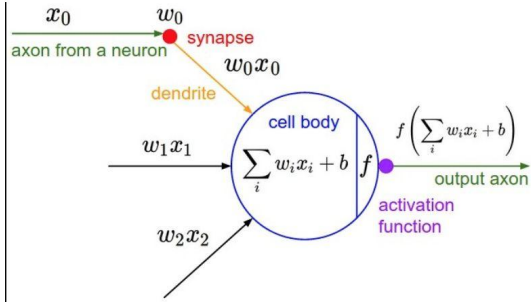
DQN


State
Cart Velocity
Cart Position
Pole Angular Velocity
Pole Angle
Action
Left
Right

DQN

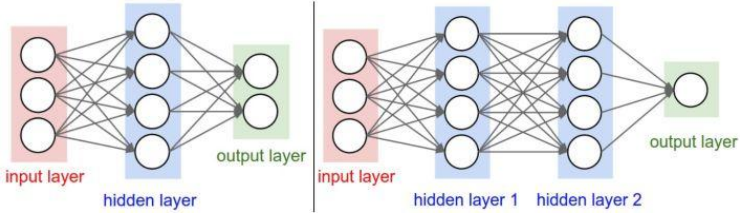
Un rappel sur les réseaux de neurones :

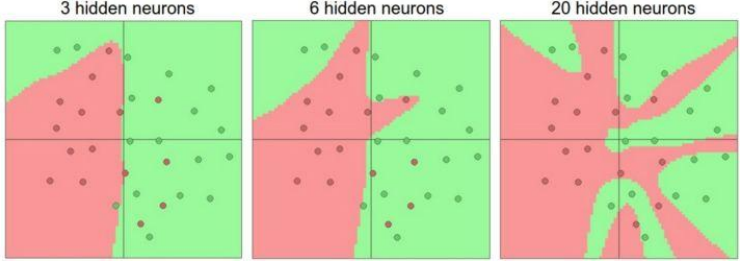




DQN

Un rappel sur les réseaux de neurones :





DQN

- DQN = Q-learning + Deep Learning
- Quelle est la fonction de perte (en anglais, *loss function*) du Q-Network ?
- Les fonctions de perte :
  - *Mean Squared Error (MSE)*
  - *Root Mean Squared Error (RMSE)*
  - *Mean Absolute Error (MAE)*
  - etc.

DQN

$$\mathrm{MSE}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}$$

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

$$\omega^* = \arg\min_{\omega} \frac{1}{N} \sum_{i=1}^{N} [r_i + \gamma \max_{a'} Q_\omega(s_i', a') - Q_w(s_i, a_i)]^2$$

Deux techniques clés du DQN

Experience Replay :

- D (un tampon de lecture) $\leftarrow < s, a, r, s' >$
- Rompre la corrélation des données (faire en sorte que les échantillons satisfassent à l'hypothèse d'indépendance).
- Améliorer l'efficacité des échantillons (chaque échantillon peut être utilisé plusieurs fois)

Deux techniques clés du DQN

Target Network :

- Rappeler l'objectif de l'algorithme :

$$Q_\omega(s,a) \approx r + \gamma \max_{a'} Q_\omega(s',a') \quad \text{ou} \quad Q(s,a;\theta) \approx r + \gamma \max_{a'} Q(s',a';\theta)$$

- **Problème** : Le côté droit de l'équation contient la sortie du côté gauche (l'objectif évolue constamment !)
- **Solution** : Utiliser un réseau distinct :

$$Q_\omega(s,a) \approx Q_{\omega^-}(s,a) \quad \text{and} \quad \omega_t^- \leftarrow \begin{cases} \omega_t & \text{if } t = nC \ (n \in \mathbb{N}) \\ \omega_{t-1}^- & \text{otherwise} \end{cases}$$

'C' est le nombre de pas (ou la longueur du pas).

DQN

1. Initialiser le réseau $Q_\omega(s, a)$ avec des paramètres de réseau aléatoires ω.
2. Copier les mêmes paramètres $\omega_t^- \leftarrow \omega_t$, pour initialiser le réseau cible $Q_{\omega^-}(s, a)$.
3. Initialiser le tampon de relecture d'expérience D.
4. **Répéter** (pour chaque épisode) :
5. Initialiser l'état s.
6. **Répéter** (pour chaque étape de l'épisode) :
7. Choisir l'action a en l'état s (en fonction du Q_ω) selon la politique ε-greedy.
8. Prendre l'action a, obtenir la récompense r, et l'état passe à s'.
9. Stocker $< s, a, r, s' >$ dans le tampon de relecture D.
10. S'il y a suffisamment de données dans D, échantillonner-en N données $< s_i, a_i, r_i, s_i' >_{i=1,\ldots,N}$.
11. Pour chaque donnée, utiliser le réseau cible Q_{ω^-} pour calculer $y_i = r_i + \gamma \max_{a'} Q_{\omega^-}(s_i', a')$.
12. Minimiser la perte cible L pour mettre à jour le réseau actuel Q_ω : $L = \frac{1}{N} \sum_{i=1}^{N} [y_i - Q_\omega(s_i, a_i)]^2$.
13. Tous les C pas, mettre à jour le réseau cible : $\omega_t^- \leftarrow \omega_t$.
14. **Jusqu'à ce que** s soit terminal.
15. **Jusqu'à ce que** la condition de résiliation soit remplie.

Fin