Cours : Méthodes de Monte Carlo en Apprentissage par Renforcement

Introduction

Les méthodes de Monte Carlo sont essentielles en apprentissage par renforcement (RL) car elles permettent d'estimer des valeurs d'états ou d'actions sans avoir de modèle prédictif de l'environnement. Contrairement aux méthodes temporelles comme TD (Temporal Difference), les méthodes Monte Carlo se basent sur des épisodes complets pour mettre à jour les valeurs d'état ou d'actions.

1 Comprendre les Méthodes de Monte Carlo

1.1 Qu'est-ce que la Méthode de Monte Carlo ?

Les méthodes de Monte Carlo en apprentissage par renforcement utilisent des **épisodes complets** pour estimer la valeur d'un état ou d'une action en se basant sur les retours observés. Elles reposent sur la technique d'**échantillonnage aléatoire** pour collecter des données et calculer des moyennes empiriques des récompenses obtenues.

Applications :

- Estimation de la fonction de valeur des états : $V(s)$
- Estimation de la fonction de valeur état-action : $Q(s, a)$

2 Prédiction Monte Carlo

2.1 Objectif de la Prédiction Monte Carlo

L'objectif est d'estimer la valeur $V(s)$ d'un état s pour une **politique fixe** π en utilisant l'observation d'épisodes successifs. Cette méthode est utilisée pour prédire le retour total attendu en partant d'un état et en suivant la politique π.

2.2 Processus de la Prédiction Monte Carlo

1. **Génération d'Épisodes** : Exécuter la politique π pour générer des épisodes complets.

2. **Calcul du Retour** G : À partir de chaque état dans chaque épisode, calculer le retour (récompense cumulée) jusqu'à la fin de l'épisode.

3. **Mise à Jour de la Valeur de l'État** : Moyenne des retours observés pour chaque état visité.

2.3 Méthodes Every-Visit et First-Visit

- **Every-Visit** : Met à jour la valeur de l'état chaque fois qu'il est visité dans un épisode.
- **First-Visit** : Met à jour la valeur de l'état uniquement lors de sa première visite dans un épisode.

Équations :

$$\text{Every-Visit} : V(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s)$$

$$\text{First-Visit} : V(s) \approx \frac{1}{N_{\text{first}}(s)} \sum_{i=1}^{N_{\text{first}}(s)} G_i(s)$$

où :

- $N(s)$ est le nombre total de visites de l'état s,
- $G_i(s)$ est le retour observé depuis s lors de l'épisode i.

3 Contrôle Monte Carlo

3.1 Objectif du Contrôle Monte Carlo

L'objectif est de trouver une **politique optimale** π^* en apprenant à partir des retours d'épisodes complets. Cela se fait en ajustant continuellement la politique en fonction des valeurs d'action estimées.

3.2 Contrôle Sur-Politique et Hors-Politique

1. **Contrôle Sur-Politique (On-Policy)** : La politique actuelle génère des épisodes et est améliorée progressivement pour maximiser les récompenses. Utilisation de la **politique epsilon-greedy** pour assurer un équilibre entre exploration et exploitation.

2. **Contrôle Hors-Politique (Off-Policy)** : Les épisodes sont générés par une politique de comportement μ différente de la politique cible π. Utilisation de l'**échantillonnage d'importance** pour corriger la différence entre la politique de comportement et la politique cible.

Exemple d'Équation (pour contrôle hors-politique) :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \frac{\pi(a|s)}{\mu(a|s)} \cdot (G - Q(s,a))$$

4 Politique Epsilon-Greedy

La politique epsilon-greedy aide à équilibrer entre :

- **Exploration** : Essayer des actions moins connues pour découvrir de nouvelles récompenses potentielles.
- **Exploitation** : Utiliser les actions connues pour maximiser les récompenses.

4.1 Algorithme de la Politique Epsilon-Greedy

1. **Initialisation** : Fixer une valeur de ϵ (taux d'exploration).
2. **Sélection de l'Action** :
   - Avec une probabilité de ϵ, choisir une action aléatoire (exploration).
   - Avec une probabilité de $1 - \epsilon$, choisir l'action qui maximise $Q(s,a)$ (exploitation).

Remarque : Un ϵ élevé favorise l'exploration, tandis qu'un faible ϵ favorise l'exploitation.

5 Implémentation en Python

5.1 Prédiction Monte Carlo (First-Visit)

Voici un exemple simple pour implémenter la prédiction Monte Carlo pour estimer $V(s)$ d'une politique donnée.

```
import numpy as np
from collections import defaultdict

# Définition de la politique et de l'environnement
states = [0, 1, 2, 3, 4]
actions = ['a', 'b']
policy = {s: np.random.choice(actions) for s in states}

# Générer un épisode aléatoire basé sur la politique
def generate_episode(policy):
    episode = []
    state = np.random.choice(states)
    while state != 4:  # état terminal
```

```
        action = policy[state]
        next_state = np.random.choice(states)
        reward = np.random.randn()  # Récompense aléatoire
        episode.append((state, action, reward))
        state = next_state
    return episode

# Prédiction Monte Carlo (First-Visit)
def monte_carlo_prediction_first_visit(policy, episodes, gamma=0.9):
    V = defaultdict(float)
    returns = defaultdict(list)
    for _ in range(episodes):
        episode = generate_episode(policy)
        G = 0
        visited = set()
        for t in reversed(range(len(episode))):
            state, _, reward = episode[t]
            G = gamma * G + reward
            if state not in visited:
                visited.add(state)
                returns[state].append(G)
                V[state] = np.mean(returns[state])
    return V

# Exécution
value_function = monte_carlo_prediction_first_visit(policy, episodes=1000)
print("Fonction de Valeur Estimée :")
for state, value in value_function.items():
    print(f"V({state}) = {value:.2f}")
```

5.2 Contrôle Monte Carlo (Sur-Politique, Every-Visit)

Cet exemple montre comment estimer $Q(s, a)$ et obtenir une politique optimale via une approche epsilon-greedy.

```
def monte_carlo_control_on_policy(episodes, gamma=0.9, epsilon=0.1):
    Q = defaultdict(lambda: np.zeros(len(actions)))
    policy = {s: np.random.choice(actions) for s in states}

    def epsilon_greedy_policy(state):
        if np.random.rand() < epsilon:
            return np.random.choice(actions)
        else:
            return actions[np.argmax(Q[state])]

    for _ in range(episodes):
```

```
        episode = []
        state = np.random.choice(states)
        while state != 4:  # tat terminal
            action = epsilon_greedy_policy(state)
            next_state = np.random.choice(states)
            reward = np.random.randn()  # R compense al atoire
            episode.append((state, action, reward))
            state = next_state

        G = 0
        for t in reversed(range(len(episode))):
            state, action, reward = episode[t]
            G = gamma * G + reward
            if (state, action) not in [(x[0], x[1]) for x in episode[:t]]:
                Q[state][actions.index(action)] += (G - Q[state][actions.index(a
                policy[state] = actions[np.argmax(Q[state])]

    return policy, Q

# Ex cution
optimal_policy, action_value_function = monte_carlo_control_on_policy(episodes=1
print("Politique Optimale :")
for state, action in optimal_policy.items():
    print(f" tat {state}: {action}")

print("Fonction de Valeur Action Estim e :")
for state, values in action_value_function.items():
    for action, value in zip(actions, values):
        print(f"Q({state}, {action}) = {value:.2f}")
```

Conclusion

Les méthodes de Monte Carlo sont des techniques puissantes pour l'apprentissage par renforcement, en particulier lorsque l'on ne dispose pas d'un modèle de l'environnement. Les algorithmes de Monte Carlo permettent d'apprendre des politiques optimales sur la base de retours observés dans des épisodes complets, offrant ainsi une alternative intéressante aux méthodes basées sur les différences temporelles.