UNIVERSITÉ TOULOUSE III PAUL SABATIER

TD 1 : MDP, Value et policy iteration

SRI - Apprentissage

Objectifs

1. Découvrir les markov decision processes.
2. Résolution des markov decision processes avec deux algorithmes itératifs.

1 Markov decision processes

Un processus de décision de Markov (MDP) est un cadre mathématique utilisé pour modéliser des situations de prise de décision dans lesquelles un agent est confronté à une séquence d'actions et d'états. Il consiste en un ensemble d'états, un ensemble d'actions et une fonction de transition qui définit la probabilité de passer d'un état à un autre en fonction de l'état actuel et de l'action choisie. Le MPD comprend également une fonction de récompense qui attribue une récompense ou un coût à chaque état et action.

Dans un MDP, l'objectif de l'agent est de trouver une politique, c'est-à-dire un ensemble de règles qui spécifient l'action à entreprendre dans chaque état, qui maximise la récompense cumulative attendue. L'agent suit la politique pour prendre des décisions et passe d'un état à l'autre jusqu'à ce qu'il atteigne un état terminal, auquel le processus s'arrête.

Les MDP sont largement utilisés en intelligence artificielle et en apprentissage automatique pour modéliser et résoudre des problèmes de prise de décision, comme la détermination de la stratégie de contrôle optimale d'un robot ou la stratégie d'investissement optimale d'un portefeuille. Ils peuvent également être utilisés pour modéliser et analyser des systèmes complexes dans des domaines tels que l'économie, la recherche opérationnelle et l'ingénierie.

Un processus de décision de Markov est un 4-tuple (S, A, P_a, R_a), où :

- S est un ensemble d'états appelé espace d'état,
- A est un ensemble d'actions appelé l'espace des actions (alternativement, A_s est l'ensemble des actions disponibles à partir de l'état s),

- $P_a(s, s') = \Pr\left(s_{t+1} = s' \mid s_t = s, a_t = a\right)$ est la probabilité que l'action a dans l'état s au temps t conduise à l'état s au temps $t + 1$,
- $R_a(s, s')$ est la récompense immédiate (ou la récompense immédiate attendue) reçue après la transition de l'état s à l'état s, grâce à l'action a.

Les espaces d'état et d'action peuvent être finis ou infinis, par exemple l'ensemble des nombres réels. Certains processus dont les espaces d'état et d'action sont discrets et infinis peuvent être réduits à ceux dont les espaces d'état et d'action sont finis.

Une fonction de politique π est une correspondance (potentiellement probabiliste) de l'espace d'état (S) à l'espace d'action (A).

Exercice

Considérons le scénario suivant : un agent tente d'atteindre un objectif dans un labyrinthe. Le labyrinthe est constitué d'une grille de cellules, dont certaines sont bloquées et ne peuvent être traversées. L'agent commence à une cellule de départ désignée et peut se déplacer vers l'une des quatre cellules adjacentes (gauche, droite, haut, bas) en un pas de temps. Le but est situé dans une cellule de préalablement désignée.

1. Définissez les états, les actions et les transitions de ce MDP.
2. Définissez la fonction de récompense pour ce MDP.
3. Proposer une fonction `transition(state,action)` et une fonction `reward(state)` qui encode le passage d'un état à l'autre et la récompense obtenue.

1. Les **états** de ce MDP sont les cellules du labyrinthe, qui peuvent être représentées par un tuple de deux entiers (i, j) indiquant la ligne et la colonne de la cellule. Les **actions** sont les quatre directions dans lesquelles l'agent peut se déplacer (gauche, droite, haut, bas). Les **transitions** peuvent être représentées par une fonction qui prend un état et une action en entrée et renvoie l'état suivant. Par exemple, la fonction de transition pourrait être définie comme suit :

```
def transition(state, action):
    i, j = state
    if action == "left":
        return (i, max(0, j-1))
    elif action == "right":
        return (i, min(n-1, j+1))
    elif action == "up":
        return (max(0, i-1), j)
    elif action == "down":
        return (min(m-1, i+1), j)
```

Cette fonction de transition suppose que le labyrinthe comporte m lignes et n colonnes, et que l'agent ne peut pas se déplacer en diagonale ni sortir du labyrinthe.

La fonction de récompense pour ce MPD peut être définie comme suit :

```
def reward(state):
    if state == goal:
```

```
        return 10
    else:
        return -1
```

Cette fonction de récompense donne une récompense de 10 pour l'atteinte de la cellule cible, et une pénalité sinon.

2 Résolution pour les MDP : value iteration

Il existe plusieurs méthodes de résolution pour les processus de décision de Markov (MPD) qui peuvent être utilisées pour trouver la politique optimale. Cette politique est un ensemble de règles qui spécifient l'action à entreprendre dans chaque état pour maximiser la récompense cumulative attendue.

Une première méthode est l'**itération de la valeur**, qui consiste à mettre à jour de manière itérative la valeur de chaque état en fonction de la récompense attendue pour chaque action possible. La **valeur d'un état** est la récompense cumulée attendue en suivant la politique optimale de cet état. Pour calculer la valeur d'un état, l'algorithme explore la récompense attendue pour chaque action, ainsi que les valeurs des états résultants, et sélectionne l'action qui maximise la récompense cumulative. Ce processus est répété jusqu'à ce que les valeurs des états convergent, auquel cas la politique optimale peut être dérivée des valeurs des états.

Algorithme d'itération de valeurs pour résoudre les MDP :

1. Initialiser les valeurs de tous les états à des valeurs arbitraires (par exemple, zéro).
2. Itérer sur tous les états du MDP :
   (a) Pour chaque état, considérez toutes les actions possibles et calculez la récompense attendue pour chaque action.
   (b) Sélectionnez l'action qui maximise la récompense attendue et mettez à jour la valeur de l'état pour qu'elle corresponde à la récompense attendue pour l'action sélectionnée.
3. Répétez le processus jusqu'à ce que les valeurs des états convergent.
4. Une fois que les valeurs des états ont convergé, la politique optimale peut être dérivée en sélectionnant l'action qui maximise la valeur de chaque état.

L'algorithme d'itération des valeurs fonctionne en améliorant de manière itérative les estimations des valeurs des états jusqu'à ce qu'elles convergent vers les valeurs optimales. Pour ce faire, il utilise l'équation de Bellman, qui exprime la valeur d'un état comme la récompense attendue pour l'action actuelle plus la valeur actualisée de l'état suivant. L'équation de Bellman pour une politique fixée peut s'écrire sous la forme :

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s').$$

Exercice

Reprendre l'exemple précédent. L'agent commence au milieu de la grille et peut se déplacer vers n'importe quelle cellule adjacente en un pas de temps. Le but du jeu est d'atteindre la cellule marquée d'un "G", qui est située dans le coin supérieur droit de la grille. La récompense pour atteindre le but est +10, et la récompense pour toutes les autres cellules est -1. L'agent ne peut se déplacer que vers des cellules situées à l'intérieur de la grille, et il ne peut pas se déplacer en diagonale.

```
  0  1  2  3  4

0 -1  -1  -1  -1

1 -1  -1  -1  -1

2 -1  -1  S  -1

3 -1  -1  -1  -1

4 -1  -1  -1  G
```

1. Implémentez l'algorithme d'itération de valeur pour trouver la politique optimale pour ce MDP. Vous pouvez supposer que la politique initiale consiste à se déplacer aléatoirement dans l'une des quatre directions avec une probabilité égale.
2. Exécutez l'algorithme d'itération de valeur et affichez la politique résultante à chaque itération.
3. Expliquez ce que fait la politique résultante et pourquoi elle est optimale.

```python
import numpy as np

# Define the MDP

# Set of states
states = [(i, j) for i in range(5) for j in range(5)]

# Set of actions
actions = ["left", "right", "up", "down"]

# Transition function
def transition(state, action):
    i, j = state
    if action == "left":
        return (i, max(0, j-1))
    elif action == "right":
        return (i, min(4, j+1))
    elif action == "up":
        return (max(0, i-1), j)
    elif action == "down":
        return (min(4, i+1), j)
```

```
# Reward function
def reward(state):
    if state == (4, 3):
        return 10
    else:
        return -1

# Initial policy (random action with equal probability)
policy = {state: {a: 1/len(actions) for a in actions} for state in states}


def value_iteration():
    # Initialize the values of all states to zero
    values = {state: 0 for state in states}

    # Set the discount factor
    gamma = 0.9

    # Repeat until the values converge
    while True:
        delta = 0
        for state in states:
            old_value = values[state]
            new_value = max(reward(state) + gamma * values[transition(state,...
                     ...action)] for action in actions)
            values[state] = new_value
            delta = max(delta, abs(new_value - old_value))
        if delta < 1e-9:
            break

    # Derive the optimal policy from the values
    policy = {state: max(actions, key=lambda a: reward(state) + gamma *...
         ...values[transition(state, a)]) for state in states}

    return policy, values

value_iteration()
```

```
({(0, 0): 'right',
  (0, 1): 'right',
  (0, 2): 'right',
  (0, 3): 'down',
  (0, 4): 'down',
  (1, 0): 'right',
  (1, 1): 'right',
  (1, 2): 'right',
  (1, 3): 'down',
  (1, 4): 'down',
  (2, 0): 'right',
  (2, 1): 'right',
  (2, 2): 'right',
```

```
  (2, 3): 'down',
  (2, 4): 'down',
  (3, 0): 'right',
  (3, 1): 'right',
  (3, 2): 'right',
  (3, 3): 'down',
  (3, 4): 'down',
  (4, 0): 'right',
  (4, 1): 'right',
  (4, 2): 'right',
  (4, 3): 'down',
  (4, 4): 'left'},
 {(0, 0): 42.61265899142269,
  (0, 1): 48.45850999142269,
  (0, 2): 54.95389999142269,
  (0, 3): 62.17099999142268,
  (0, 4): 54.95389999228041,
  (1, 0): 48.45850999142269,
  (1, 1): 54.95389999142269,
  (1, 2): 62.17099999142268,
  (1, 3): 70.1899999914227,
  (1, 4): 62.17099999228043,
  (2, 0): 54.95389999142269,
  (2, 1): 62.17099999142268,
  (2, 2): 70.1899999914227,
  (2, 3): 79.09999999142269,
  (2, 4): 70.18999999228042,
  (3, 0): 62.17099999142268,
  (3, 1): 70.1899999914227,
  (3, 2): 79.09999999142269,
  (3, 3): 88.9999999914227,
  (3, 4): 79.09999999228043,
  (4, 0): 70.1899999914227,
  (4, 1): 79.09999999142269,
  (4, 2): 88.9999999914227,
  (4, 3): 99.9999999914227,
  (4, 4): 88.99999999228044})
```

3 Résolution pour les MDP : policy iteration

Une autre méthode est l'**itération de la politique**, qui consiste à améliorer itérativement la politique en alternant les étapes d'évaluation et d'amélioration de la politique. Dans l'évaluation de la politique, la valeur de chaque état sous la politique actuelle est calculée en utilisant l'équation de Bellman, qui exprime la valeur d'un état comme la récompense attendue pour l'action actuelle plus la valeur actualisée de l'état suivant. Dans l'amélioration de la politique, la politique actuelle est améliorée en sélectionnant l'action qui maximise la valeur de chaque état. Ce processus est répété jusqu'à ce que la politique converge vers la politique optimale.

Algorithme d'itération de policy pour résoudre les MDP :

1. Initialiser la politique à une politique arbitraire (par exemple, en sélectionnant une action aléatoire dans chaque état).
2. Répétez les étapes suivantes jusqu'à ce que la politique converge :
   (a) Effectuer l'évaluation de la politique :
      i. Initialiser les valeurs de tous les états à des valeurs arbitraires (par exemple, zéro).
      ii. Itérer sur tous les états du MDP :
         A. Pour chaque état, calculez la valeur de l'état sous la politique actuelle en utilisant l'équation de Bellman, qui exprime la valeur d'un état comme la récompense attendue pour l'action actuelle plus la valeur actualisée de l'état suivant.
   (b) Améliorer la politique :
      i. Pour chaque état, sélectionnez l'action qui maximise la valeur de l'état.
3. Une fois que la politique a convergé, la politique optimale a été trouvée.

L'algorithme d'itération de la politique fonctionne en alternant les étapes d'évaluation et d'amélioration de la politique. Dans l'évaluation de la politique, les valeurs des états sous la politique actuelle sont calculées en utilisant l'équation de Bellman. Dans l'amélioration de la politique, la politique actuelle est améliorée en sélectionnant l'action qui maximise la valeur de chaque état. Ces étapes sont répétées jusqu'à ce que la politique converge vers la politique optimale.

Exercice

1. Implémentez l'algorithme policy iteration pour trouver la politique optimale. Vous pouvez supposer que la politique initiale consiste à se déplacer aléatoirement dans l'une des quatre directions avec une probabilité égale.
2. Exécutez l'algorithme et affichez la politique résultante à chaque itération.
3. Expliquez ce que fait la politique résultante et pourquoi elle est optimale.

Cette fonction initialise les valeurs de tous les états à zéro et fixe le facteur d'actualisation (gamma) à 0,9. Elle itère ensuite sur tous les états et met à jour la valeur de chaque état en utilisant l'équation de Bellman, qui exprime la valeur d'un état comme la récompense attendue pour l'action actuelle plus la valeur actualisée de l'état suivant. L'étape de mise à jour des valeurs est répétée jusqu'à ce que les valeurs convergent, ce qui est indiqué par le fait que la variable delta est inférieure à un petit seuil (1e-9). Une fois que les valeurs ont convergé, la politique optimale est dérivée des valeurs de la manière suivante

```
def policy_iteration():
    # Initialize the values of all states to zero
    values = {state: 0 for state in states}

    # Repeat until the policy converges
    while True:
```

```
        # Perform policy evaluation
        for state in states:
            value = 0
            for action, prob in policy[state].items():
                next_state = transition(state, action)
                value += prob * (reward(state) + values[next_state])
            values[state] = value

        # Perform policy improvement
        unchanged = True
        for state in states:
            old_action = max(policy[state], key=policy[state].get)
            new_action = max(actions, key=lambda a: reward(state) + ...
                ...values[transition(state, a)])
            if old_action != new_action:
                policy[state] = {a: 1/len(actions) if ...
                    ...a == new_action else 0 for a in actions}
                unchanged = False

        # If the policy has not changed, return the policy and values
        if unchanged:
            return policy, values

# Run

policy_iteration()
({(0, 0): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (0, 1): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (0, 2): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (0, 3): {'left': 0, 'right': 0, 'up': 0, 'down': 0.25},
  (0, 4): {'left': 0.25, 'right': 0, 'up': 0, 'down': 0},
  (1, 0): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (1, 1): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (1, 2): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (1, 3): {'left': 0, 'right': 0, 'up': 0, 'down': 0.25},
  (1, 4): {'left': 0.25, 'right': 0, 'up': 0, 'down': 0},
  (2, 0): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (2, 1): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (2, 2): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (2, 3): {'left': 0, 'right': 0, 'up': 0, 'down': 0.25},
  (2, 4): {'left': 0.25, 'right': 0, 'up': 0, 'down': 0},
  (3, 0): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (3, 1): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (3, 2): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (3, 3): {'left': 0, 'right': 0, 'up': 0, 'down': 0.25},
  (3, 4): {'left': 0.25, 'right': 0, 'up': 0, 'down': 0},
  (4, 0): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (4, 1): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (4, 2): {'left': 0, 'right': 0.25, 'up': 0, 'down': 0},
  (4, 3): {'left': 0, 'right': 0, 'up': 0, 'down': 0.25},
  (4, 4): {'left': 0.25, 'right': 0.25, 'up': 0.25, 'down': 0.25}},
 {(0, 0): -0.3327617608010769,
```

```
(0, 1): -0.3320903740823269,
(0, 2): -0.3294048272073269,
(0, 3): -0.3186626397073269,
(0, 4): -0.3296656599268317,
(1, 0): -0.3320903740823269,
(1, 1): -0.3294048272073269,
(1, 2): -0.3186626397073269,
(1, 3): -0.2756938897073269,
(1, 4): -0.3189234724268317,
(2, 0): -0.3294048272073269,
(2, 1): -0.3186626397073269,
(2, 2): -0.2756938897073269,
(2, 3): -0.10381888970732689,
(2, 4): -0.2759547224268317,
(3, 0): -0.3186626397073269,
(3, 1): -0.2756938897073269,
(3, 2): -0.10381888970732689,
(3, 3): 0.5836811102926731,
(3, 4): -0.10407972242683172,
(4, 0): -0.2756938897073269,
(4, 1): -0.10381888970732689,
(4, 2): 0.5836811102926731,
(4, 3): 3.333681110292673,
(4, 4): -0.36383457598276436})
```

4 Différences entre policy iteration et value iteration

L'itération de valeur et l'itération de politique sont deux algorithmes pour résoudre les MDP. Les deux algorithmes sont utilisés pour trouver la politique optimale, qui est un ensemble de règles spécifiant l'action à entreprendre dans chaque état pour maximiser la récompense cumulative attendue.

La principale différence entre l'itération de valeur et l'itération de politique est la façon dont ils abordent le problème de la recherche de la politique optimale.

L'itération de valeur est un algorithme itératif qui consiste à mettre à jour la valeur de chaque état en fonction de la récompense attendue pour chaque action possible. La valeur d'un état est la récompense cumulée attendue en suivant la politique optimale de cet état. Pour calculer la valeur d'un état, l'algorithme examine la récompense attendue pour chaque action, ainsi que les valeurs des états résultants, et sélectionne l'action qui maximise la récompense cumulative attendue. Ce processus est répété jusqu'à ce que les valeurs des états convergent, auquel cas la politique optimale peut être dérivée des valeurs des états.

L'itération de la politique est également un algorithme itératif, mais il implique une alternance entre les étapes d'évaluation et d'amélioration de la politique. Dans l'évaluation de la politique, la valeur de chaque état sous la politique actuelle est calculée en utilisant l'équation de Bellman, qui exprime la valeur d'un état comme la récompense attendue pour l'action actuelle plus la valeur actualisée de l'état suivant.

Dans l'amélioration de la politique, la politique actuelle est améliorée en sélectionnant l'action qui maximise la valeur de chaque état. Ce processus est répété jusqu'à ce que la politique converge vers la politique optimale.

Globalement, l'itération des valeurs est généralement plus rapide que l'itération des politiques, mais elle peut nécessiter plus de mémoire pour stocker les valeurs des états. L'itération de politique est plus lente mais nécessite moins de mémoire, car elle ne doit stocker que la politique actuelle. Les deux algorithmes sont utilisés dans différentes situations, en fonction des caractéristiques du MDP et des ressources informatiques disponibles.