یادگیری تقویتی- مینی پروژه اول

سحر محمدی ۴۰۲۱۳۹۰۱۰۹

۱. پیاده سازی دو الگوریتم policy iteration and value iteration و بررسی در محیط Frozen lake

Policy iteration شامل دو بخش ارزیابی سیاست و بهبود سیاست است که در دو گام جداگانه انجام می‌شوند.

اما در value iteration دو گام جدا نیست.

Optimal policy and value function خروجی هر دو الگوریتم policy iteration and value iteration یکسان است.

ساختار Policy تعریف شده در الگوریتم به صورت یک ماتریس است که سطرهای آن شماره استیت و ستون‌های آن شماره اکشن را مشخص می‌کند. که اگر کارگزار در فلان استیت بود، اگر شتون شماره ۰ برابر یک باشد یعنی اکشن ۰ را انجام می‌دهد.

```
Optimal Policy:
[[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]]
```

برای اینکه بیشتر قابل فهم باشد سیاست را به شکل دیگری چاپ می‌کنیم.

```
['d' 'r' 'd' 'l']
['d' 'l' 'd' 'l']
['r' 'd' 'd' 'l']
['l' 'r' 'r' 'l']
```

که معادل خانه‌های مدل است و نشان می‌دهد اگر در هر استیت باشد اکش راست چپ یا بالا یا پایین را انجام می‌دهد.

ارزیابی هم طبق مقادیر بدست آمده از رابطه بلمن و پاداش و ضریب کاهش، چاپ شده است.

```
Optimal Value Function:
[0.03125 0.0625  0.125   0.0625 ]
[0.0625 0.     0.25   0.    ]
[0.125 0.25  0.5   0.   ]
[0.  0.5 1.  0. ]
```

- بررسی تاثیر گاما

در گاما بزرگتر و نزدیک به ۱ که حالت farsighted هست، با gamma=0.9 از همان استیت اول، ارزش ۰.۵۹ بدست آمده و آینده را هر چند دور که بالاخره به goal و reward می‌رسد را در نظر گرفته و در استیت شماره ۱۳ هم ۰.۹ بدست آمده که خیلی نزدیک به یک است که در استیت بعدی ریوارد یک گرفته می‌شود.

```
Optimal Policy:
[[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]]
['d' 'r' 'd' 'l']
['d' 'l' 'd' 'l']
['r' 'd' 'd' 'l']
['l' 'r' 'r' 'l']

Optimal Value Function:
[0.59049 0.6561  0.729   0.6561 ]
[0.6561 0.     0.81   0.    ]
[0.729 0.81  0.9   0.   ]
[0.  0.9 1.  0. ]
```

و اما در گاما کوچکتر و نزدیک به صفر که حالت myopic می‌باشد، با gamma=0.1 ارزش‌ها بسیار کوچک شده طوری که در استیت اول برابر 10^{-5} و در استیت شماره ۱۳ ارزش برابر ۰.۱ است. که نشان می‌دهد در این حالت بیشتر به ریوارد نقد ارزش می‌دهد تا چیزی که قرار است در آینده گرفته شود.

```
Optimal Policy:
[[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]]
['d' 'r' 'd' 'l']
['d' 'l' 'd' 'l']
['r' 'd' 'd' 'l']
['l' 'r' 'r' 'l']

Optimal Value Function:
[1.e-05 1.e-04 1.e-03 1.e-04]
[0.0001 0.    0.01  0.   ]
[0.001 0.01  0.1   0.   ]
[0.  0.1 1.  0. ]
```

اما در هر دو حالت ارزش استیت شماره ۱۴ برابر ۱ است.

- بررسی تاثیر غیر قطعی کردن محیط

در محیط غیر قطعی، ۳۳ درصد احتمال دارد agent در جهت اکشنی که مد نظر است حرکت کند و ۶۶ درصد در جهت‌های عمود بر آن. یعنی اگر قرار باشد پایین برود ۳۳ درصد احتمال دارد پایین، ۳۳ درصد احتمال دارد راست و ۳۳ درصد هم احتمال دارد چپ برود.

```
# prob, next_state, reward, done
#[state][action]
env.P[6][1]
```

output

```
[(0.3333333333333333, 5, 0.0, True),
 (0.3333333333333333, 10, 0.0, False),
 (0.3333333333333333, 7, 0.0, True)]
```

و اما پالیسی بهینه‌ای که توسط الگوریتم پیدا شده بسیار جالب است.

طوری که مثلا در استیت شماره ۴ که سمت راست آن یک چاله وجود دارد، پالیسی اکشن چپ را انتخاب کرده که احتمال راست رفتن agent صفر باشد فقط یا بالا برود یا چپ که چون سمت چپ آن دیوار است در همین

استیت می‌ماند و یا پایین می‌رود. اگر شانسی همان چپ رفتن انتخاب شود انقدر چپ می‌رود تا بالاخره یک بار پایین برود.

و همین طور در استیتی که پایین آن چاله است پالیسی همواره بالا رفتن انتخاب شده.

```
Optimal Policy:
[[1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]]
['l' 'u' 'l' 'u']
['l' 'l' 'l' 'l']
['u' 'd' 'l' 'l']
['l' 'r' 'd' 'l']

Optimal Value Function:
[0.06889086 0.06141454 0.07440974 0.0558073 ]
[0.09185451 0.         0.1122082  0.        ]
[0.14543633 0.24749694 0.29961758 0.        ]
[0.         0.37993589 0.63902014 0.        ]
```

۲. پیاده سازی دو الگوریتم sarsa and q-learning و بررسی در محیط cliff walking

هر دو با epsilon greedy و طبق TD پیاده سازی می‌شوند. و اما در q-learning در max TD-target گرفته می‌شود.

در sarsa با مشخصات

```
num_episodes = 1000
alpha = 0.1
gamma = 1.0
epsilon = 0.1
```

همان طور که انتظار می‌رفت مسیر امن تر و دورتر از لبه پرتگاه انتخاب شده.

```
# 0: Move up
# 1: Move right
# 2: Move down
# 3: Move left
```
output
Learned Optimal Policy:
[[1 1 1 1 1 1 2 1 1 1 1 2]
 [1 1 1 1 1 1 1 1 1 1 1 2]
 [0 0 0 0 0 1 0 1 1 3 1 2]
 [0 0 0 0 0 0 0 0 0 0 0 0]]

با تعداد اپیزود ۲۰۰۰ به نتیجه حتی امن تر هم رسیده طوری که اکثر استیت‌های کنا پرتگاه صفر شده‌اند که یعنی بالا برود و از پرتگاه دور شود.

```
Learned Optimal Policy:
[[1  1  1  1  1  1  1  1  1  1  1  2]
 [1  1  1  1  1  1  1  1  1  1  1  2]
 [0  0  0  0  0  0  0  1  0  0  1  2]
 [0  0  0  0  0  0  0  0  0  0  0  0]]
```

اگر نرخ یادگیری را کوچکتر کنیم، سیاست نتیجه زیاد مطلوب نیست و به نظر می‌رسد که بهینه نباشد، چون با توجه به قطعی بودن محیط اینکه در یک استیت بالا برود و در استیت بعدی پایین، حلقه است و طبق این سیاست هیچ وقت به جواب نخواهیم رسید.

```
Learned Optimal Policy:
[[2  3  2  0  2  1  2  2  1  0  2  2]
 [0  1  1  3  2  3  1  2  3  3  1  0]
 [0  1  1  0  3  3  1  1  0  3  0  3]
 [0  0  0  0  0  0  0  0  0  0  0  0]]
```

و اما با کوچکتر کردن اپسیلون و به عبارتی کاشتن از اکتشاف، سیاست نتیجه شبیه جواب q-learning شد، همان طور که انتظار می‌رفت.

```
num_episodes = 1000
alpha = 0.1
gamma = 1.0
epsilon = 0.0001
# Run SARSA algorithm
Q = sarsa(env, num_episodes, alpha, gamma, epsilon)
# 0: Move up
# 1: Move right
# 2: Move down
# 3: Move left
```
output
Learned Optimal Policy:
[[1 1 1 1 1 2 1 0 1 1 2 0]

```
[0 3 2 1 1 1 1 2 1 2 1 2]
[1 1 1 1 1 1 1 1 1 1 1 2]
[0 0 0 0 0 0 0 0 0 0 0 0]]
```

با کاهش گاما هم مسیر امن‌تر و دورتر از صخره انتخاب شد.

و اما در q-learning

```
num_episodes = 1000
alpha = 0.1
gamma = 1.0
epsilon = 0.1
# Run Q-learning algorithm
Q = q_learning(env, num_episodes, alpha, gamma, epsilon)
# 0: Move up
# 1: Move right
# 2: Move down
# 3: Move left
```

output

```
Learned Optimal Policy:
[[1 3 0 1 3 1 1 1 2 1 1 2]
 [0 1 2 1 1 1 1 1 1 1 1 2]
 [1 1 1 1 1 1 1 1 1 1 1 2]
 [0 0 0 0 0 0 0 0 0 0 0 0]]
```

۳. بررسی q-learning and double q-learning در محیط frozen lake

Q-learning از ارزش Q بیشینه برای عمل بعدی استفاده می‌کند بدون درنظر گرفتن عملی که در واقعیت انتخاب شده است. و اما Double Q-learning برای انتخاب عمل بعدی، از دو تابع Q مجزا استفاده می‌کند و به صورت دوره‌ای آن‌ها را به‌روز می‌کند.

در این محیط پس از چندین بار امتحان کردن برای تنظیم پارامترها، چون در تلاش‌های اولیه خروجی سیاست تمام صفر می‌داد که یعنی سیاست یاد گرفته نشده بود، به نتیجه رسیدیم.

```
# Create the Frozen Lake environment
env = gym.make('FrozenLake-v1',is_slippery=False)
# Define hyperparameters
num_episodes = 1000
alpha = 0.001
gamma = 0.5
epsilon = 1
# Run Q-learning algorithm
```

```
Q = q_learning(env, num_episodes, alpha, gamma, epsilon)
# 0: Move left
# 1: Move down
# 2: Move right
# 3: Move up
```

output

Learned Optimal Policy:
[2 2 1 0]
[1 0 1 0]
[2 2 1 0]
[0 2 2 0]

اینجا هم با کاهش نرخ یادگیری به نتیجه غیر مطلوب رسیدیم.

و اما برای محیط غیرقطعی هم سیاست بهینه پیدا شد.

```
# Create the Frozen Lake environment
env = gym.make('FrozenLake-v1',is_slippery=True)
# Define hyperparameters
num_episodes = 1000
alpha = 0.1
gamma = 1
epsilon = 1
# Run Q-learning algorithm
Q = q_learning(env, num_episodes, alpha, gamma, epsilon)
# 0: Move left
# 1: Move down
# 2: Move right
# 3: Move up
```

output

Learned Optimal Policy:
[1 3 0 3]
[0 0 0 0]
[3 1 2 0]
[0 2 3 0]

طوری که اکشن‌های را انتخاب می‌کند که با هیچ احتمالی کارگزار را در چاله نیندازد.

Double q-learning هم با تنظیمات زیر می‌تواند سیاست بهینه را پیدا کند.

```
# Create the Frozen Lake environment
env = gym.make('FrozenLake-v1',is_slippery=False)
# Define hyperparameters
num_episodes = 1000
alpha = 0.1
gamma = 0.9
epsilon = 1
```

```
# Run Double Q-learning algorithm
Q1, Q2 = double_q_learning(env, num_episodes, alpha, gamma, epsilon)
# Combine Q1 and Q2 to get the learned Q-value function
Q = (Q1 + Q2) / 2
# 0: Move left
# 1: Move down
# 2: Move right
# 3: Move up
```

output

Learned Optimal Policy (Double Q-learning):
[2 2 1 0]
[1 0 1 0]
[2 2 1 0]
[0 2 2 0]

تعداد اپیزود هم مهم است. مثلا با تعداد اپیزود ۱۰۰ سایست بهینه پیدا نمی‌شود.

البته در محیط غیر قطعی با تنظیمات مختلف double q-learning نتوانست سیاست بهینه را پیدا کند، سیاستی که کاملا از چاله‌ها پرهیز کند.

```
Learned Optimal Policy (Double Q-learning):
[1 3 0 0]
[0 0 2 0]
[3 1 2 0]
[0 3 3 0]
```

مثلا در اینجا در استیت شماره ۱۳ باید راست برود که بالا انتخاب شده و با بالا رفتن احتمال دارد در چاله سمت چپ بیفتد.

```
Learned Optimal Policy (Double Q-learning):
[0 3 0 3]
[0 0 2 0]
[3 2 1 0]
[0 2 1 0]
```

یا در اینجا در استیت شماره ۹ نباید راست برود که احتمال چاله بالایی موجود باشد.