دانشکده مهندسی برق

یادگیری تقویتی در کنترل

تمرین چهارم: برنامه ریزی پویا و مونت کارلو

استاد: دکتر سعید شمقدری

دانشجو: سیده ستاره خسروی

پائیز ۱۴۰۳

چکیده

در تمرین سری چهارم یادگیری تقویتی در کنترل با ۶ سوال از مباحث برنامه ریزی پویا و مونت کارلو مواجه هستیم، که در هر فصل به سوال و یا سوالات مطرح شده پاسخ داده شده است.

واژه‌های کلیدی: یادگیری تقویتی، برنامه ریزی پویا، مونت کارلو

فهرست مطالب

فهرست تصاویر و نمودارها

عنوان صفحه

فصل ۱: مفاهیم برنامه ریزی پویا و مونت کارلو

۱.۱ مقدمه

در این فصل به ۴ سوال اول تمرین پرداخته می‌شود.

۱.۲ سوال اول

صورت سوال: تفاوت‌های الگوریتم‌های Asynchronous DP و Synchronous DP در چیست؟

پاسخ:

این دو الگوریتم در مواردی چون الگوری به روز رسانی، نحوه پیاده سازی، همگرایی و کارایی محاسباتی تفاوت دارند.

۱) الگوی به روزرسانی: در ADP، تمام حالت‌ها را به طور همزمان در هر تکرار به‌روز می‌کند. نیاز به نگهداری دو نسخه از توابع ارزش دارد، یکی برای تکرار فعلی و یکی برای ذخیره مقادیر جدید. مقادیر فقط پس از ارزیابی همه حالت‌ها به‌روز می‌شوند. در SDP، حالت‌ها را یکی یکی به‌روز می‌کند و فوراً از مقادیر جدید برای به‌روزرسانی‌های بعدی استفاده می‌کند. فقط به نگهداری یک نسخه از تابع ارزش نیاز دارد زیرا به‌روزرسانی‌ها inplace انجام می‌شوند.

۲) کارایی محاسباتی: ADP به حافظه بیشتری نیاز دارد (به دلیل نگهداری دو نسخه) و ممکن است محاسبات غیرضروری انجام دهد با به‌روزرسانی همه حالت‌ها حتی اگر برخی حالت‌ها قبلاً همگرا شده باشند. SDP معمولاً کارآمدتر است زیرا از حافظه کمتری استفاده می‌کند، می‌تواند اولویت‌بندی کند که کدام حالت‌ها به‌روز شوند، می‌تواند فوراً از تخمین‌های جدید ارزش استفاده کند، می‌تواند به‌روزرسانی حالت‌هایی که قبلاً همگرا شده‌اند را رد کند.

۳) همگرایی: هر دو روش تحت شرایط مناسب به مقادیر بهینه همگرا می‌شوند، SDP، می‌تواند سریع‌تر همگرا شود زیرا فوراً از جدیدترین تخمین‌های ارزش استفاده می‌کند.

۴) پیاده سازی: ADP، پیاده‌سازی ساده‌تری دارد زیرا همه به‌روزرسانی‌ها به صورت موازی انجام می‌شوند.

۱.۳ سوال دوم

صورت سوال: از رابطه ۷-۵ رابطه‌ی ۸-۵ را بدست آورید.

پاسخ:

رابطه‌ی ۷-۵ مطابق زیر است:

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \qquad n \geq 2,$$

همچنین رابطه‌ی ۸-۵ نیز به شرح زیر است:

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n}\Big[G_n - V_n\Big], \qquad n \geq 1,$$

مشابه آنچه در فصل دوم داشتیم، عمل می‌کنیم. ابتدا فرض می‌کنیم که $C_0 = 0$ است و $C_n = \sum_{k=1}^{n} W_k$ می‌باشد. برای V_{n+1} داریم:

$$V_{n+1} = \frac{\sum_{k=1}^{n} W_k G_k}{C_n}$$

سپس ساده سازی ها را انجام می‌دهیم: (تبدیلات در خط دوم وسوم با توجه به معادله ۷-۵ انجام شده)

$$V_{n+1} C_n = \sum_{k=1}^{n} W_k G_k$$

$$V_{n+1} C_n = W_n G_n + \sum_{k=1}^{n-1} W_k G_k$$

$$V_{n+1} C_n = W_n G_n + V_n \sum_{k=1}^{n-1} W_k$$

$$V_{n+1} C_n = W_n G_n + V_n C_{n-1}$$

$$V_{n+1} C_n = W_n G_n + V_n \left(C_n - W_n\right)$$

در نهایت خواهیم داشت:

$$V_{n+1} = V_n + \frac{W_n}{C_n}\left[G_n - V_n\right]$$

۱.۴ سوال سوم

صورت سوال:

۹ مرحله (episode) زیر را در نظر بگیرید که توسط یک فرایند مارکوف ناشناخته ایجاد شده‌اند و در آن A و B حالت‌ها و اعداد نشان‌دهنده پاداش ها هستند.

A,0,B,6	B,0,A,2,B,2	B,6
A,3	B,0,A,3	B,2
A,2,B,0,A,3	B,2	B,6

الف- تابع مقدار متناظر به حالات A و B را با استفاده از این مجموعه داده‌ها توسط روش مونت کارلو در اولین بازدید از مجموعه دریافت می‌شود (با فرض $\gamma = 0.8$)، بدست آورید.

ب- چنانچه یک مدل maximum-likelihood از فرآیند پاداش مارکوف بر اساس مراحل فوق (و فقط این مراحل) تشکیل دهید، آن چه خواهد بود؟

پاسخ:

بازگشت را برای هر state محاسبه می‌کنیم، ابتدا برای A state، داریم:

برای A :

A,0,B,6 → $0 + 0,8(6) = 4,8$

A,3 → 3

A,2,B,0,A,3 → $2 + 0,8(0 + 0,8(3)) = 3,92$

B,0,A,2,B,2 → $2 + 0,8(2) = 3,6$

B,0,A,3 → 3

Averages $(4,8 + 3 + 3,92 + 3,6 + 3) \times \frac{1}{5} = 3,664$

برای B نیز داریم:

برای B:

$A, 0, B, 6 \rightarrow 6$

$A, 2, B, 0, A, 3 \rightarrow 0 + 3 \times 0.8 = 2.4$

$B, 0, A, 2, B, 2 \rightarrow 0 + 0.8(2 + 0.8(2)) = 3.28$

$B, 0, A, 3 \rightarrow 0 + 0.8(3) = 2.4$

$B, 2 \rightarrow 2$

$B, 6 \rightarrow 6$ Average $(6 + 2.4 + 3.24 + 2.4 + 2 + 6 + 2 + 6)/8 = 3.76$

$B, 2 \rightarrow 2$

$B, 6 \rightarrow 6$

و درنهایت برای بخش دوم سوال خواهیم داشت:

تست آخر:

$A \rightarrow B \rightarrow 0$ $\frac{1}{4}$

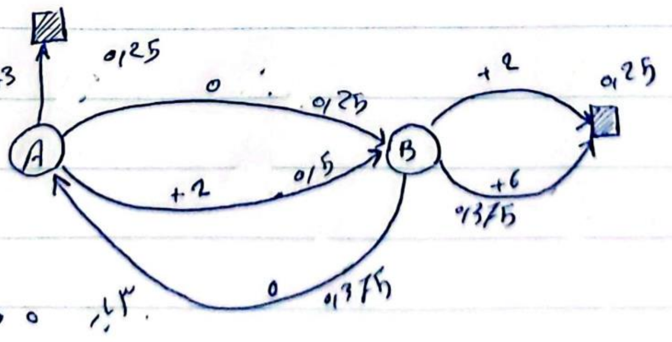
$A \rightarrow B \rightarrow 2$ $\frac{2}{4}$ ۲ بار

$A \rightarrow +3$ ۱ بار $\frac{1}{4}$

$B \rightarrow A \rightarrow 0$ ۳ بار

$B \rightarrow 2$ ۳ بار

$B \rightarrow 6$ ۳ بار



۱.۵ سوال چهارم

صورت سوال:

سوال ۴.۵ کتاب: چگونه الگوریتم تکرار سیاست (Policy iteration) با استفاده از مقادیر عمل-ارزش q تعریف می‌شود؟ یک الگوریتم کامل برای محاسبه q^*، مشابه الگوریتم فوق که برای محاسبه v^* ارائه شده است، پیشنهاد دهید.

پاسخ:

الگوریتم تکرار سیاست به صورت زیر است:

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\qquad v \leftarrow V(s)$
   $\qquad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

شکل ۱: الگوریتم تکرار سیاست

الگوریتم جدید را بر اساس الگوریتم قبلی به صورت زیر می‌نویسیم:

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $q(s,a)$ and $\pi(s)$ initialised arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each state-action pair (s,a), $s \in \mathcal{S}, a \in \mathcal{A}$:
   $q \leftarrow Q(s,a)$
   $Q(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s')Q(s',a')\right]$
   $\Delta \leftarrow max(\Delta, |q - Q(s,a)|)$

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   Loop for each state-action pair (s,a), s $\in \mathcal{S}, a \in \mathcal{A}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s',a')\right]$
   if old-action $\neq \pi(s)$, then policy-stable $\leftarrow$ *False*
   If policy-stable, then stop and return Q $\approx q_* and \pi \approx \pi_*$; else return to 2.

شکل ۲: الگوریتم تکرار سیاست جدید

درواقع تنها تفاوتی که ایجاد شد فرمولاسیون Q بود که جایگزین v کردیم.

فصل ۲: شبیه سازی برنامه ریزی پویا و مونت کارلو

۲.۱ مقدمه

در این بخش به پاسخ سوالات پنجم و ششم پرداخته می‌شود.

۲.۲ سوال پنجم

صورت سوال:

Gridworld زیر را در نظر بگیرید (اعداد شماره خانه‌ها هستند.)

۱	۲	۳	۴
۵	۶	تله۷	۸
۹	۱۰	۱۱	۱۲
۱۳	۱۴	۱۵	هدف

چهار عمل چپ، بالا، راست و پایین امکان‌پذیر هستند و هر کدام عامل را به صورت قطعی به همان سمت جا به جا می‌کنند. اعمالی که عامل را بیرون از صفحه می‌برند تاثیری ندارند. رسیدن به خانه هدف پاداش مثبت ده و رفتن به خانه تله پاداش منفی ده دارد و پاداش دیگری وجود ندارد.
الف- برنامه‌ای بنویسید و از طریق آن سیاست بهینه و ارزش‌های بهینه را طبق الگوریتم VI زیر تعیین کنید.
ب- برنامه دیگری نوشته و این بار مقادیر بهینه را با استفاده از الگوریتم PI زیر بدست آورید.

پاسخ:

ابتدا برای پیاده سازی این بخش، کدی نوشتیم که شامل کلاس GridWorld است. اگر کد را مشاهده کنید، ابتدا در این کد فضای action ها را مشخص می‌کنیم، سپس کلاسی نوشتیم تا بتوانیم هر grid world دلخواهی را بسازیم. در آغاز ساخت کلاس، تابع init آن وجود دارد، که ابعاد و نقطه شروع را به آن می‌دهیم.

در ادامه تابعی برای تعیین عمل‌ها، پاداش‌ها و احتمالات آن‌ها نوشتیم. در تابع سوم، حالت فعلی را مشخص می‌کنیم، در تابع چهارم state فعلی برگردانده می‌شود و در تابع پنجم چک می‌کنیم آیا این state ترمینال است یا خیر. در تابع ششم اکشن را به محیط اعمال می‌کنیم، که در اینجا ۴ اکشن داریم، توابع دیگر نیز کمکی هستند برای چاپ کردن کل فضای حالات، ریست کردن محیط و غیره.

در ادامه در تابع q_grid نیز، gridworld خواسته شده در سوال را می‌سازیم. مطابق الگوریتم‌های PI و VI که شبه کد آن‌ها در ادامه موجود است، کدهایی این دو روش را نیز در پایتون پیاده سازی می‌کنیم.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   - $\Delta \leftarrow 0$
   - Loop for each $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - *old-action* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

شکل ۳: شبه کد PI

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
   $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$

شکل ۴: شبه کد VI

برای تست عملکرد، ۲ کد VI_test.py و PI_test.py را نوشتیم که سیاست بهینه و ارزش‌های بهینه را بازگرداند.

کدها را که اجرا کردیم مطابق معمول با ارور مواجه شدیم:

```
(base) setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV:~/Downloads/HW4/Q5$ python VI_test.py
Traceback (most recent call last):
  File "/home/setare/Downloads/HW4/Q5/VI_test.py", line 24, in <module>
    V, policy = value_iteration(g)
                ^^^^^^^^^^^^^^^^^^
  File "/home/setare/Downloads/HW4/Q5/Value_iteration.py", line 15, in value_iteration
    transition_probs, rewards = calculate_probs_and_rewards(grid)
                                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/setare/Downloads/HW4/Q5/Value_iteration.py", line 8, in calculate_probs_and_rewards
    for (s,a), v in grid.probs.items():
                    ^^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'probs'
(base) setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV:~/Downloads/HW4/Q5$ python PI_test.py
Traceback (most recent call last):
  File "/home/setare/Downloads/HW4/Q5/PI_test.py", line 25, in <module>
    policy = policy_iteration(g)
             ^^^^^^^^^^^^^^^^^^^
  File "/home/setare/Downloads/HW4/Q5/Policy_iteration.py", line 60, in policy_iteration
    transition_probs, rewards = calculate_probs_and_rewards(grid)
                                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/setare/Downloads/HW4/Q5/Policy_iteration.py", line 23, in calculate_probs_and_rewards
    for (s,a), v in grid.probs.items():
                    ^^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'probs'
```

شکل ۵: خطای کد

اشکال مربوط به تشکیل Grid بود که رفع گردید، تابع آن درواقع هیچ خروجی نداشت.

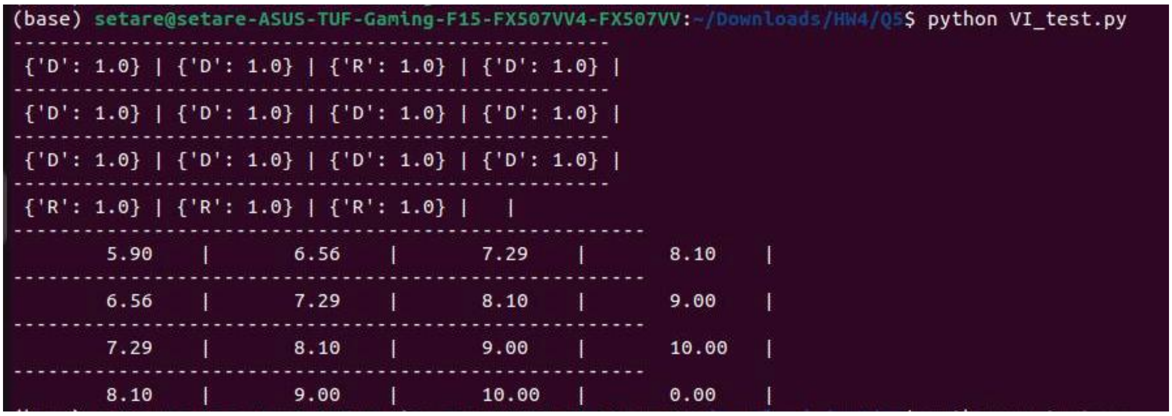
سپس کد PI را اجرا کردیم.

```
(base) setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV:~/Downloads/HW4/Q5$ python PI_test.py
---------------------------------------------------
 {'D': 1.0} | {'D': 1.0} | {'R': 1.0} | {'D': 1.0} |
---------------------------------------------------
 {'D': 1.0} | {'D': 1.0} | {'D': 1.0} | {'D': 1.0} |
---------------------------------------------------
 {'D': 1.0} | {'D': 1.0} | {'D': 1.0} | {'D': 1.0} |
---------------------------------------------------
 {'R': 1.0} | {'R': 1.0} | {'R': 1.0} |            |
```

شکل ۶: policy بهینه پس از انجام PI

همانطور که مشاهده می‌گردد، سیاست بهینه مطابق شکل ۶ بدست می‌آید، نکته قابل توجه این است که در خانه‌های اطراف تله، هیچگونه حرکتی به سمت تله نداریم.

پس از اجرای Value Iteration نیز، خروجی به صورت زیر است:



```
(base) setare@setare-ASUS-TUF-Gaming-F15-FX507VV4-FX507VV:~/Downloads/HW4/Q5$ python VI_test.py
------------------------------------------------------------------------
 {'D': 1.0} | {'D': 1.0} | {'R': 1.0} | {'D': 1.0} |
------------------------------------------------------------------------
 {'D': 1.0} | {'D': 1.0} | {'D': 1.0} | {'D': 1.0} |
------------------------------------------------------------------------
 {'D': 1.0} | {'D': 1.0} | {'D': 1.0} | {'D': 1.0} |
------------------------------------------------------------------------
 {'R': 1.0} | {'R': 1.0} | {'R': 1.0} |    |
------------------------------------------------------------------------
       5.90    |       6.56    |       7.29    |       8.10    |
------------------------------------------------------------------------
       6.56    |       7.29    |       8.10    |       9.00    |
------------------------------------------------------------------------
       7.29    |       8.10    |       9.00    |       10.00   |
------------------------------------------------------------------------
       8.10    |       9.00    |       10.00   |       0.00    |
```

شکل ۷: خروجی VI، policy بهینه و ارزش‌های بهینه

در خروجی نیز سیاست بهینه مطابق PI بدست آمد، مقادیر Value نیز در جدول مربوطه آمده است، برای خانه‌های منتهی به هدف بیشترین value را داریم، برای خانه‌های دو سطر اول چون ممکن است باعث رفتن به تله بشوند این ارزش کمتر است و همچنین دور از خانه‌ی هدف نیز هستند.

۲.۳ سوال ششم

این بخش فعلا انجام نشده است.