دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره 6

درس سیستم های هوشمند

پاییز 1401

امیرحسین بیرژندی

...

810198367

...

سوال 1: یادگیری تقویتی مبتنی بر مدل(تحلیلی):

Initialization :

←	↑	↓	3
→	→	→	-2
↓		←	↓

Initial policy

0	0	0	3
0	0	0	-2
0		0	0

Initial values

$\gamma = 0.2$ حرکت به جهت دلخواه $\rightarrow 0.6$

حرکت به جهت مجاور $\rightarrow 0.2$

$$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

First Iteration :

Policy Evaluation:

$V(1,1) = 0.6 \times [0] + 0.4 [0] = 0$

$V(1,2) = 0$

$V(1,3) = 0.6 [0] + 0.2 [0 + 0.2 \times 3] = 0.12$

$V(1,4) = 3$

$V(2,1) = 0$

$V(2,2) = 0$

$V(2,3) = 0.6 \times [0 + 0.2 \times -2] = -0.24$

$V(2,4) = -2$

$V(3,1) = 0$

$V(3,3) = 0$

$V(3,4) = 0$

Policy Improvement

0	0	0.12	3
0	0	-0.24	-2
0		0	0

←	→	→	3
→	↑	↑	-2
↓		←	↓

Second Iteration :

Bliey Evaluation :

$V(1,1) = 0$

$V(1,2) = 0.6 \times [0 + 0.2 \times 0.12] = 0.0144$

$V(1,3) = 0.6 \times [0.2 \times 3] + 0.2 [0.2 \times 0.12] + 0.2 [0.2 \times -0.24] = 0.355$

$V(1,4) = 3$

$V(2,1) = 0$

$V(2,2) = 0.2 \times [0.2 \times -0.24] = -0.0096$

$V(2,3) = 0.6 \times [0.2 \times 0.12] + 0.2 \times [0.2 \times -2] = -0.066$

$V(2,4) = -2$

$V(3,1) = 0$

$V(3,3) = 0.2 \times 0.2 \times -0.24 = -0.0096$

$V(3,4) = 0$

Policy Improvement :

0	0.0144	0.355	3
0	-0.0096	-0.066	-2
0		-0.0096	0

→	→	→	3
↑	↑	↑	-2
↓		→	↓

Third Iteration :

Policy Evaluation :

$V(1,1) = 0.6 \times 0.2 \times 0.0144 = 0.001728$

$V(1,2) = 0.6 \times 0.2 \times 0.355 + 0.2 \times 0.2 \times 0.0144 + 0.2 \times 0.2 \times -0.0096 = 0.04279$

$V(1,3) = 0.6 \times 0.2 \times 3 + 0. \times 0.2 \times 0.355 + 0.2 \times 0.2 \times -0.066 = 0.3715$

$V(1,4) = 3$

$V(2,1) = 0.2 \times 0.2 \times (-0.0096) = -0.0004$

$V(2,2) = 0.6 \times 0.2 \times 0.0144 + 0.2 \times 0.2 \times -0.066 = -0.0009$

$V(2,3) = 0.6 \times 0.2 \times 0.355 + 0.2 \times 0.2 \times -2 + 0.2 \times 0.2 \times -0.0096 = -0.0377$

$V(2,4) = -2$

$V(3,1) = 0$

$V(3,3) = 0.2 \times 0.2 \times -0.066 + 0.2 \times 0.2 \times -0.0096 = -0.003$

$V(3,4) = 0.2 \times 0.2 \times -0.0096 = -0.0004$

Policy Improvement :

0.00173	0.0428	0.3715	3
-0.0004	-0.0009	-0.0377	-2
0		-0.003	-0.004

→	→	→	3
↑	↑	↑	-2
↓		→	↓

سوال 2: یادگیری تقویتی مبتنی بر مدل(پیاده‌سازی):

در این بخش با یک مسئله یادگیری تقویتی model-based سروکار داریم. می‌دانیم در مسائلی که مدل محیط را داریم به سراغ روش هایی چون policy iteration و value iteration می‌رویم. در این بخش مسئله را value iteration حل می‌کنیم.

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

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

در شکل بالا الگوریتم value iteration را مشاهده می‌فرمایید. کافیست با درک مسئله و مدلسازی صحیح فضای حالت و فضای اقدام مسئله را حل کنیم.

فضای حالت: در این مسئله فضای حالت برابر مجموعه تمام اعداد صحیح از صفر تا 100 می‌باشد. زیرا باتوجه به مقدار شرط‌بندی شده می‌توانیم در هر کدام از این اعداد قرار بگیریم. برای مثال اگر 20 دلار داشته باشیم و 15 دلار شرط ببندیم و برنده شویم از استیت 20 به استیت 35 منتقل شده‌ایم. دو استیت 0 و 100 دو استیت ترمینال ما هستند و در صورت قرار گرفتن در هر کدام از این دو حالت شرط‌بندی تمام می‌شود.

$$state \in \{0,1,\dots,100\}$$

فضای اقدام: در این مسئله فضای اقدام در واقع مقدار پول مورد نظر ما برای شرط‌بندی است.این فضا به گونه‌ای است که اگر مقدار دارایی ما در لحظه کوچکتر و یا مساوی 50 باشد می‌توانیم حداکثر برابر دارایی خودمان شرط بندی کنیم و اگر مقدار دارایی ما بیشتر از 50 باشد می‌توانیم حداکثر برابر تفاوت 100 دلار و مقدار دارایی خود شرط‌بندی کنیم. به عبارتی:

$$action \in \{0,1,\dots,\min(state, 100 - state)\}$$

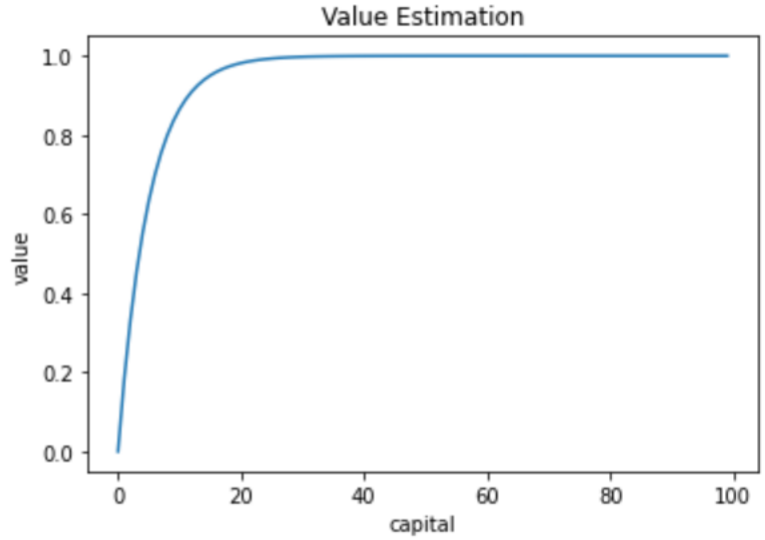
پاداش: نحوه پاداش‌دهی در این مسئله به گونه‌ای است که اگر شرط بندی موجب به برنده شدن ما شود پاداش +1 دریافت می‌کنیم و در غیر این صورت پاداش دیگری دریافت نمی‌کنیم.

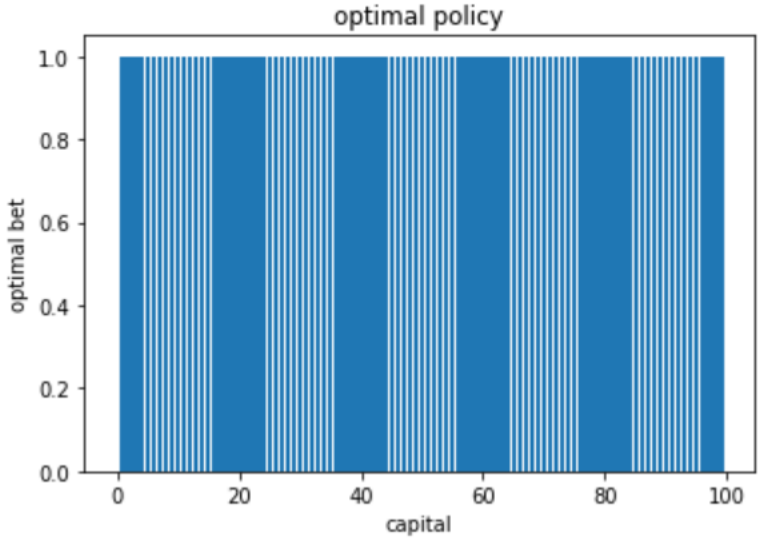
شبیه سازی

$p_h = 0.55$

```
optimal policy for P_h=0.55:
[0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1.]
```

در تصویر بالا مقدار بهترین مقدار پول برای هر استیت در شرط بندی برابر 1 شده است که جلوتر این موضوع را تحلیل می‌کنیم.

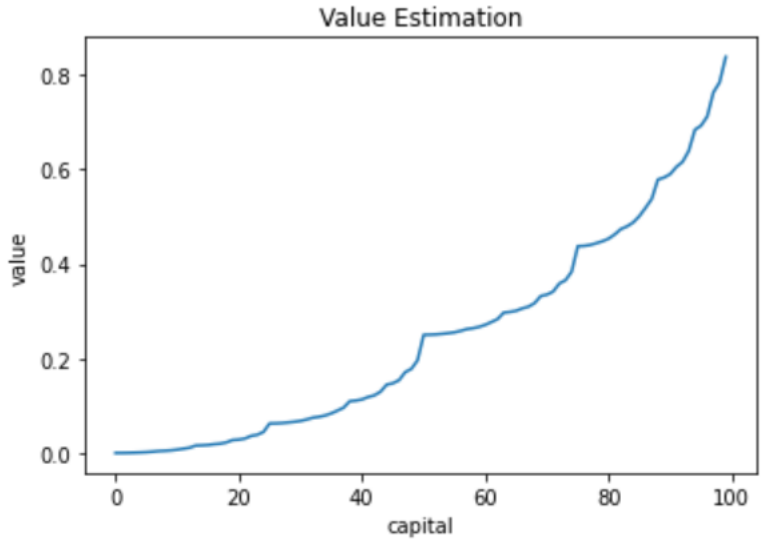


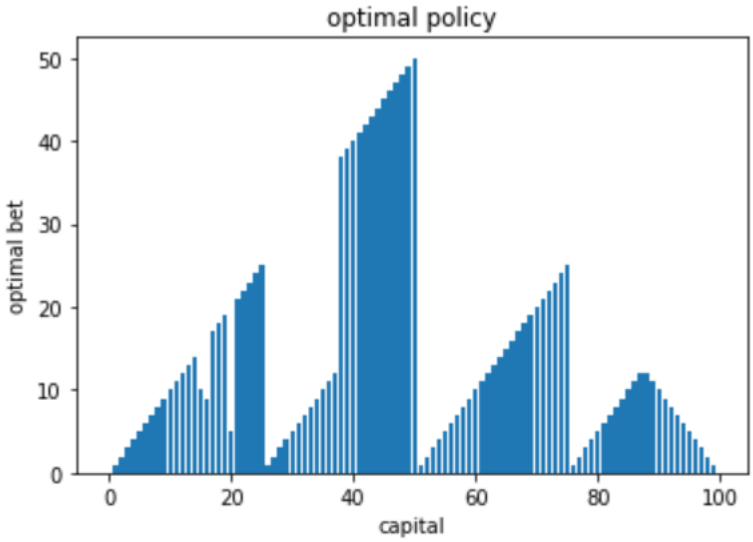


$p_h = 0.25$

```
optimal policy for P_h=0.25:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 10.  9. 17.
 18. 19.  5. 21. 22. 23. 24. 25.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.
 11. 12. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50.  1.  2.  3.
  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21.
 22. 23. 24. 25.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 12. 11.
 10.  9.  8.  7.  6.  5.  4.  3.  2.  1.]
```

در تصویر بالا مقدار بهینه برای شرط‌بندی در هر استیت نمایش داده شده است.





تحلیل:

در حالتی که $p_h = 0.55$ است بهترین نتیجه را زمانی دریافت می‌کنیم که در همه استیت ها فقط 1 دلار شرط بندی کنیم که این کار منطقی است زیرا زمانی که با احتمال بیش از نیم در هر سری پیروز می‌شویم قطعا با شرطبندی های 1 دلاری بالاخره به مقدار 100 دلار می‌رسیم و در واقع کمترین ریسک را در این کار داریم.

حال در حالتی که $p_h = 0.25$ است به دلیل اینکه احتمال باختن شرط بالاتر است بهتر است که تا زمانی که کمتر از 50 دلار پول داریم ریسک کنیم و خود را بالا بکشیم. البته دقت شود این ریسک یک ریسک منطقی است زیرا در مقادیر پایین با مقدار شرط های کوچک فقط خود احتمال از دست دادن پول بیشتر است زیرا باید چندین بار شانس بیاوریم و از احتمال 0.25 استفاده کنیم که در واقع غیر ممکن است.

نکته دیگر این است که مشاهده می‌کنیم در هر دو نمودار ارزش استیت ها صعودی است که این موضوع کاملا روشن است زیرا هر چه به 100 دلار نزدیک تر باشیم در واقع حاشیه امنیت بیشتری داریم پس در آن استیت قرار گرفتن ارزش بالاتری دارد.

توضیحات پیاده‌سازی:

برای پیاده‌سازی این بخش یک class به نام Gamblers_Problem تعریف شده است. این کلاس به عنوان ورودی 3 مقدار دریافت می‌کند:

1. احتمال شیر آمدن 2. مقدار تخفیف 3. مقدار تتا

توابع این کلاس عبارت هستند از:

Available actions: با توجه به استیت، اکشن های قابل استفاده را مشخص می‌کند.

Calculate reward: با توجه به استیت و اکشن در نظر گرفته شده مقدار پاداش را مشخص می‌کند.

Max value: با توجه به استیت، اکشنی که بیشترین ارزش را برای این استیت دارد پیدا می‌کند و به همراه آن مقدار ارزش را مشخص می‌کند.

Value iteration: این تابع الگوریتم را پیاده‌سازی می‌کند.

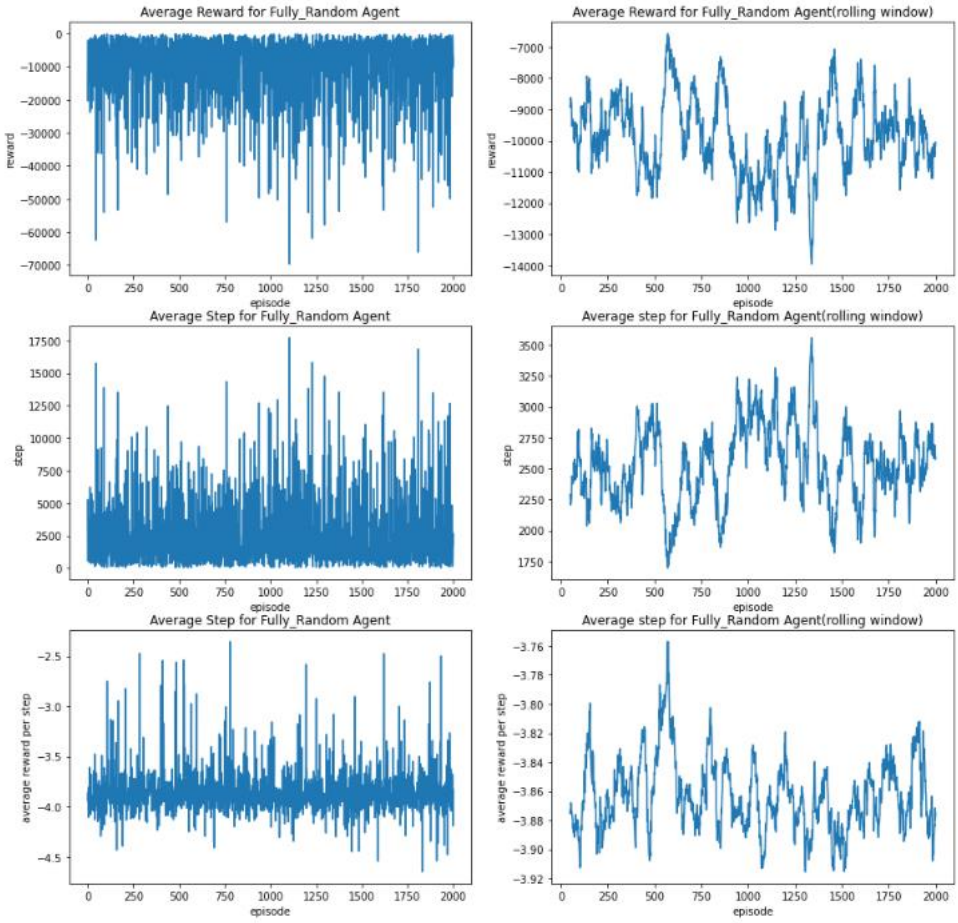
سوال 3: یادگیری تقویتی غیر مبتنی بر مدل(پیاده‌سازی):

در این بخش قرار است که یک مسئله یادگیری تقویتی model-free را حل کنیم. می‌دانیم در این گونه مسائل سراغ روش هایی چون Sarsa و Q-Learning می رویم. در این بخش قرار است الگوریتم Q-Learning را پیاده‌سازی کنیم.

الف) حل محیط بازی بدون استفاده از روش Q-Learning و مبتنی بر پیمایش رندوم:

برای پیاده سازی این بخش کافی است در هر گام یک اکشن رندوم به وسیله دستور self.Enivronment.action_space.sample() تولید کنیم و تا جایی در هر اپیزود باقی می مانیم که بالاخره تاکسی مسافر را در مکان درست پیاده کند. حال در این قسمت برای پیاده‌سازی 2000 اپیزود را در نظر می‌گیریم.

برای مشخص‌تر شدن نتایج روی نمودار از تکنیک Rolling window استفاده شده است.



در تصاویر سمت راست از همان تکنیک Rolling window ذکر شده استفاده شده است.

دو نمودار بالا: نمودار های بالا مجموع پاداش های دریافتی در هر اپیزود را مشخص می‌کند. توقع داریم در حالتی که رندوم اکشن ها را تعیین میکنیم پاداشی بسیار منفی و متغیر داشته باشیم که نمودار های ما آن را نشان می‌دهند.

دو نمودار میانی: نمودار های میانی تعداد گام های برداشته شده در هر اپیزود را نشان می‌دهند. توقع داریم در حالتی که رندوم اکشن ها را تعیین می‌کنیم تعداد گام ها بسیار زیاد و متغیر باشد.

دو نمودار پایینی: نمودار های پایینی میانگین پاداش در هر گام را نشان می‌دهد. این نمودار با تقسیم مجموع پاداش هر اپیزود به تعداد گام آن اپیزود محاسبه می‌شود.

ب) حل محیط بازی با استفاده از روش Q-Learning و مبتنی بر پیمایش هوشمندانه

در این بخش به حل مسئله با Q-Learning می‌پردازیم.

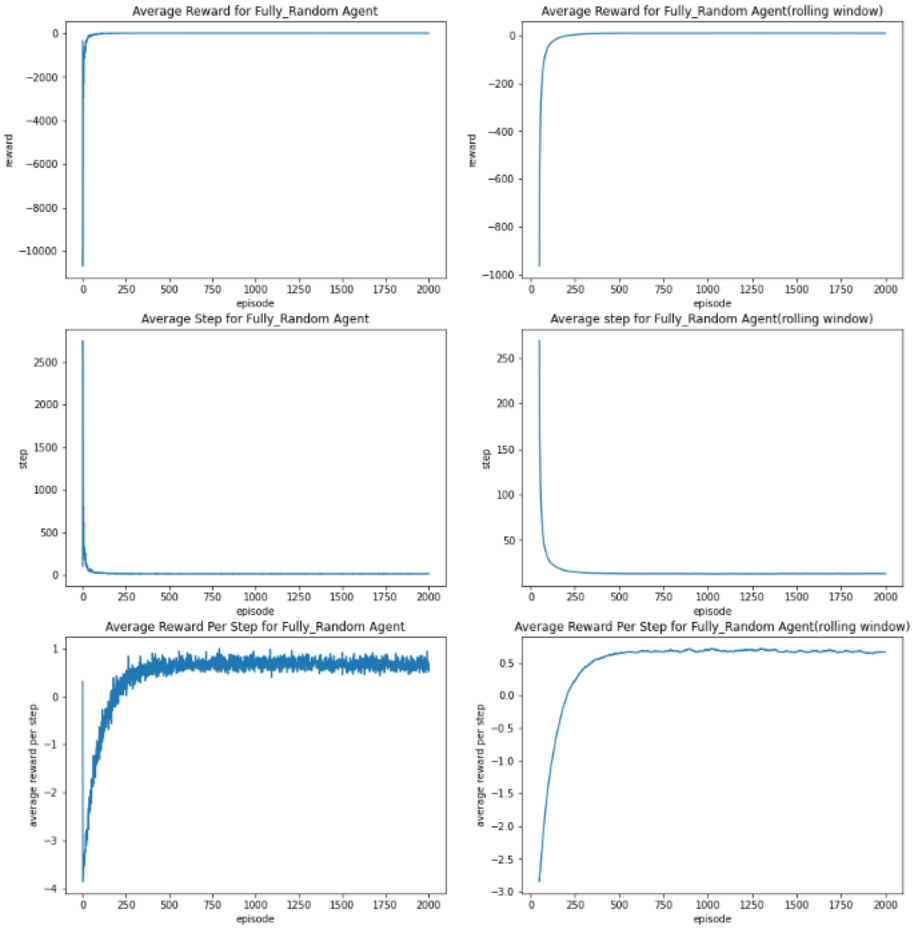
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

در تصویر بالا الگوریتم Q-Learning را مشاهده می‌کنیم. برای پیاده‌سازی این الگوریتم باید در نظر بگیریم که در هر گام از اپیزود باید اکشن استفاده شده را با توجه به یک سیاست مشخص بدست آوریم. در این پیاده سازی از سیاست epsilon-greedy استفاده می‌کنیم. در این سیاست در هر استیت اکشنی که بیشترین ارزش یا Q دارد با احتمال بالاتری انتخاب می‌شود و مابقی اکشن‌ها با احتمال کوچکتری انتخاب می‌شوند. حال کافیست اکشن انتخاب شده توسط سیاست epsilon-greedy را با استفاده از دستور self.Enivronment.step(action) اجرا کنیم. پس از اجرا استیت بعدی و مقدار پاداش را دریافت کرده و به وسیله آن ارزش های Q خود را آپدیت می‌کنیم. این چرخه را تا جایی ادامه می‌دهیم که تاکسی مسافر را در مکان صحیح پیاده کند.

برای پیاده سازی این بخش از 20 تکرار که هر یک دارای 2000 اپیزود است استفاده شده است.



نتایج شبیه‌سازی را در نمودار های بالا مشاهده می‌کنیم.

توضیح نتایج حاصل از Q-learning

دو نمودار بالا: نمودار های بالا مجموع پاداش های دریافتی در هر اپیزود را مشخص می‌کند. توقع داریم در حالتی که از روش Q-learning اکشن ها را تعیین میکنیم مجموع پاداش در مرور زمان افزایش و به ریوارد مطلوب همگرا شود.

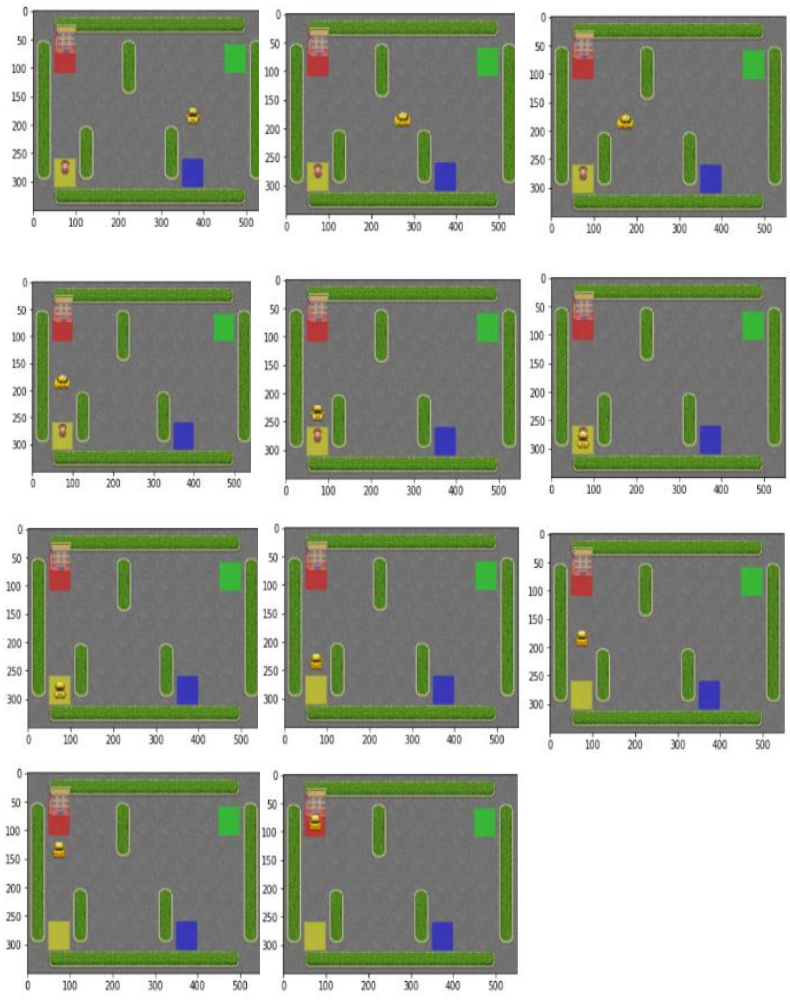
دو نمودار میانی: نمودار های میانی تعداد گام های برداشته شده در هر اپیزود را نشان می‌دهند. توقع داریم در حالتی که از روش Q-learning اکشن ها را تعیین میکنیم تعداد گام ها به مرور زمان کاهش پیدا کند و به اندازه مورد نیاز باشد.

دو نمودار پایینی: نمودار های پایینی میانگین پاداش در هر گام را نشان می‌دهد. این نمودار با تقسیم مجموع پاداش هر اپیزود به تعداد گام آن اپیزود محاسبه می‌شود در نتیجه باید مانند نمودار مجموع پاداش کاملا صعودی باشد و به مقدار نزدیک 1 همگرا شود.

محاسبه پاداش همگرا شده

تصاویر زیر را مشاهده کنید همانطور که مشخص است تاکسی با 10 گام به هدف خود دست می‌یابد در نتیجه توقع داریم به پاداش 10-20 که حاصل تفریق 10 تا پاداش جابه‌جایی از پاداش نهایی که 20 است همگرا شویم. یعنی به عدد 10 همگرا شویم. اما از آنجایی که ممکن تاکسی در مکان های اولیه مختلف قرار گیرد و تعداد گام ها متفاوت باشد نمودار اصلی باید به مقداری در حوالی 10 همگرا شود که آن مقدار را محاسبه کرده‌ایم.

```
Q-Learning Average reward in last 100 episodes: 7.99
```



در تصاویر بالا یک نمونه مسیر یابی تاکسی پس از یادگیری توسط Q-Learning را مشاهده می‌کنیم. در واقع ارزش ها به گونه‌ای آپدیت شده است که تاکسی با دانستن استیت خود می تواند به استیتی برود که بیشترین ارزش را دارد.

مقایسه نتایج:

همانطور که در نمودار های دو بخش مشاهده می‌کنیم در حالت رندوم مجموع پاداش ها مقدار بسیار منفی‌ای شده است و در واقع به ازای هر اپیزود بسیار جریمه شده ایم و در مقادیری چون 8000- سیر می‌کنیم. اما با استفاده از Q-Learning مشاهده می‌کنیم در مرور زمان مقدار جریمه ها کاهش یافته و به مقدار پاداش صفر می‌رسیم که نشان دهنده هوشمندانه بودن روش می‌باشد. همین تحلیل برای میانگین پاداش ها نیز برقرار است.

همچنین در تعداد گام های هر اپیزود مشاهده می‌کنیم که در روش رندوم به صورت میانگین 2500 گام در هر اپیزود طی می‌کنیم. از آن طرف در روش Q-learning تعداد گام ها به مرور زمان کاهش یافته و به حداقل خود رسیده است.

تغییر مقدار پاداش ها:

با تغییر مقدار پاداش ها می‌توانیم به همگرایی سریع‌تری برسیم. این تغییر باید از دیدگاه کسی باشد که به محیط واقف است و باید این تغییرات معقول و هشمندانه باشد. برای همگرایی سریع تر باید دریابیم که در کدام استیت ها می‌توانیم پاداش را تغییر دهیم تا به همگرایی سریع‌تری برسیم. در این پیاده سازی دو تغییر را ایجاد کرده که در ادامه آن ها را توضیح می‌دهیم.

***تغییر 1:** می دانیم زمانی که مسافر را سوار کردیم خانه های نزدیک مقصد ارزش بالایی دارند پس کافیست جریمه جابه‌جایی در این خانه ها را کاهش دهیم تا قدر این خانه ها دانسته شود. برای این کار تغییر زیر داده شده است.

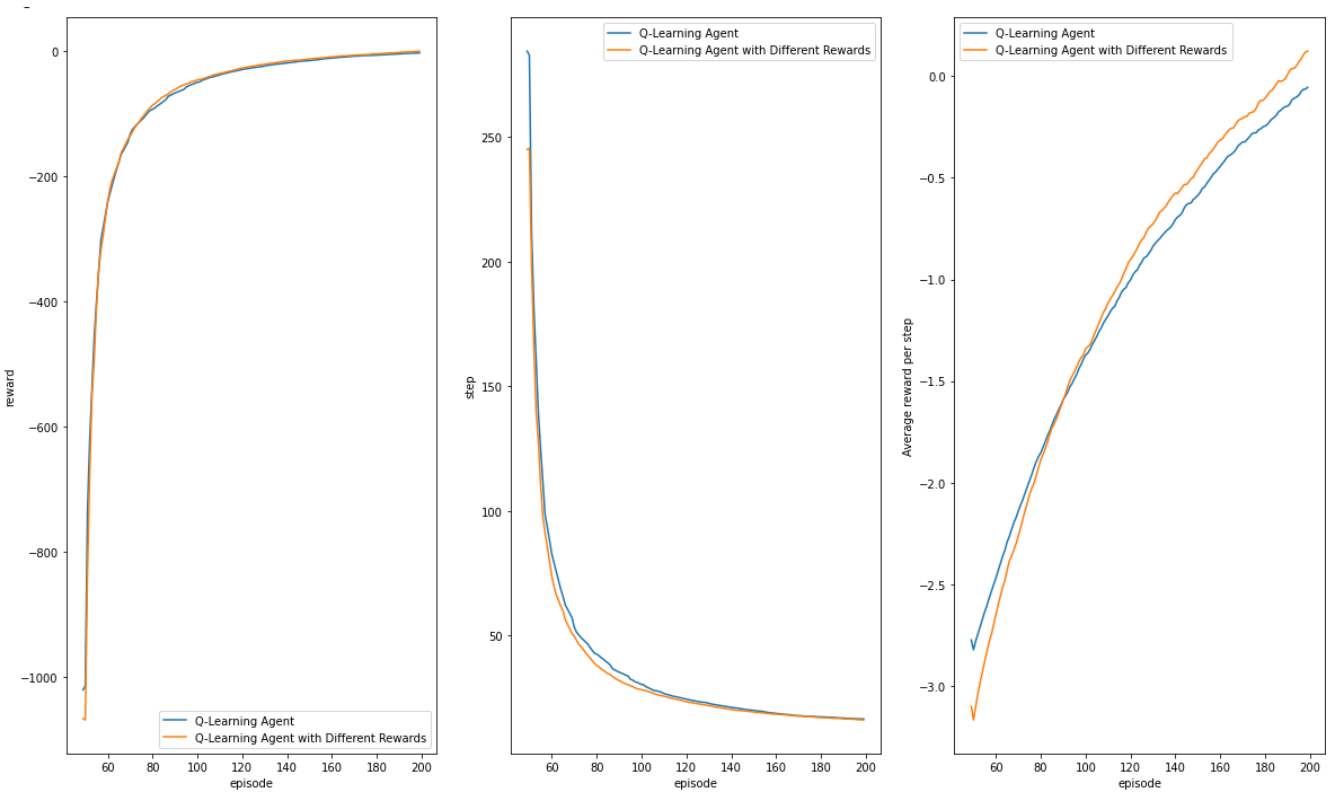
```
pass_row , pass_col , pass_idx , dest_idx = self.Enivronment.decode(next_state)
if(pass_idx == 4):
  if(dest_idx == 0):
    if(([pass_row , pass_col] == [0,0]) or ([pass_row , pass_col] == [0,1]) or ([pass_row , pass_col] == [1,0]) or ([pass_row , pass_col] == [1,1])):
      reward = -0.1
  if(dest_idx == 1):
    if(([pass_row , pass_col] == [0,3]) or ([pass_row , pass_col] == [1,3]) or ([pass_row , pass_col] == [1,4]) or ([pass_row , pass_col] == [0,4])):
      reward = -0.1
  if(dest_idx == 2):
    if(([pass_row , pass_col] == [2,0]) or ([pass_row , pass_col] == [3,0]) or ([pass_row , pass_col] == [4,0])):
      reward = -0.1
  if(dest_idx == 3):
    if(([pass_row , pass_col] == [3,3]) or ([pass_row , pass_col] == [3,4]) or ([pass_row , pass_col] == [4,4]) or ([pass_row , pass_col] == [4,3])):
      reward = -0.1
```

همانطور که مشاهده می‌کنیم ابتدا با دستور decode اطلاعات حالت فعلی را پیدا کرده سپس با یک سری شرط که مشخص می‌کنند آیا با مسافر در نزدیکی مقصد هستیم یا خیر، پاداش را از 1- به 0.1- تغییر می‌دهیم.

***تغییر 2:** می دانیم هیچ کدام از مقاصد ما در وسط صفحه موجود نمی‌باشند برای همین پیاده کردن مسافر در این نواحی کار عاقلانه‌ای نیست پس کافیست در این مکان ها پاداش منفی تری قرار دهیم که برای این کار تغییر زیر در پاداش ها داده شده است.

```
if(([pass_row , pass_col] == [2,1]) or ([pass_row , pass_col] == [3,1]) or ([pass_row , pass_col] == [4,1]) or ([pass_row , pass_col] == [0,2]) or ([pass_row , pass_col] == [1,2])or([pass_row , pass_col] == [2,2])or([pass_row , pass_col] == [3,2])or([pass_row , pass_col] == [4,2])):
  if(reward == -10):
    reward = -20
```

برای مقایسه تاثیر این تغییرات در سرعت همگرایی 20 بار 2000 اپیزود را ران کرده و نتایج را با نتایج بدست آمده در قسمت قبل مقایسه می‌کنیم. در این نمودار ها 200 اپیزود اول را نشان داده‌ایم که این تغییر سرعت مشهود باشد.



در تصاویر بالا مشاهده می‌کنیم که در هر سه نمودار کمی سریع‌تر همگرا شده‌ایم و این تغییر در نمودار تعداد گام ها باید خود را کامل نشان دهد و مشاهده می‌کنیم که در این نمودار به صورت کاملاً مشهود سریع تر همگرا شده‌ایم به عبارتی تعداد گام کمتری برای یادگیری محیط خرج کرده‌ایم.

ایده برای بهینه کردن الگوریتم:

یکی از ایده های مورد استفاده برای tune کردن هایپر پارامتر ها استفاده از epsilon و learning rate کاهشی می باشد که در ادامه توضیح می‌دهیم که چرا این تغییرات باعث پیشرفت الگوریتم ما می‌شود.

اپسیلون کاهشی

یکی از مهمترین چالش ها در یادگیری تقویتی رو به رو شدن با مسئله exploration exploitation balance است. در ابتدای یادگیری زمانی که هنوز اطلاعات کافی راجب محیط نداریم باید explore بیشتری انجام دهیم که معادل این است که اپسیلون در سیاست epsilon-greedy مقدار بزرگی باشد و در مرور زمان که اطلاعات خود را از محیط بیشتر بردیم exploitaion بیشتری انجام دهیم که معادل اپسیلون کوچک در سیاست epsilon-greedy می‌باشد. در نتیجه اگر مقدار اپسیلون را به مرور زمان به صورت نمایی کاهش دهیم نتیجه خیلی خوبی خواهیم گرفت. برای پیاده‌سازی کافیست اپسیلون را برابر یک و با دستور زیر با گذشت از هر اپیزود آن را کاهش دهیم.

```
self.Epsilon = np.exp(-0.01*current_episode)
```

نرخ یادگیری کاهشی

همانطور که در مسائل عمومی یادگیری ماشین که نیاز به یک نرخ یادگیری داریم در مسائل یادگیری نیز تقویتی به این هایپرپارامتر نیاز داریم. اگر نرخ یادگیری در مرور زمان کاهش یابد میتوانیم از واریانس نتایج کم کنیم و جواب های بهتری دریافت کنیم. برای پیاده‌سازی کافیست نرخ یادگیری را از مقدار 0.1 شروع کرده و با دستور زیر با هر اپیزود آن را تغییر دهیم.

```
self.Learning_rate = np.exp(-0.001*current_episode)
```

توضیحات پیاده‌سازی:

برای پیاده سازی هر بخش یک کلاس متناظر آن‌ها تعریف شده است که در این بخش به توضیح پیاده‌سازی کلاس Q_Learning_Agent می‌پردازیم.

این کلاس به عنوان ورودی متغیر و پارامتر های زیر را دریافت می‌کند.

1. محیط 2. تعداد تکرار الگوریتم 3. تعداد اپیزود هر تکرار 4. مقدار اپسیلون سیاست epsilon-greedy 5. نرخ یادگیری 6. مقدار تخفیف

توابع مورد استفاده در این کلاس عبارت‌اند از:

Choose_action: در این تابع با استفاده از سیاست epsilon-greedy با توجه به استیت قرار گرفته ارزش تمام اکشن ها بررسی می‌شود و برای اکشنی که ارزش ماکسیمم دارد بیشترین احتمال و برای ما بقی اکشن ها احتمال برابر کوچکتری در نظر گرفته می‌شود. سپس یک مقدار رندوم بین صفر و یک تولید می‌کنیم اگر بین بازه احتمالی اکشن با ارزش بیشتر بود آن اکشن به عنوان خروجی داده می‌شود و در غیر این صورت رندوم از بین مابقی اکشن ها انتخاب می‌کنیم.

Update_Q: در این تابع مقدار ارزش استیت-اکشن ها آپدیت می‌شود. در واقع رابطه زیر پیاده می‌کنیم.

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

Q_Learning: در این تابع که هسته اصلی الگوریتم است در هر اپیزود تا زمانی که به هدف خود نرسیده‌ایم متوقف نمی‌شویم. در گام با توجه به سیاست epsilon-greedy اکشن مورد نظر را انتخاب کرده و پاداش و استیت بعدی را دریافت می‌کنیم. با این دو مقدار دریافتی Q ها را بروزرسانی می‌کنیم و این چرخه ادامه پیدا خواهد کرد. این تابع به عنوان خروجی مقدار پاداش و گام هر اپیزود را تحویل می‌دهد.

Routing_with_updated_Q_values: این تابع پس از یادگیری برای نشان دادن نحوه مسیریابی و rendering استفاده می‌شود.

Q_Learning_algorithm: این تابع صرفا Q_Learning را برای 20 بار اجرا می‌کند.

منابع:

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.