



دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



گزارش تمرین شماره ۴

درس یادگیری تعاملی

پاییز ۱۴۰۱

محمد سعادت

۸۱۰۱۹۸۴۱۰

فهرست

چکیده.....	۴
سوالات تحلیلی.....	۵
سوال ۱.....	۵
۱-۱.....	۵
۲-۱.....	۵
۳-۱.....	۵
سوال ۲.....	۷
مسئله پیاده سازی.....	۸
سوال ۱.....	۹
هدف سوال.....	۹
توضیح پیاده سازی.....	۹
روند اجرای کد پیاده سازی.....	۱۰
نتایج.....	۱۰
سوال ۲.....	۱۲
توضیح پیاده سازی.....	۱۲
روند اجرای کد پیاده سازی.....	۱۲
سوال ۳.....	۱۳
هدف سوال.....	۱۳
توضیح پیاده سازی.....	۱۴
روند اجرای کد پیاده سازی.....	۱۵
نتایج.....	۱۵

سوال ۴ ۱۷

هدف سوال ۱۷

توضیح پیاده سازی ۱۷

روند اجرای کد پیاده سازی ۱۸

نتایج ۱۸

سوال ۵ ۲۱

امتیازی ۲۱

توضیح پیاده سازی ۲۱

روند اجرای کد پیاده سازی ۲۲

نتایج ۲۲

منابع ۲۵

چکیده

هدف این تمرین آشنایی با الگوریتم‌هایی برای حل مسئله MDP با فرض ناشناخته بودن محیط می باشد. از این روش‌ها در ادبیات به عنوان روش‌های بدون مدل (Model-Free) یاد می شود. در این تمرین دو سوال تحلیلی و یک مسئله پیاده سازی که شامل بخش‌های مختلف می شود در نظر گرفته شده است که طی آن با الگوریتم‌های SARSA, Expected SARSA, Q-learning, Tree Backup n-step آشنا می شویم.

سوالات تحلیلی

سوال ۱

در این سوال الگوریتم های SARSA و Expected SARSA را در یک محیط گسسته از منظر میزان پشیمانی در حالت های مختلف مقایسه می کنیم.

بطور کلی سوالات این بخش جواب یکتایی ندارند زیرا هر یک از این دو الگوریتم برتری مطلق بر دیگری ندارند و با توجه به task ای که داریم، یک الگوریتم ممکن است نسبت به دیگری عملکرد بهتری داشته باشد و در task دیگری این موضوع برعکس باشد.

۱-۱

الگوریتم Expected SARSA عموماً با میزان حسرت کمتری نسبت به الگوریتم SARSA به یک سیاست epsilon-optimal همگرا می شود زیرا Expected SARSA مجموعی وزن دار را بر روی اکشن ها محاسبه می کند در صورتی که SARSA به طور noisy نمونه برداری می کند. چون Expected SARSA از نمونه برداری تصادفی استفاده نمی کند، واریانس کمتری نیز دارد. [1] [2]

۲-۱

الگوریتم SARSA ممکن است سیاست بهینه را پیدا نکند. SARSA به یک سیاست "تقریباً" بهینه همگرا می شود. با این حال می تواند به یک سیاست بهینه همگرا شود تا زمانی که همه جفت های state-action به تعداد بی نهایت بازدید شوند و سیاست در حد سیاست ϵ -greedy همگرا شود. Expected SARSA یادگیری مقادیر سیاست greedy بهینه را هدف قرار می دهد، در حالی که SARSA یادگیری مقادیر سیاست تقریباً بهینه ϵ -greedy را هدف قرار می دهد. به همین دلیل عموماً میزان حسرت الگوریتم Expected SARSA نسبت به الگوریتم SARSA در یافتن سیاست بهینه کمتر است. همچنین دلایل قسمت قبل نیز برای این بخش قابل استفاده است. [1] [2] [3] [4]

۳-۱

الگوریتم Expected SARSA از نظر محاسباتی پیچیده تر و در نتیجه کندتر از SARSA است زیرا expectation، Expected SARSA را بر روی اکشن های بعدی محاسبه می کند. تفاوت در هزینه های

محاسباتی بین دو روش زمانی افزایش می یابد که فضای عمل یک محیط افزایش یابد. محاسبه expected value نسبت به اکشن ها در محیط هایی با تعداد اکشن های بیشتر در مقایسه با محیط های ساده، نسبتاً پرهزینه تر است، در حالی که هزینه های محاسباتی نمونه برداری تصادفی بر روی اکشن ها با تغییر تعداد اکشن ها تغییر نمی کند. [1]

SARSA نسبت به انتخاب step-size یا نرخ یادگیری حساسیت بیشتری دارد زیرا هدف آن واریانس بالاتری دارد. در نتیجه سرعت کاهش نرخ یادگیری Expected SARSA نسبت به SARSA بیشتر است. از آنجایی که Expected SARSA نسبت به SARSA محاسبات بیشتری در رسیدن به سیاست بهینه و سیاست epsilon-optimal انجام می دهد، در نتیجه SARSA نسبت به Expected SARSA با سرعت بیشتری epsilon را کاهش می دهد و زودتر به یک سیاست همگرا می شود.

سوال ۲

expected value همه n-step return ها تضمین می‌شود که به روشی خاص نسبت به تابع value فعلی به عنوان تقریبی به تابع value واقعی بهبود یابد. برای هر V ، n-step return expected value با استفاده از V تضمین می‌شود که تخمین بهتری از V^π نسبت به V در بدترین حالت باشد. یعنی بدترین خطا در برآورد جدید تضمین می‌شود که کمتر یا مساوی γ^n برابر بدترین خطا در زیر V باشد:

$$\max_s \underbrace{\left| E_\pi \{ R_t^n \mid s_t = s \} - V^\pi(s) \right|}_{\text{n step return}} \leq \gamma^n \underbrace{\max_s |V(s) - V^\pi(s)|}_{\text{Maximum error using V}}$$

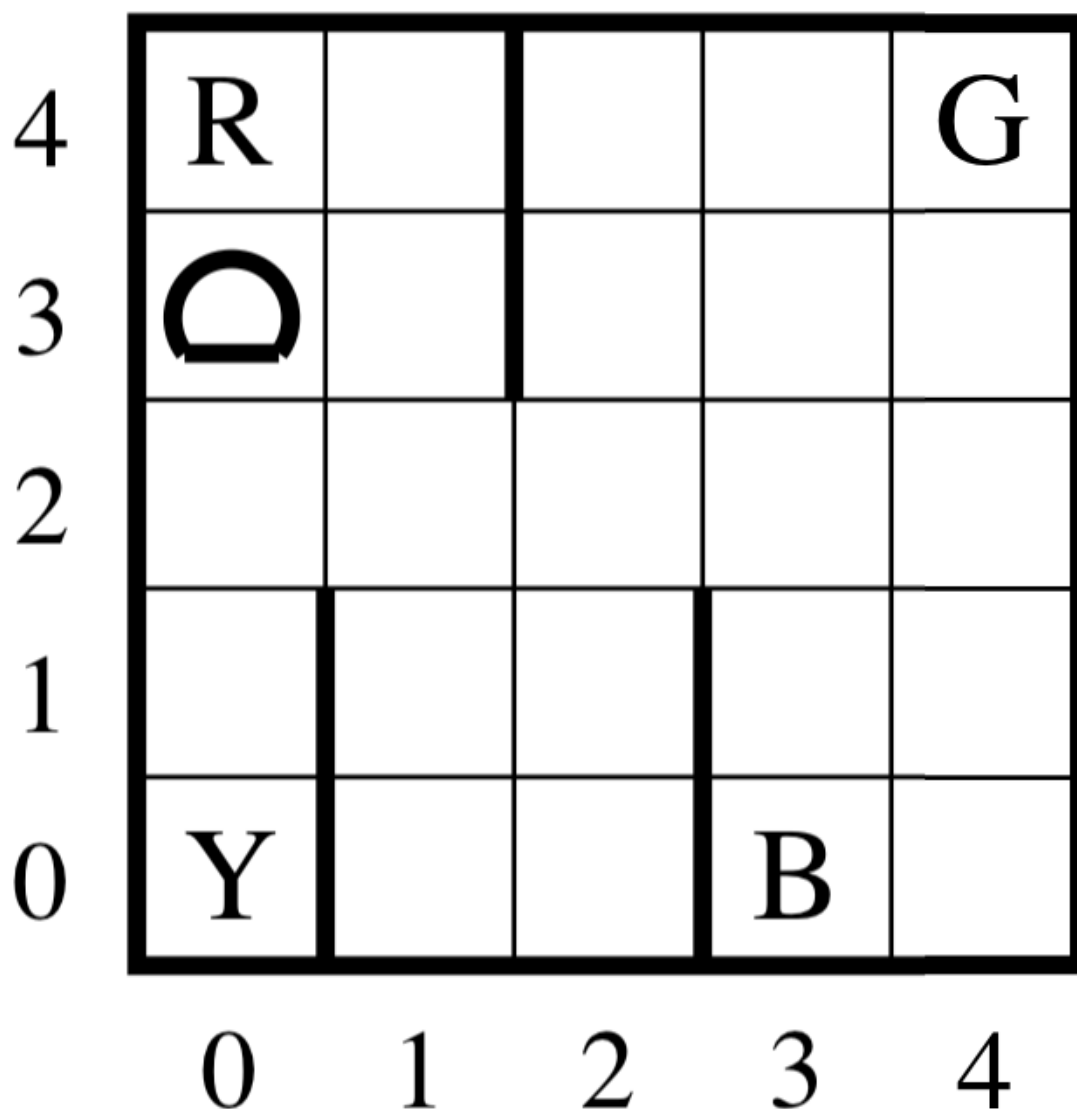
Maximum error using n-step return Maximum error using V

به این error-correction property n-step return error-reduction property می‌گویند. به دلیل TD آنلاین و آفلاین با استفاده از پشتیبان گیری n-step return به پیش بینی های صحیح تحت شرایط فنی مناسب همگرا می‌شوند. بنابراین، روش های TD -step n ، خانواده ای از روش های معتبر را تشکیل می‌دهند. [5]

اثبات رابطه بالا در این [لینک](#) آورده شده است. [6]

مسئله پیاده سازی

ما از محیط OpenAI Gym's Taxi-v3 برای طراحی الگوریتمی استفاده می‌کنیم تا به یک مأمور تاکسی آموزش دهد تا در یک gridworld کوچک حرکت کند.



States: ۵۰۰ حالت ممکن وجود دارد که مربوط به ۲۵ موقعیت شبکه ممکن، ۵ مکان برای مسافر و ۴ مقصد است.

Actions: ۶ عمل ممکن وجود دارد که مربوط به حرکت به سمت شمال، شرق، جنوب یا غرب، سوار کردن مسافر و پیاده کردن مسافر است.

سوال ۱

هدف سوال

در این سوال قرار است مسئله را با استفاده از الگوریتم Q-Learning حل کنیم. برای پیاده سازی الگوریتم مطابق شبه کد آن در کتاب ساتون بارتو عمل می کنیم:

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  $Q(\text{terminal}, \cdot) = 0$ 
Loop for each episode:
  Initialize  $S$ 
  Loop for each step of episode:
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
```

مسئله را یکبار با نرخ یادگیری ثابت و یکبار با نرخ یادگیری کاهشی حل می کنیم و نتایج را تحلیل می کنیم.

توضیح پیاده سازی

کد های مربوط به پیاده سازی این سوال در فایل [Codes.ipynb/html](https://codes.ipynb/html) قسمت **Question 1** قرار دارد.

- در کلاس **QLearning** الگوریتم Q-Learning پیاده سازی شده است.

○ کاهش مقدار اپسیلون در تابع **decay_epsilon** این کلاس صورت می گیرد. در اولین اپیزود مقدار اپسیلون برابر ۱ است و در اپیزود های بعدی مقدار اپسیلون برابر رابطه زیر قرار می گیرد:

$$\text{Epsilon} = e^{-(\text{epsilon decay rate} * \text{episode number})}$$

که مقدار **decay_rate** برابر ۰.۰۰۳ است.

○ کاهش مقدار نرخ یادگیری در تابع **decay_epsilon** این کلاس صورت می گیرد. در اولین اپیزود مقدار نرخ یادگیری برابر ۰.۹۹ است و در اپیزود های بعدی مقدار نرخ یادگیری برابر رابطه زیر قرار می گیرد:

$$\text{Learning rate} = 0.99 * e^{-(\text{decay rate} * \text{episode number})}$$

که مقدار **decay_rate** برابر ۰.۰۰۲۳ است.

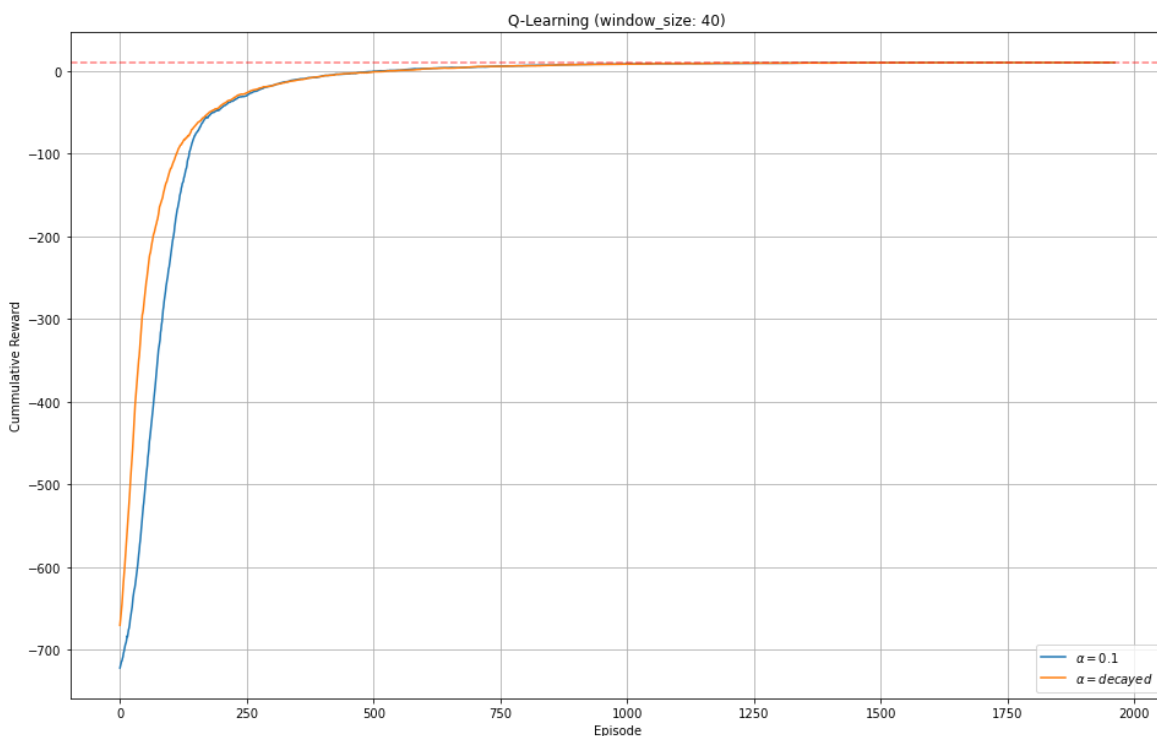
- تابع `agent_QLearning_run` الگوریتم Q-Learning را با ۲۰ بار تکرار و به تعداد ۲۰۰۰ اپیزود با توجه به نرخ یادگیری ثابت یا کاهشی اجرا می کند و پاداش های دریافت شده و پاداش سیاست بهینه را بر می گرداند.
- تابع `run_optimal_policy` برای پیدا کردن پاداش حاصل از اجرا سیاست بهینه استفاده می شود.
- تابع `run_20_episode` برای اجرا عامل آموزش داده شده به اندازه ۲۰ اپیزود استفاده می شود.

روند اجرای کد پیاده سازی

برای اجرا کد های این بخش سوال است تا تمام کد های مقابل این بخش و کد های این سوال را اجرا کنید.

نتایج


نتایج حاصل از اجرای الگوریتم در دو حالت نرخ یادگیری ثابت و کاهشی به شکل زیر است:



مشاهده می شود که در هر دو حالت در نهایت الگوریتم به سیاست بهینه همگرا می شود اما در حالتی که نرخ یادگیری کاهش است، سرعت همگرایی بیشتر و الگوریتم زودتر به سیاست بهینه همگرا می شود. همچنین حالتی که نرخ یادگیری کاهشی است نسبت به حالتی که نرخ یادگیری ثابت است، دارای حسرت کمتری است و مقدار پاداش دریافتی آن در هر اپیزود بیشتر است. دلیل این برتری آن است که در حالت

نرخ یادگیری کاهشی ما از یک نرخ یادگیری زیاد آموزش عامل را شروع می کنیم و به مرور نرخ یادگیری کم می شود که این باعث می شود در ابتدا که عامل تجربه کمی دارد، یادگیری زیادی از داده های جدید داشته باشد و به مرور که تجربه عامل افزایش می یابد، کمتر از داده های جدید یادگیری می کند و بیشتر به تجربه کسب شده خود اتکا می کند. این باعث می شود تا سرعت همگرایی افزایش و میزان حسرت کاهش یابد. اما در حالتی که نرخ یادگیری برابر یک مقدار کوچک و ثابت است، آموزش عامل به کندی صورت می گیرد. در نتیجه سرعت همگرایی کمتر و میزان حسرت بیشتر است.

عملکرد عامل بهینه پس از آموزش به اندازه ۲۰ ایزود به شکل زیر است:

+	-	-	-	-	-	+
	R :		:		:	G
	:		:	:	:	
	:	:	:	:	:	
	:	:	:	:	:	
	Y	:		B :	:	
+	-	-	-	-	-	+

سوال ۲

توجه داشته باشید که ۴۰۰ حالت وجود دارد که واقعاً می توان در طول یک episode به آنها رسید. حالت ترمینال زمانی است که مکان مسافر با مکان مقصد یکی باشد. از نظر فنی حالت های غیرقابل دسترسی وجود دارد که مسافر را در مقصد و تاکسی را در جای دیگری نشان می دهند، زیرا این حالت معمولاً پایان یک قسمت را نشان می دهد. اما state representation همچنان شامل این موارد می شود، زیرا انجام این کار آسان تر است. چهار حالت دیگر را می توان بلافاصله پس از یک episode موفق مشاهده کرد، زمانی که مسافر و تاکسی هر دو در مقصد هستند. این در مجموع ۴۰۴ حالت گسسته قابل دسترسی را به دست می دهد.

توضیح پیاده سازی

کد های مربوط به پیاده سازی این سوال در فایل `Codes.ipynb/html` قسمت **Question 2** قرار دارد. برای بدست آوردن شماره این حالت، روی تمام حالات پیمایش می کنیم و تابع `decode` را بر روی آن حالت فراخوانی می کنیم. اگر مقدار `pass_idx` و `dest_idx` در خروجی تابع `decode` با هم برابر بود، یعنی مسافر در مقصد قرار دارد. پس آن حالت دست نیافتی است. بعد پیدا کردن تمام این حالات شماره آنها را چاپ می کنیم.

روند اجرای کد پیاده سازی

برای اجرا کد های این سوال لازم است تا تمام کد های ماقبل این بخش و کد های این سوال را اجرا کنید.

سوال ۳

هدف سوال

در این سوال قرار است مسئله را با استفاده از دو الگوریتم Q-Learning و n-step Tree Backup حل کنیم. برای پیاده سازی الگوریتم ها مطابق شبه کد آن در کتاب ساتون بارتو عمل می کنیم:

***n*-step Tree Backup for estimating $Q \approx q_*$ or q_π**

```

Initialize  $Q(s, a)$  arbitrarily, for all  $s \in \mathcal{S}, a \in \mathcal{A}$ 
Initialize  $\pi$  to be greedy with respect to  $Q$ , or as a fixed given policy
Algorithm parameters: step size  $\alpha \in (0, 1]$ , a positive integer  $n$ 
All store and access operations can take their index mod  $n + 1$ 

Loop for each episode:
  Initialize and store  $S_0 \neq \text{terminal}$ 
  Choose an action  $A_0$  arbitrarily as a function of  $S_0$ ; Store  $A_0$ 
   $T \leftarrow \infty$ 
  Loop for  $t = 0, 1, 2, \dots$ :
    If  $t < T$ :
      Take action  $A_t$ ; observe and store the next reward and state as  $R_{t+1}, S_{t+1}$ 
      If  $S_{t+1}$  is terminal:
         $T \leftarrow t + 1$ 
      else:
        Choose an action  $A_{t+1}$  arbitrarily as a function of  $S_{t+1}$ ; Store  $A_{t+1}$ 
     $\tau \leftarrow t + 1 - n$  ( $\tau$  is the time whose estimate is being updated)
    If  $\tau \geq 0$ :
      If  $t + 1 \geq T$ :
         $G \leftarrow R_T$ 
      else
         $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a)$ 
      Loop for  $k = \min(t, T - 1)$  down through  $\tau + 1$ :
         $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k)Q(S_k, a) + \gamma \pi(A_k|S_k)G$ 
       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$ 
      If  $\pi$  is being learned, then ensure that  $\pi(\cdot|S_\tau)$  is greedy wrt  $Q$ 
  Until  $\tau = T - 1$ 

```

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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 $Q(\text{terminal}, \cdot) = 0$
Loop for each episode:
 Initialize S
 Choose A from S using policy derived from Q (e.g., ε -greedy)
 Loop for each step of episode:
 Take action A , observe R, S'
 Choose A' from S' using policy derived from Q (e.g., ε -greedy)
 $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$
 $S \leftarrow S'; A \leftarrow A';$
 until S is terminal

الگوریتم n-step Tree Backup را به ازای چهار مقدار ۱، ۲، ۳ و ۴ برای n اجرا می کنیم.

توضیح پیاده سازی

کد های مربوط به پیاده سازی این سوال در فایل [Codes.ipynb/html](https://codes.ipynb/html) قسمت **Question 3** قرار دارد.

- در کلاس **SARSA** الگوریتم **SARSA** پیاده سازی شده است.
 - کاهش مقدار اپسیلون در تابع **decay_epsilon** این کلاس صورت می گیرد. در اولین اپیزود مقدار اپسیلون برابر ۱ است و در اپیزود های بعدی مقدار اپسیلون برابر رابطه زیر قرار می گیرد:

$$\text{Epsilon} = e^{-(\text{epsilon decay rate} * \text{episode number})}$$

که مقدار **decay_rate** برابر ۰.۰۰۳ است.

- تابع **agent_SARSA_run** الگوریتم **SARSA** را با ۲۰ بار تکرار و به تعداد ۲۰۰۰ اپیزود اجرا می کند و پاداش های دریافت شده و پاداش سیاست بهینه را بر می گرداند.
- تابع **run_optimal_policy** برای پیدا کردن پاداش حاصل از اجرا سیاست بهینه استفاده می شود.
- تابع **run_20_episode** برای اجرا عامل آموزش داده شده به اندازه ۲۰ اپیزود استفاده می شود.
- در کلاس **NStepTreeBackup** الگوریتم n-step Tree Backup پیاده سازی شده است.
 - کاهش مقدار اپسیلون در تابع **decay_epsilon** این کلاس صورت می گیرد. در اولین اپیزود مقدار اپسیلون برابر ۱ است و در اپیزود های بعدی مقدار اپسیلون برابر رابطه زیر قرار می گیرد:

$$\text{Epsilon} = e^{-(\text{epsilon decay rate} * \text{episode number})}$$

که مقدار **decay_rate** برابر ۰.۰۰۳ است.

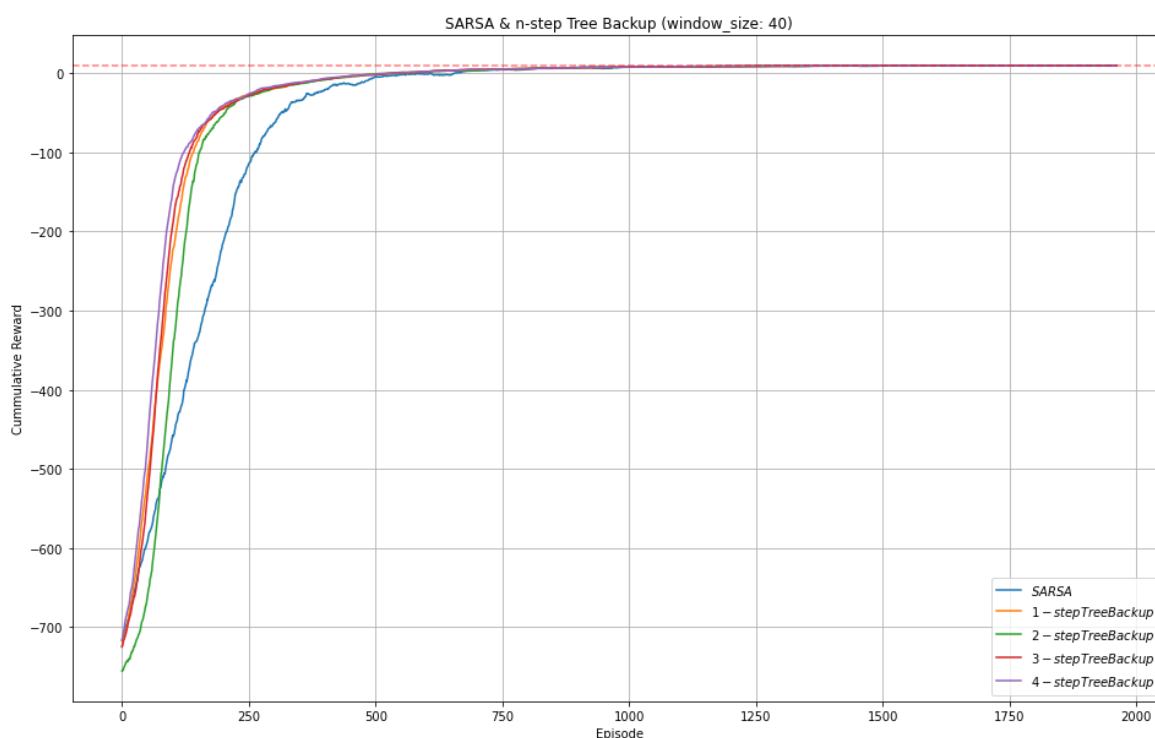
- تابع `agent_n_step_Tree_Backup_run` الگوریتم `n-step Tree Backup` را با ۲۰ بار تکرار و به تعداد ۲۰۰۰ اپیزود با توجه به مقدار `n` ورودی اجرا می کند و پاداش های دریافت شده و پاداش سیاست بهینه را بر می گرداند.
- تابع `run_optimal_policy` برای پیدا کردن پاداش حاصل از اجرا سیاست بهینه استفاده می شود.
- تابع `run_20_episode` برای اجرا عامل آموزش داده شده به اندازه ۲۰ اپیزود استفاده می شود.

روند اجرای کد پیاده سازی

برای اجرا کد های این بخش سوال است تا کد های بخش ابتدایی و کد های این سوال را اجرا کنید.

نتایج

نتایج حاصل از اجرای الگوریتم به شکل زیر است:



مشاهده می شود که هر دو الگوریتم به سیاست بهینه همگرا می شوند اما الگوریتم `n-step Tree Backup` نسبت به الگوریتم `SARSA` دارای حسرت کمتری است و مقدار پاداش دریافتی آن در هر اپیزود بیشتر است؛ همچنین سرعت همگرایی بالاتری نیز دارد و الگوریتم زودتر به سیاست بهینه همگرا می شود. همچنین در الگوریتم `n-step Tree Backup` هر چه مقدار `n` بیشتر می شود، سرعت همگرایی الگوریتم بالاتر و میزان حسرت در هر اپیزود کمتر می شود و الگوریتم از نقطه بهتری یادگیری را آغاز می کند.

عملکرد عامل بهینه پس از آموزش به اندازه ۲۰ ایزود به شکل زیر است:

+-----+			
R:	:	:G	
:	:	:	
:	:	:	
:	:	:	
Y	:	B:	
+-----+			

سوال ۴

هدف سوال

در این سوال قرار است مسئله را با استفاده از الگوریتم On-Policy MC حل کنیم. برای پیاده سازی الگوریتم مطابق شبه کد آن در کتاب ساتون بار تو عمل می کنیم:

On-policy first-visit MC control (for ε -soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε -soft policy
 $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
 $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

مسئله را یکبار با اپسیلون ثابت و یکبار با اپسیلون کاهشی حل می کنیم و نتایج را تحلیل می کنیم.

توضیح پیاده سازی

کد های مربوط به پیاده سازی این سوال در فایل [Codes.ipynb/html](#) قسمت **Question 4** قرار دارد.

- در کلاس **On_Policy_MC** الگوریتم On-Policy MC پیاده سازی شده است.

○ کاهش مقدار اپسیلون در تابع **decay_epsilon** این کلاس صورت می گیرد. در اولین

اپیزود مقدار اپسیلون برابر ۱ است و در اپیزود های بعدی مقدار اپسیلون برابر رابطه زیر

قرار می گیرد:

$$\text{Epsilon} = e^{-(\text{epsilon decay rate} * \text{episode number})}$$

که مقدار **decay_rate** برابر ۰.۰۰۳ است.

- تابع `agent_On_Policy_MC_run` الگوریتم On-Policy MC را با ۲۰ بار تکرار و به تعداد ۲۰۰۰ اپیزود با توجه به اپسیلون ثابت یا کاهشی اجرا می کند و پاداش های دریافت شده و پاداش سیاست بهینه را بر می گرداند.
- تابع `generate_episode` به ایجاد یک اپیزود با توجه به سیاست π می پردازد.
- تابع `run_optimal_policy` برای پیدا کردن پاداش حاصل از اجرا سیاست بهینه استفاده می شود.
- تابع `run_20_episode` برای اجرا عامل آموزش داده شده به اندازه ۲۰ اپیزود استفاده می شود.

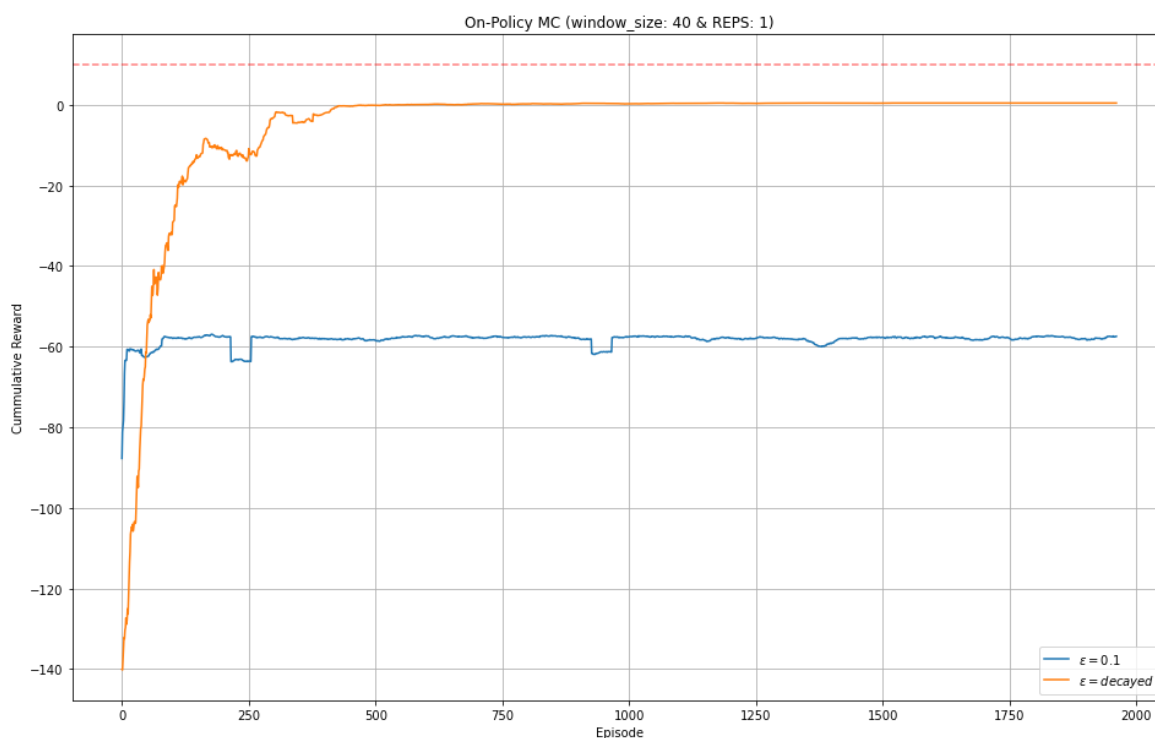
روند اجرای کد پیاده سازی

برای اجرا کد های این بخش سوال است تا کد های بخش ابتدایی و کد های این سوال را اجرا کنید.

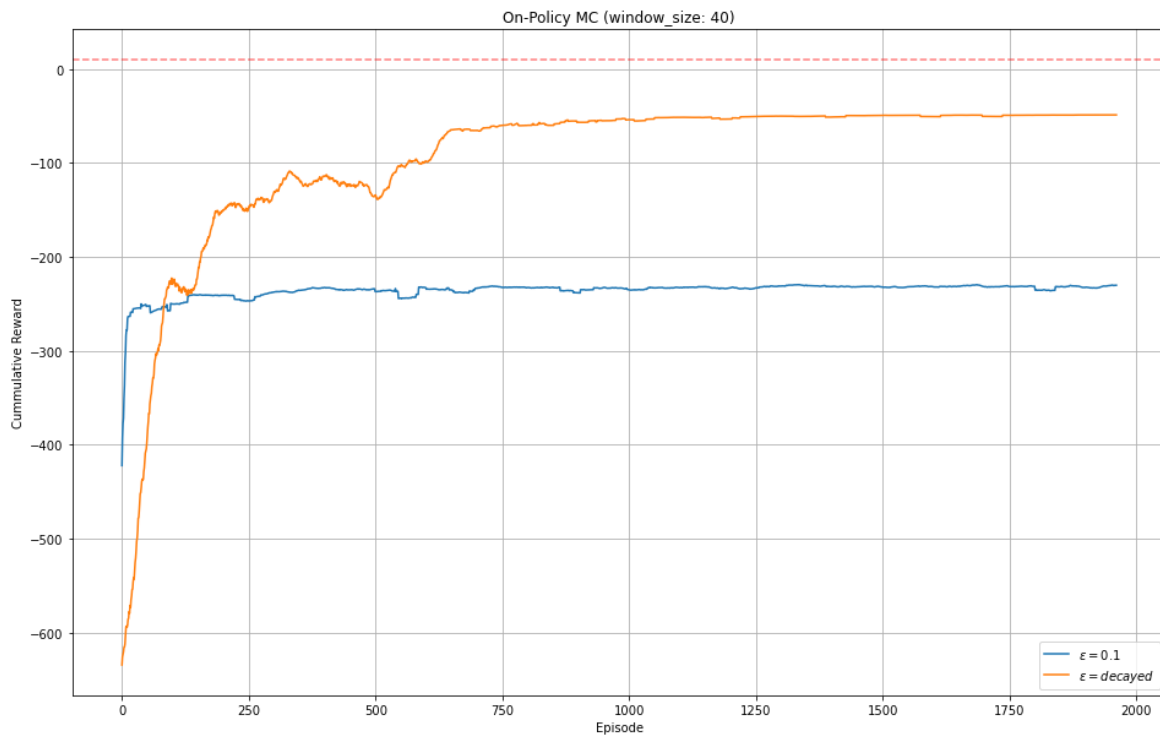
نتایج

نتایج حاصل از اجرای الگوریتم در دو حالت اپسیلون ثابت و کاهشی به شکل زیر است:

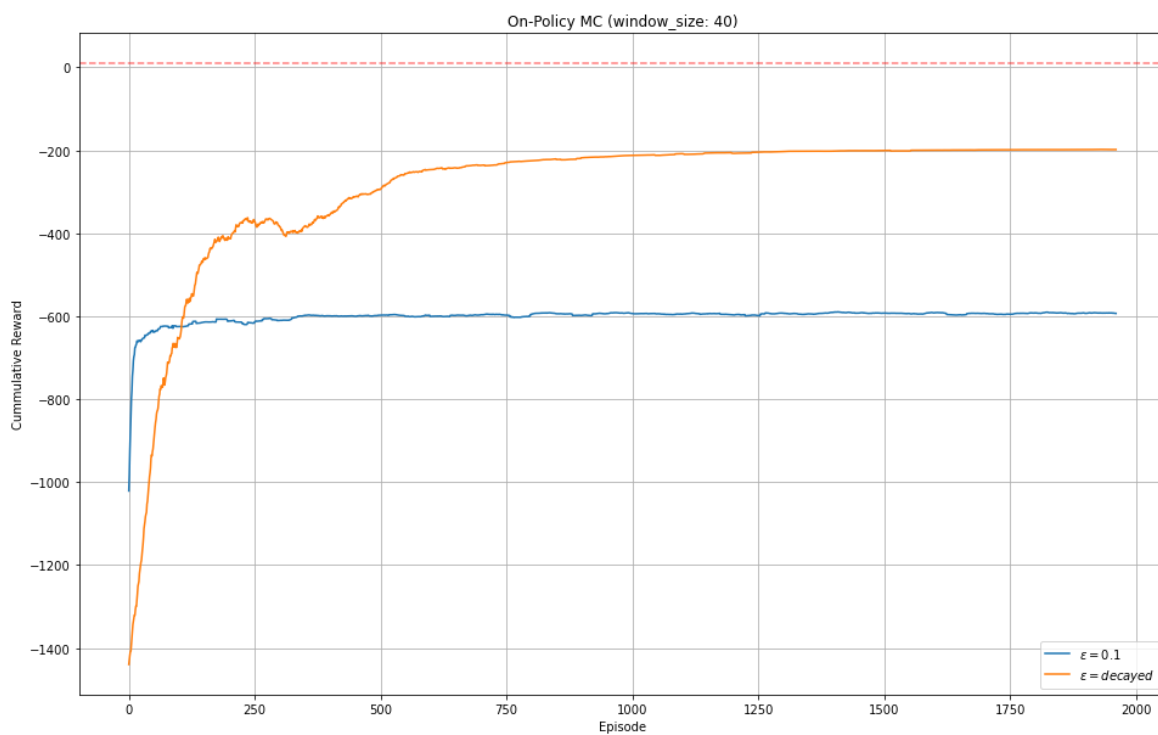
- REPS = 1



- REPS = 4



REPS = 10 •

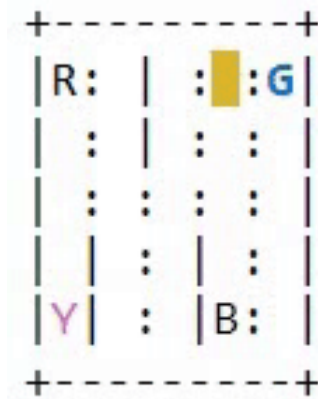


در حالتی که اپسیلون ثابت است، الگوریتم به سیاست بهینه همگرا نمی شود اما در حالتی که اپسیلون کاهشی است، الگوریتم به سیاست بهینه همگرا می شود. علت می تواند آن باشد که در حالت اپسیلون ثابت، عامل نمی تواند به خوبی محیط را explore کند زیرا مقدار کم اپسیلون

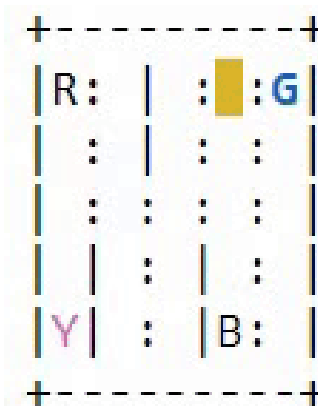
ثابت باعث می شود تا عامل بیشتر exploit کند؛ در نتیجه عامل نمی تواند سیاست راندم اولیه را به خوبی بهبود دهد و در مینیمم محلی گیر می کند. اما در حالتی اپسیلون کاهشی است، عامل ابتدا با احتمال بالاتری explore می کند و به مرور زمان رو به exploit می آورد و با greedy عمل کردن، به سیاست بهینه همگرا می شود. سرعت همگرایی این الگوریتم در هر دو حالت نسبت به الگوریتم های سوال های قبل کمتر است اما در حالتی اپسیلون ثابت است، الگوریتم زودتر همگرا می شود چون به دلیل عدم توانایی در اکتشاف بیشتر محیط، الگوریتم بعد از مدتی بهبود نمی یابد.

عملکرد عامل بهینه پس از آموزش به اندازه ۲۰ اپیزود به شکل زیر است:

- اپسیلون ثابت:



- اپسیلون کاهشی:



سوال ۵

سرعت یادگیری در سوال آخر نسبت به سوال های پیشین کمتر است. علت آن است که این الگوریتم از اطلاعات گرادیان استفاده نمی کنند. این بدان معنی است که، در مقایسه با تکنیک های تمایز یا استنتاج متغیر، برای نزدیک شدن به پاسخ های صحیح، به اجرای بسیار بیشتری از مدل نیاز دارند. همچنین وقتی تعداد زیادی متغیر محدود به محدودیت های مختلف داریم، برای تقریب یک راه حل با استفاده از این روش، به زمان و محاسبات زیادی نیاز است. اگر پارامترها و محدودیت های ضعیفی به مدل وارد شوند، نتایج ضعیف به عنوان خروجی داده می شود. [8] [7]

امتیازی

برای افزایش سرعت این الگوریتم می توان از راهکار های زیر استفاده کرد:

- ۱- با زیاد کردن پاداش رساندن مسافر به مقصد (کم کردن جریمه اتلاف وقت) مدل سریع تر آموزش می بیند. منطق آن است که مدل ابتدا یاد بگیرد تا از اعمال جریمه دار (سوار و پیاده کردن اشتباه مسافر) دوری کند و سپس به فکر راه نزدیک باشد و این مسئله حسرت را کاهش می دهد.
 - ۲- می توان از تکنیک های کاهش واریانس برای کاهش تعداد تکرارهای مورد نیاز استفاده کرد. [9]
- ما از روش اول برای افزایش سرعت استفاده می کنیم. بدین صورت که پاداش رسیدن به مقصد را به جای ۲۰ برابر ۷۰ قرار می دهیم.

توضیح پیاده سازی

کد های مربوط به پیاده سازی این سوال در فایل `Codes.ipynb/html` قسمت **Question 5** قرار دارد.

- در کلاس `On_Policy_MC_v2` الگوریتم `On-Policy MC v2` پیاده سازی شده است.
 - کاهش مقدار اپسیلون در تابع `decay_epsilon` این کلاس صورت می گیرد. در اولین اپیزود مقدار اپسیلون برابر ۱ است و در اپیزود های بعدی مقدار اپسیلون برابر رابطه زیر قرار می گیرد:

$$\text{Epsilon} = e^{-(\text{epsilon decay rate} * \text{episode number})}$$

که مقدار `decay_rate` برابر ۰.۰۰۳ است.

- تابع `agent_On_Policy_MC_v2_run` الگوریتم `On-Policy MC v2` را با ۲۰ بار تکرار و به تعداد ۲۰۰۰ اپیزود با توجه به اپسیلون ثابت یا کاهشی اجرا می کند و پاداش های دریافت شده و پاداش سیاست بهینه را بر می گرداند.

- تابع `generate_episode_v2` به ایجاد یک اپیزود با توجه به سیاست π می پردازد.
- تابع `run_optimal_policy_v2` برای پیدا کردن پاداش حاصل از اجرا سیاست بهینه استفاده می شود.
- تابع `run_20_episode_v2` برای اجرا عامل آموزش داده شده به اندازه ۲۰ اپیزود استفاده می شود.

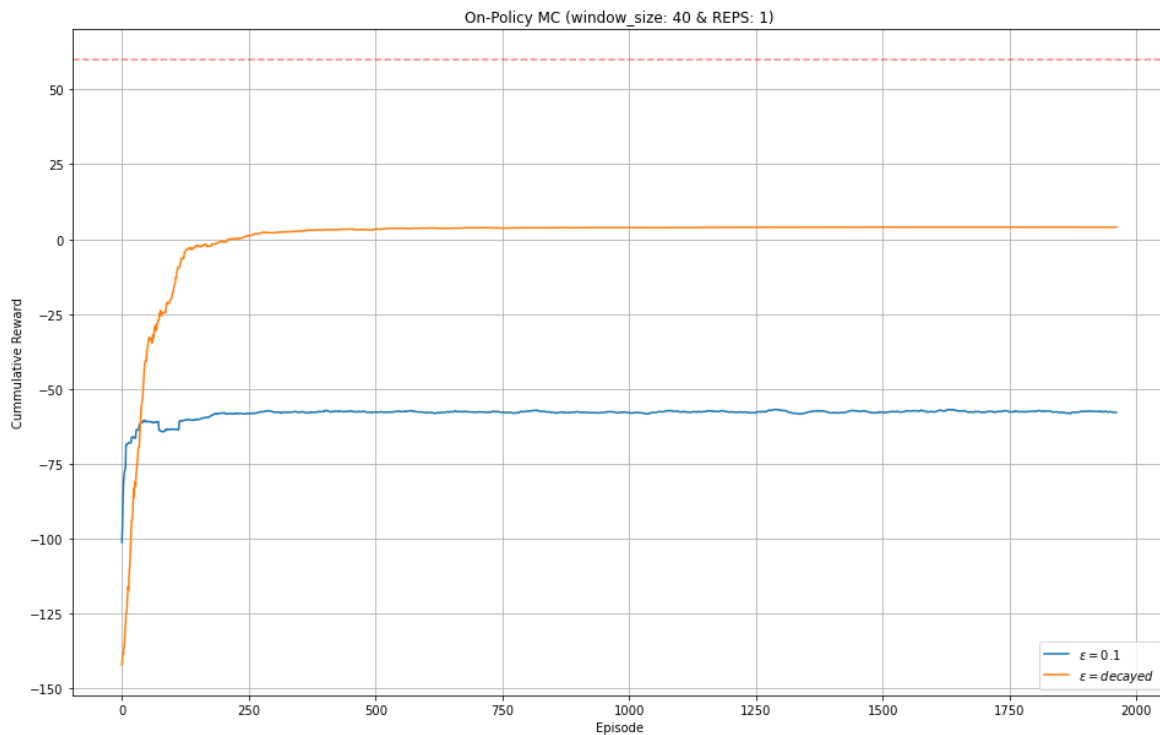
روند اجرای کد پیاده سازی

برای اجرا کد های این بخش سوال است تا کد های بخش ابتدایی و کد های این سوال را اجرا کنید.

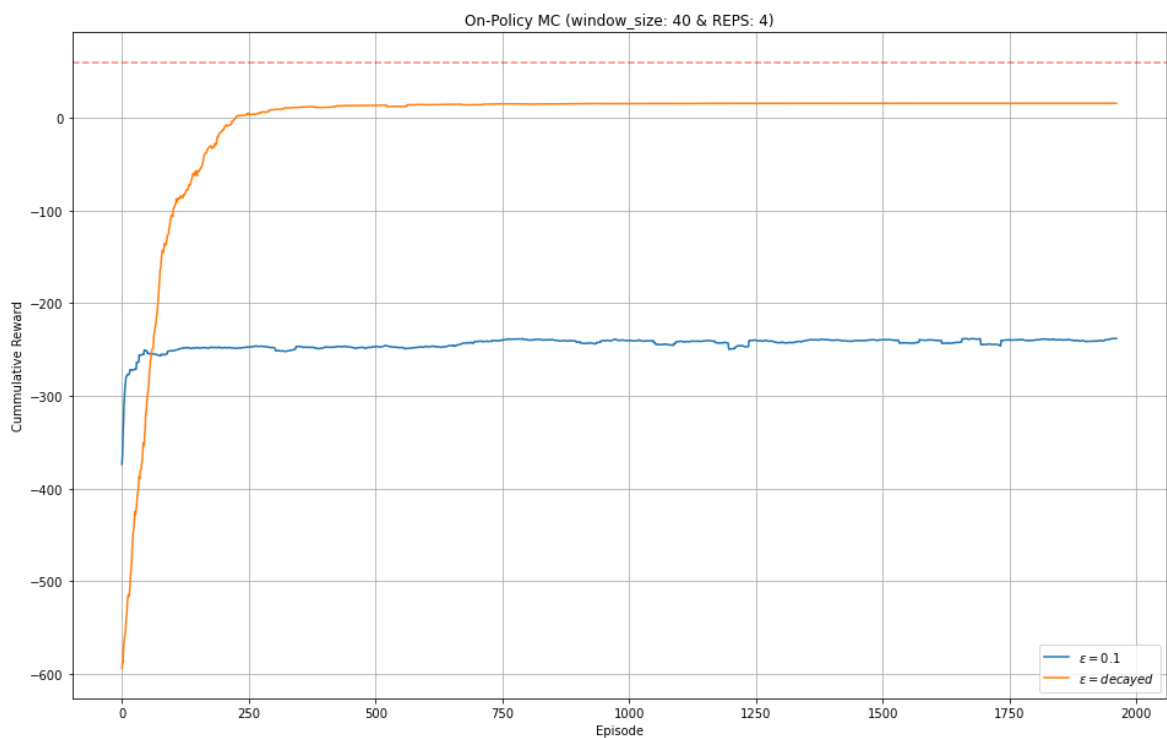
نتایج

نتایج حاصل از اجرای الگوریتم در دو حالت اپسیلون ثابت و کاهشی به شکل زیر است:

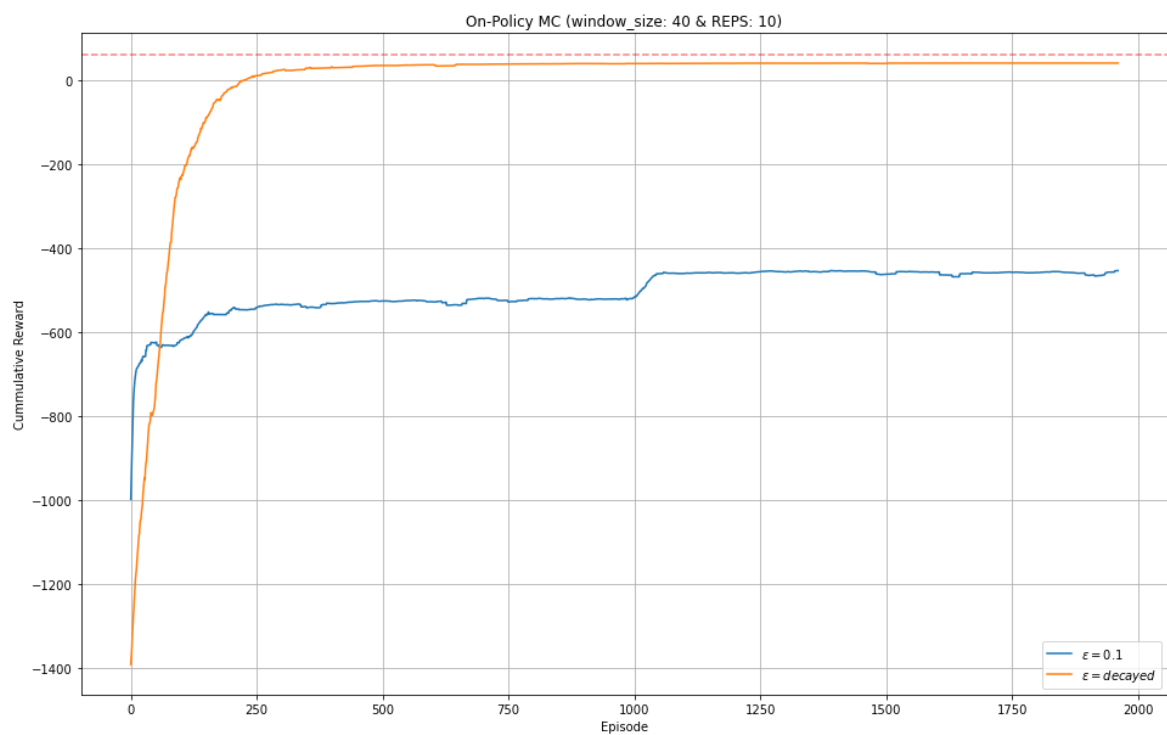
- REPS = 1



- REPS = 4




REPS = 10 •



مشاهده می شود که در هر سه حالت، سرعت یادگیری در این سوال نسبت به سوال قبل بیشتر شده است و الگوریتم ها در تعداد اپیزود کمتری به یک سیاست همگرا می شوند. علت این امر تصمیم اتخاذ شده در مورد افزایش پاداش رسیدن به مقصد می باشد.

عملکرد عامل بهینه پس از آموزش به اندازه ۲۰ ایزود:

+-----+				
R:		:  :	G	
:		:	:	
:	:	:	:	
	:	:		
Y	:	B:		
+-----+				

- 1- <https://jochemsoons.medium.com/a-comparison-between-sarsa-and-expected-sarsa-66b931202c75>
- 2- <https://www.cs.ox.ac.uk/people/shimon.whiteson/pubs/vanseijenadprl09.pdf>
- 3- <https://arshren.medium.com/reinforcement-learning-expected-sarsa-bd12f8fa0e42>
- 4- <https://medium.com/analytics-vidhya/q-learning-expected-sarsa-and-comparison-of-td-learning-algorithms-e4612064de97>
- 5- <http://www.incompleteideas.net/book/7/node2.html>
- 6- <https://ai.stackexchange.com/questions/9396/how-do-we-prove-the-n-step-return-error-reduction-property>
- 7- <https://www.quora.com/Why-are-MCMC-methods-considered-slow>
- 8- <https://towardsdatascience.com/monte-carlo-method-explained-8635edf2cf58>
- 9- https://en.wikipedia.org/wiki/Variance_reduction#Crude_Monte_Carlo_simulation