بخش تحلیلی

سوال اول

شبه کد الگوریتم SARSA به شرح زیر است:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

شبه کد الگوریتم Q-Learning به شرح زیر است:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (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

با توجه به اینکه سیاست A نسبت به سیاست B به ازای گام های کمتری به حالت هدف رسیده است پاداش بالاتری دریافت کرده است. Q-Learning یک روش off-policy است و در مقابل SARSA یک روش on-policy است. انتظار می رود سیاست A ناشی از الگوریتم Q-Learning باشد زیرا این الگوریتم همواره به دنبال یافتن optimal policy است در حالی که SARSA ممکن است به sub-optimal policy همگرا شود.
علت این امر این است که روش های off-policy در exploration نقطه قوت بالایی نسبت به روش on-policy دارند و این موضوع سبب می شوند در یافتن سیاست بهینه عملکرد بهتری داشته باشند.

سوال دوم

الف

اگر $r_s = -1$ آنگاه عامل به ازای هر عمل در non-terminal states جریمه می شود و سعی میکند با کوتاه ترین مسیر به goal state برسد.
در این سوال از value iteration برای یافتن مقدار بهینه ارزش هر state استفاده می کنیم:

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

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

Iteration =1:

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Iteration =2:

-1	-1	-1	-1
-5	-1	-1	-1
-1	-1	-1	5
-1	-1	1-	-1

Iteration =3:

-2	-2	-2	-2
-5	-2	-2	4
-2	-2	4	5
-2	-2	-2	-2

Iteration =4:

-3	-3	-3	3
-5	-3	3	4
-3	3	4	5
-3	-3	-3	-3

Iteration =5:

-4	-4	2	3
-5	2	3	4
2	3	4	5
-4	-4	-4	-4

Iteration =6:

-5	1	2	3
-5	2	3	4
2	3	4	5
1	-5	-5	-5

Iteration =7:

0	1	2	3
-5	2	3	4
2	3	4	5
1	0	-6	-6

Iteration =8:

0	1	2	3
-5	2	3	4
2	3	4	5
1	0	-1	-7

Iteration =9:

0	1	2	3
-5	2	3	4
2	3	4	5
1	0	-1	-2

Optimal policy:

→	→	→	↓
Hell	→	→	↓
→	→	→	Goal
↑	←	←	←

ب.

$c = 3 \rightarrow r_s = 2,\ r_r = -2,\ r_g = 8$

Iteration =1:

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Iteration =2:

2	2	2	2
-2	2	2	2
2	2	2	8
2	2	2	2

Iteration =3:

4	4	4	4
-2	4	4	10
4	4	10	8
4	4	4	4

Iteration =4:

6	6	6	12
-2	6	12	12
6	12	12	8
6	6	6	6

Iteration =5:

8	8	14	14
-2	14	14	14
14	14	14	8
8	8	8	8

Iteration =6:

10	16	16	16
-2	16	16	16
16	16	16	8
16	10	10	10

Iteration =7:

18	18	18	18
-2	18	18	18
18	18	18	8
18	18	12	12

Iteration =8:

20	20	20	20
-2	20	20	20
20	20	20	8
20	20	20	12

Iteration =9:

22	22	22	22
-2	22	22	22
22	22	22	8
22	22	22	22

…

Optimal policy:

┼	┼	┼	┼
Hell	├	┼	┴
┬	┼	┤	Goal
┼	┼	┼	┼

ارزش مربع های سفید چندین برابر می شود و این نشان می دهد هر عملی به جز رفتن به terminal state ارزش بالاتری دارد و عامل سعی می کند در طول زمان با اجتناب از رفتن به terminal state ها از محیط پاداش کسب کند. فرض شد عامل محدودیت زمانی در محیط ندارد در غیر اینصورت بسته به میزان زمان، سیاست بهینه تغییر می کرد.

ج

هر چه discount factor نزدیک به صفر انتخاب شود عامل اصطلاحا myopic و هر چه به یک نزدیک تر انتخاب شود عامل اصطلاحا far-sighted یا non-myopic است.

در بخش قبل دیدیم discount factor نزدیک یک سبب می شود عامل برای پاداشی که در آینده دریافت می کند ارزش بیشتری قائل باشد و در نتیجه از terminal states پرهیز کند؛ در مقابل discount factor نزدیک صفر سبب می شود عامل صرفا پاداش لحظه ای را در نظر بگیرد؛ و در نتیجه تمام مربع های سفید دارای ارزش یکسان 2 واحدی خواهند بود. عامل در مجاورت terminal states عمل بهینه را انتخاب و در دیگر حالات random انتخاب می کند.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
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 $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

$$V_{old}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_{old}(s')]$$

$$r \leftarrow r + c$$

$$\Rightarrow V_{new}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + c + \gamma V_{new}(s')]$$

$$\Rightarrow V_{new}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + c + \gamma[\sum_a \pi(a|s') \sum_{s'',r} p(s'', r|s', a)[r + c + \gamma V_{new}(s'')]]$$

$$\Rightarrow V_{new}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma c + \gamma^2 c + \ldots + \gamma V_{old}(s')]$$

$$\Rightarrow V_{new}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \sum_{n=1}^{N} \gamma^n c + \gamma V_{old}(s')]$$

$$V_{new}^{\pi}(s) = \sum_{n=1}^{N} \gamma^n c + V_{old}^{\pi}(s)$$

سوال سوم

شبه کد الگوریتم SARSA به شرح زیر است:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

در الگوریتم Expected SARSA، قاعده بروزرسانی مطابق زیر تفاوت دارد:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma E_{\pi}[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)]$$

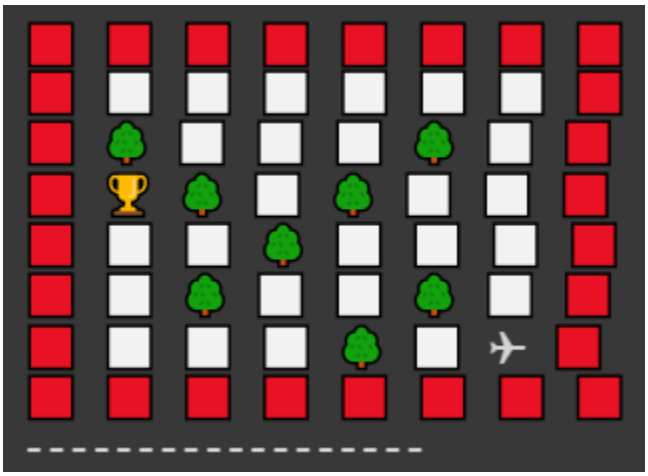
Trade-off ای بین compute-time و sample efficiency وجود دارد. در یک طرف، Expected SARSA پیچیده تر است و در نتیجه محاسبات کند تری از SARSA دارد و در طرف دیگر Expected SARSA به ازای تجربه یکسان، معمولا SARSA را outperform می کند.

- Expected SARSA در یافتن سیاست e-optimal الگوریتم SARSA را outperform می کند، زیرا نسبت به شرایط نویزی موجود در محیط و سیاست robust تر است.
- هر دو توانایی همگرایی به سیاست بهینه را داند اما Expected SARSA در Horizon کوتاه تری همگرا می شود و معمولا SARSA را outperform می کند.

منبع

بخش پیاده سازی

- در پیاده سازی محیط مکان drone به عنوان observation عامل در نظر گرفته شده است.
- عامل اگر به کمتر از 5 درصد باتری و یا 15 درصد سلامت خود برسد شبیه سازی truncated و اگر به هدف برسد شبیه سازی terminated می شود.
- نمونه ای از render محیط در ادامه گزارش شده است.



.1

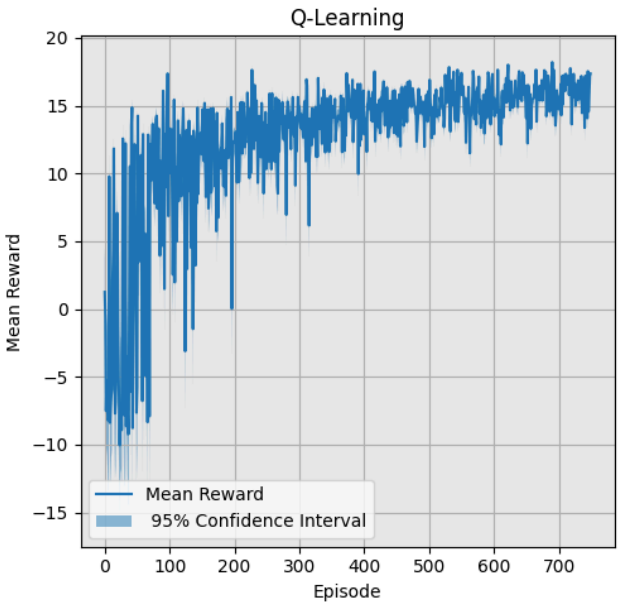
شبه کد الگوریتم Q-learning به شرح زیر است:

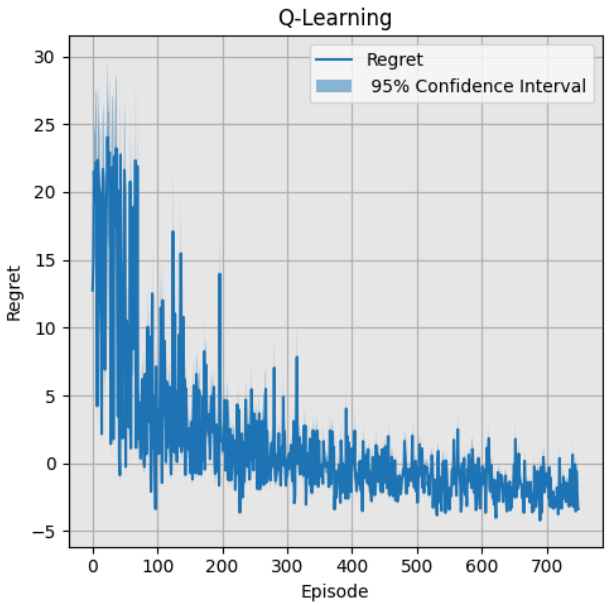
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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 $Q(terminal, \cdot) = 0$

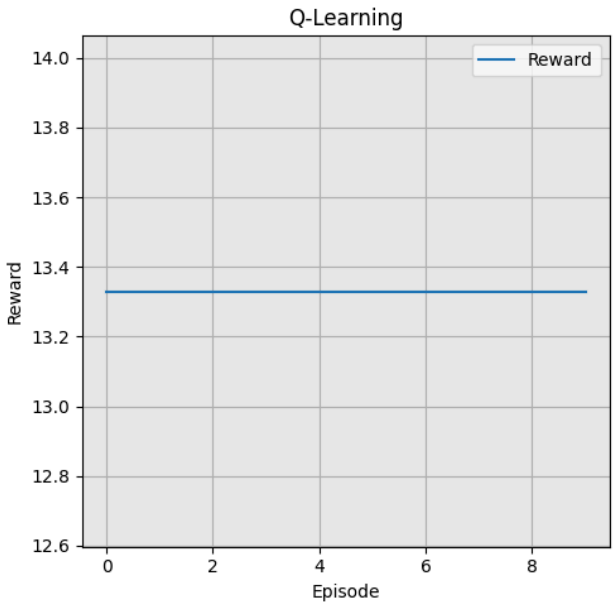
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

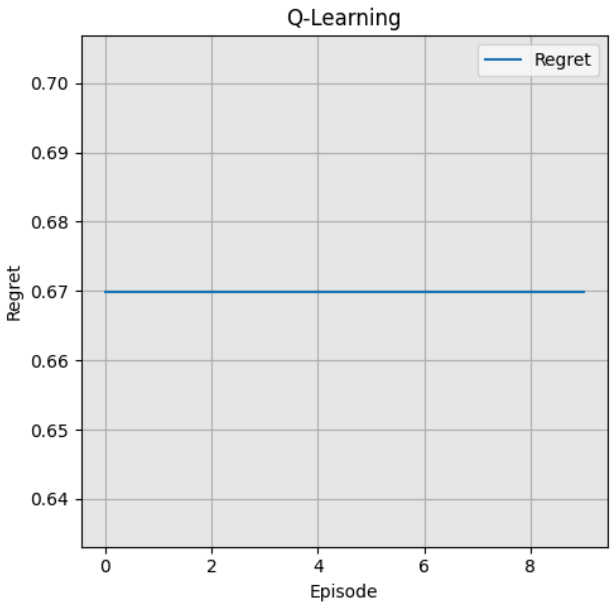
به ازای α=0.1 ثابت نتایج آموزش الگوریتم مطابق زیر است:
اپسیلون به صورت نمایی با نرخ 0.999 کاهش یافته است.



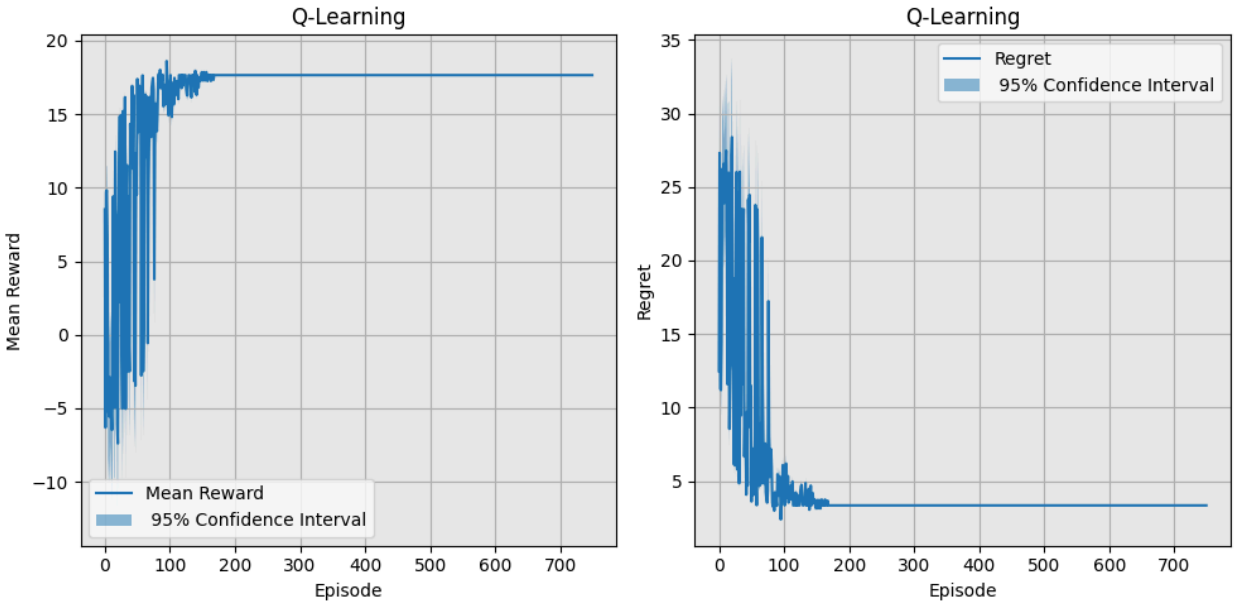


همچنین به ازای 10 اپیزود تست عامل نتایج به شرح زیر است:

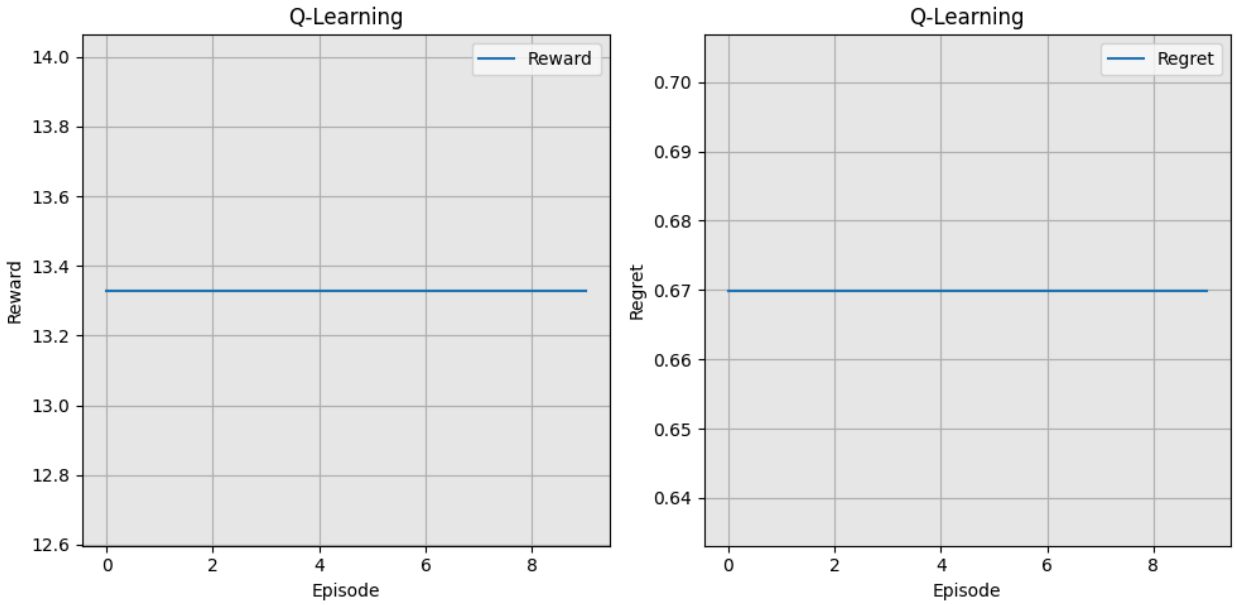




به ازای α=0.1 و نرخ کاهش نمایی 0.99 نیز نتایج الگوریتم به شرح زیر است:



همچنین به ازای 10 اپیزود تست عامل، نتایج به شرح زیر است:



مشاهده می شود به ازای کاهش نرخ یادگیری، سرعت همگرایی و پشیمانی کاهش می یابد.

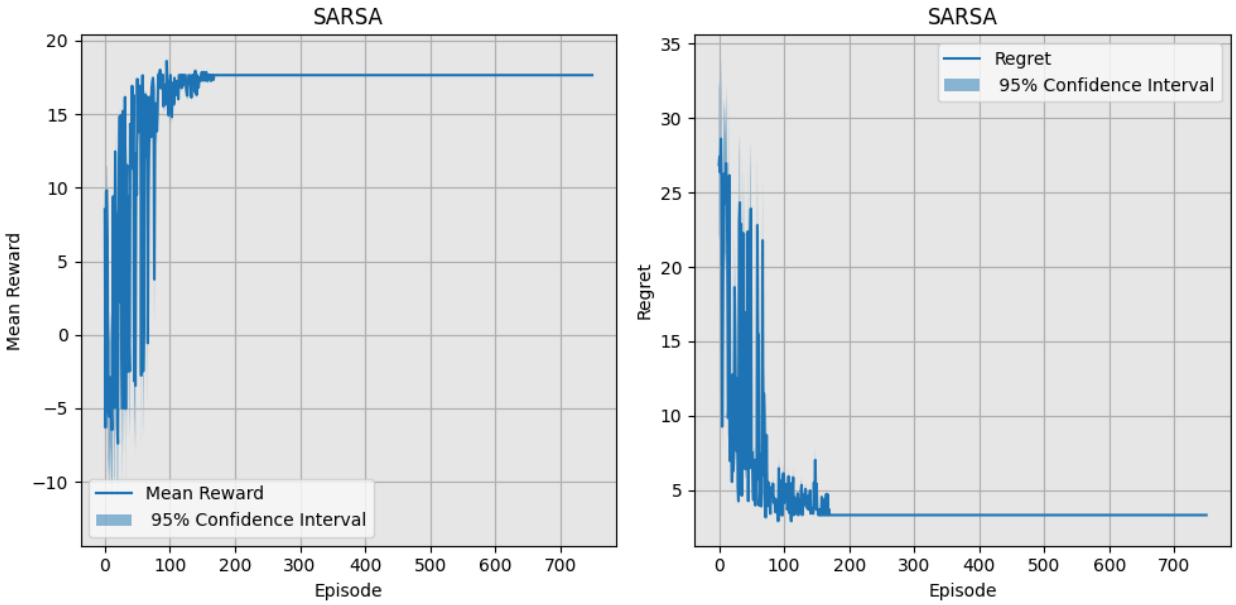
- توجه کنید متفاوت بودن طول هر اپیزودو تصادفی بودن محیط سبب می شود تخمین دقیق از مجموع پاداش بهینه دشوار باشد.
- اپسیلون نیز با نرخ کاهش نمایی 0.99 در طول یادگیری کاهش یافته است.

.2

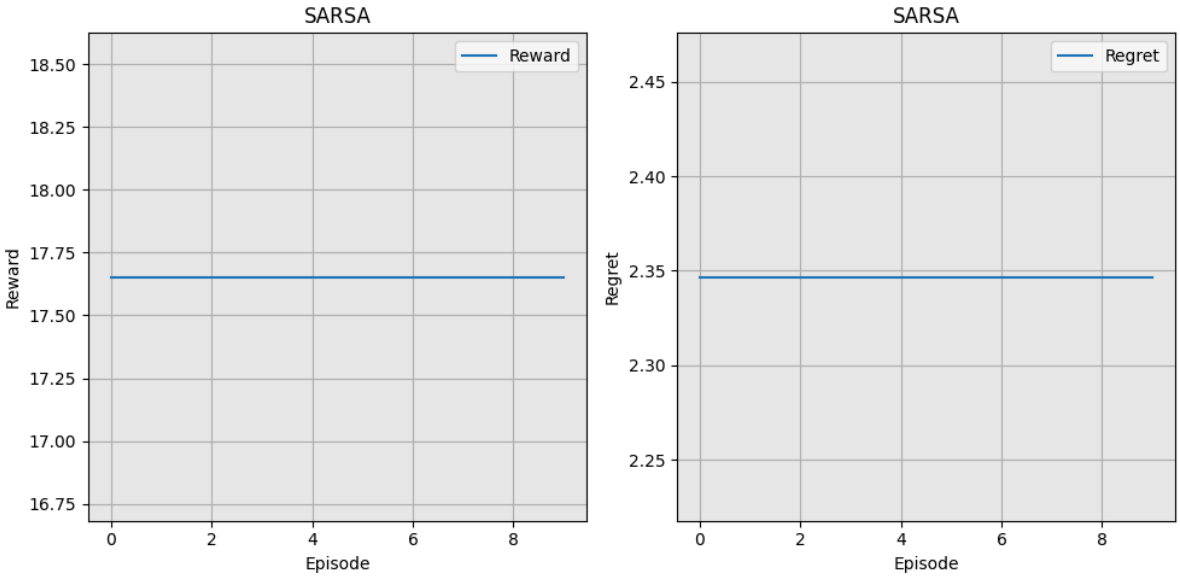
شبه کد الگوریتم SARSA به شرح زیر است:

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

نتایج آموزش این الگوریتم مطابق زیر است:



نتایج تست عامل در 10 اپیزود مطابق زیر است:



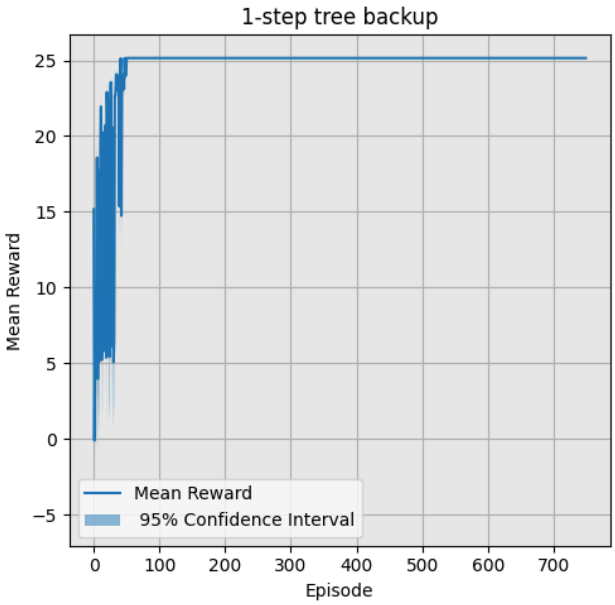
شبه کد الگوریتم n-step Tree Backup به شرح زیر است:

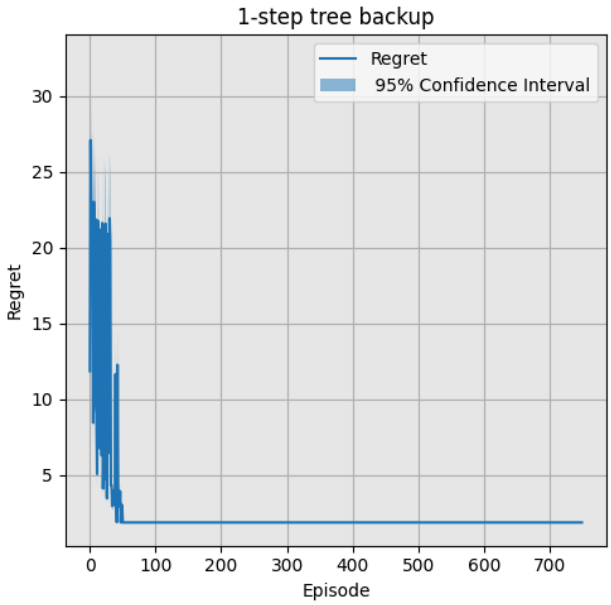
n-step Tree Backup for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π 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 $n+1$

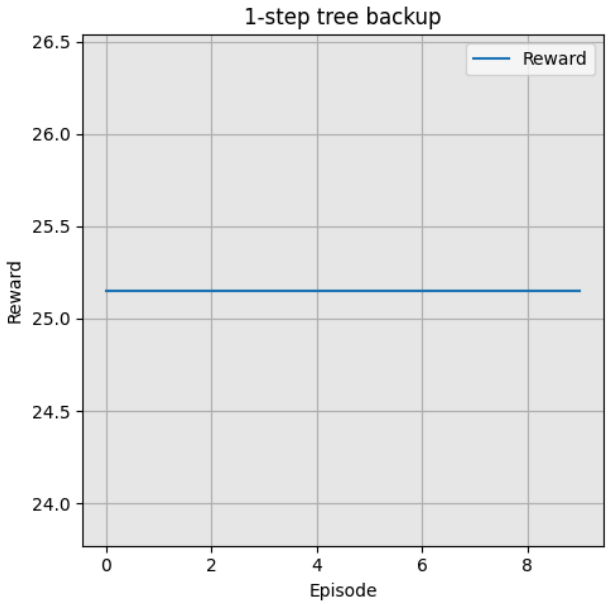
Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose an action A_0 arbitrarily as a function of S_0; Store A_0
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$:
  | Take action A_t; observe and store the next reward and state as R_{t+1}, S_{t+1}
  | If S_{t+1} 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}
  | $\tau \leftarrow t+1-n$ (τ 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 $\tau+1$:
  | $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau) \right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T-1$

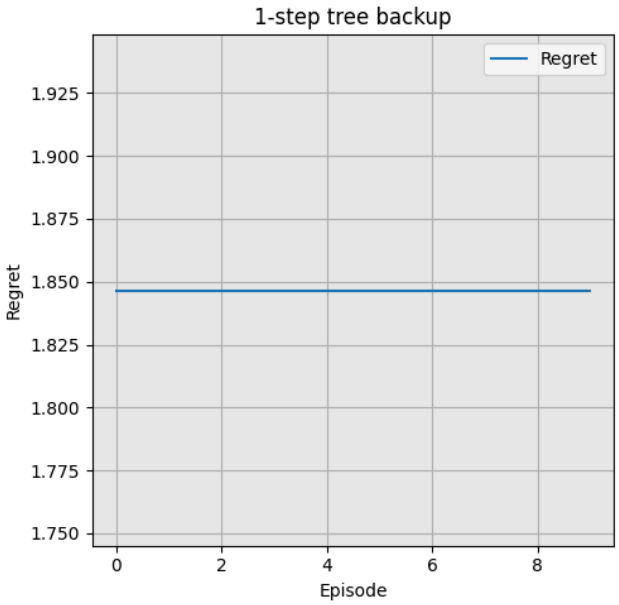
نتیجه یادگیری الگوریتم one step tree back up به شرح زیر است:



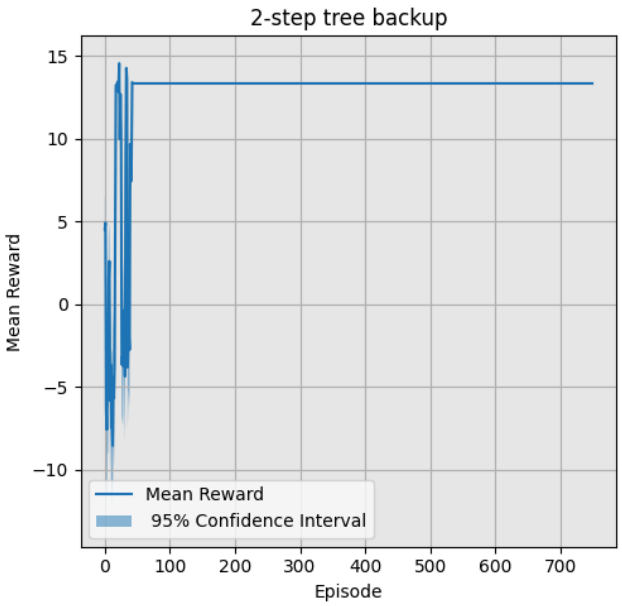


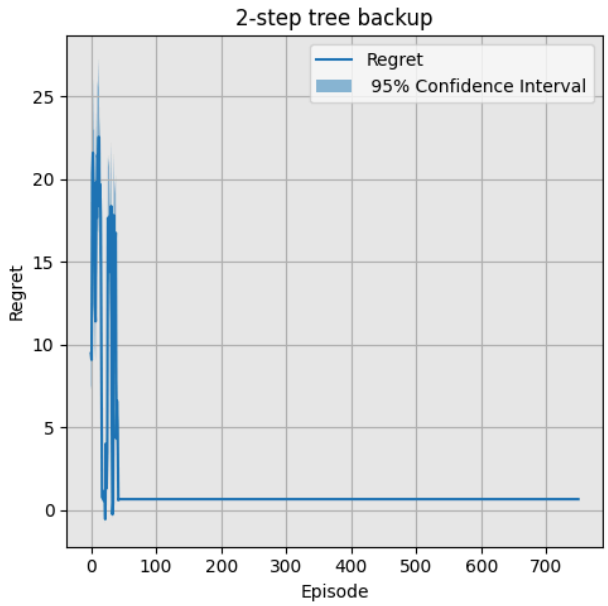
و نتیجه تست عامل به طول 10 اپیزود مطابق زیر است:



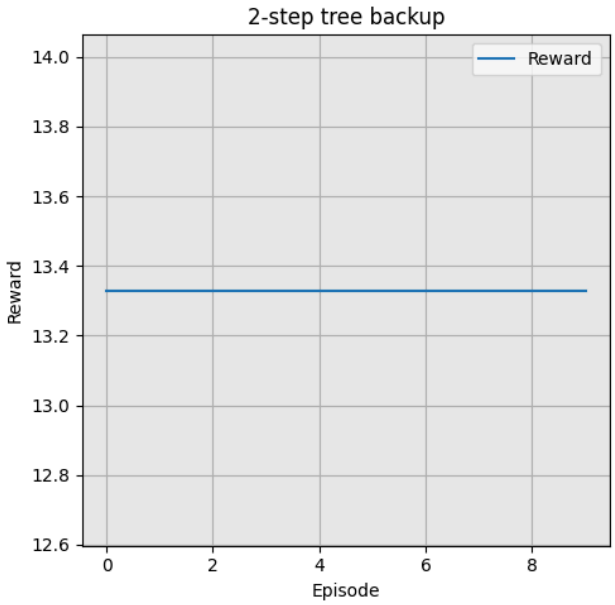


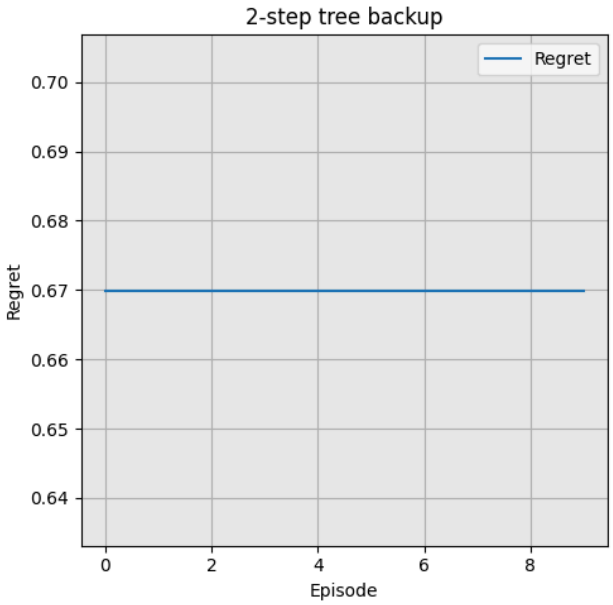
نتیجه یادگیری الگوریتم two step tree backup مطابق زیر است:



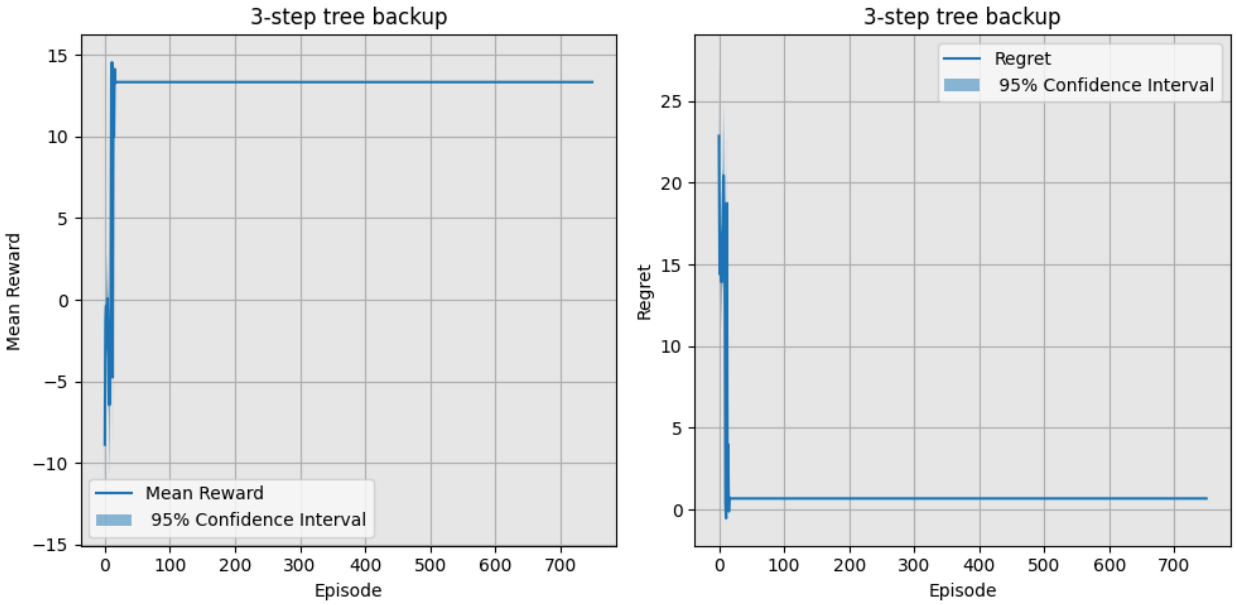


عامل در 10 اپیزود مطابق زیر تست شده است:

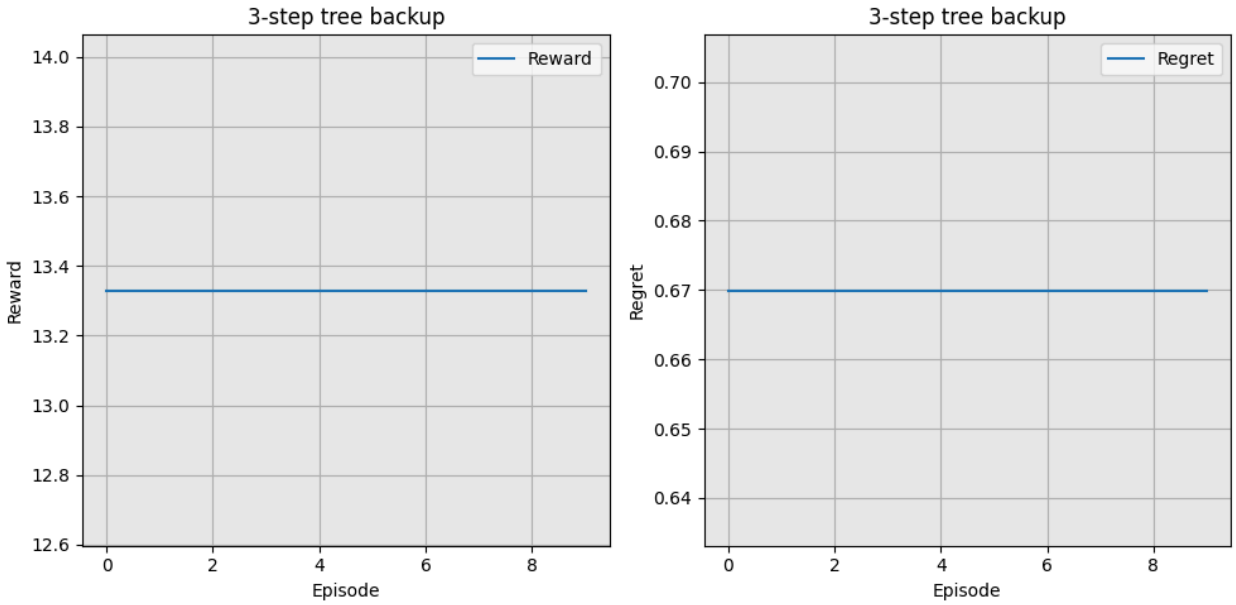




نتیجه یادگیری الگوریتم three step tree backup مطابق زیر است:



عامل در 10 اپیزود مطابق زیر تست شده است:



مشاهده می کنیم با افزایش n سرعت همگرایی عامل افزایش می یابد که علت آن تخمین کامل تر عامل از پاداش آینده است. الگوریتم n step tree backup چون برخلاف روش n step TD به صورت sample based عمل نمی کند بلکه نسبت به n step آینده bootstrap می کند در این محیط تصادفی عملکرد بسیار خوبی از خود نشان می دهد.
صرفا در این الگوریتم نسبت به مقدار دهی ابرپارامتر ها باید دقت کرد.
همچنین این روش در مقایسه با الگوریتم SARSA سرعت همگرایی بالاتر و نوسان به نسبت کمتری دارد.

.3

.3.1

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

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

$G \leftarrow 0$

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

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

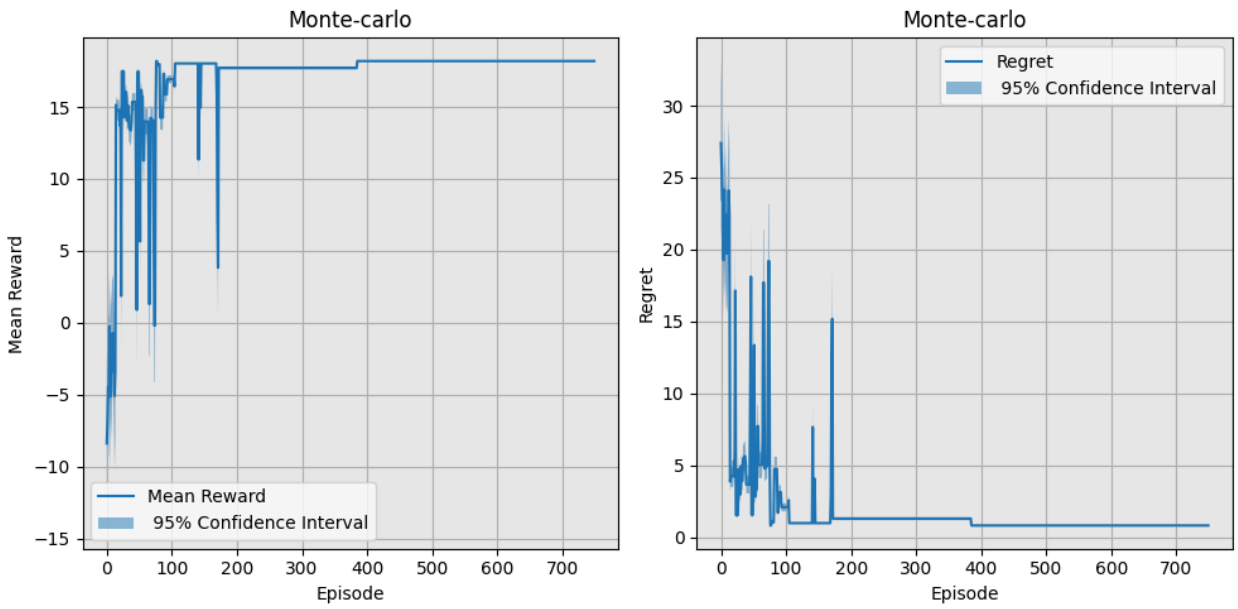
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

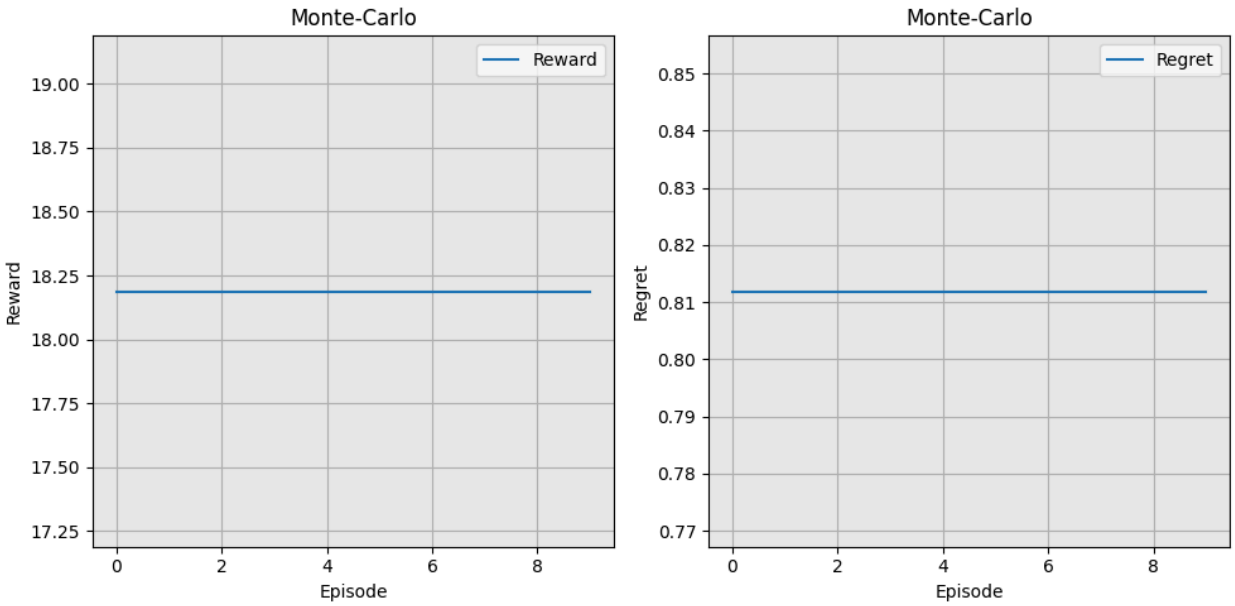
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

نتیجه آموزش عامل Every visit monte-carlo مطابق زیر است:

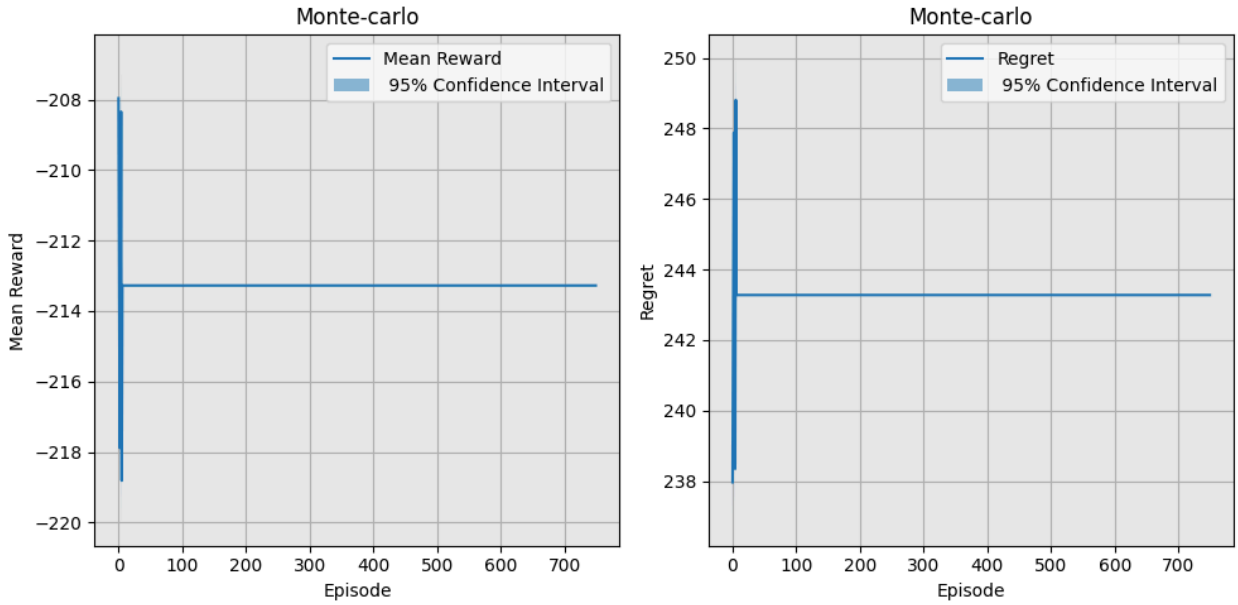


نتیجه تست عامل در10 اپیزود نیز مطابق زیر است:



در مقابل عدم کاهش epsilon در طول یادگیری سبب می شود موجب یادگیری ناپایدار عامل می شود و سبب می شود عامل به سیاست بهینه همگرا نشود.

در این محیط به ازای epsilon=0.1 ثابت نتیجه یادگیری عامل مطابق زیر است که به سبب عدم exploration کافی در محیط، عامل توانایی یادگیری خود را از دست می دهد.



:3.2

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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

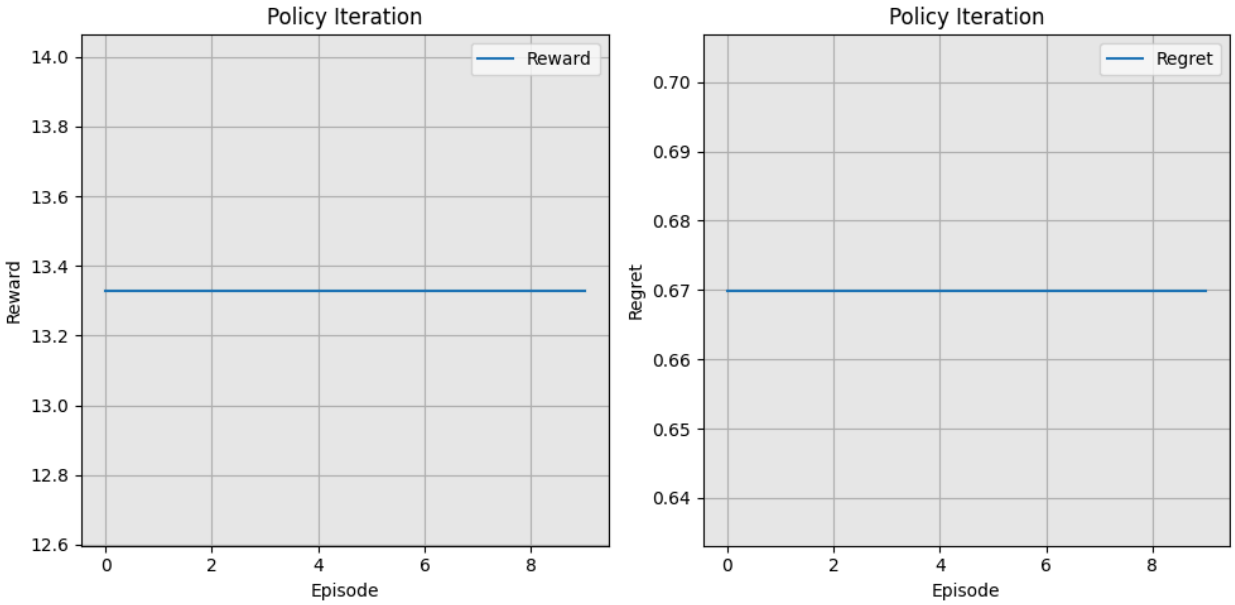
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*,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

ابرپارامتر های این الگوریتم شامل **θ** و **γ** است.

γ: Discount factor
مقدار این ابرپارامتر بین 0 و 1 است و هر چه به 1 نزدیک تر به پاداش در آینده اهمیت بیشتری می دهد و اصطلاحا far-sighted است و هرچه به 0 نزدیک تر باشد به پاداش آنی اهمیت بیشتری می دهد و اصطلاحا myopic است.

θ آستانه خطای مجاز برای ارزش حالات است. تا زمانی که خطای همگرایی ارزش استیت ها کمتر از این مقدار نشده باشد، بروزرسانی ارزش ها ادامه می یابد.

در انتها عامل به ازای 10 اپیزود مطابق زیر تست شده است که نشان دهنده همگرایی به سیاست بهینه است



در هر مرحله عامل اکشنی را انتخاب می کند که امید بازگشت پاداش آن بیشینه است.
در این روش چون به ساختار محیط شامل state transition probability و reward function دسترسی داشتیم همان طور که انتظار می رفت به سیاست بهینه همگرا شدیم.

:4

الگوریتم های پیاده سازی شده را از سه نظر زیر مقایسه می کنیم:

- مقایسه off-policy vs. on-policy:
  - در این مسئله الگوریتم Q-learning یک الگوریتم off-policy بود و در مقابل SARSA یک الگوریتم on-policy بود. هر دو الگوریتم به سیاست بهینه همگرا شدند اما در این مسئله نوسان یادگیری on-policy کمتر از off-policy بود که دلیل آن این است که الگوریتم های on-policy مستقیما ارزش های بهینه را از سیاست اجرایی محاسبه می کنند.
- مقایسه sample-based vs. bootstrap:
  - در این مسئله الگوریتم SARSA یک الگوریتم bootstrap اما در مقابل الگوریتم Monte-carlo یک الگوریتم sample-based بود. الگوریتم SARSA عملکرد به مراتب بهتری نسبت Monte-carlo داشت که توانایی بالای الگوریتم های bootstrap در محیط های تصادفی را نشان می دهد.
- مقایسه Reinforcement Learning vs. Dynamic Programing:
  - در Policy Iteration به دلیل داشتن ویژگی های محیط یک مسئله بهینه سازی Dynamic programing است و همواره به ارزش و سیاست بهینه همگرا می شود. در مقابل اکثر اوقات ما از ویژگی های محیط اطلاعی نداریم بلکه با الگوریتم های Reinforcement Learning سعی داریم در تعامل با محیط سیاست بهینه را یاد بگیریم.

:5

تصادفی بودن محیط سبب می شود نرخ همگرایی سیاست عامل ها کند تر باشد زیرا عامل باید بیشتر در محیط تعامل کند تا تخمین دقیقی از ارزش حالات داشته باشد.
این ذات تصادفی بودن محیط سبب می شود تا روش های bootstrap روش های sample-based را اصطلاحا out-perform کنند که در شبیه سازی نیز مشاهده کردیم الگوریتم های Q-learning و SARSA نسبت به Monte-carlo عملکرد به مراتب بهتری داشتند.
این لینک را برای اطلاعات بیشتر مطالعه نمایید.