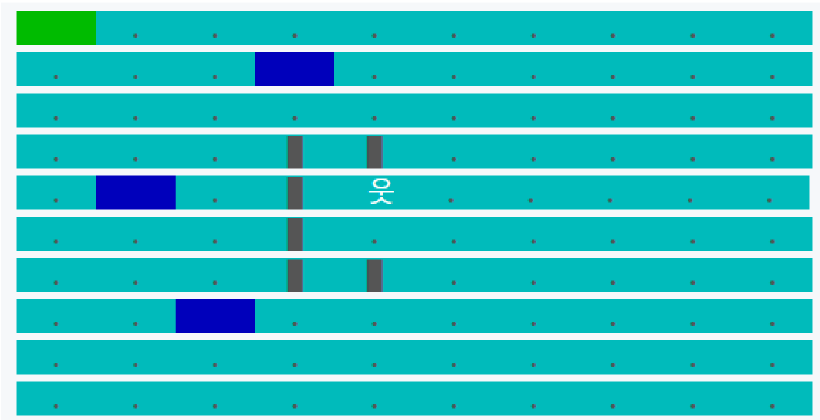
گزارش تمرین شماره ۴

نام و نام خانوادگی	علی عدالت
شماره دانشجویی	۸۱۰۱۹۹۳۴۸

سوال 1 – حرکت به سوی جزیره

در این سوال قصد داریم الگوریتم‌های متفاوتی را برای حل مسئله MDP بدون دینامیک محیط را پیاده سازی کنیم و در ادامه به مقایسه و بررسی عملکرد آن‌ها بپردازیم. در این سوال محیط یک gridworld است و مانند یک ماتریس است. خانه‌ها به این صورت شماره گذاری شده است که در ادامه آمده است. شماره خانه‌های ردیف اول از چپ به راست از ۰ تا ۹ است. شماره گذاری ردیف دوم از چپ به راست از ۱۰ تا خود ۱۹ است. برای دیگر خانه‌ها نیز به همین شکل شماره گذاری انجام شده است و شماره خانه‌ها از ۰ تا ۹۹ است. در یک خانه در مرکز ماتریس، اگر به پایین حرکت کنیم، شماره‌ی خانه ۱۰ واحد زیاد می‌شود. اگر به جای پایین به بالا حرکت کنیم، ۱۰ واحد شماره‌ی خانه کم می‌شود. اگر به چپ و راست حرکت کنیم به ترتیب شماره خانه یک واحد کم و یک واحد زیاد می‌شود. شماره‌ی خانه در این مسئله state ما را مشخص می‌کند و ما در ابتدا در خانه‌ی ۴۴ هستیم. هدف ما رسیدن به خانه صفر و انجام حرکت در آن است. نکته قابل توجه در این جا وجود موانع در محیط است.



موانع تیره رنگ، موانعی هستند که اگر در خانه‌ی همسایه آن قرار بگیریم نمی توانیم به خانه‌ی شامل این نوع مانع منتقل شویم. در صورت انتخاب اکشن برای انتقال به خانه شامل مانع تیره در جای خود می‌مانیم و پاداش 5- می‌گیریم. موانع نیلی رنگ، موانعی هستند که ما می‌توانیم به خانه‌ی شامل این نوع مانع منتقل شویم ولی در ازای انتقال به این نوع خانه‌ها پاداش 15- دریافت می‌کنیم. اگر در خانه‌ی صفر یا جزیره حرکت کنیم پاداش 3 دریافت می‌کنیم. در موارد دیگر جابه‌جایی پاداش صفر دارد. نکته‌ی مهم در این جا خانه‌های ابتدا و انتهای هر ردیف و ستون است. اگر در انتهای یک ردیف بخواهید به راست حرکت کنید، در جای خود می‌مانید و پاداش صفر می‌گیرید. در ابتدای هر سطر حرکت به چپ، در انتهای ستون حرکت به پایین و در ابتدای ستون حرکت به بالا به این شکل است. طبق این ویژگی ممکن است آدمک مدت زیادی در انتهای سطر گیر کند که مورد نظر ما نیست و سودی برای ما ندارد. دلیل گیر کردن پاداش صفر دریافتی است. دریافت این پاداش باعث می‌شود عامل فکر کند که جابه‌جایی عادی داشته اما

این گونه نیست. ما در این شرایط جابه‌جا نمی‌شویم و به هدف نزدیک نمی‌شویم. پس طبق این موارد حرکت به راست در انتهای یک ردیف یک اکشن غیر مجاز است. در ابتدای ردیف حرکت به چپ، در ابتدای ستون حرکت به بالا و در انتهای ستون حرکت به پایین غیر مجاز است. لازم است در سیاست زندگی در محیط این موضوع را مورد نظر قرار دهیم.

همان طور که در ابتدا گفته شد ما از مجموعه اکشن‌های مجاز در هر state و مجموعه‌ی state ها اطلاع داریم ولی احتمال جابه‌جایی از یک state به state دیگر و متوسط پاداش به ازای این جابه‌جایی ها را نمی‌دانیم. پس برای حل مسئله باید در محیط زندگی‌کنیم تا بتوانیم سیاست بهینه را بیابیم. در این جا سیاست بهینه مورد نظر یک سیاست است که ما را بدون دریافت جزا به جزیره برساند. همچنین ما نیاز داریم هزینه‌ی متوسط در مسیر یادگیری کمینه باشد. برای زندگی در محیط ما از سیاست epsilon greedy استفاده می‌کنیم که در آن در هر state برای انتخاب هر عمل یک احتمال وجود دارد. برای state های انتها و ابتدای سطرها و ستون‌ها باید احتمال انتخاب اکشن‌های عیر مجاز را در سیاست epsilon greedy برابر صفر قرار دهیم. برای این کار از تابع mask استفاده شده است. این تابع در هر state احتمال اکشن‌های غیر مجاز را صفر می‌کند برای این موضوع باید این تابع را در احتمال اکشن‌ها در یک state ضرب کنیم. در زیر این تابع آمده است.

```
def mask(self, s):
  prob = [1,1,1,1]
  if s in range(0,10):
    prob[0] = 0
  if s in range(90,100):
    prob[2] = 0
  if s in range(0,100,10):
    prob[3] = 0
  if s in range(9,100,9):
    prob[1] = 0
  return prob
```

در الگوریتم egreedy در هر state، احتمال اکشن با بیشترین مقدار $Q(s,a)$ به اندازه‌ی $1-\varepsilon$ از بقیه اکشن‌ها بیشتر است. برای پیدا کردن اکشن با بیشترین مقدار q نباید اکشن‌های غیر مجاز را در نظر بگیریم. برای این موضوع از تابع mask2 استفاده می‌کنیم. این تابع را اگر با q های یک state جمع کنیم، مقدار q برای اکشن‌های غیر مجاز در هر state برابر منهای بی‌نهایت می‌شود که این موضوع باعث می‌شود اکشن غیر مجاز در محاسبات در نظر گرفته نشود. تابع mask2 و نحوه‌ی به روز کردن سیاست زندگی egreedy در زیر آمده است.

```
def mask2(self, s):
  prob = [0,0,0,0]
  if s in range(0,10):
    prob[0] = float('-inf')
  if s in range(90,100):
    prob[2] = float('-inf')
  if s in range(0,100,10):
    prob[3] = float('-inf')
  if s in range(9,100,9):
    prob[1] = float('-inf')
  return prob
```

```
self.b = [[self.eps / float(self.act_num) for i in range(self.act_num)] for i in range(self.s_len)]
for i in range(self.s_len):
  index = np.argmax(np.array(self.mask2(i))+np.array(self.q[i]))
  temp = (np.array(self.b[i])*np.array(self.mask(i)))*(float(self.act_num)/(np.array(self.mask(i)) == 1).sum
  self.b[i] = list(temp)
  self.b[i][index] += (1.0 - self.eps)
```

نکته‌ی دیگر مشترک در الگوریتم‌ها این است که در هر گام باید state کنونی، اکشن انجام شده، پاداش دریافتی و state حاصل از اکشن را نگه‌داری کنیم. این اطلاعات در به روز کردن مقدار q ها و به روز رسانی سیاست به کار ما می‌آیند. همچنین برای مقایسه‌ی الگوریتم‌ها نیز به این داده‌ها نیاز داریم. برای این که در جزیره پاداش دریافت کنیم باید در خانه صفر حرکت کنیم. این در حالی است که وقتی به خانه‌ی صفر وارد شویم done برابر True می‌شود. برای این که episode تمام شود و جایزه دریافت کنیم از یک flag به اسم laststep استفاده می‌کنیم. ابتدا این flag برابر false است. وقتی done را دیدیم آن را True می‌کنیم. در این شرایط عبارت d == False or self.laststep == False برابر false می‌شودکه این به معنی این است که گام آخر در episode انجام شده است. یعنی با دیدن done انجام گام در episode را متوقف نمی‌کنیم و یک گام بعد از دیدن done هم انجام می‌دهیم تا پاداش دریافت کنیم. در ادامه درباره‌ی هر یک از الگوریتم‌های پیاده سازی شده و نتایج آن‌ها صحبت می‌کنیم.

On-policy Monte-Carlo

در این الگوریتم سیاست زندگی‌کردن با سیاستی که ارزیابی می‌کنیم یکسان است. در این الگوریتم از سیاست epsilon greedy استفاده می‌کنیم. شبه کد و کلیت این الگوریتم در زیر آمده است. در انتها این الگوریتم به یک سیاست epsilon optimal همگرا می‌شود. برای این که در این الگوریتم خود را به سیاست بهینه‌ی greedy نزدیک کنیم، نیاز است که epsilon را به تدریج کم کنیم تا به صفر میل کند. همان طور که دیده می‌شود، هر بار یک episode با توجه به سیاست تهیه می‌شود و سپس به بروز رسانی مقادیر q

پرداخته می‌شود. برای این کار از انتهای episode به ابتدا حرکت می‌کنیم. در این جا ما first-visit عمل می‌کنیم به همین دلیل نیاز است که به روز رسانی برای اولین بار دیده شدن state و اکشن در episode انجام شود. الگوریتم مطابق با شبه کد پایین پیاده‌سازی شده است. بعد از به روز رسانی برای یک episode، ما سیاست را باتوجه به مقادیر جدید q به روز می‌کنیم و برای این بروزرسانی از epsilon کوچکتری استفاده می‌کنیم. epsilon به شکل $eps = eps * 0.999$ تغییر می‌کند که مقدار اولیه‌ی eps برابر یک است.

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 $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

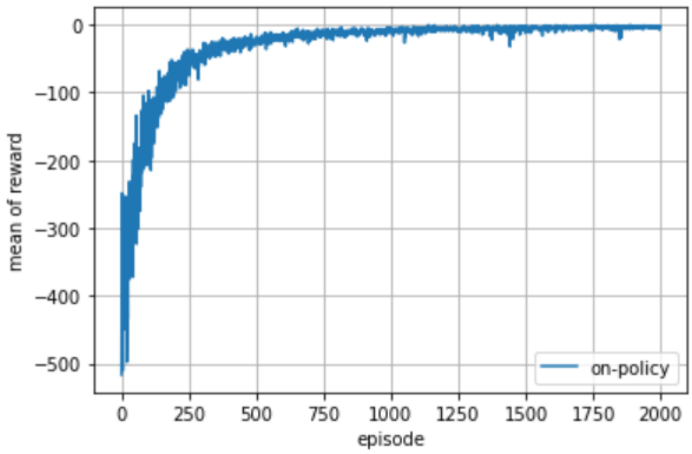
Repeat forever (for each episode):
- Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow G + R_{t+1}$
  - Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
    - Append G to $Returns(S_t, A_t)$
    - $Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
    - $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
    - For all $a \in \mathcal{A}(S_t)$:

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

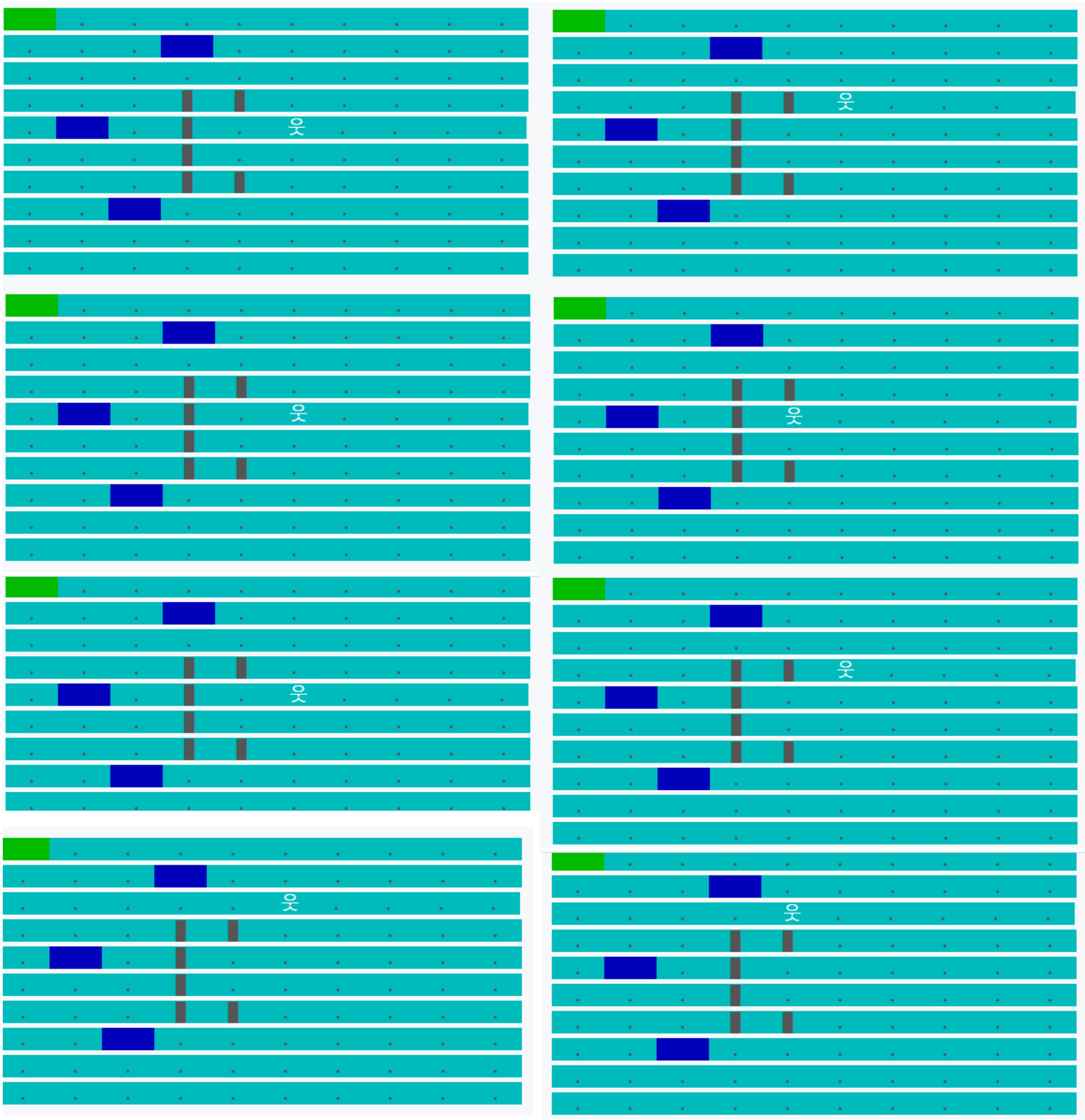
در ابتدا ما اطلاعات خاصی ندارم به همین دلیل تمام مقادیر q برابر صفر است. در ابتدا برای بدست آوردن اطلاعات نیاز داریم که به exploration بپردازیم. به همین دلیل با epsilon برابر یک شروع می‌کنیم که باعث می‌شود احتمال تمام اکشن‌ها در هر state یکسان شود. به تدریج epsilon را با نرخ 0.999 کاهش می‌دهیم که دلیل آن زیاد شدن اطلاعات، دقیق شدن مقادیر q و مشخص شدن مسیر و سیاست بهینه است. با این کاهش به تدریج از فاز exploration به exploration می‌رویم. به گونه‌ای که در انتها احتمال اکشن با بیشترین q در state از بقیه بسیار بیشتر است. این روند باعث می‌شود متوسط هزینه‌ی مسیر یادگیری ما پایین بیاید و سیاست نهایی را به سیاست بهینه‌ی greedy بسیار نزدیک کنیم. همچنین این کار باعث می‌شود که sample efficiency بالاتر برود. در این الگوریتم حتما باید یک episode کامل انجام شود تا بتوان به بروزرسانی پرداخت. همچنین با توجه به مسئله که ما می‌خواهیم مسیر مناسب از خانه ۴۴ به صفر را بیابیم، تمام episode ها را از خانه ۴۴ شروع می‌کنیم. نمودار متوسط پاداش به ازای تعداد episode در زیر آمده است. برای یادگیری مناسب agent به تعداد ۲۰۰۰ تا episode با توجه به نرخ تغییر epsilon نیاز است که در پیاده‌سازی مورد نظر قرار گرفته است. همچنین الگوریتم یادگیری ماهیت

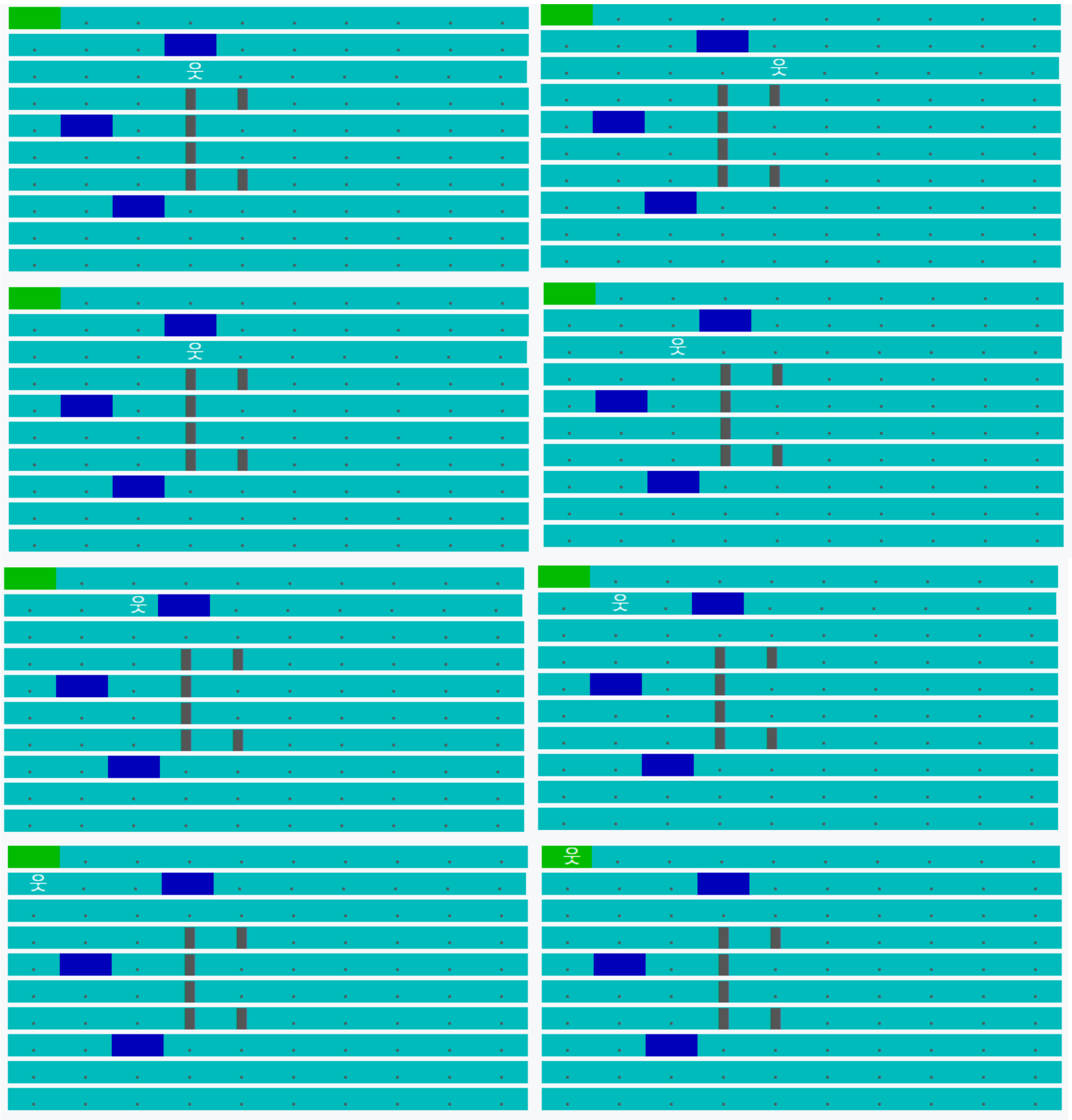
stochastic دارد و نیاز است چندین بار اجرا شود و میانگین نتایج اعلام شود که برای این موضوع از ۲۰ بار اجرا استفاده شده است. در عکس زیر دیده می‌شود که الگوریتم همگرا شده است. این موضوع را با ثابت شدن نمودار در انتها می‌توان دریافت کرد. همچنین که دیده می‌شود به تدریج و با زیادتر شدن episode های دیده شده، اطلاعات ما بیشتر می‌شود و ما هر بار بهتر از قبل عمل می‌کنیم. این موضوع باعث شده که متوسط پاداش با بیشتر شدن episode های دیده شده بیشتر شود. متوسط کل پاداش‌های دریافتی به ازای کل episode ها در این نمودار برابر 30.861375- است. ما به دنبال سیاستی هستیم که این متوسط در آن بیشینه باشد و همچنین تعداد episode های لازم برای آن تا ثابت شدن در مقدار متوسط بیشینه از همه کمتر باشد.

```
on-policy  : -30.861375
```



در نمودار بالا دیده می‌شود که نمودار در مقداری نزدیک به صفر ثابت شده است که از مقدار سه کمتر است که دلیل آن پیدا کردن سیاست epsilon greedy بهینه با کمترین مقدار epsilon است. یک پاسخ agent با توجه به سیاست یادگرفته شده در زیر آمده است. همانطور که دیده می‌شود بعضی اوقات رفت و برگشت داریم که به خاطر on-policy بودن و این موضوع است که ما سیاست egreedy بهینه با کمترین epsilon را یافته ایم. ما در این جا به سیاست بهینه‌ی greedy نرسیده ایم. برای تهیه‌ی پاسخ از سیاست یادگرفته شده توسط الگوریتم استفاده شده است. ترتیب عکس‌ها از چپ به راست و از بالا به پایین است. اولین عکس سمت چپ بالا شماره یک، اولین عکس سمت راست بالا شماره دو و دومین عکس سمت چپ از بالا شماره سه است. این شماره گذاری تا انتها به همین شکل ادامه دارد.





Off-policy Monte-Carlo

در این الگوریتم سیاست زندگی که b نام دارد با سیاستی که می‌خواهیم ارزیابی کنید یعنی سیاست p متفاوت است. سیاست b یک سیاست epsilon greedy است و p یک سیاست greedy می‌باشد. سیاست b مانند الگوریتم قبل می‌باشد. نکته قابل توجه در این جا این است که ما با سیاست b زندگی می‌کنیم و نمونه‌هایی که از محیط می‌گیریم با توجه به سیاست b است. در state ها با توجه به سیاست b اکشن برای انجام انتخاب می‌کنیم. به همین دلیل لازم است که نمونه‌های گرفته شده را به اطلاعات متناسب با سیاست p تبدیل کنیم. در این الگوریتم q با توجه به نمونه‌های تبدیلی برای سیاست p به روز می‌شوند. باتوجه به این q ها سیاست p و b به روز می‌شود. کم کردن تدریجی epsilon باعث می‌شود مانند الگوریتم قبل به

سیاست بهینه همگرا شویم و از exploration به exploitation برویم. کلیت و شبه کد این الگوریتم در زیر آمده است. الگوریتم با توجه به این شبه کد پیاده سازی شده است.

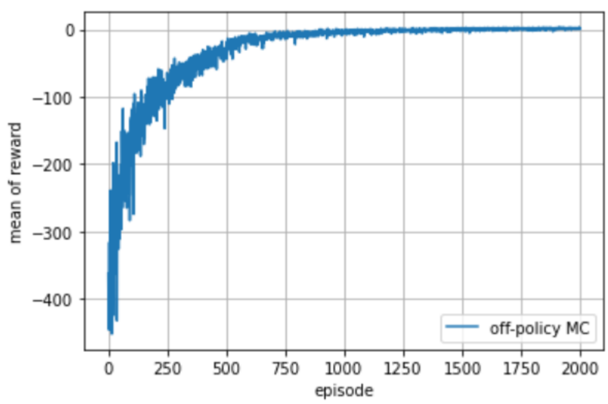
Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
 $Q(s, a) \in \mathbb{R}$ (arbitrarily)
 $C(s, a) \leftarrow 0$
 $\pi(s) \leftarrow \arg\max_a Q(s, a)$ (with ties broken consistently)

Loop forever (for each episode):
 $b \leftarrow$ any soft policy
 Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
 $G \leftarrow 0$
 $W \leftarrow 1$
 Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  $G \leftarrow \gamma G + R_{t+1}$
  $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
  $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
  If $A_t \neq \pi(S_t)$ then exit For loop
  $W \leftarrow W \frac{1}{b(A_t|S_t)}$

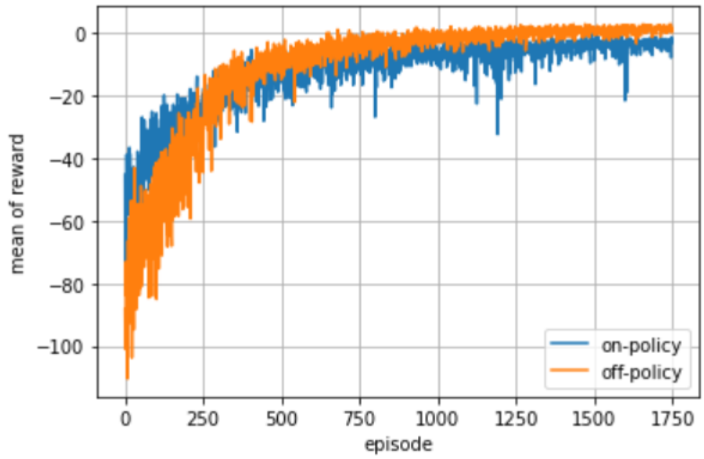
در زیر نمودار متوسط پاداش به ازای تعداد episode دیده شده آمده است. تعداد اجرا و تعداد episode در هر اجرا مانند الگوریتم گذشته است. در این الگوریتم نیز حتما باید یک episode کامل دیده شود که بتوان مقدار q ها را به روز کرد. مانند قبل دیده می‌شود که در انتها نمودار ثابت شده است که این موضوع نشان دهنده‌ی همگرایی الگوریتم است. همچنین دیده می‌شود که با افزایش تعداد episode های دیده شده، متوسط پاداش افزایش پیدا می‌کند. نتایج الگوریتم گذشته را می‌توان در نمودار این الگوریتم دید. نکته قابل توجه در این نمودار این است که نمودار به مقداری بیشتر از مقدار در الگوریتم قبل همگرا شده است.

```
off-policy MC  : -31.541875
```

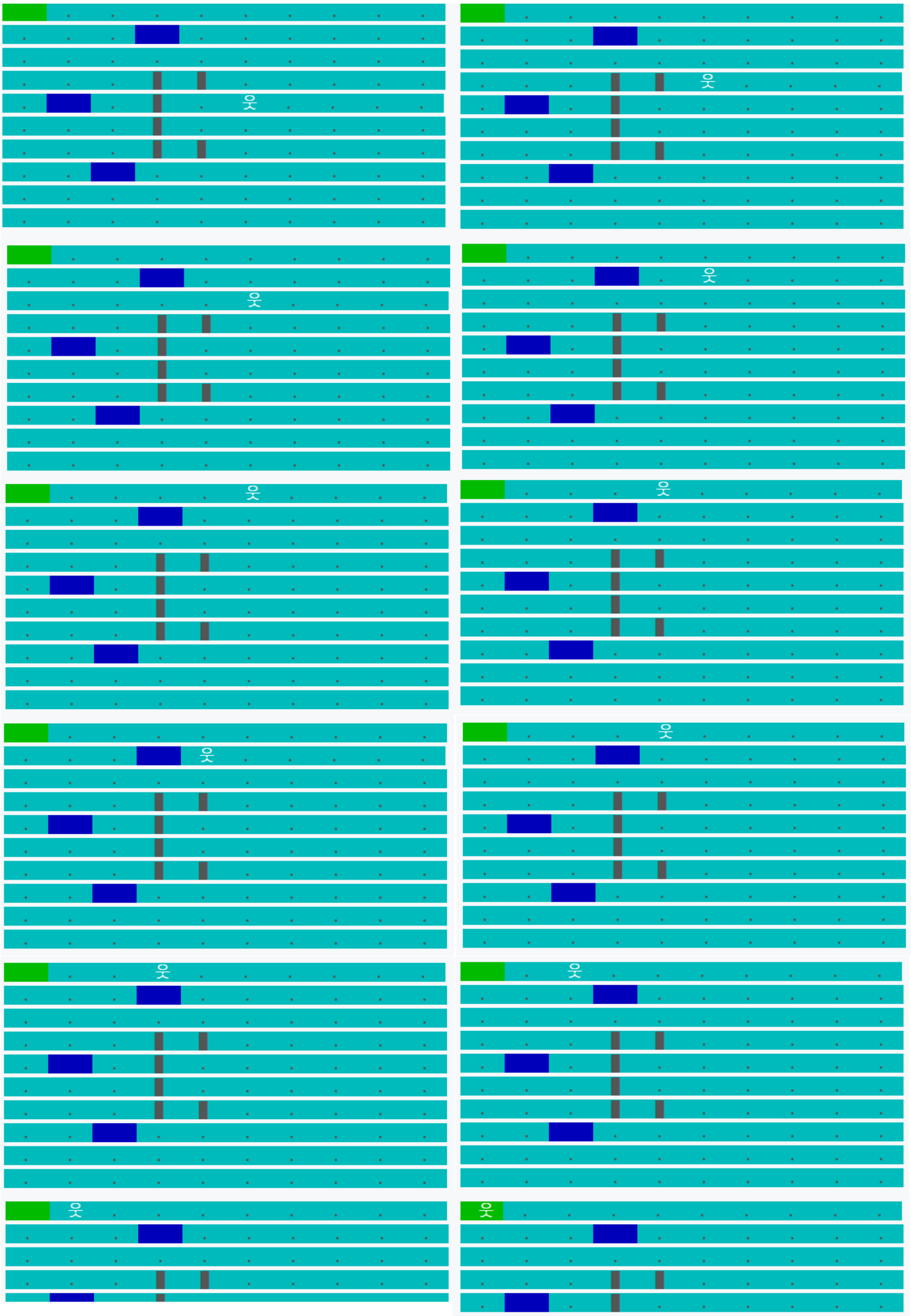


همچنین دیده می‌شود که نمودار دیر تر از نمودار الگوریتم گذشته به مقدار ثابت رسیده است و همگرا شده است. متوسط پاداش در کل فرآیند یادگیری برابر 31.541875- است که از الگوریتم قبل کمتر است. نمودار این دو الگوریتم در کنار هم در زیر آمده است. برای دیدن بهتر جزئیات نمودارها از ۲۵۰ episode به بالا در زیر آمده است. نتایج برای ۲۵۰ episode با تعداد صفر episode نشان داده شده است.

```
on-policy  : -30.861375
off-policy  : -31.541875
```



همانطور که دیده می‌شود الگوریتم off-policy دیرتر از on-policy همگرا می‌شود و مقدار همگرایی در off-policy بیشتر است. مقدار همگرایی در off-policy تقریبا برابر سه است. همگرا شدن به مقدار سه به دلیل این است که در off-policy ما به دنبال سیاست greedy بهینه هستیم که سیاست بهینه در کل می‌باشد. در on-policy ما سیاست egreedy بهینه با کمترین مقدار epsilon را پیدا کردیم که چون بی‌نهایت بار epsilon کوچک نشده است، ما به سیاست greedy بهینه نرسیدیم ولی سیاست پیدا شده در این الگوریم به سیاست بهینه نزدیک است. همگرایی در off-policy دیرتر صورت گرفته و هزینه‌ی یادگیری ما بیشتر بوده است. هزینه بیشتر بوده است که دلیل آن کمتر بودن متوسط پاداش در کل episode ها در off-policy است. دلیل این موضوع sample efficiency پایین تر در off-policy به خاطر تبدیل نمونه‌های سیاست b به نمونه‌های سیاست p است. ممکن است در episode تهیه شده با سیاست b ما در state اکشنی انجام دهیم که در سیاست p ممکن نباشد. در این صورت نمونه ایجاد شده به درد به روزرسانی q نمی‌خورد. این موارد باعث همگرایی دیرتر می‌شود. در زیر نمونه جواب الگوریتم off-policy MC آمده است. ترتیب عکس‌ها در زیر مانند قسمت قبل می‌باشد.



SARSA

در این الگوریتم فرض کردم که الگوریتم on-policy مد نظر است. با توجه به این موضوع سیاست زندگی کردن و سیاست مورد ارزیابی یکی هستند. سیاست مانند سیاست‌ای است که در on-policy MC مورد بررسی ما بود. از یک سیاست epsilon greedy استفاده می‌کنیم که مانند گذشته بعد از هر episode مقدار epsilon را با نرخ 0.999 کم می‌کنیم. الگوریتم منطبق با شبه کد زیر پیاده سازی شده است.

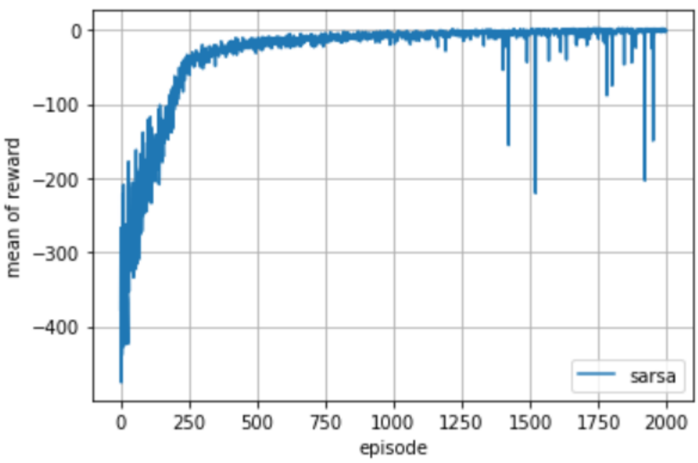
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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 $Q(terminal, \cdot) = 0$

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)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

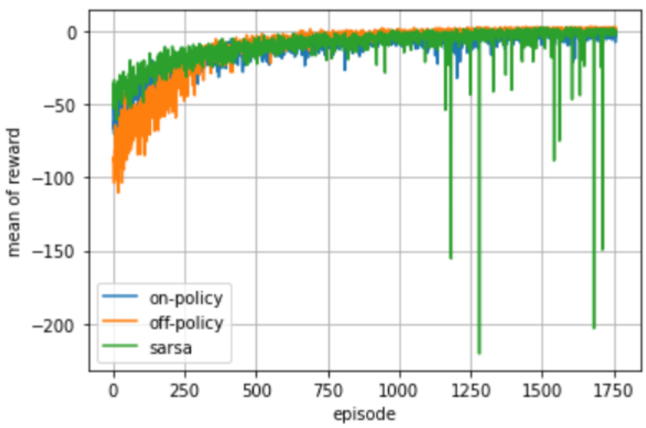
در این الگوریتم برای به روز رسانی از یک گام به اسم learning rate استفاده می‌کنیم که مشخص می‌کند به چه اندازه مقدار q گذشته را با توجه به نمونه‌ی جدید تغییر دهیم. با توجه به شبه کد، lr را برابر 0.2 قرار دادیم که یک مقدار کوچک مثبت است. در زیر نمودار متوسط پاداش به ازای تعداد episode آمده است. متوسط پاداش در طول یادگیری نیز در زیر آمده است که برابر -31.023625 است.

```
sarsa  : -31.023625
```

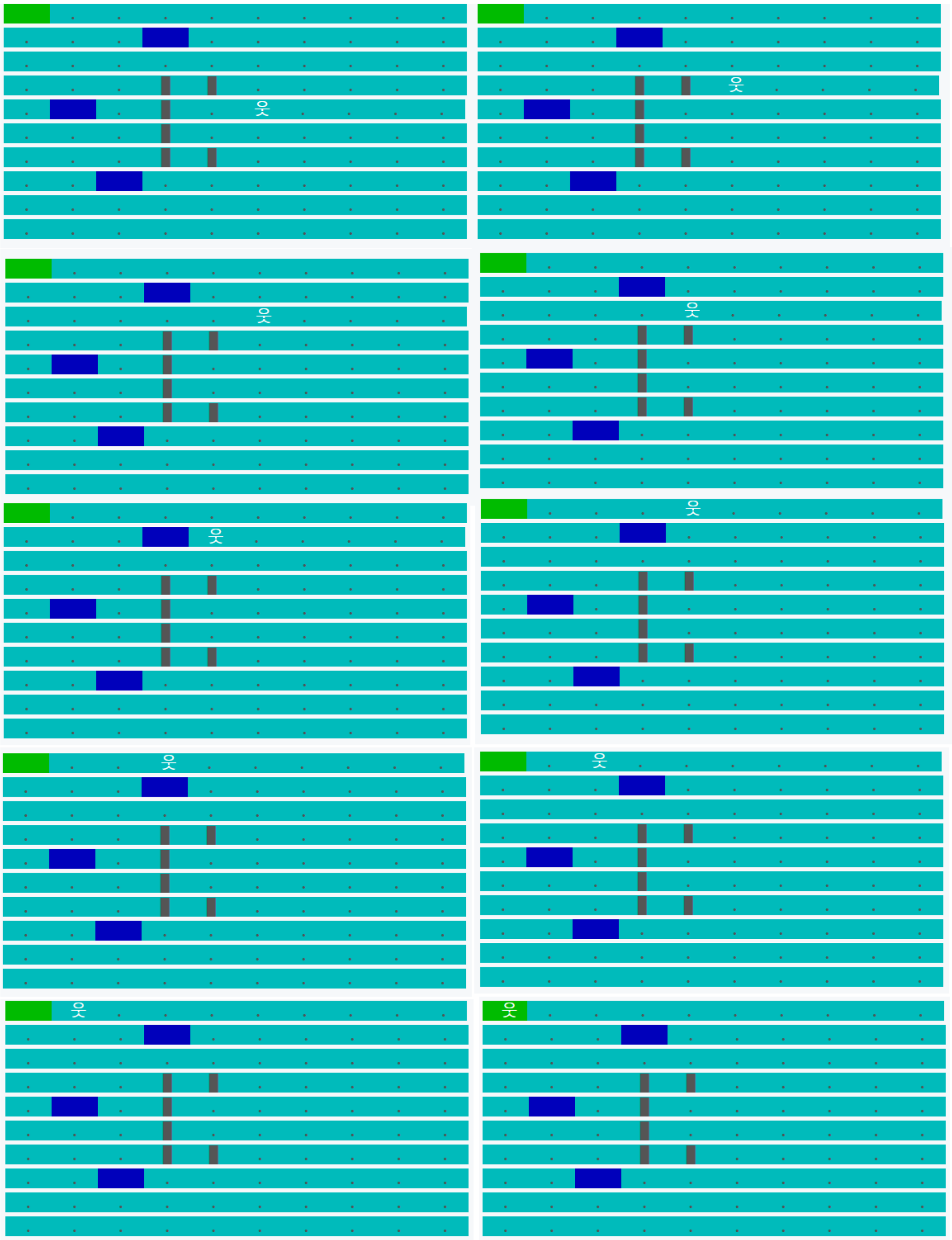


همان طور که دیده می‌شود نمودار در انتها ثابت شده است. این موضوع نشان دهنده‌ی همگرا شدن سیاست است. با بیشتر شدن تعداد episode دیده شده متوسط پاداش بیشتر می‌شود. برای تهیه‌ی این نمودار مانند گذشته از ۲۰ بار اجرا با ۲۰۰۰ episode استفاده شده است. با این که در این الگوریتم نیازی به دیدن episode کامل برای به روز کردن مقادیر q نیست، در این جا هر episode به طور کامل است. یعنی زمانی episode تمام می‌شود که ما به جزیره برسیم. در زیر نمودار متوسط پاداش به ازای تعداد episode برای سه الگوریتم مورد بررسی تا کنون آمده است. برای دیدن بهتر جزئیات نمودارها از ۲۵۰ episode به بالا در زیر آمده است. نتایج برای ۲۵۰ episode با تعداد صفر episode نشان داده شده است.

```
on-policy  : -30.861375
off-policy  : -31.541875
sarsa  : -31.023625
```



همان طور که دیده می‌شود، sarsa عملکری مشابه با on-policy MC دارد. مقدار متوسطی که این الگوریتم به آن همگرا شده کمتر از سه است که دلیل آن یافتن سیاست epsilon greedy با کمترین epsilon و پیدا نکردن سیاست greedy است. سرعت sarsa با on-policy MC تفاوت چندانی ندارد و سرعت sarsa کمی بیشتر است. اما متوسط کل پاداش دریافتی در طول یادگیری آن از on-policy MC کمتر است. دلیل این موضوع می‌تواند ثابت بودن lr و تاثیر بالای داده‌های لحظه‌ای جدید در تعداد episode بالا باشد. هر چه تعداد episode های دیده شده بیشتر می شود دقت مقادیر q بیشتر می‌شود و لازم است کمتر داده‌های جدید در مقادیر q تاثیر بگذارند. دلیل آن این است که اطمینان ما از مقادیر q بسیار بالا رفته و یک داده‌ی لحظه‌ای نباید مقدار با اطمینان زیاد حاصل چندین episode را خیلی تغییر دهد. پس نیاز است که lr به تدریج به صفر میل کند. این lr نامناسب باعث می‌شود sample ها به خوبی استفاده نشوند و sample efficiency پایین بیاید. پاسخ الگوریتم برای بردن در بازی در زیر آمده است.



Two-Step Expected SARSA

در این الگوریتم مانند SARSA از یک سیاست برای زندگی و ارزیابی استفاده می‌کنیم. در این جا از یک سیاست epsilon greedy استفاده می‌کنیم. کلیت الگوریتم مانند SARSA است فقط به جای q در state نتیجه، از V در آن state برای به روز رسانی q در state اولیه استفاده می‌کنیم. همچنین الگوریتم به صورت two-step است. یعنی بعد از دو بار انجام اکشن بعد از قرار گیری در یک state به بروزرسانی q state اولیه می‌پردازیم. شبه کد الگوریتم n-step SARSA در زیر آمده است.

n-step Sarsa for estimating $Q \approx q_* \text{ or } q_\pi$

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

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for each step of episode, $t = 0, 1, 2, \dots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

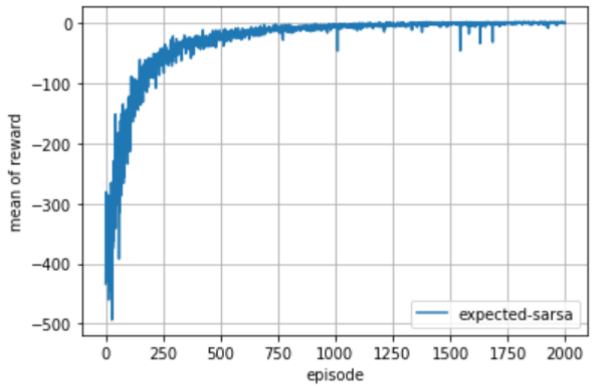
تنها تفاوت الگوریتم n-step expected SARSA در نحوه‌ی محاسبه‌ی G در قسمت مشخص شده است که در زیر آمده است.

$$G \leftarrow G + \gamma^n v(S_{\tau+n})$$

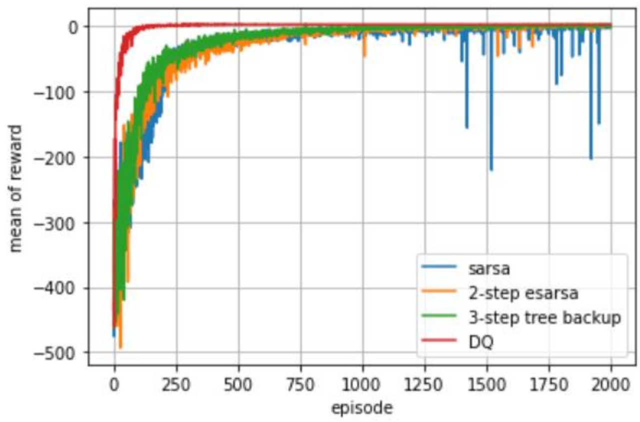
برای پیاده سازی شبه کد بالا، ابتدا صبر می‌کنیم تا دو اکشن در episode انجام دهیم. در گامی که دومین اکشن را انجام دادیم، مقدار $Q(S_{t-1}, A_{t-1})$ را به روز می‌کنیم که مربوط به state است که قبل از انجام دو اکشن در آن بوده‌ایم. این کار را ادامه می‌دهیم تا گام آخر را برداریم. ممکن است state در گام آخر به خاطر طول فرد episode به روز نشده باشد که آن را بر اساس پاداش دریافتی اکشن انجام شده به عنوان G به روز می‌کنیم. مانند گذشته epsilon را به تدریج با نرخ 0.999 از یک کاهش می‌دهیم. در ابتدای یادگیری ما اطلاع خاصی نداریم پس نیاز است داده‌ی جدید به طور کامل در مقدار q تاثیر بگذارد. پس نیاز است در ابتدا lr برابر یک باشد. هر چه تعداد episode های دیده شده بیشتر می شود دقت مقادیر

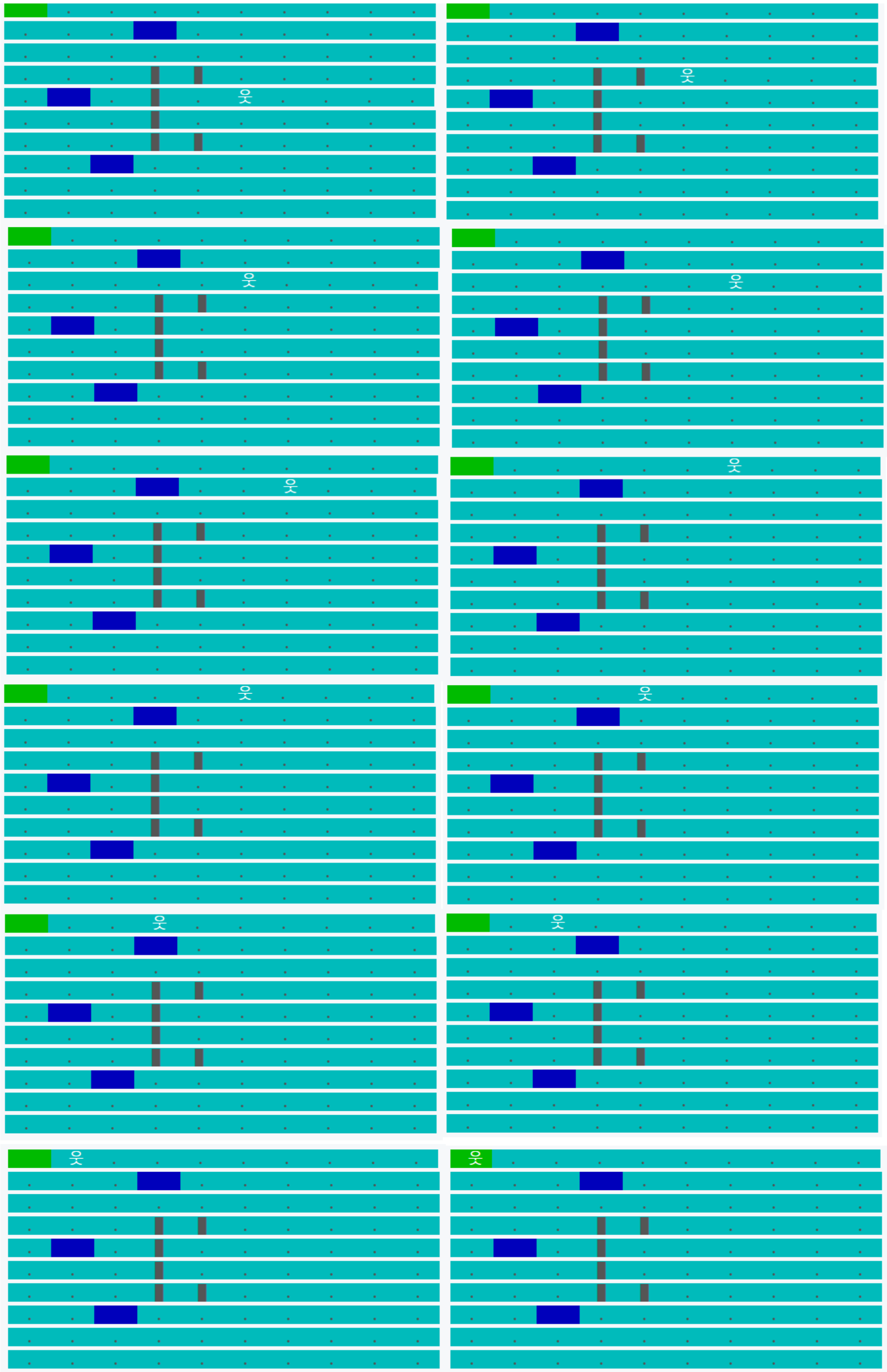
q بیشتر می‌شود و لازم است کمتر داده‌های جدید در مقادیر q تاثیر بگذارند. دلیل آن این است که اطمینان ما از مقادیر q بسیار بالا رفته و یک داده‌ی لحظه‌ای نباید مقدار با اطمینان زیاد حاصل چندین episode را خیلی تغییر دهد. پس نیاز است که lr نیز به صورت $lr = lr \times 0.999$ تغییر کند که ابتدا lr برابر یک است. در زیر نمودار متوسط پاداش به ازای تعداد episode آمده است.

```
expected-sarsa  : -28.2425
```



در این جا برای بدست آوردن نتایج بالا از ۲۰ اجرا که هر یک شامل ۲۰۰۰ episode است استفاده کردیم. در این الگوریتم لازم نیست که در هر episode به state نهایی برسیم تا بتوانیم به روزرسانی q ها و سیاست را داشته باشیم. در هر گام می‌توان q ها را به روز کرد. به همین دلیل بیشینه تعداد گام‌ها را در این الگوریتم برابر ۲۰۰۰۰ قرار دادیم. این به این معنی است که اگر ۲۰۰۰۰ گام انجام دهیم و به state نهایی نرسیدیم، به episode بعدی می‌رویم. همان طور که دیده می‌شود الگوریتم به متوسط پاداش سه همگرا شده است. سرعت همگرایی در آن طبق نمودارهای زیر از sarsa بیشتر است و متوسط کل پاداش در طول یادگیری آن برابر 28.2425- است که از تمام الگوریتم‌های گذشته بیشتر می‌باشد. در بازه‌ی ۰ تا ۲۵۰ episode نمودار این الگوریتم بر روی sarsa قرار دارد و تا انتها این موضوع وجود دارد به همین دلیل الگوریتم سریعتر از sarsa همگرا می‌شود. یک نمونه پاسخ الگوریتم به بازی در ادامه آمده است.





Tree Back-up

در این الگوریتم مانند گذشته سیاست زندگی و ارزیابی یکی است و ما از یک سیاست epsilon greedy استفاده می‌کنیم. با توجه به توزیع داده شده در این قسمت 3-step Tree Back-up مدنظر است. شبه کد n-step Tree Back-up در زیر آمده است که این شبه کد برای سه گام پیاده سازی شده است.

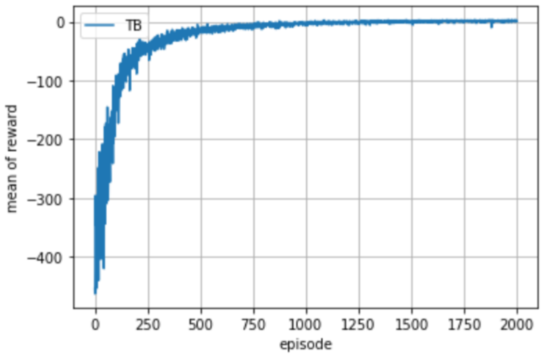
n-step Tree Backup for estimating $Q \approx q_*$ or q_π

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

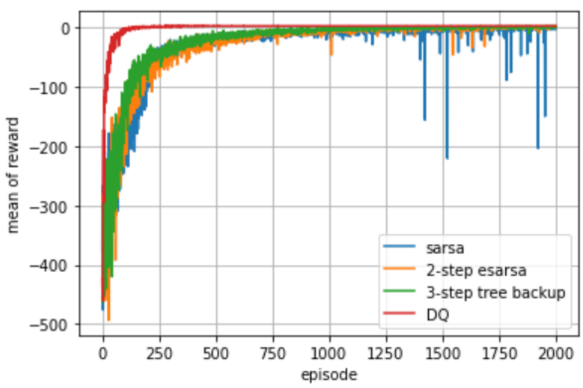
برای پیاده سازی مانند الگوریتم قبل در ابتدا صبر می‌کنیم که سه اکشن در episode انجام شود. سپس زمانی که اکشن سوم انجام شد، مقدار $Q(S_{t-2}, A_{t-2})$ را به روز می‌کنیم. این مقدار q مربوط به اولین state است که در آن بودیم و بعد سه اکشن انجام دادیم. State مربوط به سه گام قبل است. این کار را برای گام‌های بعد نیز انجام می دهیم. ممکن است طول episode به سه بخش پذیر نباشد و ما به انتها برسیم و تعدادی state به روز نشده باشند. این state ها را با توجه به جمع discount شده‌ی پاداش‌های دریافتی تا انتها به روز می‌کنیم. epsilon و lr را مانند قبل کاهش می‌دهیم. در زیر نمودار متوسط پاداش به ازای تعداد episode این الگوریتم امده است.

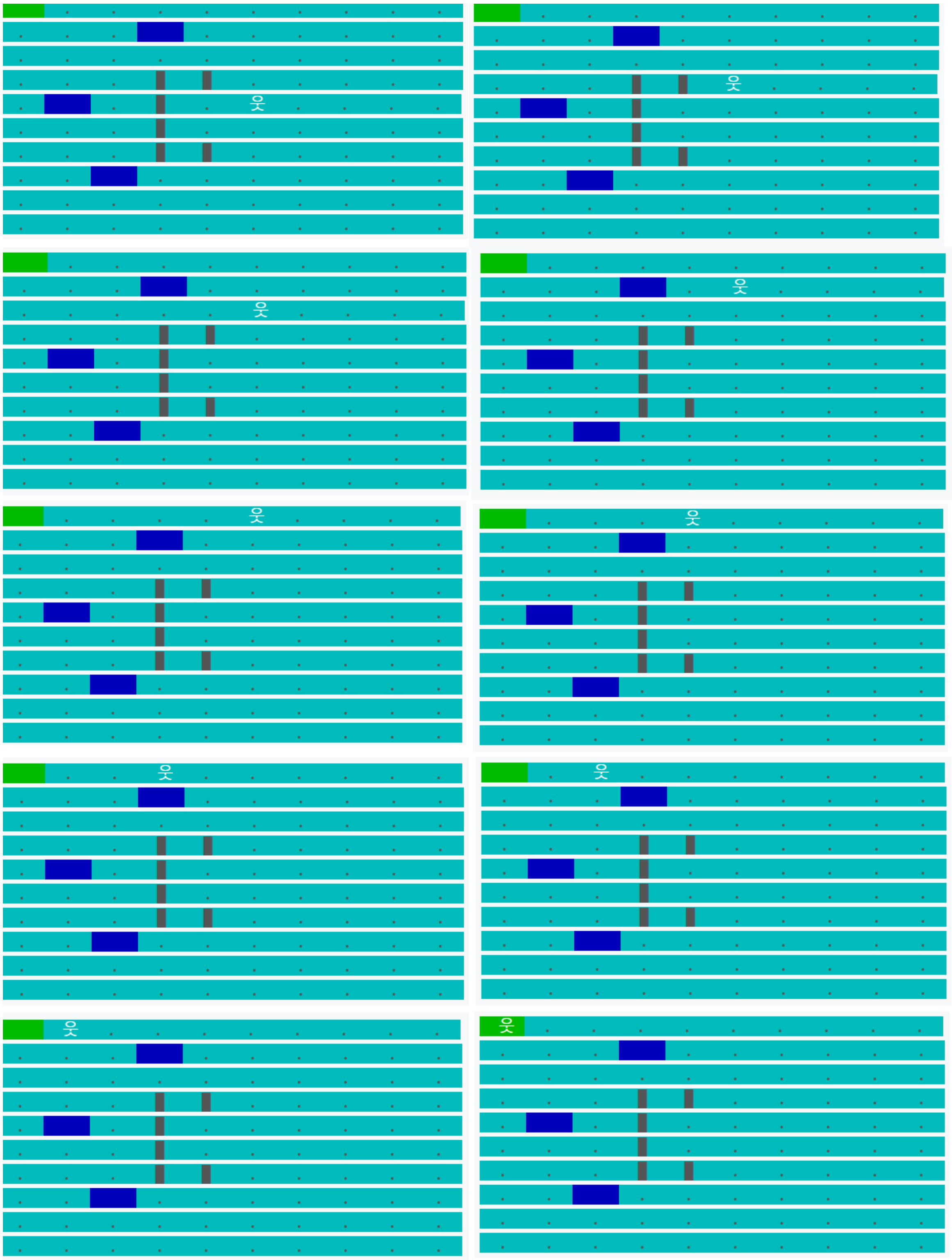
```
TB  : -23.695375
```



همانطور که در نمودار بالا دیده می‌شود، الگوریتم به متوسط پاداش سه و سیاست بهینه همگرا شده است. با افزایش تعداد episode های دیده شده، می توان دید که متوسط پاداش افزایش می‌یابد و سیاست به سمت سیاست بهینه حرکت می‌کند. سرعت همگرایی الگوریتم از two-step expected sarsa بیشتر است که دلیل آن استفاده از داده‌های q اکشن‌های انتخاب نشده در طول episode است. با توجه به یادگیری سریعتر دیده می‌شود که متوسط پاداش سریع تر به سه نزدیک می‌شوند. متوسط کل پاداش در مسیر یادگیری برابر 23.695375- است که از تمام الگوریتم‌های پیشین بیشتر است. این بیشتر بودن به خاطر همگرایی سریع تر است که این همگرایی باعث شده بیشتر مواقع متوسط پاداش از الگوریتم‌های قبل بیشتر باشد. در زیر نمودار این الگوریتم در کنار نمودار sarsa و two-step expected sarsa دیده می‌شود. سرعت بیشتر همگرایی الگوریتم در این نمودار قابل مشاهده است. به وضوح مشخص است که مقدار متوسط پاداش در طول یادگیری در الگوریتم tree-back بیشتر از two-step expected sarsa است. این موضوع سرعت همگرایی بالا تر را نشان می‌دهد. پاسخ الگوریتم برای حل بازی در زیر آمده است. شرایط اجرا مانند الگوریتم قبل است.

```
sarsa  : -31.023625
2-step esarsa  : -28.2425
3-step tree backup  : -23.695375
DQ  : -0.31975
```





هزینه‌ی محاسبات در هر گام از الگوریتم قبل بیشتر است. چون در هر گام ما باید از اطلاعات اکشن‌های انتخاب نشده در مسیر هم استفاده کنیم.

Double Q-learning

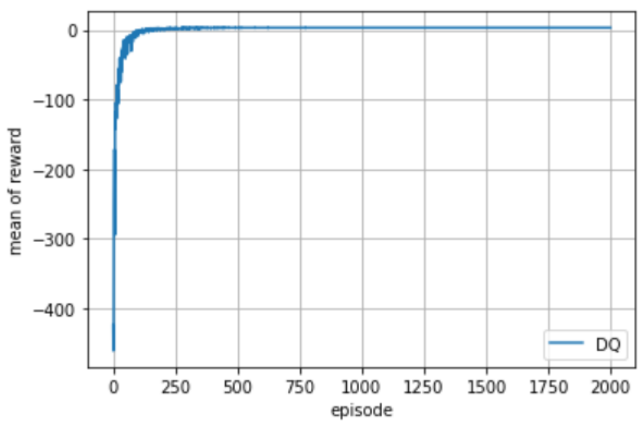
در این الگوریتم ما با یک سیاست epsilon greedy زندگی می‌کنیم و به ارزیابی یک سیاست greedy می‌پردازیم. در این جا مقادیر q را با توجه به سیاست greedy به روز می‌کنیم. هدف ما بهبود سیاست greedy و رسیدن به سیاست بهینه است. شبه کد الگوریتم در زیر آمده است که با توجه به آن پیاده سازی انجام شده است.

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q.(terminal, \cdot) = 0$

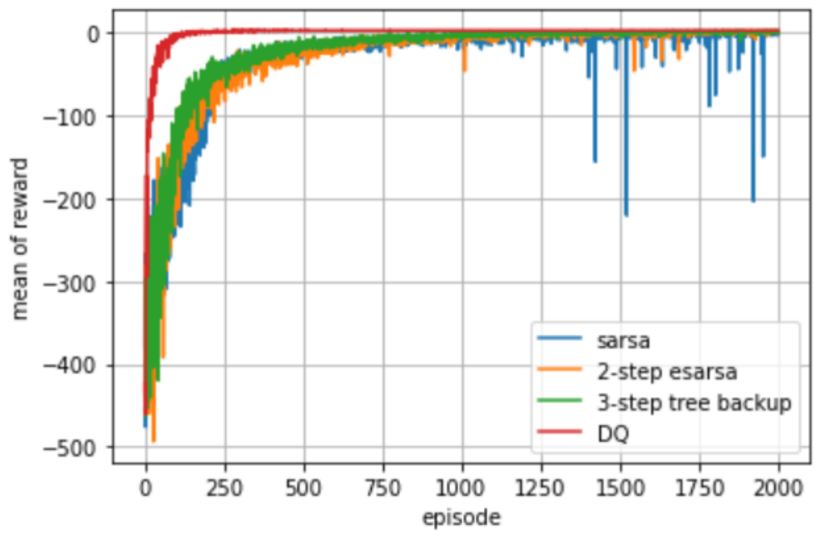
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \mathrm{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \mathrm{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
    $S \leftarrow S'$
  until S is terminal

همان طور که در الگوریتم‌های قبل بحث شد، نیاز است که epsilon و lr به تدریج از مقدار یک کاهش پیدا کنند. در این الگوریتم از نرخ 0.99 برای کاهش این دو مقدار استفاده کرده‌ایم. برای ارزیابی این الگوریتم باز هم از ۲۰ بار اجرا شامل ۲۰۰۰۰ episode استفاده کردیم. در این الگوریتم نیازی به دیدن کل episode برای به روز رسانی q ها نیست و بعد از هر گام می‌توان به بروزرسانی پرداخت. در اجرای الگوریتم ما طول بیشینه‌ی episode را برابر بی نهایت در نظر گرفتیم. نمودار متوسط پاداش به ازای تعداد episode در زیر آمده است.

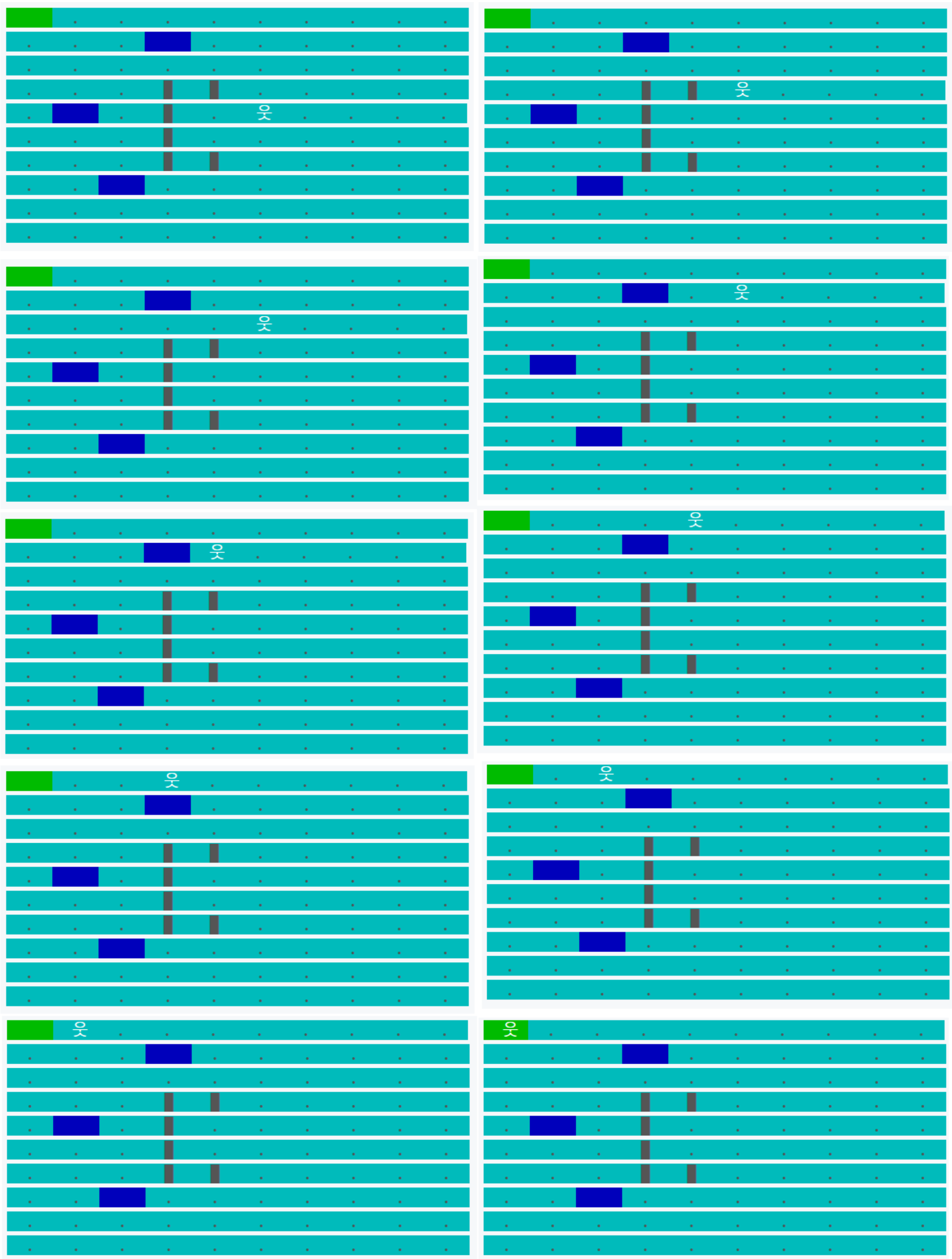


همان طور که دیده می‌شود الگوریتم به سیاست بهینه همگرا شده است. این موضوع از ثابت شدن نمودار در انتها به مقدار متوسط پاداش سه قابل استنباط است. متوسط کل پاداش‌ها در طول یادگیری برابر 0.31975- است. این مقدار از تمام متوسط‌های الگوریتم‌های دیگر کمتر است. دلیل آن این است که الگوریتم در تعداد پایین episode توانسته سیاست بهینه را پیدا کند و از آن به بعد با این سیاست بهینه بازی را انجام دهد. این موضوع نشان دهنده‌ی همگرایی سریع تر این الگوریتم نسبت به الگوریتم‌های قبل است که در نمودار زیر نیز قابل مشاهده است.

```
sarsa  : -31.023625
2-step esarsa  : -28.2425
3-step tree backup  : -23.695375
DQ  : -0.31975
```



همان طور که دیده می‌شود نمودار الگوریتم DQ در طول یادگیری بالاتر از سایر نمودارها قرار دارد. این بدین معنا است که در طول یادگیری با دیدن یک تعداد episode متوسط پاداش دریافتی این الگوریتم از الگوریتم‌ها گذشته بیشتر است. این موضوع به دلیل یافتن سیاست بهتر است. یعنی در هر تعداد episode سیاستی که الگوریتم double q-learning بدست می‌آورد از سیاست‌های دیگر الگوریتم‌ها به سیاست بهینه نزدیک تر است. از طرف دیگر دیده می‌شود که در ۲۵۰ episode این الگوریتم به متوسط پاداش سه می‌رسد و به سیاست بهینه‌ی greedy هم گرا می‌شود. این در حالی است که دیگر الگوریتم‌ها بعد از ۷۵۰ episode به همگرایی می‌رسند. این نشان می‌دهد که سرعت همگرایی در آن از بقیه بیشتر است. در بررسی سرعت ما تعداد episode مورد بررسی را مقایسه می‌کنیم. هزینه‌ی محاسباتی هر گام از این الگوریتم از الگوریتم tree back-up کمتر است که دلیل آن محاسبه بر اساس پاداش دریافتی لحظه‌ای است. در زیر پاسخ این الگوریتم برای حل بازی قابل مشاهده است.

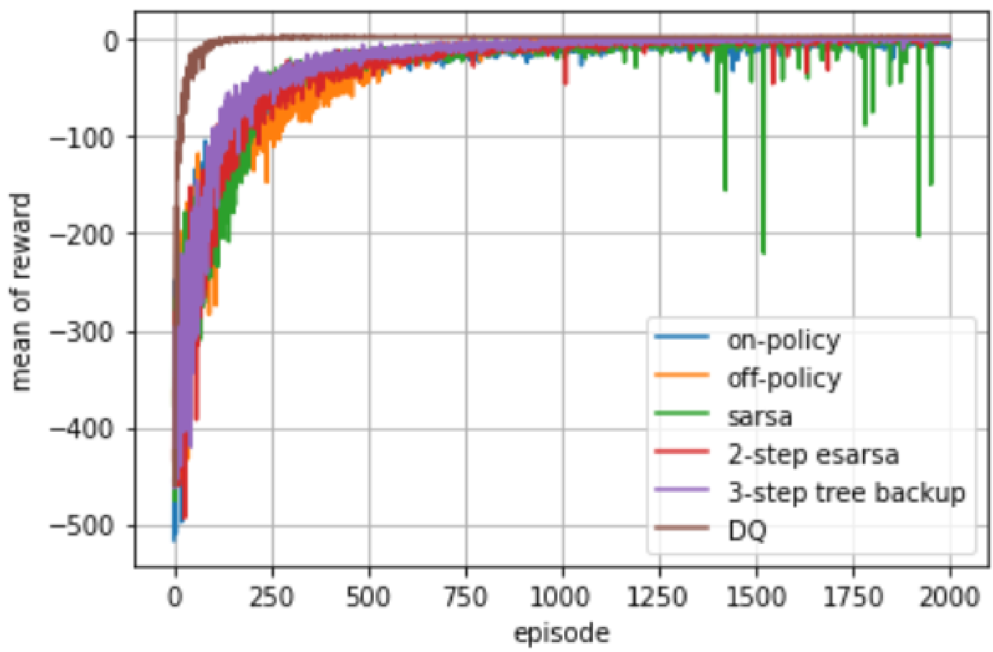


پاسخ پرسش‌ها

پرسش اول

در زیر نمودار متوسط پاداش به ازای تعداد episode برای تمام الگوریتم‌ها در کنار هم آمده است. همچنین متوسط کل پاداش در طول یادگیری برای هر الگوریتم به صورت عدد در مقابل نام الگوریتم آمده است.

```
on-policy  : -30.861375
off-policy  : -31.541875
sarsa  : -31.023625
2-step esarsa  : -28.2425
3-step tree backup  : -23.695375
DQ  : -0.31975
```

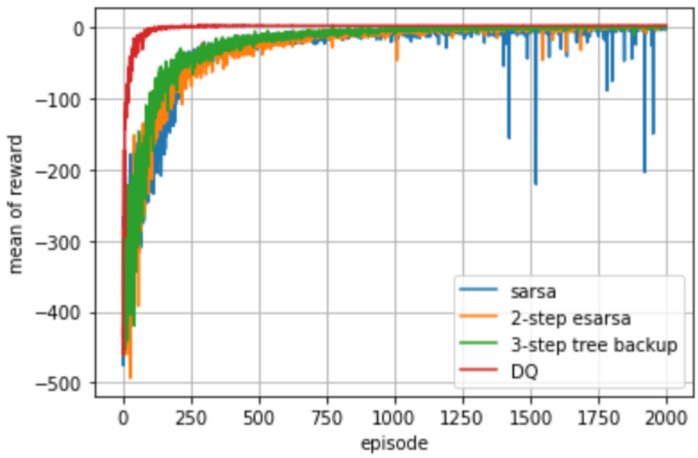


همانطور که دیده می‌شود در تمام الگوریتم‌ها با افزایش تعداد episode های دیده شده متوسط پاداش دریافتی افزایش می‌یابد و تمام الگوریتم‌ها همگرا می‌شوند و متوسط پاداش دریافتی آن‌ها از تعداد episode ای به بعد ثابت می‌شود. ما به دنبال سیاست بهینه هستیم که راه حلی برای بازی است که پاداش متوسط کل راه آن برابر سه باشد. پس رسیدن به متوسط پاداش سه به این معنی است که در بارهای مختلف اجرا راه حل پیشنهادی الگوریتم پاداش سه داشته است و این به این معنی است که ما به سیاست بهینه

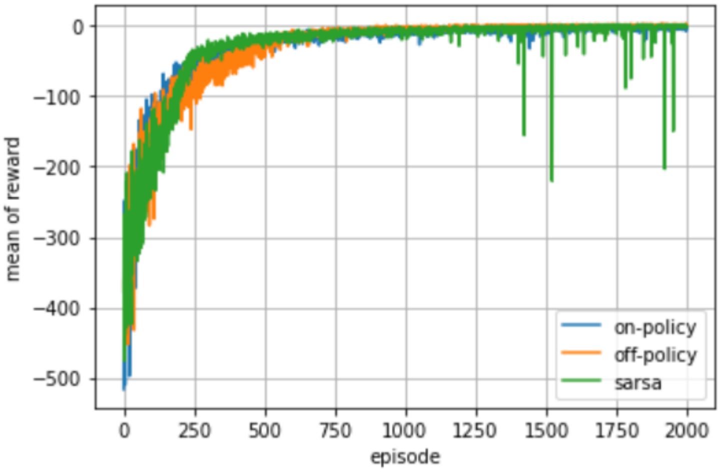
رسیده‌ایم. همان طور که دیده می‌شود نمودار الگوریتم DQ در طول یادگیری بالاتر از سایر نمودارها قرار دارد. این بدین معنا است که در طول یادگیری با دیدن یک تعداد episode متوسط پاداش دریافتی این الگوریتم از دیگر الگوریتم‌ها بیشتر است. این موضوع به دلیل یافتن سیاست بهتر است. یعنی در هر تعداد episode سیاستی که الگوریتم double q-learning بدست می‌آورد از سیاست‌های دیگر الگوریتم‌ها به سیاست بهینه نزدیک تر است. در تمام تعداد episode ها مقدار متوسط پاداش الگوریتم DQ از همه بالا تر است. به همین دلیل متوسط کل پاداش‌های دریافتی در آن از همه بیشتر می‌باشد. در قبل برای هر الگوریتم دیدیم که نمودار متوسط پاداش به ازای تعداد episode در تمام الگوریتم‌ها یکسان و اکیدا صعودی است که در انتها ثابت می‌شود. تمام الگوریتم‌ها جدودا از یک مقدار متوسط پاداش شروع می‌کنند و همه به یک مقدار همگرا می‌شوند. پس اگر نمودار یک الگوریتم بالا تر از دیگری باشد، در این صورت متوسط پاداش در کل برای آن الگوریتم بیشتر خواهد بود. الگوریتم های DQ، 3-step tree back-up ، two-step expected sarsa، sarsa و off-policy MC این شرایط را دارند که همه به سه همگرا می‌شوند. بر اساس متوسط پاداش کل داریم که به ازای هر تعداد episode باید مقدار متوسط پاداش الگوریتم ها به صورت زیر باشد.

Off-policy MC < sarsa < 2-step expected sarsa < tree back-up < double q-learning

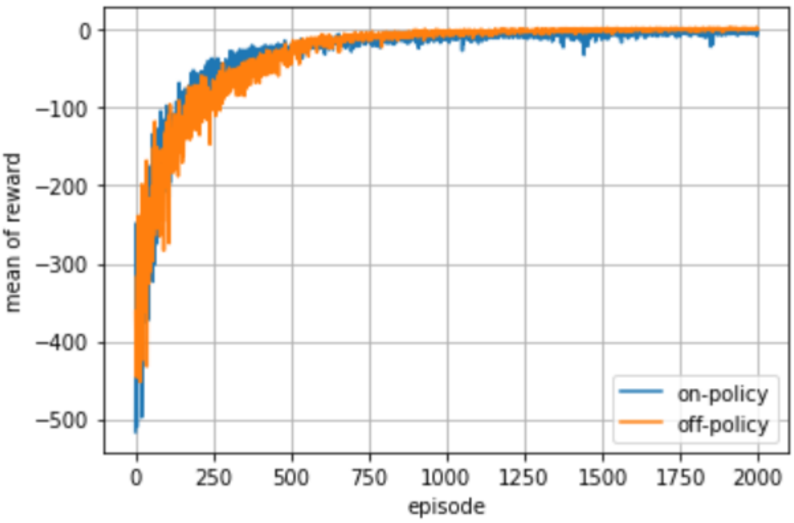
همان طور که در نمودار بالا دیده می‌شود، بعد از الگوریتم DQ الگوریتم 3-step tree back-up از بقیه الگوریتم‌ها در تمام تعداد episode ها مقدار بیشتری دارد. در زیر نمودار تعدادی از الگوریتم‌های باقی مانده در کنار هم آمده است.



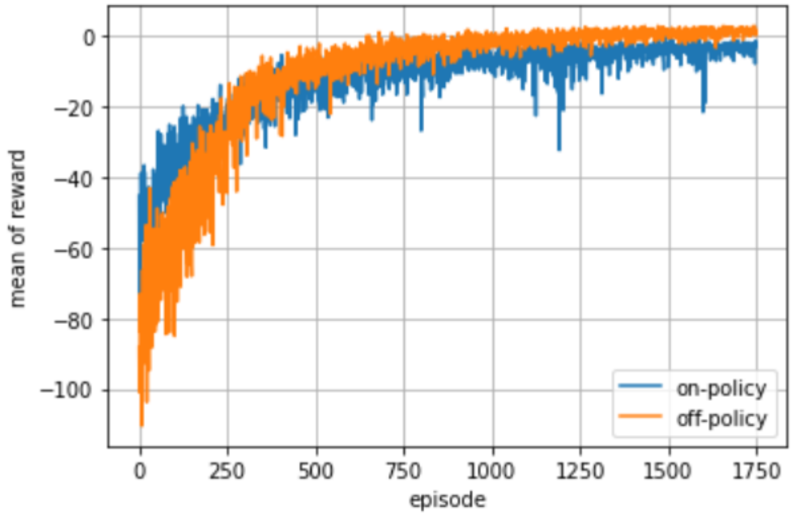
همان طور که دیده می‌شود مقدار متوسط پاداش در هر تعداد episode در الگوریتم 2-step expected sarsa از sarsa بیشتر است. در زیر نمودار الگوریتم‌های باقی مانده آمده است.



همان طور که دیده می‌شود در هر تعداد episode مقدار متوسط پاداش در الگوریتم sarsa بیشتر مساوی این مقدار در الگوریتم off-policy است. این موارد در بالا در یک خط نیز بیان شدند. الگوریتم on-policy به سیاست بهینه greedy همگرا نمی‌شود و مقدار پاداش متوسطی که الگوریتم به آن همگرا می‌شود کمتر از سه است. در زیر نمودار برای دو الگوریتم off-policy MC و on-policy آمده است.

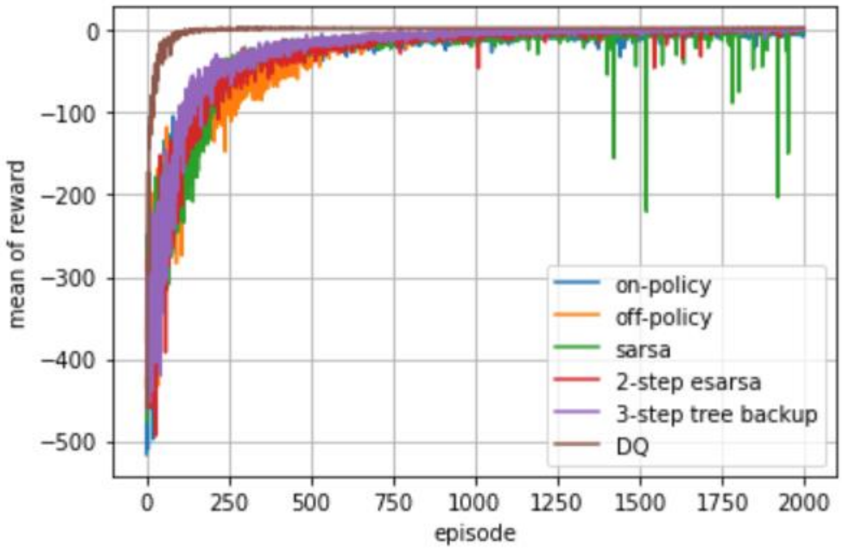


در زیر نمودار بالا برای تعداد episode بیشتر از ۲۵۰ آمده است. نمودار از تعداد ۲۵۰ episode به بالا با شروع از صفر نشان داده شده است. یعنی مقادیر متوسط پاداش در episode برابر صفر در نمودار پایین مربوط به ۲۵۰ تا episode است. همان طور که دیده می‌شود در تعداد ۵۰۰ تا episode نمودار دو الگوریتم به هم برخورد می‌کنند. تا قبل از ۵۰۰ تا episode مقدار متوسط پاداش در الگوریتم on-policy بیشتر از off-policy است. از ۵۰۰ episode تا انتها مقدار متوسط پاداش الگوریتم off-policy از on-policy بیشتر است. نکته قابل توجه دیگر این است که on-policy زودتر از off-policy همگرا شده است که دلیل آن sample efficiency پایین در الگوریتم off-policy است.



پرسش دوم

الگوریتم double q-learning از نظر سرعت یادگیری عملکرد بهتری دارد. این موضوع به وضوح در زیر قابل مشاهده است. با توجه به تعریف مسئله و پاداش‌ها، سیاست بهینه سیاستی است که در هر بار اجرا ما پاداش برابر سه دریافت کنیم. یعنی زمانی که یک الگوریتم یادگیری به متوسط پاداش سه همگرا شود، به سیاست بهینه رسیده است. همانطور که در زیر دیده می‌شود، الگوریتم double q-learning قبل از دیدن ۲۵۰ تا episode به متوسط پاداش سه همگرا می‌شود. این در حالی است که دیگر الگوریتم‌ها بعد از دیدن ۷۵۰ تا episode همگرا می‌شوند و سیاست بهینه‌ی خود را می‌یابند.



قسمت امتیازی

برای رسیدن به همگرایی سریعتر از استاندارد سازی پاداش‌ها استفاده می‌کنیم. دیدیم که پاداش‌ها با توجه به ماهیت فیزیکی و به صورت معقول تهیه شده‌اند. ما برای سریع تر کردن همگرایی این مقادیر پاداش را به شکل زیر استاندارد می‌کنیم. برای استاندارد سازی ابتدا تمام پاداش‌ها را منهای میانگین پاداش‌ها می‌کنیم. این کار برای این انجام می‌شود که پاداش حرکت بدون برخورد به موانع که برابر صفر است، صفر باقی بماند. در ادامه از استاندارد سازی زیر استفاده می‌کنیم.

Standardization:

$$z = \frac{x-\mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$$

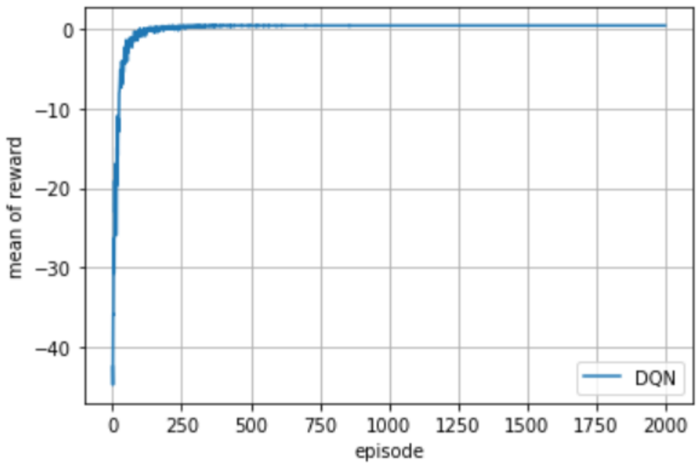
```
mu_ = np.mean(np.array([3,-5,-15,0]))
std_ = np.std(np.array([3,-5,-15,0])-np.array([mu_,mu_,mu_,mu_]))
mu_ = 0

r0 = (0-mu_)/std_
rm5 = (-5-mu_)/std_
rm15 = (-15-mu_)/std_
rp3 = (3-mu_)/std_

print(r0,rm5,rm15,rp3)
```

```
0.0 -0.7317617332646023 -2.195285199793807 0.4390570399587614
```

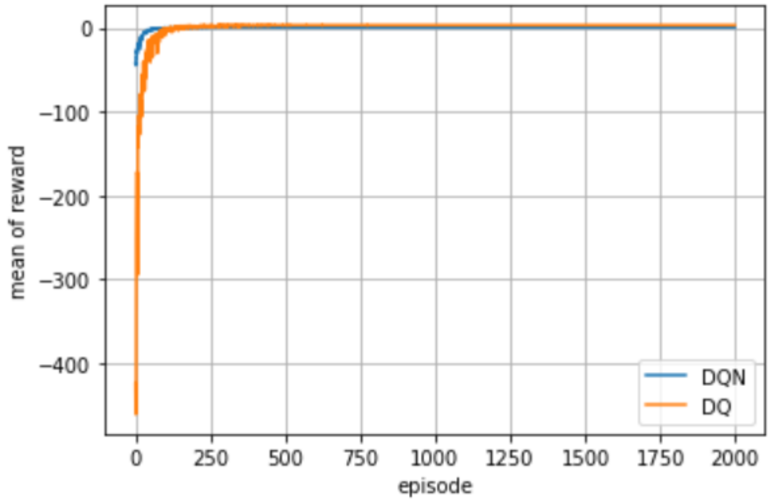
بعد از این عملیات، مقادیر جدید پاداش‌ها در بالا آمده است. در این روش پاداش متناظر با حرکت بدون برخورد به مانع صفر باقی مانده است(r0). پاداش برخورد به موانع منفی هستند و پاداش برخورد به مانع تیره (rm5) از گرفتاری در نواحی نیلی (rm15) مانند گذشته کمتر است. پاداش حرکت در جزیره (rp3)

مثبت است و نسبت اندازه‌ی پاداش‌ها حفظ شده است. این استاندارد سازی range پاداش‌ها را یکسان می‌کند و میانگین پاداش‌ها را نزدیک به صفر و واریانس را برابر یک می‌کند. این به عامل RL اجازه می دهد اقدامات خوب و بد را به طور موثرتری تشخیص دهد. یک مثال: تصور کنید ما در حال تلاش برای ایجاد یک عامل برای عبور از خیابان هستیم ، و اگر از خیابان عبور کند ، پاداش 1 می گیرد. اگر به یک ماشین برخورد کند ، پاداش -1 دریافت می کند ، و هر گام برداشتن پاداش 0 دارد. از نظر درصد ، پاداش موفقیت کاملاً بالاتر از پاداش ناکامی (ضربه خوردن از ماشین) است. با این حال ، اگر ما برای عبور موفقیت آمیز از جاده ، مبلغ 1،000،000،001 پاداش به نماینده بدهیم و به دلیل برخورد با اتومبیل به او 999،999،999 پاداش بدهیم (این سناریو و موارد فوق در صورت نرمال و استاندارد شدن یکسان هستند) ، موفقیت دیگر به اندازه قبل مشخص نیست. تشخیص موفقیت و شکست با استفاده از Discounted return با توجه به چنین پاداش های بالایی دشوارتر می شود. به همین دلیل باید از استاندارد سازی به شکل بالا استفاده کنیم.

با توجه به این پاداش‌های جدید ما environment را به روز کردیم. در قسمت‌های قبل دیدیم که الگوریتم double q-learning سرعت همگرایی از بقیه الگوریتم‌ها بیشتر است. در اینجا این الگوریتم را بر روی محیط جدید مانند قبل اجرا می‌کنیم تا تاثیر پاداش‌های جدید در همگرایی این الگوریتم را ببینیم. اگر همگرایی از قبل پریعتر و در تعداد کمتری رخ دهد، این موضوع بدین معنا است که پاداش‌های جدید به همگرایی سریعتر کمک می‌کنند. نمودار متوسط پاداش بر اساس تعداد episode برای الگوریتم DQ با پاداش‌های جدید در زیر آمده است.

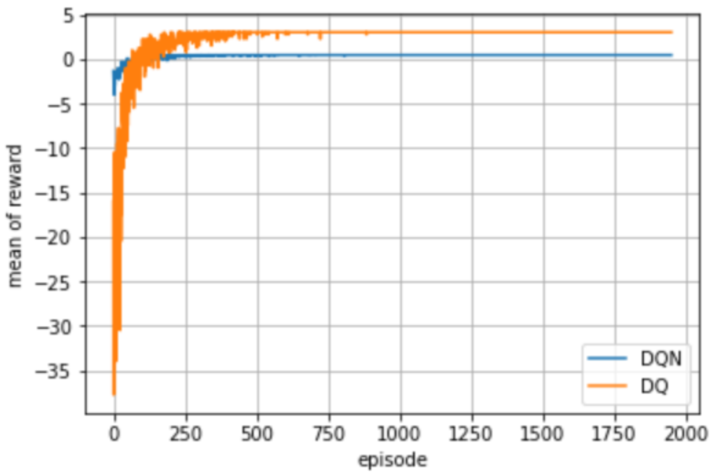


هیچ چیز در الگوریتم و نحوه‌ی اجرا تغییر نکرده است و فقط مقدار پاداش‌ها تعییر کرده است. همانطور که دیده‌می‌شود مانند گذشته الگوریتم همگرا می‌شود. الگوریتم double qlearning با پاداش‌های جدید را DQN می‌نامیم. در زیر نمودار DQN را در کنار نمودار DQ رسم کرده‌ایم.



در زری نمودار بالا از تعداد ۵۰ تا episode به بالا آمده است. مقادیر مربوز به ۵۰ تا episode در نمودار زیر در تعداد صفر episode به نمایش درآمده است.

```
DQN  : 0.0137754146287063
DQ  : -0.31975
```



همانطور که دیده می شود نمودار DQN در بین تعداد episode صفر تا ۲۵۰ در نمودار بالا همگرا شده است و این در حالی است که نمودار DQ بعد از ۲۵۰ تا episode در نمودار بالا همگرا شده است. این موضوع نشان می‌دهد که همگرایی DQN سریع تر DQ است. این به این معنی است که پاداش‌های جدید باعث سرعت در همگرایی می‌شود.