دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره چهارم

درس یادگیری تعاملی

پاییز ۱۴۰۰

نام و نام خانوادگی	علی ساعی زاده
شماره دانشجویی	۸۱۰۱۹۶۴۷۷

فهرست

چکیده

این تمرین یک مسئله مسیریابی برای یک دریاچه یخی است که هر خانه آن با احتمالی می‌شکند که در این تمرین میخواهیم به کمک روش های یادگیری مختلف مسیر بهینه را پیدا کنیم. در این گزارش روش های model-based و model-free و ترکیب آن‌ها بررسی خواهد شد. در روش model-based روش value iteration پیاده‌سازی و اجرا می شود و به عنوان مرجع سیاست بهینه برای مقایسه و تعیین عملکرد روش های دیگر مورد استفاده قرار می‌گیرد.

روش های مختلف model-free اعم از Monte Carlo off policy ، Q-Learning، SARSA و ۲-step tree backup پیاده‌سازی شده و مطابق دستور کار با یکدیگر در شرایط مختلف مقایسه خواهند شد. در قسمت پایانی نیز ترکیب دو روش value iteration و Q-Learning در یک محیط نسبتا پویا پیاده می‌شود. در این گزارش به تفصیل در مورد این روش ها و عملکرد آن ها بحث خواهد شد.

سوال ۱ - بررسی روش های Model-Based

هدف سوال

در این قسمت عاملی واقف بر شرایط محیط توسعه داده می‌شود تا سیاست بهینه و متوسط پاداش بهینه برای این محیط مشخص شود تا بتوان روش های مختلف را از منظر حسرت و نزدیکی به سیاست بهینه مقایسه کنیم. مدل محیط و احتمال سقوط در شکل زیر نشان داده شده است. در این محیط چهار عمل اصلی بالا، پایین، چپ و راست قابل اجراست همچنین این محیط شامل ۱۶ خانه است که حالت های مسئله مارا تشکیل می‌دهند.

```
-----------------------------
| 0.000 | 0.181 | 0.467 | 0.018 |
-----------------------------
| 0.001 | 0.092 | 0.622 | 0.358 |
-----------------------------
| 0.001 | 0.001 | 0.001 | 0.001 |
-----------------------------
| 0.153 | 0.717 | 0.942 | 0.000 |
-----------------------------
```

شکل ۱ محیط مسئله

توضیح پیاده سازی

پیاده سازی این قست درست مانند تمرین قبلی پیاده شد که شبه کد آن طبق کتاب مرجع [۱] در شکل ۲ آورده شده است.

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

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

شکل ۲ شبه کد **Value Iteration**

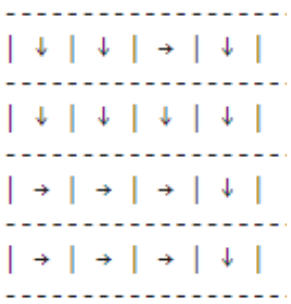
این عامل با اطلاعات کامل از محیط مسئله MDP را به کمک روش Value Iteration حل می‌کند لازم به ذکر است که سیاست بدست آمده سیاست بهینه خواهد بود. تنها پیچیدگی این مسئله احتمال سقوط در هر خانه بود که به کمک تابع امید ریاضی از پاداش دریافتی این مسئله حل شد یعنی احتمال سقوط و عدم سقوط و برد عامل و مقادیر آنها محاسبه شد و ارزش

هر خانه به کمک آن محاسبه شد. همچنین حل این مسئله در حدود ۶,۸ ثانیه به طول انجامید. لازم به ذکر است که مقدار discount factor = ۰,۹ و theta = ۰,۱ در نظر گرفته شد.

نتایج

پس از حل و همگرایی روش، عمل بهینه روی نقشه برای هر خانه با فلش در شکل ۳ مشخص شده است. همچنین مقادیر Q-Value برای هر حالت-عمل بدست آمده است که به کمک تابع get_q_values() از عامل ما قابل بازیابی است. نمونه ای از این مقادیر را برای حالت (۰,۰) برای چهار عمل ما آورده شده است.

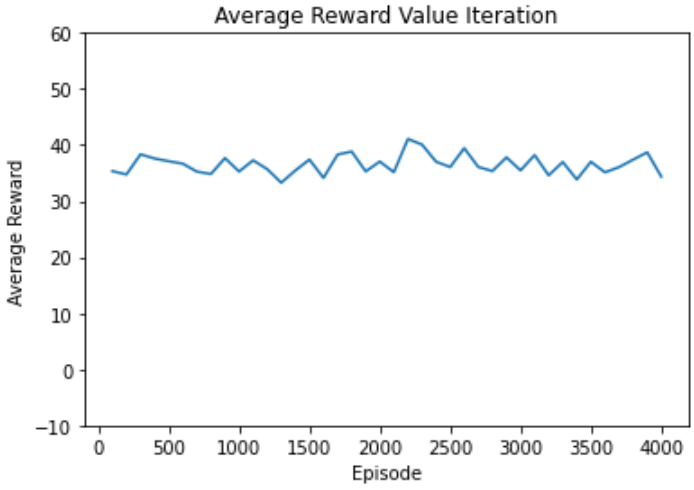
```
Q((۰,۰),a) =  [۲۲۸,۰۷۶۱۰۱۵۱ ۲۵۳,۱۵۳۲۴۸۲۷ ۲۵۰,۴۶۵۰۷۳۱  ۲۲۸,۰۷۶۱۰۱۵۱]
```



شکل ۳ عمل بهینه برای هر خانه

همچنین سیاست بهینه توسط تابع get_policy() قابل دریافت است.

در این قسمت متوسط پاداش دریافتی در ۴۰۰۰ اپیزود در یک بار اجرا رسم شد که در شکل قابل مشاهده است.



شکل ۴ متوسط مجموع پاداش دریافتی در یک بار اجرا برای عامل **Value Iteration**

روند اجرای کد پیاده‌سازی

لازم به ذکر به کلاس محیط دو تابع برای همخوانی بیشتر با کد اضافه شده که کلاس جدید تحت عنوان NSFrozenLake_modeified از مدل اولیه نشات گرفته است و با اجرای سلول ربوط به آن قابل استفاده است. توابع اضافه

شده به منظور دریافت مستیم فضای حالت-عمل از محیط اضافه شده است. همچنین طول پنرجه برای کلیه قسمت برابر با ۵۰ درنظر گرفته شده است.

البته برای پیاده سازی نکته خاصی نیاز نیست تنها سلول ها از اول باید اجرا شود.

سوال ۲ - بررسی روش‌های Model-Free

هدف سوال

در این قسمت هدف این است که روش های مختلف Model-Free و پارامترهای آن‌ها از منظر حسرت و مجموع پاداش دریافتی با یکدیگر مقایسه شوند. در این سوال روش مختلف اعم ازOff-Policy MC ، Q-Leaning، SARSA و ۲-Step Tree Backup پیاده سازی شد و طبق خواسته صورت تمرین با یکدیگر مقایسه شدند.

روش Off-Policy MC

توضیح پیاده سازی

در این قسمت با استفاده از روش Off-Policy MC به کمک سیاست رفتاری epsilon-greedy حل شد. طبق خواسته سوال این روش در دو حالت با اپسیلون ثابت ۰,۱ و کاهشی پیاده سازی شد و عملکرد آن در محیط مقایسه شد. در این روش یک اپیزود کاملا به کمک سیاست رفتاری اجرا می شود و سپس حالت های این اپیزود از آخر بررسی می شود و مقادیر Q-Value ها بروز رسانی می شود شبه کد [۱] این قست که کد براساس آن توسعه داده شده است به شکل زیر 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)

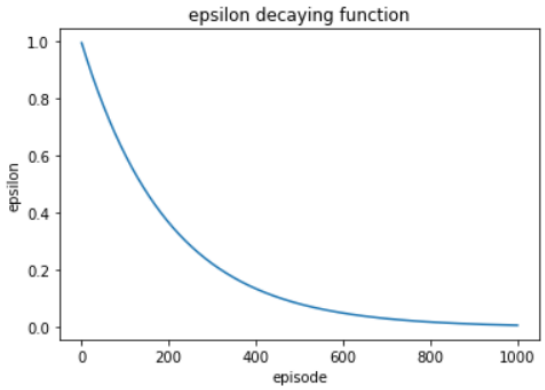
Loop forever (for each episode):
$b \leftarrow$ any soft policy
Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t)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

شکل ۵ شبه کد روش **Off-Policy MC**

اپسیلون و ثابت یا کاهشی بودن آن به شکل ورودی به عامل داده می‌شود. برای کاهش اپسیلون از تابع زیر در معادله ۱ استفاده شد که عملکرد خوبی را در این مسئله نشان داد که در آن i شماره اپیزود است. در شکل ۶ رفتار این تابع نشان داده شده است.

معادلهٔ ۱

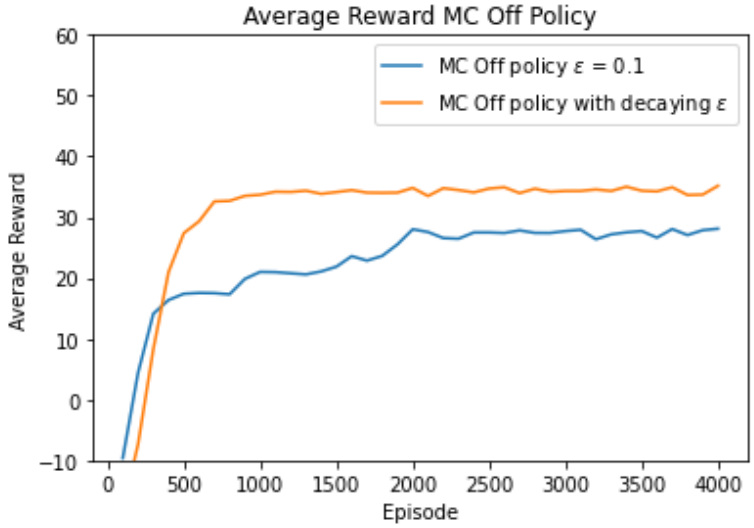
$$\epsilon = \frac{\epsilon}{۷}(۰.۹۹۵)^i$$



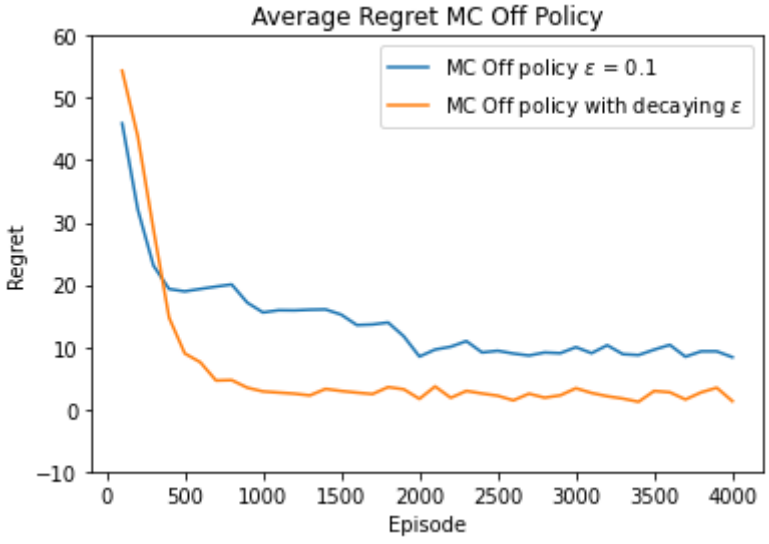
شکل ۶ رفتار تابع کاهشی برای اپسیلون

نتایج

برای این دو روش برای ۴۰۰۰ اپیزود و ۲۰ بار اجرا توسط مجموع پاداش دریافتی در شکل ۷ و حسرت در شکل ۸ ضمیمه شده است (حسرت به کمک پاداش دریافتی روش بهینه Value Iteration محاسبه شد).



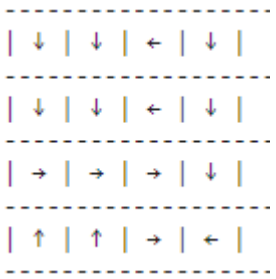
شکل ۷ متوسط مجموع پاداش دریافتی دو روش **Off-Policy MC**



شکل ۸ حسرت در افق ۴۰۰۰ اپیزود برای دو مدل از روش **Off-Policy MC**

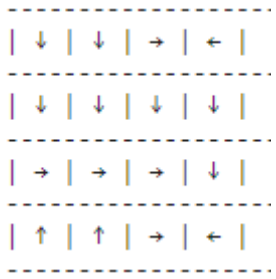
همانظور که در شکل ۷ و ۸ مشخص شده است روش اپسیلون ثابت به مقدار ۱۰ برای حسرت همگرا می‌شود که بهترین نتیجه نیست اما به نسبت قابل قبول است اما در مقابل، روش اپسیلون کاهشی برای این روش بسیار عالی عمل کرده و تقریبا حسرت را به صفر رسانده است و همچنین سرعت همگرایی آن به مراتب بیشتر است؛ دلیل آن است که محیط برای عامل در اپیزودهای اولیه کاملا ناشناخته است و برای یادگیری، عامل نیاز دارد بیشتر به explore بپردازد و محیط را پیمایش کند. برای همین مقادیر بالای اپسیلون در مراحل اولیه بسیار در سرعت همگرایی کمک کننده خواهد بود اما در پایان اپیزود های پایانی به دلیل اینکه محیط کوچک است و عامل تقریبا همه حالت ها را دیده است و محیط را یادگرفته است نیاز دارد تا greedy عمل کند بنابراین اپسیلون‌های نزدیک صفر به خوبی عامل را در exploit کردن و رسیدن به مقدار همگرایی (حداکثر پاداش) کمک خواهند کرد.

همچنین اعمال حریصانه این روش ها با روش بهینه مقایسه شد. برای اپسیلون ثابت به طور متوسط در پایان ۴۰۰۰ اپیزود و ۲۰ بار اجرا ۵,۵۸ عمل بهینه تفاوت است و و برای یک بار اجرا در پایان افق سیاست به شکل ۹ است.



شکل ۹ اعمال بهینه برای عامل **off-policy MC** با اپسیلون ثابت

روش اپسیلون کاهشی به به طور متوسط در پایان افق ۴۰۰۰ اپیزود و ۲۰ بار اجرا ۵,۶۸ عمل متفاوت با سیاست بهینه داشت که این اعمال ربوط به خانه هایی هستند که تفوات در تصمیم گیری در آنها حیاتی نیست و قابل چشم پوشی است (برای مثال خانه های چپ پایین که راست رفتن و بالا رفتن فرق چندانی نمی کند).



شکل ۱۰ اعمال بهینه برای عامل **off-policy MC** با اپسیلون کاهشی

روش Q-Learning

توضیح پیاده سازی

در این قسمت الگوریتم Q-learning پیاده سازی و در محیط اجرا شد. طبق خواسته تمرین در دو حالت با نرخ یادگیری ثابت (۰٫۱) و کاهشی این روش پیاده سازی گردید و مقایسه شد این الگوریتم به کمک شبه کد [۱] در شکل پیاده سازی شده است. برای این قسمت نیز از همان تابع قسمت قبل (معادله ۱) برای کاهش دادن نرخ یادگیری استفاده شد.

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

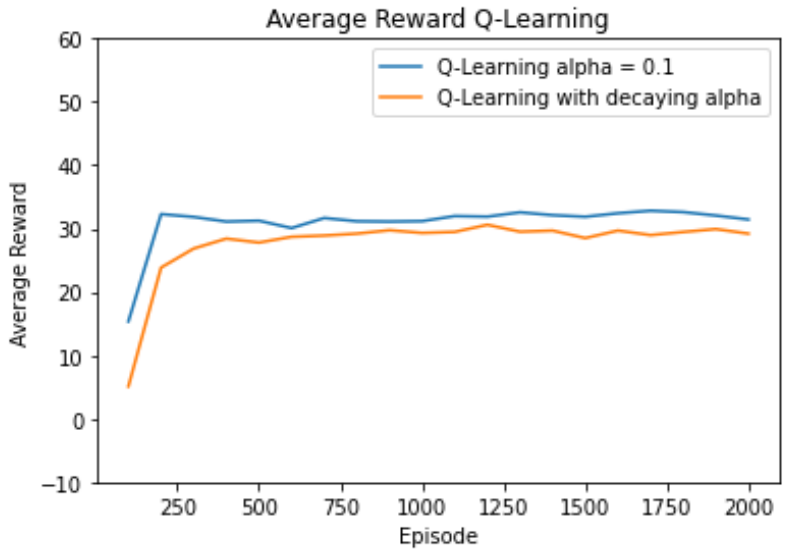
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

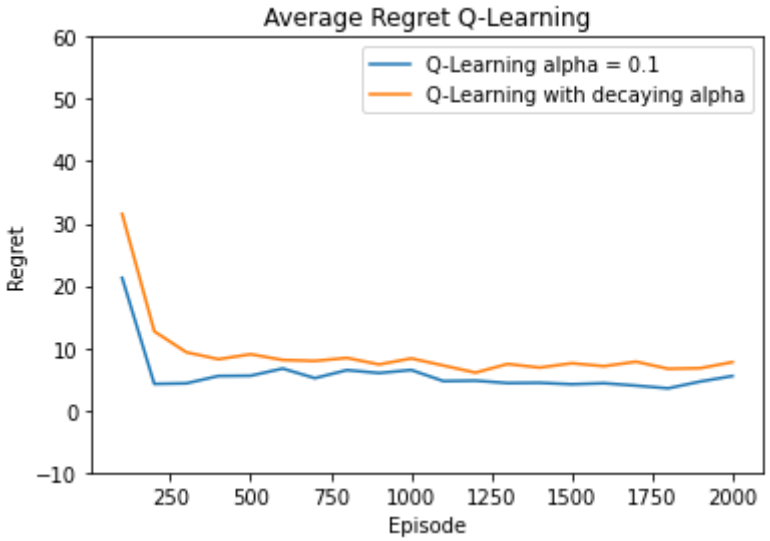
شکل ۱۱ شبه کد الگوریتم **Q-Learning**

نتایج

نتایج بدست آمده از لحاظ متوسط مجموع پاداش دریافتی و حسرت در افق ۲۰۰۰ اپیزود در ۲۰ بار اجرا به ترتیب در شکل ۱۲ و ۱۳ آمده است.



شکل ۱۲ متوسط مجموع پاداش دریافتی برای دو حالت از رورش**Q-Learning**



شکل ۱۳ حسرت برای دو حالت از روش **Q-Learning**

بطور کلی سرعت همگرایی برای نرخ یادگیری کاهشی کمی کمتر است و همچنین حسرت این روش به مقدار بیشتری همگرا می‌شود (حدودا ۱۰) اما تفاوت مقدار همگرا شده فاحش نیست. همانطور که مشخص است کاهش نرخ یادگیری کمک خاصی به بهتر کردن این روش در این مسئله نمی کند. شاید دلیل این اتفاق این باشت که به دلیل کوچک بودن فضای مسئله و مقادیر Q، یادگیری اتفاق می افتد مستقل از اینکه نرخ یادگیری چه باشد. اما اگر فضای مسئله بزرگ تر بود نیاز بود با پیدا کردن مسیر بهینه (حتی محلی) دست از یادگیری بکشیم و پاداش دریافتی خود را حداکثر کنیم. یعنی ابتدا نرخ یادگیری بالا داشته باشیم سپس آن را به صفر سوق دهیم.

توضیح پیاده سازی

در این قسمت دو روش Sarsa و ۲-step tree backup پیاده سازی شده اند. که در شکل های ۱۴ و ۱۵ شبه کد [۱] آن ها آورده شده است. اما در روش دوم برای سادگی کار، شبه کد شکل ۱۵ بسیار ساده تر اجرا شده (با توجه به اینکه مقدار n برابر با ۲ است) و در هر گام دو گام برداشته شده و با توجه به معادله موجود در کتاب [۱] که در معادله ۲ آورده شده است، پیاده‌سازی و ساده‌سازی شده است. در این روش Sampling Efficiency افزایش می یابد بدلیل اینکه از اطلاعات موجود در حالت های بعدی نیز استفاده می کنیم برای درک بهتر به شکل ۱۶ رجوع کنید.

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

شکل ۱۴شبه کد الگوریتم **Sarsa**

معادلهٔ ۲ معادله مربوط به روش ۲-**step tree backup**

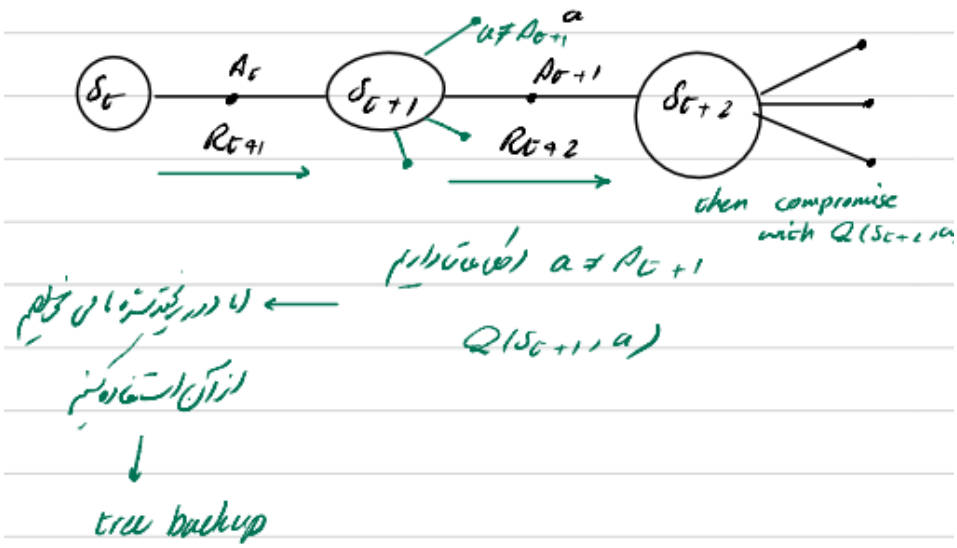
$$
\begin{aligned}
G_{t:t+2} &\doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\
&\qquad + \gamma \pi(A_{t+1}|S_{t+1}) \Big(R_{t+2} + \gamma \sum_{a} \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \Big) \\
&= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2},
\end{aligned}
$$

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 $\tau = T - 1$

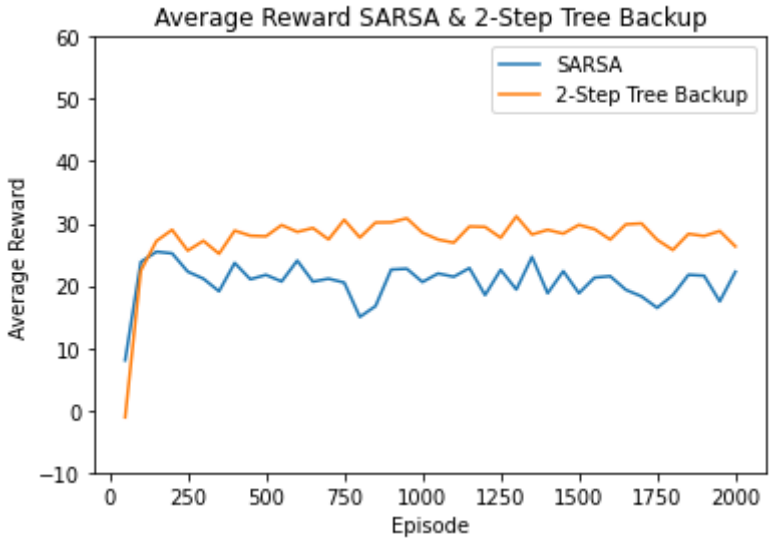
شکل ۱۵ شبه کد الگوریتم **n-step tree backup**



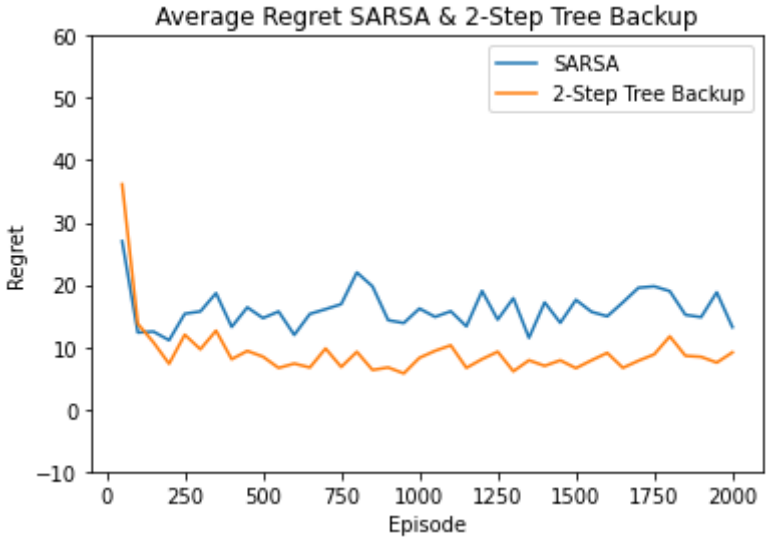
شکل ۱۶ چگونگی عملکرد الگوریتم **n-step tree backup**

نتایج

در این قسمت متوسط مجموع پاداش دریافتی و حسرت برای افق ۲۰۰۰ اپیزود برای ۲۰ اجرا، در شکل ۱۷ و ۱۸ مقایسه شده است. همانطور که مشخص است در سرعت همگرایی این دو روش تفاوت چندانی ندارند اما در مقدار همگراشده روش ۲-step tree backup به مراتب بهتر عمل می‌کند. دلیل این عملکرد بهتر را می توان در Sample Efficiency بیشتر این روش جست و جو کرد. همانطور که در تعریف این روش آمده است در این روش از اطلاعات ۲ گام بعدی نیز، برای بروزرسانی مقادیر Q استفاده می کند که باعث شناخت بهتر و بیشتر محیط می‌شود که این خود باعث می‌شود عامل بهینه تر عمل کند و به پاداش حداکثری برسد.



شکل ۱۷ متوسط پاداش دریافتی برای دو روش **sarsa** و **n-step tree backup**



شکل ۱۸ حسرت برای دو روش **sarsa** و **n-step tree backup**

سوال ۳ – ترکیب روش‌های Model-Based و Model-Free

هدف سوال

در این سوال فرض ثابت بودن احتمال شکستن یخ ها حذف شده است و محیط به محیط ناپایا تبدیل می‌شود. به عبارت دیگر، محیط با زمان تغییر می‌کند طبق صورت سوال برای حل این محیط از ترکیب روش Q-Learning و Value Iteration استفده شده است و این روش ها با وزن های متفاوت در حل مسئله و تعیین مقادیر Q نقش بازی می‌کنند. هدف از این این سوال پیدا کردن وزن بهینه است.

توضیح پیاده سازی

در این قسمت ابتدا محیط در یک حالت خود توسط عامل Value Iteration حل می شود و مقادیر Q_Model_Based استخراج می‌شود سپس عامل Q-Learning در محیط زندگی می کند و Q_Model_Free را بروزرسانی می‌کند. دقت شود در هر اپیزود احتمال شکستن یخ ها تغییر می‌کند. سپس برای گرفتن تصمیم از سیاست epsilon greedy با مقادیر Q(s,a) استفاده می‌شود که Q از معادله ۳ بدست می‌آید.

معادلهٔ ۳

$$Q(s.a) = w\, Q_{Model\ Free}(s.a) + (1 - w)\, Q_{ModelFree}\,(s.a)$$

سپس مقادیر مختلف w به روش فوق داده ی شود تا مقدار بهینه آن تعیین شود.

توجه: در معادله ای در صورت تمرین داده شده جای Q_Model_Based و Q_Model_Free نسبت به معادله ما، جابجاست بدلیل زمان اجرای طولانی (۴۰ دقیقه) و عدم تغییر در ماهیت مسئله این موضوع اصلاح نگردید.

نتایج

نتایج این قسمت در شکل ۱۹ نشان داده شده است. همانطور که انتظار می رفت هر چه نقش روش Model Based در تصمیم گیری بیشتر باشد پاداش کمتری دریافت خواهیم کرد اما با وجود اینکه این روش بهینه است، اما برای محیط پایا صادق است. در این مسئله محیط دائما در حال تغییر است و حل MDP اپیزود قبل برای اپیزود بعدی بهینه نخواهد بود. بنابراین روش Model Free به کمک ما می‌آید تا در این محیط ناپایا مسیر بیهنه را پیدا کنیم زیر برای هر حالت محیط تنها مسئله برای آن گام و حالت بروزرسانی می‌شود که باعث عملکرد خوب این روش در محیط های ناپایا می‌شود.

همانظور در شکل مشخص است w > ۰،۸ عملکرد خوبی از خود نشان داده‌اند. اما با توجه به جنس سئله این مقدار تغییر خواهد کرد. بنظر می‌آید هرچه تغییرات محیط بیشتر و قابل توجه تر باشد باید w های بزرگتری را انتخاب کنیم و بلعکس.

شکل ۱۹ توسط مجموع پاداش های دریافتی براساس **w**های مختلف

منابع

[۱] Sutton, Richard S,Andrew G. Barto, Reinforcement learning: An introduction, MIT press, ۲۰۱۸.