
بسم الله الرحمن الرحيم

عنوان

تمرین چهارم درس یادگیری ماشین (یادگیری تقویتی)

دانشجو

امیرحسین جراره – ۴۰۰۶۱۶۰۰۴

استاد درس

دکتر عبدی هجراندوست

۱) پیاده سازی Mountain Car با الگوریتم Sarsa

الگوریتم سارسا یا Sarsa که سرواژه عبارت State-Action-Reward-State-Action است شباهت زیادی با الگوریتم یادگیری Q دارد. تفاوت کلیدی این دو الگوریتم در این است که SARSA برخلاف الگوریتم یادگیریQ ، در دسته الگوریتم‌های On-Policy قرار می‌گیرد. بنابراین، الگوریتم SARSAمقدار Q-value را با توجه به اقدامی که ناشی از سیاست فعلی است محاسبه می‌کند نه اقدام ناشی از سیاست حریصانه.

اکنون قصد داریم از توابعی که به تازگی ایجاد کرده ایم برای پیاده سازی الگوریتم Sarsa استفاده کنیم. سارسا مخفف عبارت State، Action، Reward، State، Action است.

برای این مورد، ما یک تابع argmax مشابه آنچه در امتحان داشتیم ، این تابع با تابع argmax که توسط numpy استفاده می‌شود، که اولین شاخص یک مقدار حداکثر را برمی‌گرداند، متفاوت است. ما می خواهیم تابع argmax ما به طور دلخواه پیوندها را قطع کند، کاری که تابع argmax وارداتی انجام می دهد. تابع argmax داده شده آرایه ای از مقادیر را می گیرد و یک int از عملکرد انتخاب شده را برمی گرداند: argmax (مقدارهای اقدام)

راه‌های متعددی وجود دارد که می‌توانیم با اعمال کدکننده کاشی مقابله کنیم. در اینجا می خواهیم از یک روش ساده استفاده کنیم - اندازه بردار وزن را برابر (iht_size، num_actions) قرار دهیم. این به ما یک بردار وزن برای هر عمل و یک وزن برای هر کاشی می دهد.

از تابع بالا برای کمک به پر کردن select_action، agent_start، agent_step و agent_end استفاده می کنیم.

نکات:

کدگذار کاشی لیستی از شاخص های فعال را برمی گرداند (به عنوان مثال [۱، ۱۲، ۲۲]). شما می توانید یک آرایه numpy را با استفاده از یک آرایه از مقادیر ایندکس کنید - این آرایه ای از مقادیر را در هر یک از آن شاخص ها برمی گرداند. بنابراین برای به دست آوردن مقدار یک حالت، می‌توانیم بردار وزن خود را با استفاده از عمل و آرایه کاشی‌هایی که کدگذار کاشی برمی‌گرداند ایندکس کنیم:

این یک آرایه از مقادیر را به ما می دهد، یکی برای هر کاشی فعال، و ما نتیجه را جمع می کنیم تا مقدار آن جفت حالت-عمل را بدست آوریم.

در مورد یک بردار ویژگی باینری (مانند رمزگذار کاشی)، مشتق در هر یک از کاشی های فعال ۱ است و در غیر این صورت صفر است.

در انتها کافیست مانند sudo code زیر عمل کنیم.

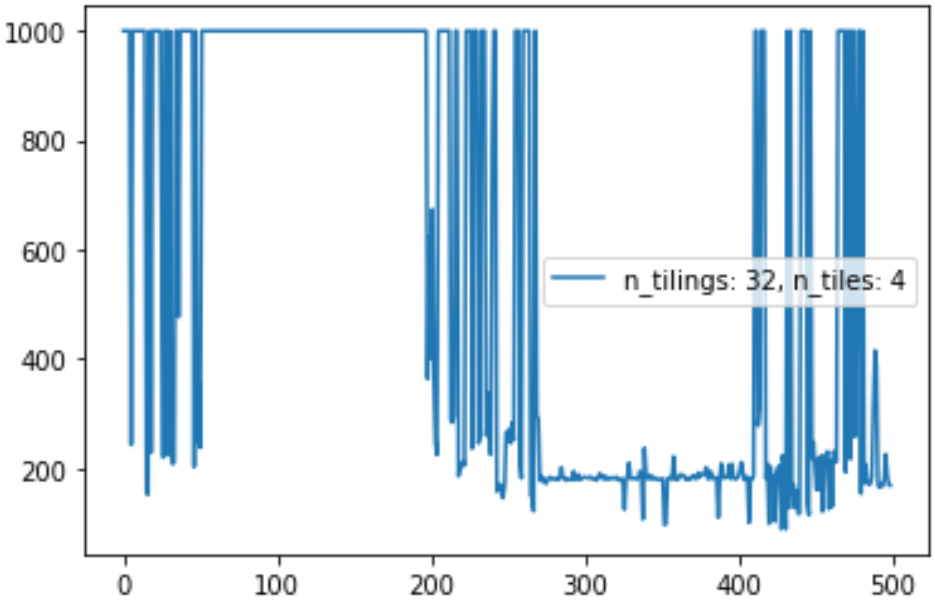
SARSA(λ): Learn function $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$

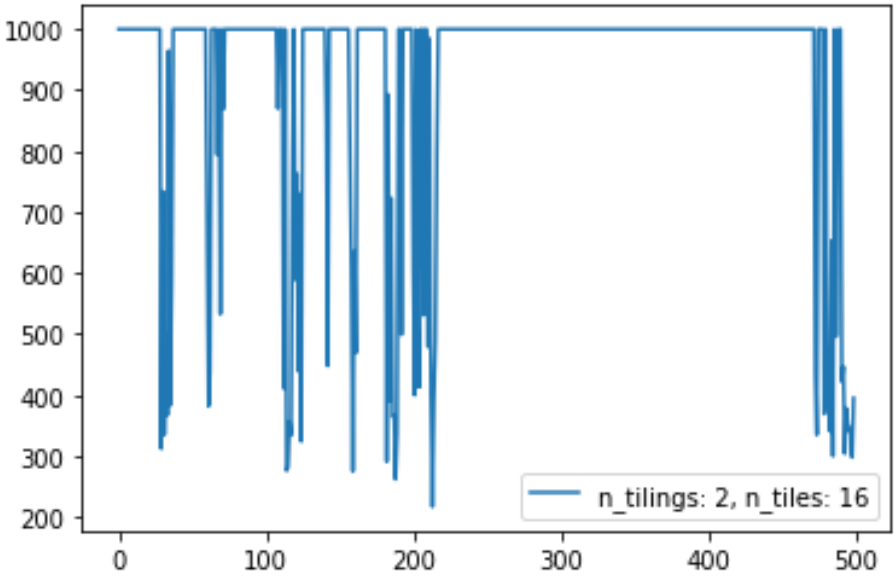
Require:
Sates $\mathcal{X} = \{1, \dots, n_x\}$
Actions $\mathcal{A} = \{1, \dots, n_a\}$, $A : \mathcal{X} \Rightarrow \mathcal{A}$
Reward function $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$
Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$
Discounting factor $\gamma \in [0, 1]$
$\lambda \in [0, 1]$: Trade-off between TD and MC
procedure QLEARNING($\mathcal{X}$, A, R, T, α, γ, λ)
    Initialize $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ arbitrarily
    Initialize $e : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ with 0. ▷ eligibility trace
    while Q is not converged **do**
        Select $(s, a) \in \mathcal{X} \times \mathcal{A}$ arbitrarily
        while s is not terminal **do**
            $r \leftarrow R(s, a)$
            $s' \leftarrow T(s, a)$ ▷ Receive the new state
            Calculate π based on Q (e.g. epsilon-greedy)
            $a' \leftarrow \pi(s')$
            $e(s, a) \leftarrow e(s, a) + 1$
            $\delta \leftarrow r + \gamma \cdot Q(s', a') - Q(s, a)$
            for $(\tilde{s}, \tilde{a}) \in \mathcal{X} \times \mathcal{A}$ **do**
                $Q(\tilde{s}, \tilde{a}) \leftarrow Q(\tilde{s}, \tilde{a}) + \alpha \cdot \delta \cdot e(\tilde{s}, \tilde{a})$
                $e(\tilde{s}, \tilde{a}) \leftarrow \gamma \cdot \lambda \cdot e(\tilde{s}, \tilde{a})$
            $s \leftarrow s'$
            $a \leftarrow a'$
    return Q

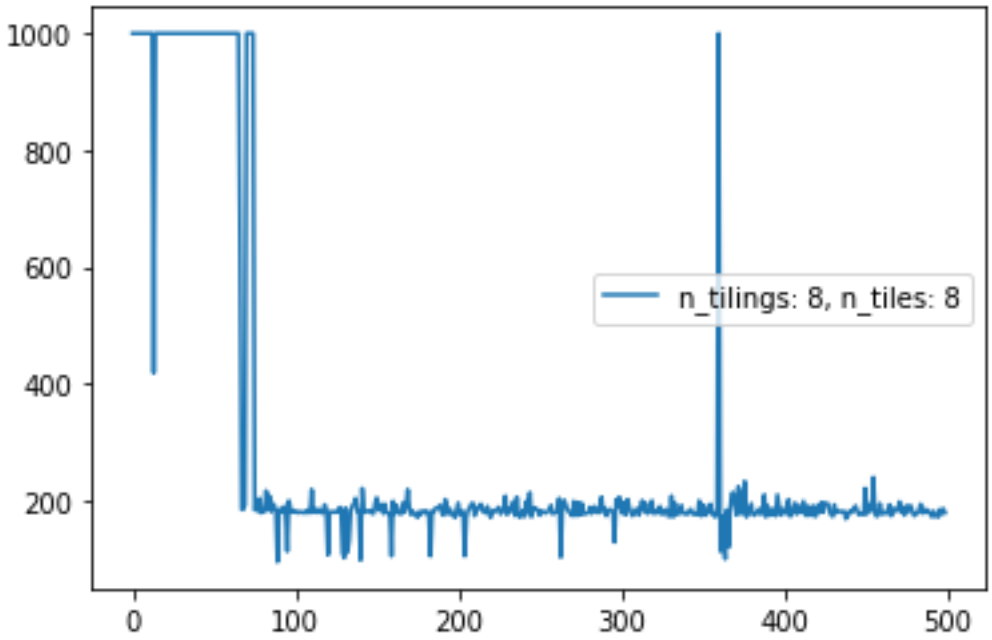
ممکن است در نمونه کد بالا متوجه شده باشید که هر دو اقدام اجرا شده از سیاست فعلی پیروی می‌کنند. اما در الگوریتم یادگیری Q، تا زمانی که اقدام بعدی بتواند مقدار Q-value برای وضعیت بعدی را حداکثر سازد، هیچ قیدی برای آن تعریف نمی‌شود. بنابراین همان‌طور که گفته شد، الگوریتم SARSA از نوع الگوریتم‌های On-policy است.

در نهایت برای ۱۵۰۰۰ تکرار خروجی به صورت زیر شده است.

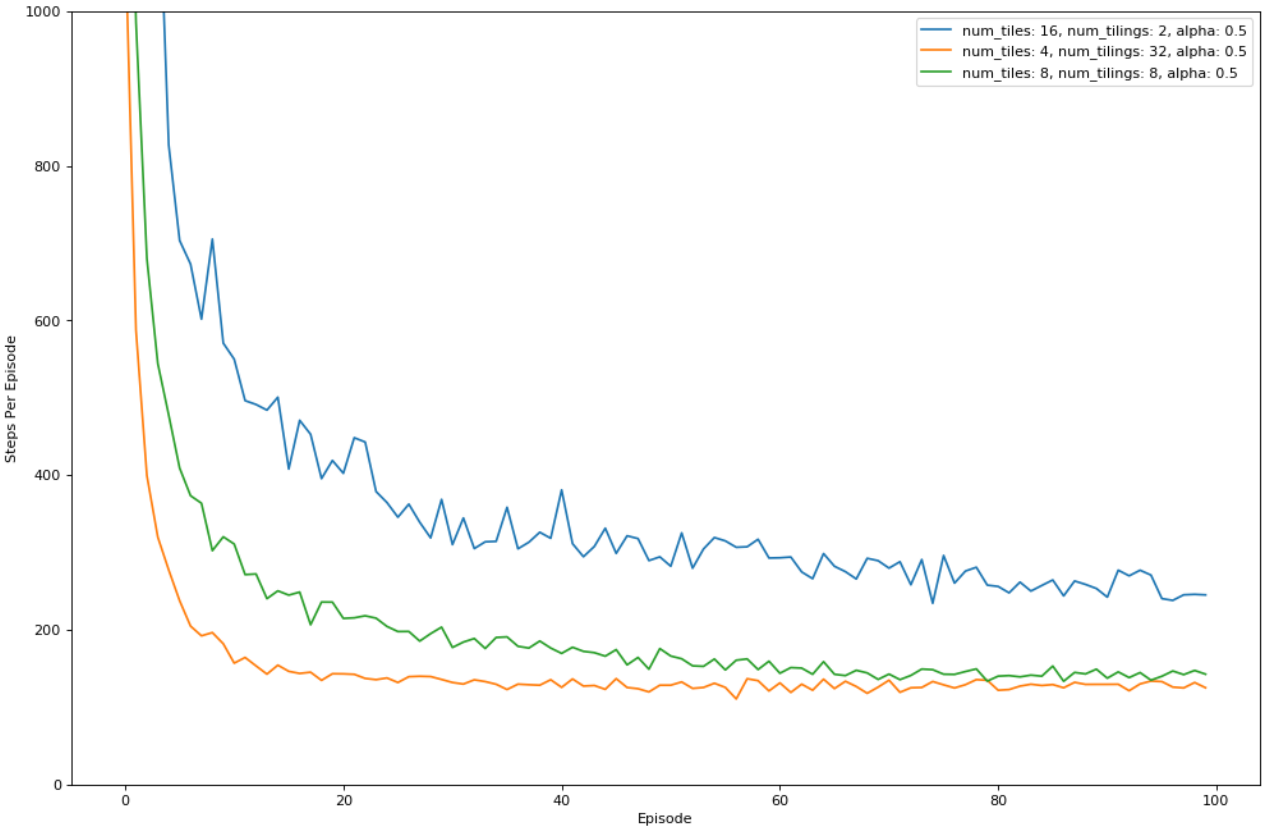
خروجی ارزیابی اولیه برای این سیستم برابر است با :







که اگر برای ۱۰۰۰ تلاش در ۱۰۰ تکرار با میانگین گیری بین ۱۰۰ نمونه آموزش به صورت زیر می باشد.



بهترین خروجی برای ۳۲ num_tiling با تعداد tiles برابر ۴ می باشد که در نهایت به صورت تقریبی برای ۸ num_tiling با تعداد tiles ۸ برابر شده است که هردو تفاوت و بهبود بهتری نسبت به حالت دیگر دارند.

ویدیو در پیوست ارائه شده است.

۲) پیاده سازی Lunar Lander با الگوریتم Deep Q-Network

متأسفانه، یادگیری تقویتی زمانی ناپایدارتر است که از شبکه های عصبی برای نمایش مقادیر عمل استفاده می شود، علیرغم استفاده از پوشش های معرفی شده در بخش قبل. آموزش چنین شبکه‌ای به داده‌های زیادی نیاز دارد، اما حتی در این صورت هم‌گرایی بر روی تابع مقدار بهینه تضمین نمی‌شود. در واقع، موقعیت‌هایی وجود دارد که وزن شبکه می‌تواند به دلیل همبستگی زیاد بین کنش‌ها و حالت‌ها، نوسان یا واگرا شود.

برای حل این مشکل، در این بخش دو تکنیک مورد استفاده توسط Deep Q-Network را معرفی می کنیم:

نکات و ترفندهای بیشتری وجود دارد که محققان برای پایدارتر و کارآمدتر کردن آموزش DQN کشف کرده‌اند که در پست‌های بعدی این مجموعه به بهترین آنها خواهیم پرداخت.

ما در حال تلاش برای تقریب یک تابع غیرخطی، Q(s، a) با یک شبکه عصبی هستیم. برای انجام این کار، باید اهداف را با استفاده از معادله بلمن محاسبه کنیم و سپس در نظر بگیریم که یک مشکل یادگیری نظارت شده در دست داریم. با این حال، یکی از الزامات اساسی برای بهینه‌سازی SGD این است که داده‌های آموزشی مستقل و به طور یکسان توزیع شده باشند و هنگامی که Agent با محیط تعامل می‌کند، توالی تاپل‌های تجربه می‌تواند به شدت همبستگی داشته باشد. الگوریتم ساده Q-learning که از هر یک از این تجربیات به ترتیب متوالی می آموزد، خطر تحت تأثیر قرار گرفتن تأثیرات این همبستگی را دارد.

ما می توانیم با استفاده از یک بافر بزرگ از تجربیات گذشته خود و نمونه داده های آموزشی از آن، به جای استفاده از آخرین تجربه خود، از نوسان یا واگرایی فاجعه بار مقادیر عمل جلوگیری کنیم. این تکنیک بافر تکرار یا بافر تجربه نامیده می شود. بافر پخش مجدد شامل مجموعه ای از تاپل های تجربه (S، A، R، S') است. تاپل ها به تدریج به بافر اضافه می شوند که ما در حال تعامل با محیط هستیم. ساده ترین پیاده سازی یک بافر با اندازه ثابت است که داده های جدیدی به انتهای بافر اضافه می شود تا قدیمی ترین تجربه را از آن خارج کند.

عمل نمونه برداری از دسته کوچکی از تاپل ها از بافر پخش مجدد به منظور یادگیری به عنوان تکرار تجربه شناخته می شود. علاوه بر شکستن همبستگی‌های مضر، بازپخش تجربه به ما امکان می‌دهد تا چندین بار از تاپل‌های منفرد بیشتر بیاموزیم، اتفاقات نادر را به خاطر بیاوریم و به طور کلی از تجربه‌مان بهتر استفاده کنیم.

به‌طور خلاصه، ایده اصلی پشت بازپخش تجربه، ذخیره تجربیات گذشته و سپس استفاده از زیرمجموعه‌ای تصادفی از این تجربیات برای به‌روزرسانی شبکه Q است، نه تنها استفاده از آخرین تجربه. به منظور ذخیره تجربیات Agent، از ساختار داده ای به نام deque در کتابخانه مجموعه های داخلی پایتون استفاده کردیم. این اساساً لیستی است که می توانید حداکثر اندازه را روی آن تنظیم کنید تا اگر سعی کنید به لیست اضافه کنید و قبلاً پر شده است، اولین مورد از لیست را حذف کرده و مورد جدید را به انتهای لیست اضافه کند. تجارب خود مجموعه‌ای از [مشاهده، عمل، پاداش، پرچم انجام شده، وضعیت بعدی] هستند تا انتقال‌های حاصل از محیط را حفظ کنند.

هر بار که Agent مرحله‌ای را در محیط انجام می‌دهد، انتقال را به بافر فشار می‌دهد و تنها تعداد ثابتی از مراحل را نگه می‌دارد (در مورد ما، $k۱۰$ انتقال). برای آموزش، دسته‌ای از انتقال‌ها را به‌طور تصادفی از بافر پخش نمونه‌برداری می‌کنیم، که به ما اجازه می‌دهد تا همبستگی بین مراحل بعدی در محیط را بشکنیم.

بیشتر کد بافر پخش مجدد تجربه کاملاً ساده است: اساساً از قابلیت کتابخانه deque سوء استفاده می کند. در متد ()sample، فهرستی از شاخص‌های تصادفی ایجاد می‌کنیم و سپس ورودی‌های نمونه‌گیری شده را در آرایه‌های NumPy برای محاسبه راحت‌تر تلفات بسته‌بندی می‌کنیم.

به یاد داشته باشید که در Q-Learning، یک حدس را با یک حدس به روز می کنیم، و این به طور بالقوه می تواند به همبستگی های مضر منجر شود. معادله بلمن مقدار Q(s, a) را از طریق Q(s', a') به ما می دهد. با این حال، هر دو state ، s و s تنها یک گام بین خود دارند. این باعث می شود که آنها بسیار شبیه باشند، و برای یک شبکه عصبی تشخیص بین آنها بسیار سخت است.

وقتی پارامترهای شبکه‌های عصبی خود را به‌روزرسانی می‌کنیم تا Q(s, a) را به نتیجه دلخواه نزدیک‌تر کنیم، می‌توانیم به‌طور غیرمستقیم مقدار تولید شده برای Q(s, a') و سایر حالت‌های اطراف را تغییر دهیم. این می تواند تمرینات ما را بسیار ناپایدار کند.

برای پایدارتر کردن آموزش، ترفندی به نام شبکه هدف وجود دارد که به وسیله آن یک کپی از شبکه عصبی خود را نگه می داریم و از آن برای مقدار Q(s', a') در معادله بلمن استفاده می کنیم.

در الگوریتم Deep Q-Learning دو مرحله اصلی وجود دارد. یکی جایی است که ما با انجام اقداماتی از محیط نمونه برداری می کنیم و تاپل های مشاهده شده مشاهده شده را در یک حافظه بازپخش ذخیره می کنیم. مورد دیگر جایی است که دسته کوچکی از تاپل ها را از این حافظه به صورت تصادفی انتخاب می کنیم و با استفاده از مرحله به روز رسانی گرادیان نزول (SGD) از آن دسته یاد می گیریم.

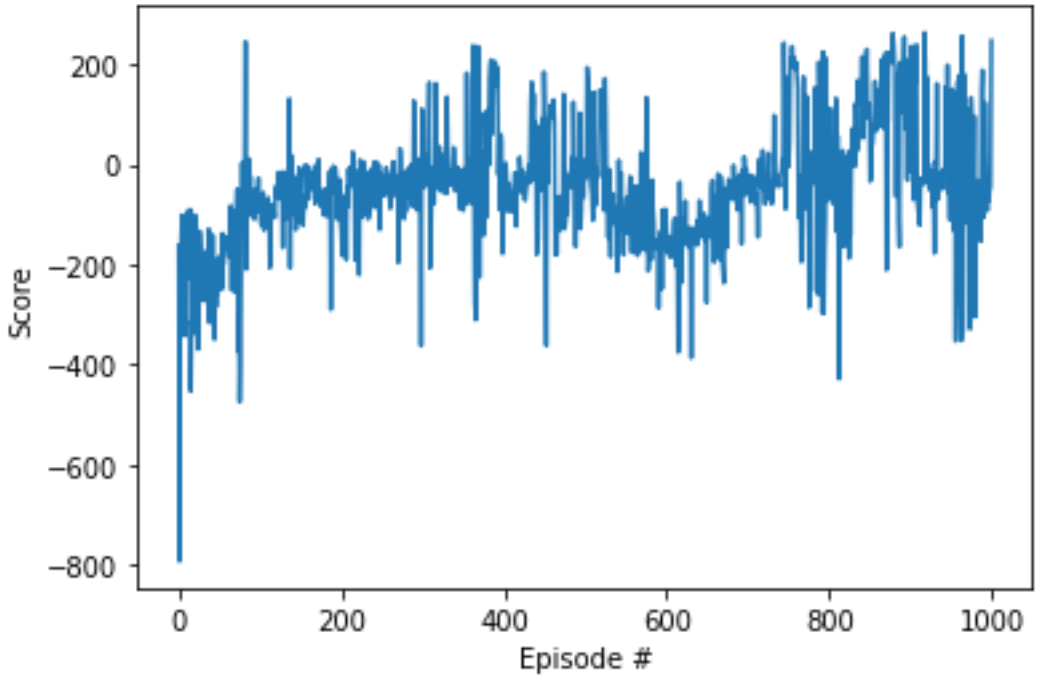
این دو مرحله مستقیماً به یکدیگر وابسته نیستند و می‌توانیم چندین مرحله نمونه‌برداری و سپس یک مرحله یادگیری یا حتی چندین مرحله یادگیری را با دسته‌های تصادفی مختلف انجام دهیم. در عمل، نمی‌توانید بلافاصله مرحله یادگیری را اجرا کنید. باید منتظر بمانید تا تجربیات کافی در D داشته باشید.

یعنی مقادیر Q پیش‌بینی‌شده این شبکه Q دوم که شبکه هدف نامیده می‌شود، برای انتشار و آموزش شبکه Q اصلی استفاده می‌شود. مهم است که مشخص شود که پارامترهای شبکه هدف آموزش داده نشده اند، اما به صورت دوره ای با پارامترهای شبکه Q اصلی هماهنگ می شوند. ایده این است که استفاده از مقادیر Q شبکه هدف برای آموزش شبکه Q اصلی، پایداری آموزش را بهبود می بخشد.

بعداً با ارائه کد حلقه آموزشی، نحوه کد دهی اولیه و استفاده از این شبکه هدف را با جزئیات بیشتری وارد خواهیم کرد.

سپس، مشاهدات را به مدل اول منتقل می کنیم و مقادیر Q خاص را برای اقدامات انجام شده با استفاده از عملیات تانسور جمع استخراج می کنیم. اولین آرگومان این فراخوانی تابع یک شاخص ابعادی است که می‌خواهیم گردآوری آن را انجام دهیم. در این حالت برابر با ۱ است، زیرا با بعد اعمال مطابقت دارد.

در نهایت خروجی برای ۱۰ اپک به Reward نزدیک ۲۰۰ نیز رسیده است که در شکل می بینید.



ویدیو در پیوست ارائه شده است.