کشف محیط با استفاده از الگوریتم های یادگیری تقویتی

علیرضا منتظری ۹۸۱۵۵۳۴
دانشگاه خواجه نصیرالدین طوسی، amontazri8@gmail.com

چکیده – در این پروژه، هدف پیاده سازی و مقایسه سه الگوریتم یادگیری تقویتی *Q-Learning* و *SARSA* و *Monte Carlo* است. برای این کار از محیط شبیه سازی *Taxi* در کتابخانه *Gym* استفاده شده است. عامل هر کدام از این الگوریتم ها در این محیط شروع به اکتشاف و یادگیری می کند. همچنین برای درک بهتر نحوه عملکرد این الگوریتم ها، پارامتر های موجود در آن ها و محیط تغییر داده شده‌اند و بر روی نتایج به دست آمده مقایسه صورت گرفته است.

کلید واژه- Grid world, Monte Carlo, SARSA, Q-Learning, Reinforcement Learning

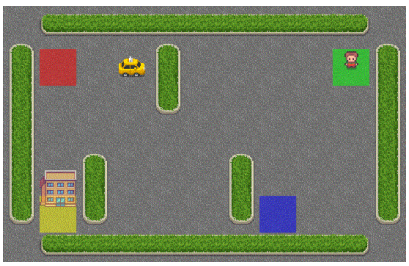
۱- مقدمه

یادگیری تقویتی (Reinforcement Learning) نوعی روش یادگیری در حوزه‌ی یادگیری ماشین (Machine Learning) است. در یادگیری ماشین دو نوع یادگیری با ناظر (Supervised Learning) و بدون ناظر (Unsupervised Learning) هم داریم. در مقایسه این سه روش، یادگیری تقویتی کمی با دو روش دیگر متفاوت است.

از یادگیری تقویتی برای حل مسائل مبتنی بر پاداش استفاده می‌شود. در یادگیری تقویتی عامل (Agent) با آزمون و خطا یاد می-گیرد و سعی می‌کند با انجام دادن برخی اقدامات در محیط (Environment) حداکثر پاداش را به دست آورد. در این پروژه از الگوریتم های یادگیری تقویتی برای شناخت محیط مشخصی که عامل از آن محیط شناخت قبلی ای ندارد استفاده شده است.

در این پروژه برای آزمایش الگوریتم ها از محیط Taxi-v3 از کتابخانه Gym استفاده شده است. در این محیط چهار مکان تعیین شده وجود دارد که با R، G، Y و B نشان داده شده‌اند. وقتی episode شروع می شود، تاکسی در یک خانه تصادفی حرکت می کند و مسافر در یک مکان تصادفی در یکی از نقاط RGYB است. تاکسی به سمت محل مسافر حرکت می کند، مسافر را سوار می کند، به مقصد مسافر (یکی دیگر از چهار مکان مشخص شده) می رود و سپس مسافر را پیاده می کند. هنگامی که مسافر پیاده می شود، episode به پایان می رسد.

در این محیط ۶ حرکت گسسته و قطعی وجود دارد. ۰ (حرکت به پایین)، ۱ (حرکت به بالا)، ۲ (حرکت به راست)، ۳ (حرکت به چپ)، ۴ (سوار کردن مسافر)، ۵ (پیاده کردن مسافر).



شکل ۱: نمای محیط Taxi-v3

در این مسئله ۵۰۰ حالت مجزا وجود دارد که عبارتند از ۲۵ موقعیت تاکسی، ۵ مکان احتمالی مسافر (از جمله زمانی که مسافر در تاکسی است) و ۴ مکان مقصد.

توجه داشته باشید که ۴۰۰ حالت وجود دارد که واقعاً می توان در طول یک episode به آنها رسید. حالت‌های از دست رفته مربوط به موقعیت‌هایی است که مسافر در همان مکان مقصد است، زیرا این حالت پایان یک قسمت را نشان می‌دهد. چهار حالت دیگر را می توان بلافاصله پس از یک قسمت موفق مشاهده کرد، زمانی که مسافر و تاکسی هر دو در مقصد هستند. این در مجموع ۴۰۴ حالت گسسته قابل دسترسی را به دست می دهد. هر فضای حالت با چند قسمت نشان داده می شود: (ردیف تاکسی، ستون تاکسی ، مکان مسافر، مقصد)

مشاهده یک عدد صحیح است که حالت مربوطه را رمزگذاری می کند. سپس حالت تاپل را می توان با تابع "decode" رمزگشایی کرد.

امتیازات برابر ۲۰ در صورت رساندن درست مسافر به مقصد و در صورت انجام عمل سوار یا پیاده کردن مسافر در جای نادرست ۱۰- و در غیر این صورت در هر حرکت ۱- است.

۲- مفاهیم

یادگیری تقویتی شامل یک عامل، مجموعه ای از حالات S و مجموعه A از اقدامات در هر حالت است. با انجام یک عمل $a \in A$، عامل از حالتی به حالت دیگر منتقل می‌شود. اجرای یک عمل در یک حالت خاص یک پاداش (امتیاز عددی) برای عامل فراهم می کند.

هدف عامل به حداکثر رساندن کل پاداش خود است. این کار را با اضافه کردن حداکثر پاداش قابل دستیابی از حالت های آینده به پاداش وضعیت فعلی انجام می دهد این پاداش بالقوه، مجموع وزنی مقادیر مورد انتظار پاداش های تمام مراحل آینده است که از وضعیت فعلی شروع می شود.

۲-۱- Q-Learning

Q-learning یک الگوریتم یادگیری تقویتی بدون مدل برای یادگیری ارزش یک عمل در یک حالت خاص است. این الگوریتم نیازی به مدلی از محیط ندارد بنابراین می‌تواند مشکلات مربوط به انتقال ها و پاداش های تصادفی را بدون نیاز به سازگاری حل کند.

برای هر فرآیند تصمیم گیری محدود مارکوف (FMDP)، Q-learning یک policy بهینه پیدا می‌کند. این به معنای به حداکثر رساندن کل پاداش های مورد انتظار در تمام مراحل با شروع از وضعیت فعلی، است. Q به پاداش های مورد انتظار برای یک اقدام انجام شده در یک وضعیت معین گفته می‌شود که الگوریتم آن را محاسبه می کند.

در هر مرحله که عامل در محیط حرکت می‌کند، در مورد عمل مرحله بعدی تصمیم می‌گیرد. وزن این مرحله به صورت γ^t محاسبه می‌شود، که در آن γ (ضریب تخفیف) عددی بین ۰ و ۱ است و تأثیر ارزش گذاری بیشتر پاداش های دریافت شده زودتر از پاداش های دریافتی دیرتر را دارد.

بنابراین، الگوریتم تابعی دارد که ارزش ترکیب state-action را محاسبه می کند. قبل از شروع یادگیری، Q به یک مقدار ثابت دلخواه (انتخاب شده توسط برنامه نویس) مقدار دهی اولیه می شود. سپس، در هر بار تکرار t عامل یک عمل را انتخاب می‌کند a_t، یک پاداش دریافت می‌کند r_t، وارد یک حالت جدید می‌شود s_{t+1} (که به حالت قبلی و عمل انتخاب شده بستگی دارد). بر اساس این مقادیر مقدار Q به روز رسانی می‌شود. هسته اصلی الگوریتم معادله بلمن به عنوان یک به روز رسانی تکرار ارزش ساده است که از میانگین وزنی مقدار قدیمی و اطلاعات جدید استفاده می کند.

$$Q(s_t, a_t) \leftarrow Q(s_t, s_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, s_t)$$

یک episode از الگوریتم زمانی به پایان می رسد که حالت s_{t+1} حالت نهایی یا پایانی باشد. با این حال، Q-Learning می تواند در کارهای غیر اپیزودیک نیز آموزش ببیند. (به خاطر خاصیت سری های بی نهایت همگرا). اگر ضریب تخفیف کمتر از ۱ باشد، مقادیر عمل محدود هستند حتی اگر مسئله دارای حلقه های بی نهایت باشد.

پارامتر α یا نرخ آموزش بیانگر این ویژگی است که در هر مرحله برای بروز رسانی مقادیر Q چه مقدار از Q جدید به قدیم اضافه شود.

همچنین در این الگوریتم پارامتر ε نیز وجود دارد که بیانگر این است که با چه احتمالی در هر step در محیط اقدام به اکتشاف یا استفاده از دانش قبلی کنیم.

۲-۲- SARSA

State–action–reward–state–action (SARSA) الگوریتمی برای یادگیری policy فرآیند تصمیم مارکوف است که در حوزه یادگیری تقویتی و یادگیری ماشین استفاده می‌شود.

این نام به سادگی نشان دهنده این واقعیت است که عملکرد اصلی برای به روز رسانی Q-value به وضعیت فعلی عامل S_1 ، عملی که عامل انتخاب می کند A_1، پاداش R که عامل برای انتخاب این عمل می گیرد، حالت S_2 که عامل پس از انجام آن عمل وارد می شود و در نهایت عمل بعدی که عامل در حالت جدید خود انتخاب می کند A_2 بستگی دارد. مخفف پنجگانه SARSA (S_t, A_t, R, S_{t+1}, A_{t+1}) است.

یک عامل SARSA با محیط تعامل می کند و policy را بر اساس اقدامات انجام شده به روز می کند، از این رو به عنوان یک الگوریتم یادگیری روی policy شناخته می شود. مقدار Q برای یک state-action با یک خطا به روز می شود که توسط نرخ یادگیری آلفا تنظیم می شود. مقادیر Q نشان دهنده پاداش احتمالی دریافت شده در مرحله زمانی بعدی برای انجام اقدام a در حالت s، به اضافه پاداش تخفیف دار آتی دریافت شده از مشاهده state-action بعدی است.

$$Q(s_t, a_t) \leftarrow Q(s_t, s_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, s_t)$$

Q-Learning تخمینی از تابع مقدار حالت-عمل بهینه Q^* را بر اساس حداکثر پاداش اقدامات موجود به روز می کند. در حالی که SARSA مقادیر Q مرتبط با policy که خودش دنبال می‌کند، یاد می‌گیرد.

۲-۳- Monte Carlo

روش Monte Carlo برای یادگیری تقویتی مستقیماً از episode های تجربه بدون هیچ گونه دانش قبلی از انتقال بین حالت های

MDP یاد می‌گیرد. یک نکته مهم این است که آن را فقط می توان برای MDP های اپیزودیک اعمال کرد. دلیل آن این است که قبل از اینکه بتوانیم پاداش را محاسبه کنیم، episode باید خاتمه یابد. در اینجا، ما بعد از هر اقدام به روز رسانی نمی‌کنیم، بلکه بعد از هر episode این کار را انجام می‌دهیم. این روش از ساده ترین ایده استفاده می کند، مقدار میانگین بازگشت تمام مسیرهای نمونه برای هر حالت است.

هدف در اینجا، یادگیری تابع ارزش حالات از episodeها تحت policy است. بازده کل برابر با پاداش تخفیف شده است. می دانیم که می توانیم هر مقدار تابع ارزش را به سادگی با جمع کردن نمونه ها و تقسیم بر تعداد کل نمونه ها تخمین بزنیم.

$$\bar{V}_{\pi}(s) = \frac{1}{N}\sum_{i=1}^{N} G_{i,s}$$

برای هر episode، دنباله‌ای از حالت ها و پاداش ها خواهیم داشت. و از این پاداش ها، می‌توانیم بازده را بر اساس تعریف محاسبه کنیم، که فقط مجموع تمام پاداش های آینده است.

در این پروژه از روش اولین مشاهده(First Visit) استفاده شده است که یعنی در هر episode تنها اولین باری که یک تابع ارزش دیده شده است بروز رسانی می‌شود.

هنگامی که تابع ارزش را برای یک policy تصادفی داریم، وظیفه مهمی که هنوز باقی می ماند یافتن policy بهینه با استفاده از Monte Carlo است. در اینجا علی رغم پیاده سازی روش epsilon-greedy از روش epsilon-soft استفاده شده است. ساده ترین ایده برای اطمینان از کاوش مداوم این است که همه اقدامات با احتمال غیر صفر امتحان شوند. 1-epsilon عملی را انتخاب می کند که تابع ارزش عمل را به حداکثر می رساند و با احتمال اپسیلون یک عمل را به صورت تصادفی انتخاب می کند.

۳- پیاده سازی

در این پروژه سه عامل Q-Learning و SARSA و Monte Carlo پیاده سازی شده‌اند. کد مربوط به این عامل در فایل Q_Agent.py و SARSA_Agent.py و MonteCarlo_Agent.py نوشته شده است. در کلاس آن ها تابع های مختلفی وجود دارند که بر اساس آن ها عامل در محیط اکتشاف می‌کند.

۳-۱- عامل های Q-Learning و SARSA

تابع __init__ که کلاس Q_Agent و SARSA_Agent را ایجاد کرده و یک محیط grid world به عنوان ورودی دریافت می‌کند.

تابع reduce_epsilon_over_epochs مقدار اپسیلون را در طول مدت یادگیری بر اساس تعداد ایپاک های سپری شده کاهش داده و آن را به عنوان خروجی بر می‌گرداند. این کاهش دادن به این خاطر است که در ابتدای یادگیری، عامل دانشی از محیط نداشته و می‌بایست در محیط به صورت تصادفی جست و جو کرده تا دانش کسب کند، ولی در ادامه روند رفته رفته دانش از محیط زیاد شده و بدون جست و جو و با تکیه به دانش خود می‌تواند مسئله را حل کند. این کاهش به شکل یک تابع نمایی با توجه به مقدار اولیه اپسیلون و تعداد کل ایپاک ها است.

تابع train_agent عامل را به تعداد ایپاک های مشخص شده به تعداد max_epoch آموزش می‌دهد. پارامتر alpha مشخص کننده نرخ آموزش، gamma فاکتور تخفیف و epsilon نرخ اکتشاف در برابر استفاده از دانش است. پارامتر policy استراتژی اولیه برای شروع آموزش را مشخص می‌کند که باید یک آرایه به اندازه تعداد حالت های محیط باشد. اگر مقدار دهی نشود در ابتدای تابع به صورت رندوم مقادیر اولیه به آن داده می‌شود. خروجی های این تابع ماتریس Q، policy بهینه، یک ارایه از تعداد step هایی که در هر ایپاک برای اتمام یک episode طول کشیده است و آرایه مقدار return جمع شونده در مراحل مختلف است.

تابع update_policy بر اساس مقادیر Q داده شده policy بهینه را به روش greedy به دست آورده و باز می‌گرداند.

برای آموزش ابتدا یک ماتریس از مقادیر Q ساخته و آن را مقدار دهی اولیه می‌کنیم. آموزش در ایپاک های مختلف انجام می‌شود. هر ایپاک از یک Episode از بازی تشکیل شده که شرط پایان آن به محیط بستگی دارد. در ابتدای هر اپیزود محیط را reset کرده تا عامل به مکان اولیه خود برود. سپس به روش $\varepsilon - greedy$ یک اکشن انتخاب کرده و به کمک تابع step این اکشن را در محیط انجام داده و استیت جدید، پاداش و اتمام episode را دریافت می‌کنیم. سپس بر اساس استیت حال و استیت آینده با توجه به فرمول های بیان شده در بخش قبل برای هر یک از الگوریتم های Q-Learning و SARSA مقدار ماتریس Q را بروز رسانی می‌کنیم. با جمع پاداش ها در هر مرحله میزان return را حساب کرده و از آن برای بدست آوردن accumulated reward استفاده می‌کنیم. در پایان هر episode نیز با توجه به ماتریس Q حاصل، policy بهینه را بدست می‌آوریم.

۳-۲- عامل Monte Carlo

تابع __init__ که کلاس MonteCarlo_Agent را ایجاد کرده و یک محیط grid world به عنوان ورودی دریافت می‌کند.

تابع reduce_epsilon_over_epochs مقدار اپسیلون را در طول

مدت یادگیری بر اساس تعداد اپیاک های سپری شده کاهش داده و آن را به عنوان خروجی بر می‌گرداند. این کاهش دادن به این خاطر است که در ابتدای یادگیری، عامل دانشی از محیط نداشته و می‌بایست در محیط به صورت تصادفی جست و جو کرده تا دانش کسب کند، ولی در ادامه روند رفته رفته دانش از محیط زیاد شده و بدون جست و جو و با تکیه به دانش خود می‌تواند مسئله را حل کند. این کاهش به شکل یک تابع نمایی با توجه به مقدار اولیه اپسیلون و تعداد کل اپیاک ها است.

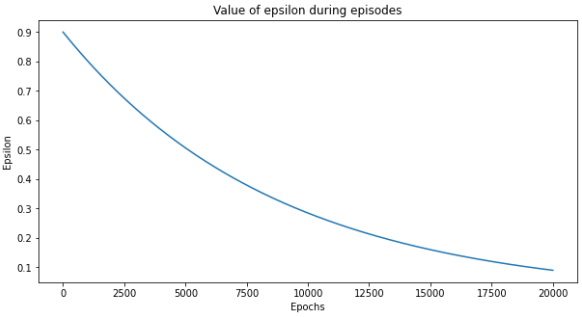
تابع train_agent عامل را به تعداد اپیاک های مشخص شده به تعداد max_epoch آموزش می‌دهد. پارامتر gamma فاکتور تخفیف و epsilon نرخ اکتشاف در برابر استفاده از دانش است. پارامتر policy استراتژی اولیه برای شروع آموزش را مشخص می‌کند که باید یک آرایه به اندازه تعداد حالت های محیط باشد. اگر مقدار دهی نشود در ابتدای تابع به صورت رندوم مقادیر اولیه به آن داده می‌شود. خروجی های این تابع ماتریس Q، policy بهینه، یک ارایه از تعداد step هایی که در هر اپیاک برای اتمام یک episode طول کشیده است و آرایه مقدار return جمع شونده در مراحل مختلف است.

برای آموزش ابتدا یک ماتریس از مقادیر Q ساخته و آن را مقدار دهی اولیه می‌کنیم. آموزش در اپیاک های مختلف انجام می‌شود. هر اپیاک از یک Episode از بازی تشکیل شده که شرط پایان آن به محیط بستگی دارد. در ابتدای هر Episode محیط را reset کرده تا عامل به مکان اولیه خود برود. سپس بر اساس policy عامل را در محیط به وسیله تابع step حرکت داده تا به حالت terminal رسیده یا از تعداد بار های مجاز بیشتر در محیط حرکت کنیم. اکشن ها در محیط به روش $\varepsilon - soft$ انتخاب می‌شوند. در طول هر episode زوج state-action و reward در یک لیست ذخیره می‌شوند. با توجه به اینکه روش first visit استفاده شده است در نتیجه از آخر episode ها به اول حرکت کرده و مقدار reward ها را به شکل discounted جمع می‌کنیم. اولین باری که هر زوج state-action مشاهده شد مقدار جمع reward ها تا آن لحظه را به return آن حالت اضافه می‌کنیم. و در انتهای تابع ارزش state-action با میانگین از return ها به دست می‌آید.

۴- آزمایش و نتایج

در این بخش آزمون هایی برای سنجش نحوه عملکرد این الگوریتم ها در دو محیط شبیه سازی شده، طراحی و نتایج آن ها مقایسه شده است. در این آزمون ها مقادیر مختلف پارامتر های موجود (α, γ) با یکدیگر مقایسه شده اند.

لازم به ذکر است که در تمامی این الگوریتم ها مقدار epsilon به صورت نمایی از ۰.۹ تا ۰.۰۹ در طول episode ها کاهش می‌یابد.



نمودار ۱: مقدار epsilon در طول episode ها

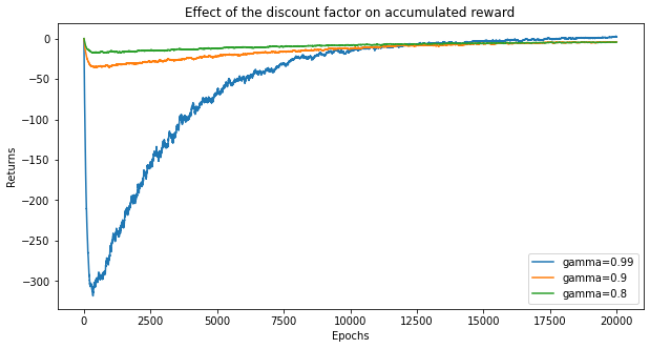
۴-۱- عامل Q-Learning

در این بخش یک Q_Agent محیط را یاد میگیرد که به ترتیب اثر ضریب فراموشی و نرخ آموزش برای آن بررسی می‌شود.

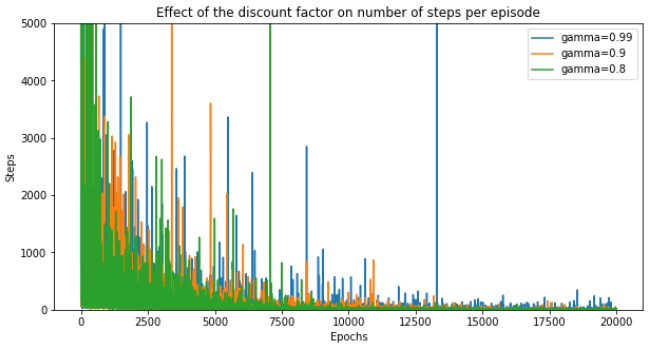
اثر ضریب فراموشی

برای این بخش نرخ آموزش ۰.۰۱، مقدار اولیه اپسیلون ۰.۹ و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می‌شود و برای ۲۰۰۰۰ episode عامل را آموزش می‌دهیم. نمودار مربوط به پاداش جمع شونده و تعداد step ها در هر episode را در ادامه مشاهده می‌کنید.

همان طور که در نمودار ۲ مشاهده می‌شود با زیاد شدن مقدار ضریب فراموشی مقدار پاداش دریافتی کلی همگرا شده نیز بیشتر شده است. همچنین در ابتدای آموزش به خاطر وجود امتیاز منفی با بزرگتر شدن نرخ آموزش، مقدار منفی نیز بیشتر شده است.



نمودار ۲: اثر ضریب فراموشی در میزان پاداش کسب شده



نمودار ۳: اثر ضریب فراموشی در تعداد step های episode

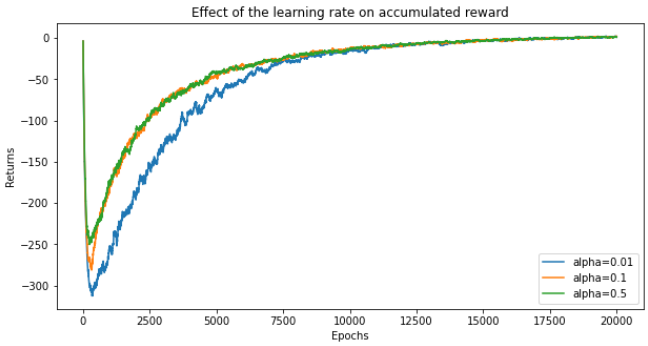
همچنین تعداد step های لازم برای اتمام episode در نمودار ۳ رسم شده است. در ابتدا، هر episode با تعداد بالایی step به پایان رسیده که این به آن علت است که عامل دانشی نداشته نمی‌توانسته خود را به نقطه های انتهایی برساند. مقدار ضریب فراموشی تاثیری در زمان همگرایی الگوریتم ندارد.

ضربه هایی که هم در نمودار پاداش و هم در نمودار step وجود دارد به خاطر اکتشاف های عامل در محیط است که باعث می‌شود مسیر هایی غیر از مسیر اصلی را امتحان کند.

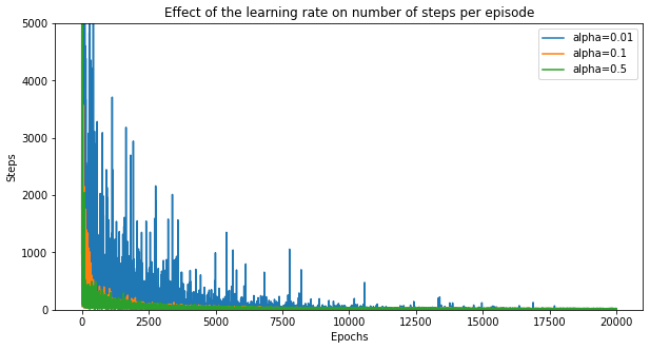
اثر نرخ آموزش

برای این بخش ضریب فراموشی ۰.۹۹، مقدار اولیه اپسیلون ۰.۹ و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می‌شود و برای ۲۰۰۰۰ episode عامل را آموزش می‌دهیم. نمودار مربوط به پاداش جمع شونده و تعداد step ها در هر episode را در ادامه مشاهده می‌کنید.

همان طور که در نمودار ۴ مشاهده می‌شود با زیاد شدن مقدار نرخ آموزش مقدار پاداش دریافتی کلی در بی نهایت تغییری نکرده و صرفا سرعت رسیدن به آن بیشتر شده است.



نمودار ۴: اثر نرخ آموزش در میزان پاداش کسب شده



نمودار ۵: اثر نرخ آموزش در تعداد step های episode

تعداد step های لازم برای اتمام episode در نمودار ۵ رسم شده است. مشخص است که عامل با نرخ آموزش بزرگتر زودتر همگرا شده است. با این حال به دلیل نزدیکی جواب مقدار ۰.۱ و ۰.۵ از مقدار ۰.۱ استفاده شده است به این خاطر که مقدار بسیار بزرگ برای نرخ آموزش باعث نوسانی شدن توابع ارزش می‌شود.

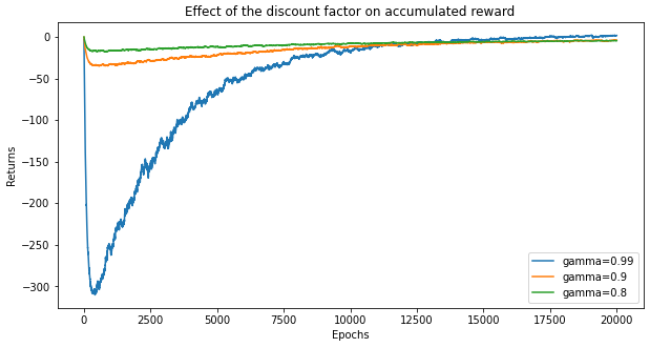
۴-۲- عامل SARSA

در این بخش یک SARSA_Agent محیط را یاد میگیرد که به ترتیب اثر ضریب فراموشی و نرخ آموزش برای آن بررسی می‌شود.

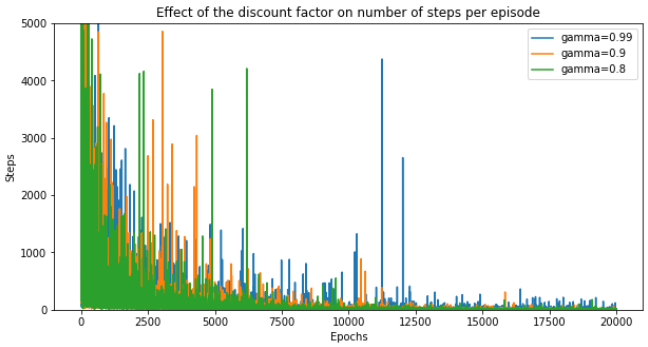
اثر ضریب فراموشی

برای این بخش نرخ آموزش ۰.۰۱، مقدار اولیه اپسیلون ۰.۹ و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می‌شود و برای ۲۰۰۰۰ episode عامل را آموزش می‌دهیم. نمودار مربوط به پاداش جمع شونده و تعداد step ها در هر episode را در ادامه مشاهده می‌کنید.

همان طور که در نمودار ۶ مشاهده می‌شود با زیاد شدن مقدار ضریب فراموشی مقدار پاداش دریافتی کلی همگرا شده نیز بیشتر شده است. همچنین در ابتدای آموزش به خاطر وجود امتیاز منفی با بزرگتر شدن نرخ آموزش، مقدار منفی نیز بیشتر شده است.



نمودار ۶: اثر ضریب فراموشی در میزان پاداش کسب شده



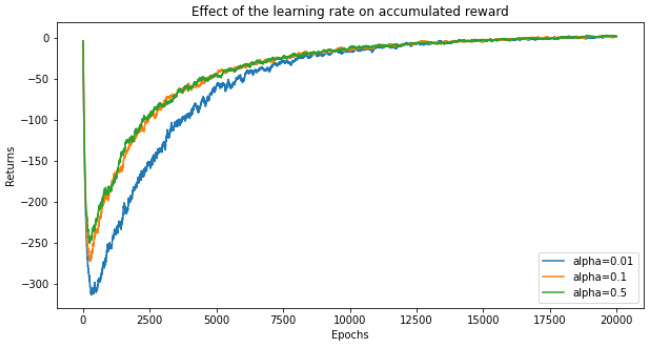
نمودار ۷: اثر ضریب فراموشی در تعداد step های episode

همچنین تعداد step های لازم برای اتمام episode در نمودار ۷ رسم شده است. در ابتدا، هر episode با تعداد بالایی step به پایان رسیده که این به آن علت است که عامل دانشی نداشته نمی‌توانسته خود را به نقطه های انتهایی برساند. مقدار ضریب فراموشی تاثیری در زمان همگرایی الگوریتم ندارد.

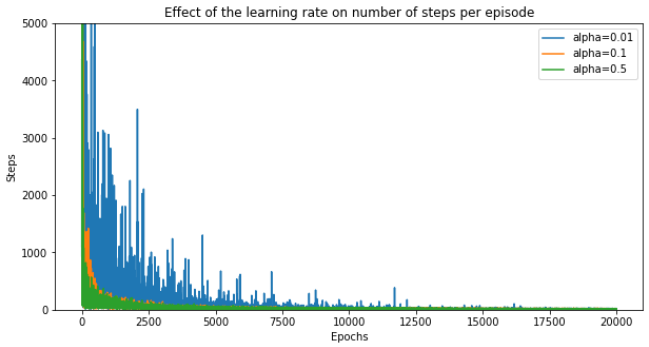
اثر نرخ آموزش

برای این بخش ضریب فراموشی ۰.۹۹، مقدار اولیه اپسیلون ۰.۹ و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می‌شود و برای ۲۰۰۰۰ episode عامل را آموزش می‌دهیم. نمودار مربوط به پاداش جمع شونده و تعداد step ها در هر episode را در ادامه مشاهده می‌کنید.

همان طور که در نمودار ۸ مشاهده می‌شود با زیاد شدن مقدار نرخ آموزش مقدار پاداش دریافتی کلی در بی نهایت تغییری نکرده و صرفا سرعت رسیدن به آن بیشتر شده است.



نمودار ۸: اثر نرخ آموزش در میزان پاداش کسب شده



نمودار ۹: اثر نرخ آموزش در تعداد step های episode

تعداد step های لازم برای اتمام episode در نمودار ۹ رسم شده است. مشخص است که عامل با نرخ آموزش بزرگتر زودتر همگرا شده است. با این حال به دلیل نزدیکی جواب مقدار ۰.۱ و ۰.۵ از مقدار ۰.۱ استفاده شده است به این خاطر که مقدار بسیار بزرگ برای نرخ آموزش باعث نوسانی شدن توابع ارزش می‌شود.

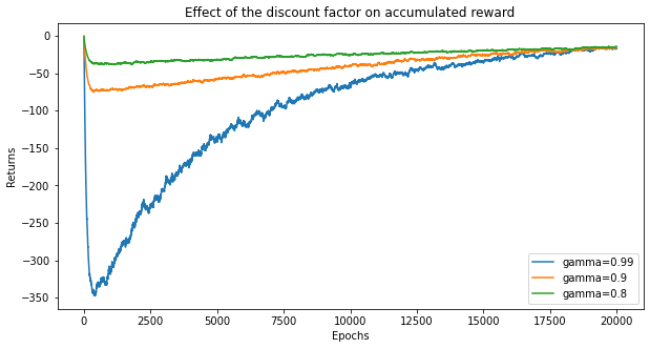
۳-۴- عامل Monte Carlo

در این بخش یک MonteCarlo_Agent محیط را یاد میگیرد که اثر ضریب فراموشی و نرخ آموزش برای آن بررسی می‌شود.

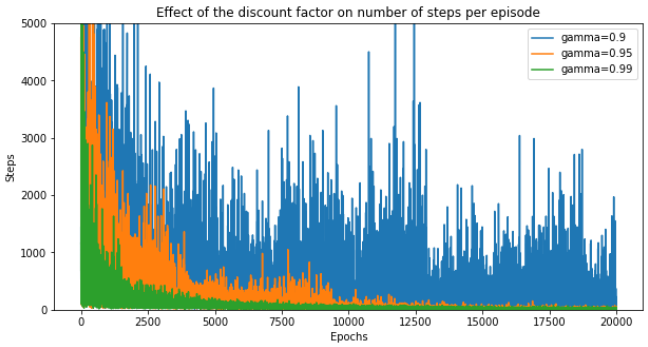
اثر ضریب فراموشی

برای این بخش مقدار اولیه اپسیلون ۰.۹ در نظر گرفته شده است. همچنین policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می‌شود و برای ۲۰۰۰۰ episode عامل را آموزش می‌دهیم. نمودار مربوط به پاداش جمع شونده و تعداد step ها در هر episode را در ادامه مشاهده می‌کنید.

همان طور که در نمودار ۱۰ مشاهده می‌شود با زیاد شدن مقدار ضریب فراموشی مقدار پاداش دریافتی کلی تغییر چندانی نکرده ولی با توجه به نمودار ۱۱ می‌توان دید که بالاتر بودن ضریب فراموشی باعث سریع تر شدن همگرایی در Monte Carlo شده است.



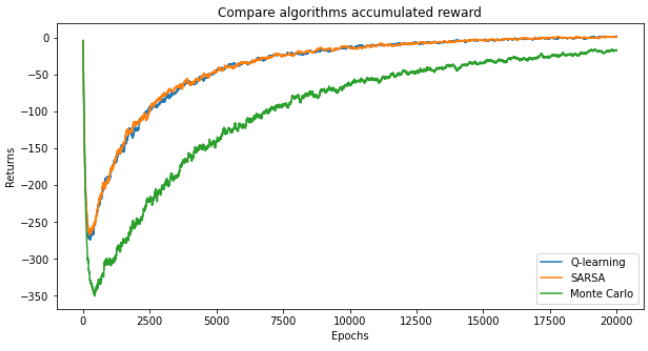
نمودار ۱۰: تاثیر ضریب فراموشی در میزان پاداش کسب شده



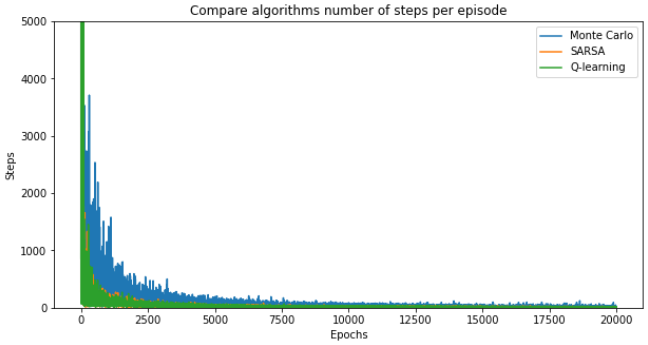
نمودار ۱۱: اثر ضریب فراموشی در تعداد step های episode

۴-۴- مقایسه عامل ها

در این قسمت بین سه الگوریتم مقایسه ای صورت می‌گیرد. برای این کار هر روش با پارامتر هایی که بهترین جواب را در قسمت های قبل داده است، پیاده سازی شده است. در Q-learning ضریب فراموشی برابر ۰.۹۹، نرخ آموزش برابر ۰.۱ و اپسیلون از مقدار ۰.۹ شروع به کاهش می‌کند. در SARSA ضریب فراموشی برابر ۰.۹۹، نرخ آموزش برابر ۰.۱ و اپسیلون از مقدار ۰.۹ شروع به کاهش می‌کند. در Monte Carlo ضریب فراموشی برابر و اپسیلون از مقدار ۰.۹ شروع به کاهش می‌کند. همچنین policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می‌شود و برای ۲۰۰۰۰ episode عامل ها را آموزش می‌دهیم. نمودار مربوط به پاداش جمع شونده و تعداد step ها در هر episode را در ادامه مشاهده می‌کنید.



نمودار ۱۲: مقایسه Q-Learning و SARSA و Monte Carloدر میزان پاداش کلی کسب شده



نمودار ۱۳: مقایسه Q-Learning و SARSA و Monte Carlo در تعداد step های episode

الگوریتم Q-learning و SARSA در این محیط بسیار شبیه یکدیگر عمل کرده اند و هم در میزان reward و هم در تعداد step های هر اپیزود شبیه یکدیگر بوده اند. ولی الگوریتم Monte Carlo هم به reward کمتری در طول زمان دست یافته و هم خیلی دیر تر نسبت به دو الگوریتم دیگر همگرا شده است. می‌توان نتیجه گرفت که به طور کلی الگوریتم های Temporal Difference در این محیط نسبت به Monte Carlo بهتر آموزش می‌بینند.

لازم به ذکر است که فرایند آموزش در ۲۰۰۰۰ اپیزود برای Q-Learning ۵۲ ثانیه، برای SARSA ۴۳ ثانیه و برای Monte Carlo ۱ دقیقه و ۲۷ ثانیه طول کشیده است که باز هم نشان دهنده بهتر بودن الگوریتم های Q-Learning و SARSA نسبت به Monte Carlo است.

برای evaluation روش ها در ۱۰۰ episode آن ها را با policy بهینه محاسبه شده توسط هر یک، آزمایش کردیم. سپس میانگین تعداد step های مورد نیاز برای انجام هر episode گزارش شده است. در صورتی که پس از ۵۰۰ step مسافر به مقصد نرسد به متغییر penalty اضافه می‌شود. در نهایت در جدول ۱ این مقادیر برای سه الگوریتم گزارش شده است.

Method	Average Step	Penalty
Q-Learning	12.82	0
SARSA	12.77	0
Monte Carlo	37.59	5

جدول ۱: مقایسه الگوریتم ها در ۱۰۰ episode

در فایل های ضمیمه شده‌ی Taxi_Q.gif و Taxi_SARSA.gif و Taxi_MC.gif حرکت هر یک از عامل ها به شکل انیمیشن برای ۵ episode وجود دارد.