کشف محیط با استفاده از الگوریتم های یادگیری تقویتی

علیرضا منتظری 9815534

دانشگاه خواجه نصیرالدین طوسی، amontazri8@gmail.com

چکيده – 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.

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

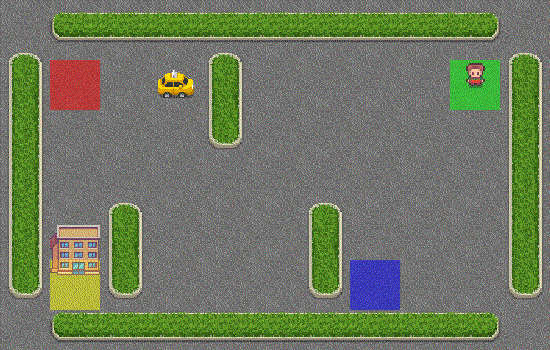
# مقدمه

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 می­گیرد 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 6 حرکت گسسته و قطعی وجود دارد. 0 ( حرکت به پایین)، 1 ( حرکت به بالا)، 2 ( حرکت به راست)، 3 ( حرکت به چپ)، 4 ( سوار کردن مسافر)، 5 (پیاده کردن مسافر).



شکل 1: نمای محیط Taxi-v3

در این مسئله 500 حالت مجزا وجود دارد که عبارتند از 25 موقعیت تاکسی، 5 مکان احتمالی مسافر (از جمله زمانی که مسافر در تاکسی است) و 4 مکان مقصد.

توجه داشته باشید که 400 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 404 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 20 در صورت رساندن درست مسافر به مقصد و در صورت انجام عمل سوار یا پیاده کردن مسافر در جای نادرست 10- و در غیر این صورت در هر حرکت 1- است.

# مفاهیم

یادگیری تقویتی شامل یک عامل، مجموعه ای از حالات S و مجموعه A از اقدامات در هر حالت است. با انجام یک عمل ، 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.

## 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 محاسبه می‌شود، که در آن (ضریب تخفیف) عددی بین 0 و 1 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 at، یک پاداش دریافت می‌کند rt، وارد یک حالت جدید می‌شود st+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.

یک episode از الگوریتم زمانی به پایان می رسد که حالت st+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 1 باشد، مقادیر عمل محدود هستند حتی اگر مسئله دارای حلقه های بی نهایت باشد.

پارامتر 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 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.

## 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 S1 ، عملی که عامل انتخاب می کند A1، پاداش R که عامل برای انتخاب این عمل می گیرد، حالت S2 که عامل پس از انجام آن عمل وارد می شود و در نهایت عمل بعدی که عامل در حالت جدید خود انتخاب می کند A2 بستگی دارد. مخفف پنجگانه (St, At, R, St+1, At+1) SARSA 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-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.

## 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 ازepisode ها 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.

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.

# 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.

## 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 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.

## 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 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.

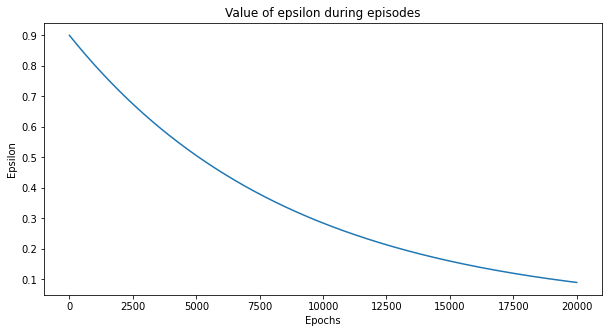
تابع train\_agent عامل را به تعداد ایپاک های مشخص شده به تعداد max\_epoch آموزش می­دهد. پارامتر gamma فاکتور تخفیف و epsilon نرخ اکتشاف در برابر استفاده از دانش است. پارامتر policy استراتژی اولیه برای شروع آموزش را مشخص می­کند که باید یک آرایه به اندازه تعداد حالت های محیط باشد. اگر مقدار دهی نشود در 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.

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 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 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.

# 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 () با یکدیگر مقایسه شده اند.

لازم به ذکر است که در تمامی این الگوریتم ها مقدار epsilon به صورت نمایی از 0.9 تا 0.09 در طول episode ها کاهش می­یابد.



نمودار 1: مقدار epsilon در طول episode ها

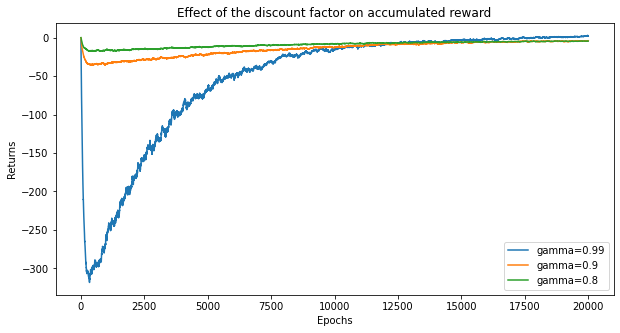
## عامل Q-Learning

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

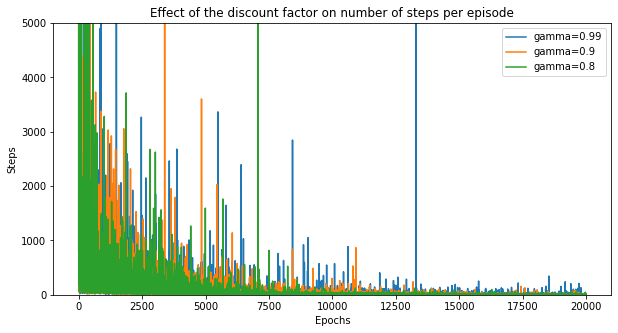
#### **اثر ضریب فراموشی**

برای این بخش نرخ آموزش 0.01، مقدار اولیه اپسیلون 0.9 و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می­شود و برای 20000 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 2 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 2: اثر ضریب فراموشی در میزان پاداش کسب شده

 نمودار 3: اثر ضریب فراموشی در تعداد step های episode

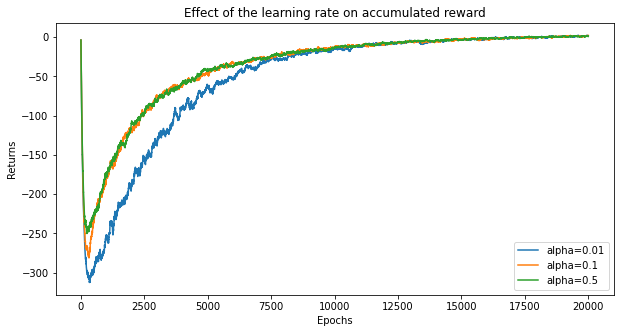
همچنین تعداد step های لازم برای اتمام episode در نمودار 3 رسم شده است. در ابتدا، هر episode با تعداد بالایی step به پایان رسیده که این به آن علت است که عامل دانشی نداشته نمی­توانسته خود را به نقطه های انتهایی برساند. مقدار ضریب فراموشی تاثیری در زمان همگرایی الگوریتم ندارد.

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.

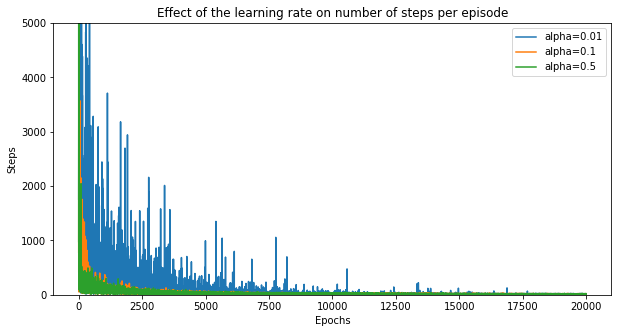
#### **اثر نرخ آموزش**

برای این بخش ضریب فراموشی 0.99، مقدار اولیه اپسیلون 0.9 و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می­شود و برای 20000 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 4 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 4: اثر نرخ آموزش در میزان پاداش کسب شده

 نمودار 5: اثر نرخ آموزش در تعداد step های episode

تعداد step های لازم برای اتمام episode در نمودار 5 رسم شده است. مشخص است که عامل با نرخ آموزش بزرگتر زودتر همگرا شده است. با این حال به دلیل نزدیکی جواب مقدار 0.1 و 0.5 از مقدار 0.1 استفاده شده است به این خاطر که مقدار بسیار بزرگ برای نرخ آموزش باعث نوسانی شدن توابع ارزش می­شود.

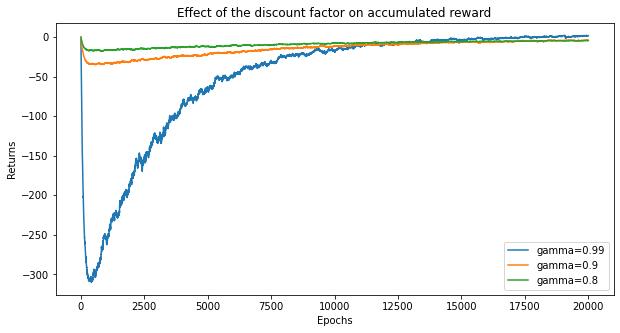
## عامل SARSA

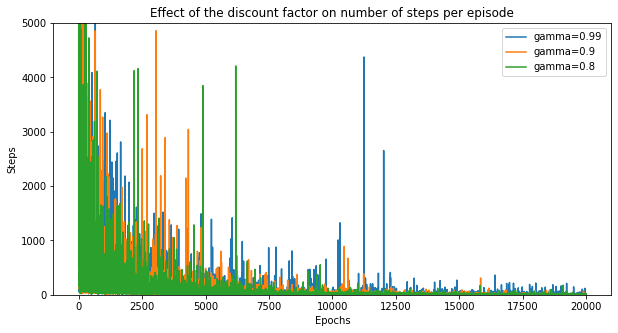
در این بخش یک SARSA\_Agent محیط را یاد میگیرد که به ترتیب اثر ضریب فراموشی و نرخ آموزش برای آن بررسی می­شود.

#### **اثر ضریب فراموشی**

برای این بخش نرخ آموزش 0.01، مقدار اولیه اپسیلون 0.9 و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می­شود و برای 20000 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 6 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 6: اثر ضریب فراموشی در میزان پاداش کسب شده

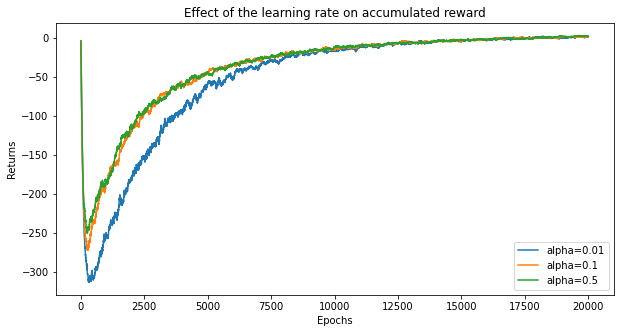
 نمودار 7: اثر ضریب فراموشی در تعداد step های episode

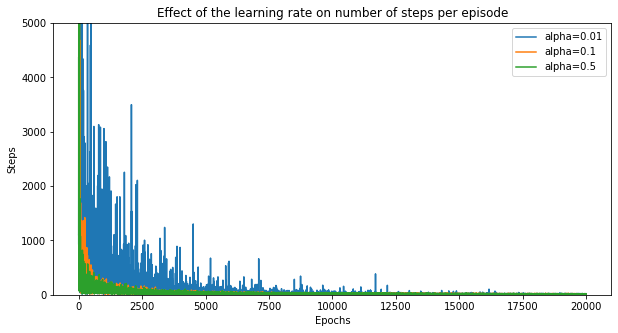
همچنین تعداد step های لازم برای اتمام episode در نمودار 7 رسم شده است. در ابتدا، هر episode با تعداد بالایی step به پایان رسیده که این به آن علت است که عامل دانشی نداشته نمی­توانسته خود را به نقطه های انتهایی برساند. مقدار ضریب فراموشی تاثیری در زمان همگرایی الگوریتم ندارد.

#### **اثر نرخ آموزش**

برای این بخش ضریب فراموشی 0.99، مقدار اولیه اپسیلون 0.9 و policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می­شود و برای 20000 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 8 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 8: اثر نرخ آموزش در میزان پاداش کسب شده

 نمودار 9: اثر نرخ آموزش در تعداد step های episode

تعداد step های لازم برای اتمام episode در نمودار 9 رسم شده است. مشخص است که عامل با نرخ آموزش بزرگتر زودتر همگرا شده است. با این حال به دلیل نزدیکی جواب مقدار 0.1 و 0.5 از مقدار 0.1 استفاده شده است به این خاطر که مقدار بسیار بزرگ برای نرخ آموزش باعث نوسانی شدن توابع ارزش می­شود.

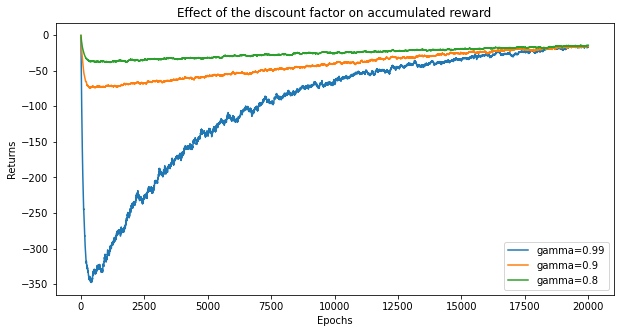
## عامل Monte Carlo

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

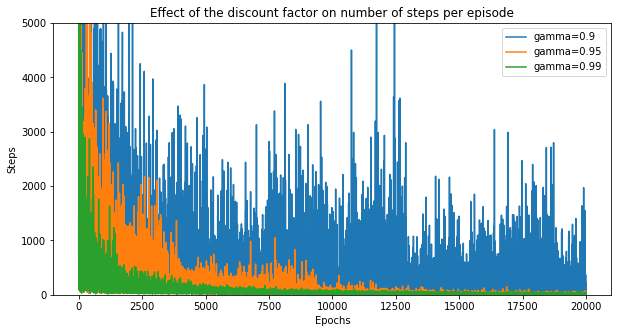
#### **اثر ضریب فراموشی**

برای این بخش مقدار اولیه اپسیلون 0.9 در نظر گرفته شده است. همچنین policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می­شود و برای 20000 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 10 مشاهده می­شود با زیاد شدن مقدار ضریب فراموشی مقدار پاداش دریافتی کلی تغییر چندانی نکرده ولی با توجه به نمودار 11 می­توان دید که بالاتر بودن ضریب فراموشی باعث سریع تر شدن همگرایی در Monte Carlo شده است.

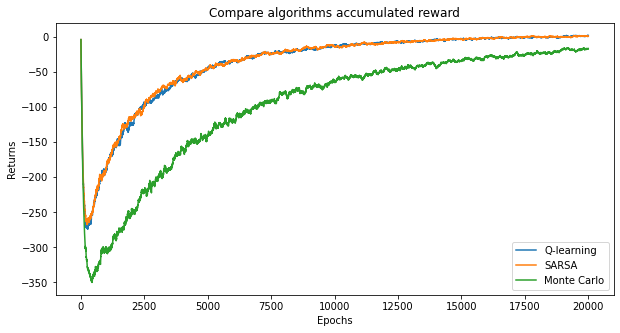


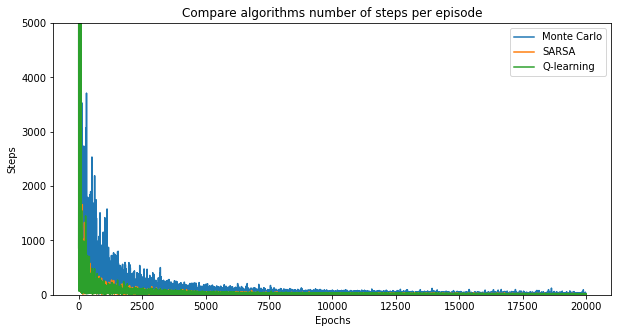
نمودار 10: تاثیر ضریب فراموشی در میزان پاداش کسب شده

 نمودار 11: اثر ضریب فراموشی در تعداد step های episode

## 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 0.99، نرخ آموزش برابر 0.1 و اپسیلون از مقدار 0.9 شروع به کاهش می­کند. در SARSA ضریب فراموشی برابر 0.99، نرخ آموزش برابر 0.1 و اپسیلون از مقدار 0.9 شروع به کاهش می­کند. در Monte Carlo ضریب فراموشی برابر و اپسیلون از مقدار 0.9 شروع به کاهش می­کند. همچنین policy اولیه در هر بار آموزش به شکل تصادفی انتخاب می­شود و برای 20000 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 12: مقایسه Q-Learning و SARSA و Monte Carloدر میزان پاداش کلی کسب شده

 نمودار 13: مقایسه Q-Learning و SARSA و Monte Carlo در تعداد step های episode

الگوریتم Q-learning وSARSA 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 20000 اپیزود برای Q-Learning 52 ثانیه، برای SARSA 43 ثانیه و برای Monte Carlo 1 دقیقه و 27 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 100 episode آن ها را با policy بهینه محاسبه شده توسط هر یک، آزمایش کردیم. سپس میانگین تعدادstep های مورد نیاز برای انجام هر episode گزارش شده است. در صورتی که پس از 500 step مسافر به مقصد نرسد به متغییر penalty اضافه می­شود. در نهایت در جدول 1 این مقادیر برای سه الگوریتم گزارش شده است.

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

جدول 1: مقایسه الگوریتم ها در 100 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 5 episode وجود دارد.