Q-learning implementation in OpenAI Gym's "Taxi-v3" environment

در این مسئله قصد داریم با پیاده سازی الگوریتم Q-Learning مدلی طراحی کنیم تا تاکسی بهترین مسیر (با بیشترین پاداش) را بین مسیر های ممکن پیدا کند و مسافر را به مقصدش برساند.

قسمت الف: حل محیط بازی بدون استفاده از روش Q-Learning و مبتنی بر پیمایش رندوم

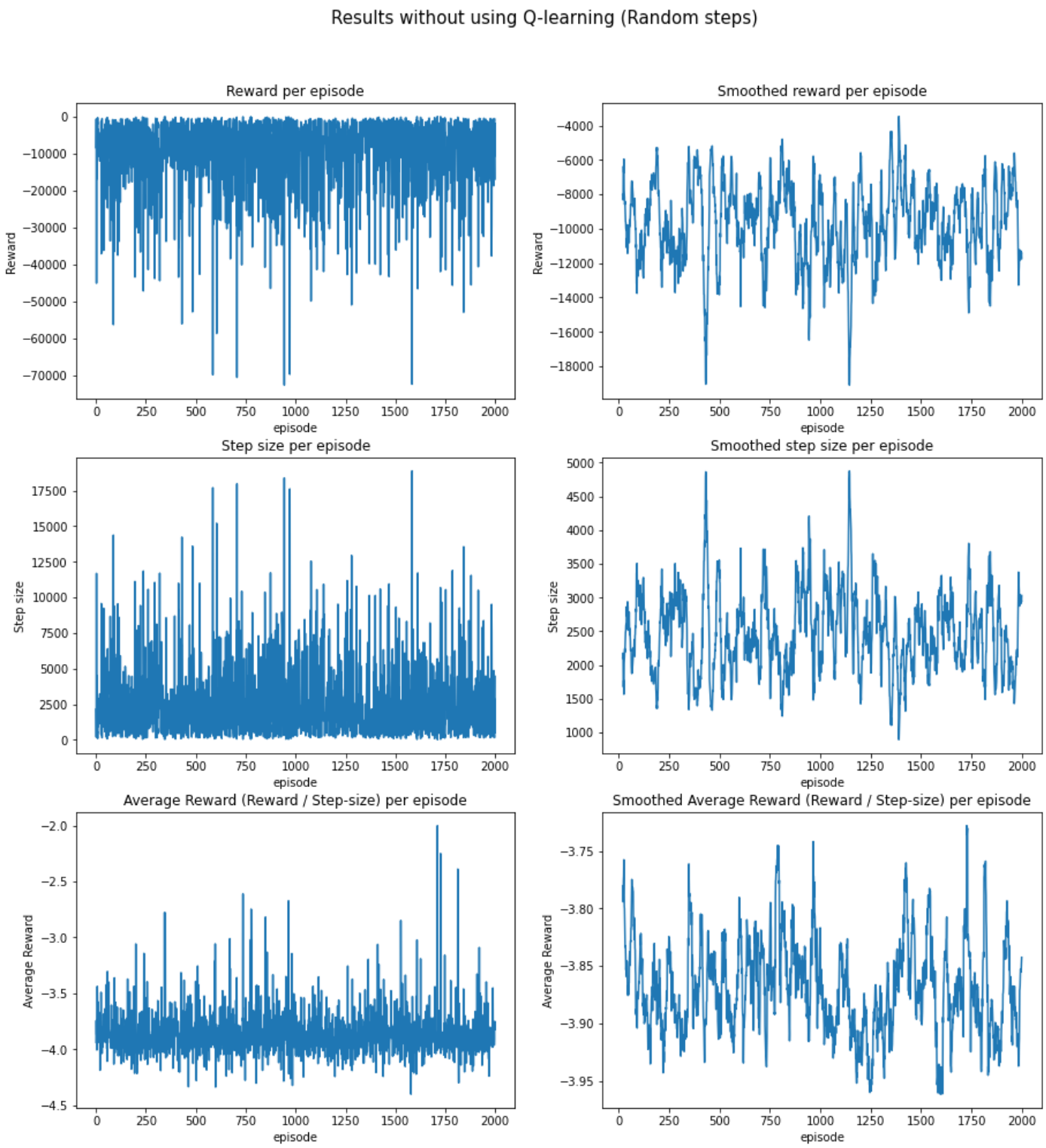
در ابتدا بدون استفاده از الگوریتم Q-Learning و با گام های رندوم مسئله را حل می‌کنیم. به این منظور در ۲۰۰۰ اپیزود، هر بار در یک حلقه بی‌نهایت، یک گام رندوم انتخاب می‌کنیم و انقدر این کار را انجام می‌دهیم تا مسافر به مقصدش برسد.

پیاده سازی: برای این بخش تابع RandomBased() نوشته شده است. در این تابع یک حلقه با ۲۰۰۰ اپیزود داریم. در هر اپیزود یک حلقه بی‌نهایت داریم که در این حلقه یک حرکت (action) رندوم انجام می‌شود و سپس در شرطی بررسی می‌شود که اگر مسافر به مقصدش رسیده است، از حلقه بی‌نهایت خارج شویم و وارد اپیزود بعدی شویم. در هر اپیزود تعداد حرکات رندوم و همچنین پاداش یا جریمه کل نیز بدست آورده می‌شود. در نهایت در تابعی به نام Result_showing() نمودار های پاداش و جریمه بر حسب اپیزود، تعداد حرکات (طول گام) برحسب اپیزود و میانگین پاداش و جریمه (نسبت پاداش و جریمه به طول گام) برحسب اپیزود رسم می‌شود. همچنین برای واضح تر شدن تغییرات و حذف اثر های منفی در نمودار از دستور rolling() از کتابخانه pandas استافده می‌کنیم. در این تابع window_size = 50 در نظر گرفته می‌شود. یعنی بجای مقدار متغیر در هر اپیزود به ۵۰ اپیزود مجاورش نگاه می‌کنیم و بین این ۵۰ اپیزود میانگین می‌گیریم.

در نهایت خروجی این بخش را مطابق با تصویر ۳-۱ رسم می‌کنیم. همانطور که مشاهده می‌شود، مطابق انتظار با استفاده از حرکات رندوم قرار نیست عملکرد خوبی را شاهد باشیم و پاداش و جرمه نیز رندوم خواهد بود و مقدار زیادی نیز خواهد داشت. همچنین مدلی نیز طراحی نشده است که عملکرد آن رو به بهبودی باشد و همواره پاداش و جریمه یک مقدار غیر وابسته به مقادیر قبلی است.

قسمت ب: حل محیط بازی با استفاده از روش Q-Learning و مبتنی بر پیمایش هوشمندانه

حال می‌خواهیم از الگوریتم Q-Learning استفاده کنیم و جدولی طراحی کنیم تا در هر state تاکسی تشخیص دهد که بهترین حرکت چیست.



تصویر ۳-۱: نمودار های مورد نظر برای حل محیط بازی بدون استفاده از روش Q-Learning و مبتنی بر پیمایش رندوم

توضیح الگوریتم و پیاده سازی: در این الگوریتم یک Q-table تعریف می‌کنیم. ابعاد این جدول وابسته به تعداد state ها (۵۰۰ تا) و تعداد حرکات (۶ حرکت) است. (دقت شود ۵۰۰ state نداریم ولی مطمئن هستیم تعداد state ها از این عدد کوچکتر است.)

بنابریان Q-table را یک ماتریس 500×6 در نظر می‌گیریم و مقادیر اولیه آن صفر است. هر درایه از این ماتریس با $Q(s,a)$ نمایش داده می‌شود که s همان state کنونی و a حرکت بعدی است. در به روز رسانی $Q(s,a)$ به صورت زیر عمل می‌کنیم.

$$Q(s,a)_{new} = Q(s,a)_{old} + \alpha[R(s,a) + \gamma \max Q(s',a') - Q(s,a)_{old}]$$

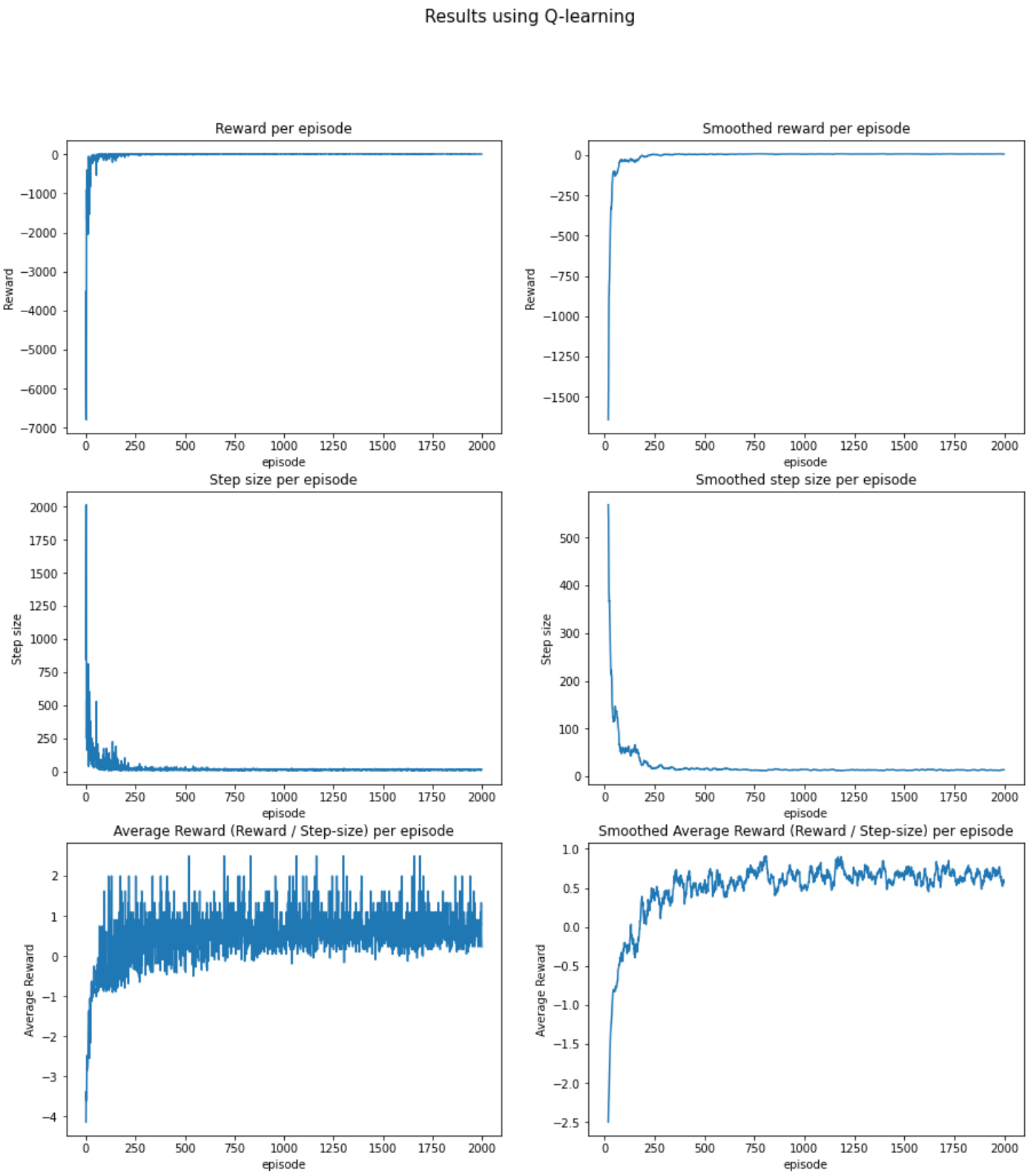
که s' همان state بعدی و $\max Q(s',a')$ بیانگر ارزش بهترین انتخاب در state بعدی است که با یک discount factor به آن ارزش می‌دهیم. در پیاده سازی این مقدار را 0.9 در نظر می‌گیریم. همچنین $R(s,a)$ پاداش یا جریمه به ازای حرکت a را نشان می‌دهد. همچنین α نرخ یادگیری (learning rate) است و در هر اپیزود آن را با یک فاکتور کاهش می‌دهیم. (با آزمون و خطا متوجه می‌شویم که یک حالت مناسب این است که ابتدا آن را ۱ در نظر بگیریم و در هر اپیزود آن را 0.999 برابر کنیم.

برای اینکه حرکت (action) a در state کنونی (s) را مشخص کنیم از الگوریتم $\epsilon - greedy$ استفاده می‌کنیم. برای این الگوریتم تابع epsilon_greedy() نوشته شده است. در این الگوریتم لزوماً بهترین حرکت در state کنونی (s) انتخاب نمی‌شود تا باقی حرکات نیز شانس انتخاب شدن داشته باشند. به این خاطر از این الگوریتم استفاده می‌کنیم تا مدل تنها به state بعدی و پاداشی که دریافت می‌کند نگاه نکند و ادامه راه نیز در نظر گرفته شود. به این منظور بهترین حرکت در state کنونی (s) با احتمال $1 - \epsilon + \frac{\epsilon}{6}$ و باقی حرکات با احتمال $\frac{\epsilon}{6}$ انتخاب می‌شوند. همچنین باید توجه کنیم که در اپیزود های ابتدایی احتمال انتخاب هر حرکت باید یکسان باشد و به مرور و در اپیزود های بعدی احتمال انتخاب بهترین حرکت افزایش یابد. برای پیاده سازی ابتدا ϵ را ۱ در نظر می‌گیریم و سپس در اپیزود های بعدی آن‌را با فاکتور 0.9، کاهش می‌دهیم.

برای پیاده سازی کل الگوریتم تابع Q_learning() را نوشته و خروجی آن‌را در تصویر ۳-۲ نشان می‌دهیم.

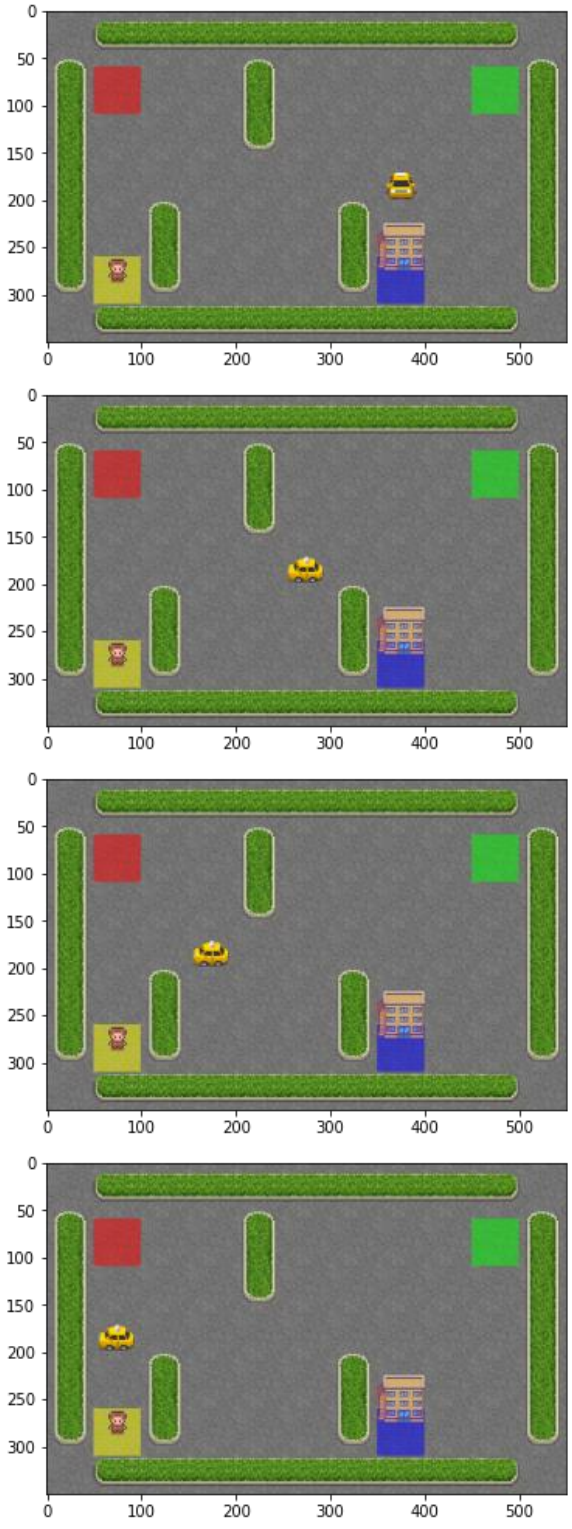
همانطور که در تصاویر ۳-۲ مشاهده می‌شود الگوریتم بعد از ۱۰۰۰ اپیزود تقریباً همگرا می‌شود و پاداش هر اپیزود مثبت می‌شود. نکته قابل توجه افزایش سرعت پایان هر اپیزود بعد از یادگیری مدل است. در گام های رندوم یادگیری و پیشرفتی وجود ندارد و به همین خاطر بسیار کند است.

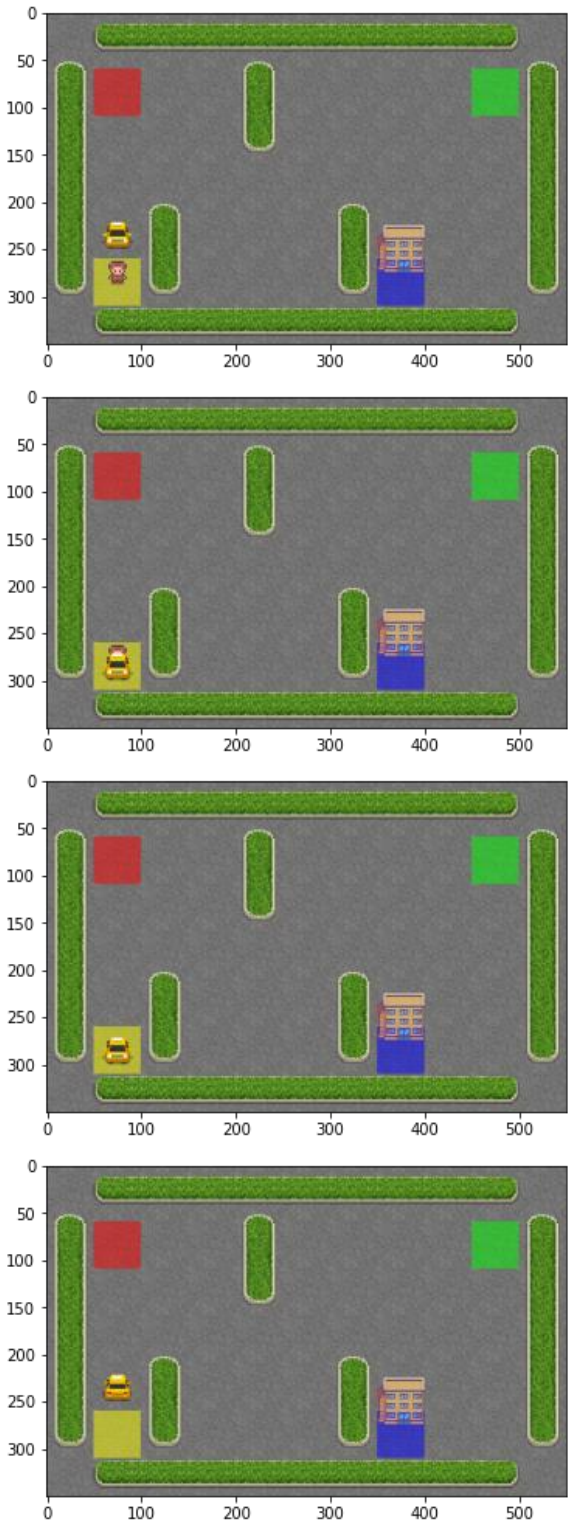
در نهایت برای تست کردن مدل نهایی تابعی به نام Test_Q_Learning() نوشته شده است که با استفاده از Q-table برای یک اپیزود الگوریتم اجرا می‌شود. تصاویر ۳-۳ نحوه حرکت تاکسی (agent) در محیط و رسیدن مسافر به مقصد را نشان می‌دهد. همانطور که مشاهده می‌شود، طول گام ۱۴ بوده و گام آخر نیز به مقصد نهایی منجر شده است. یعنی جریمه ۱۳- به ازای گام های مختلف و پاداش ۲۰+ برای رسیدن مسافر به مقصد دریافت می‌شود که مجموع پاداش ۷+ می‌شود که بیانگر بهینه بودن مدل است.

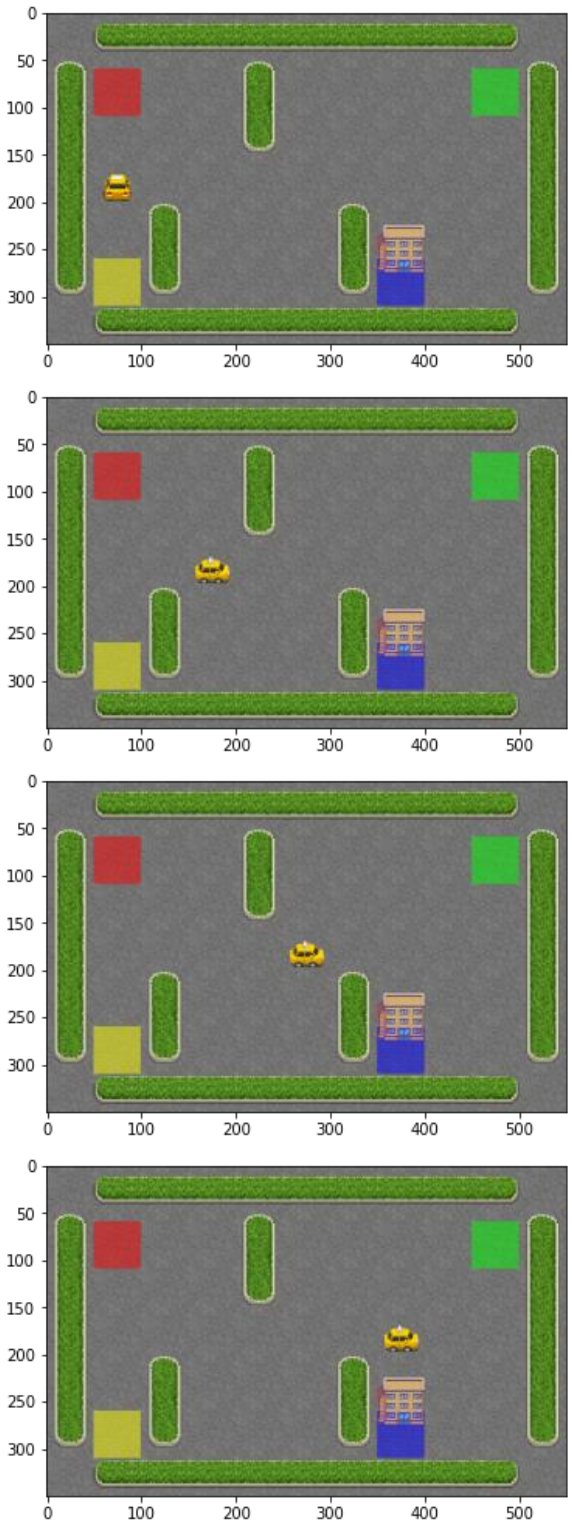


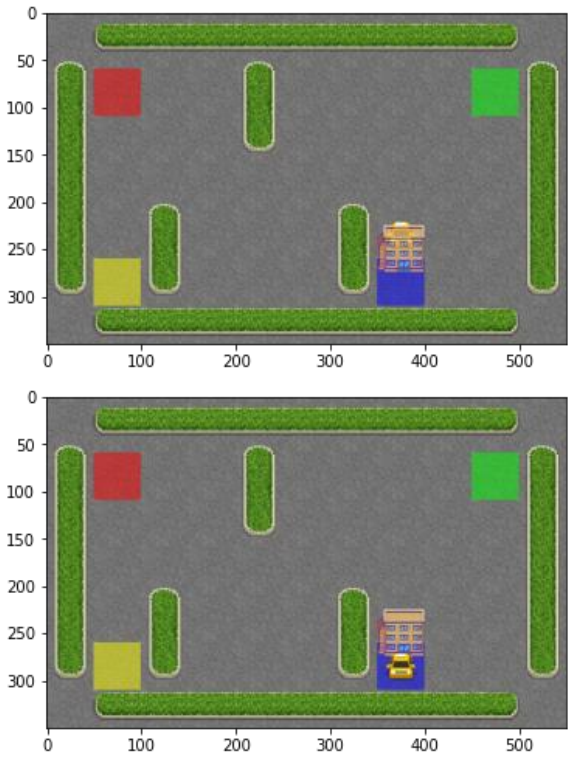
تصویر ۳-۲: نمودار های مورد نظر برای حل محیط بازی با استفاده از روش Q-Learning و مبتنی بر پیمایش هوشمند

Step-size = 14, Reward = 7



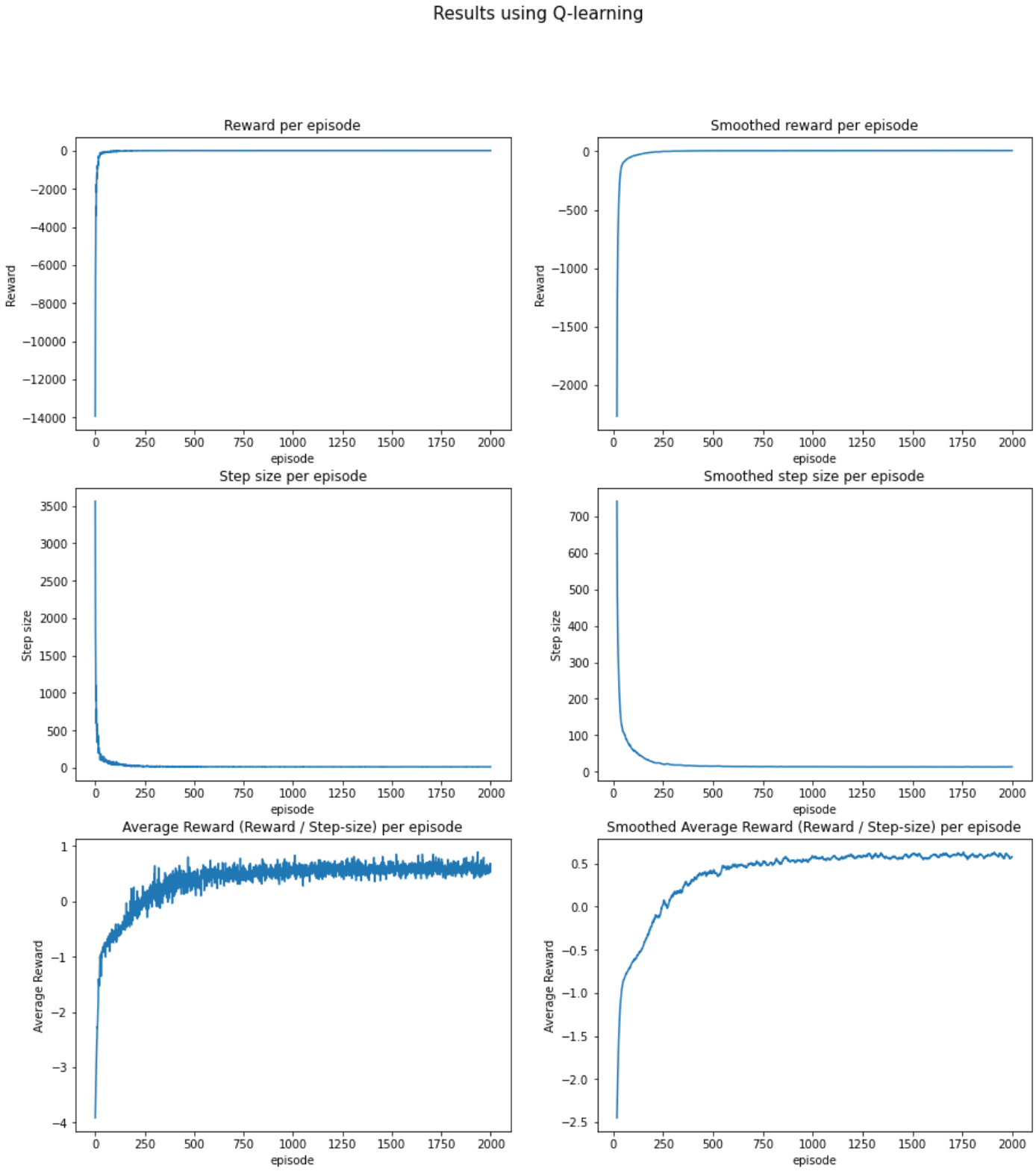

0
50
100
150
200
250
300
0
100
200
300
400
500


0
50
100
150
200
250
300
0
100
200
300
400
500



تصویر ۳-۳: نحوه حرکت تاکسی (agent) در محیط و رسیدن مسافر به مقصد با استفاده از روش Q-Learning و مبتنی بر پیمایش هوشمند

بهبود عملکرد: برای اینکه عملکرد مدل بهبود پیدا کند، می‌توانیم یک حلقه خارجی دیگر استفاده کنیم که الگوریتم به تعداد آن مجدداً تکرار شود و سپس بین اعداد Q-table خروجی میانگین می‌گیریم. به این منظور تابع Repeated_Q_Learning()نوشته شده است. تصویر ۳-۴ نمودار های مورد نظر برای مدل خروجی پس از بهبود Q-learning را نشان می‌دهد.

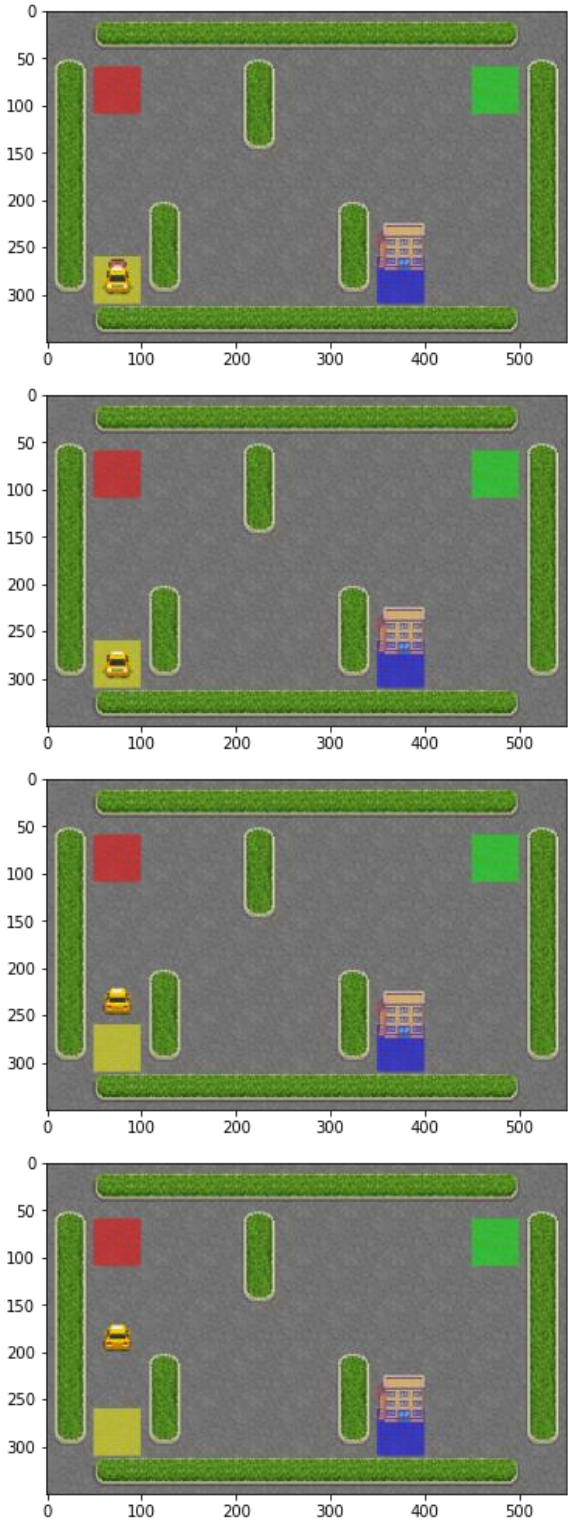


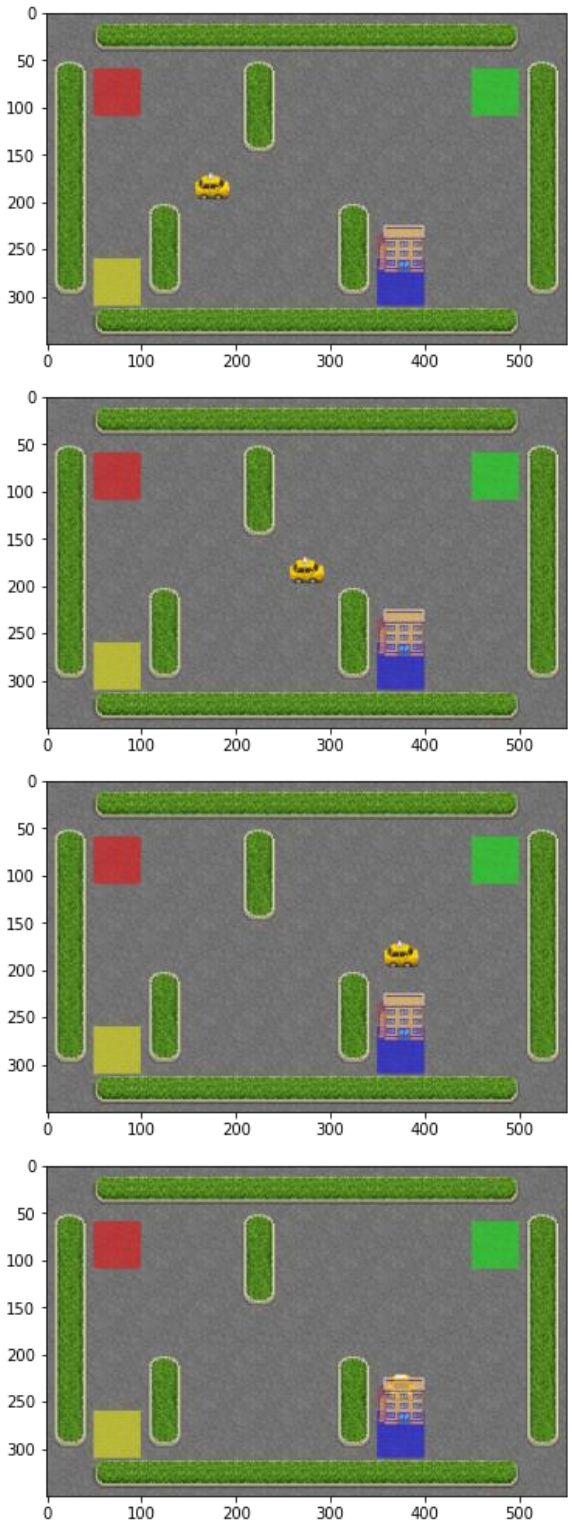
تصویر ۳-۴: نمودار های مورد نظر برای حل محیط بازی با استفاده از روش Q-Learning (بهبود یافته) و مبتنی بر پیمایش هوشمند

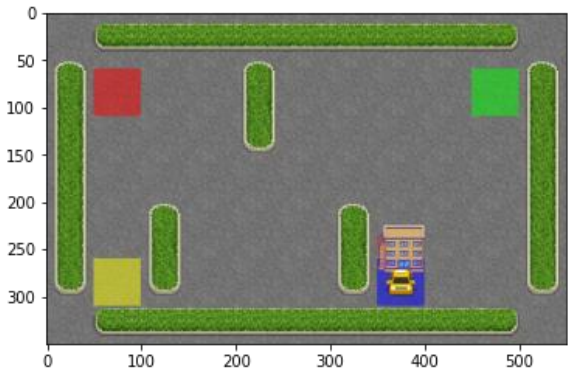
همچنین تصاویر ۵-۳ نحوه حرکت تاکسی (agent) در محیط و رسیدن مسافر به مقصد برای مدل بهبود یافته را نشان می‌دهد.

همانطور که مشاهده می‌شود، طول گام ۹ بوده و گام آخر نیز به مقصد نهایی منجر شده است. یعنی جریمه ۸- به ازای گام های مختلف و پاداش ۲۰+ برای رسیدن مسافر به مقصد دریافت می‌شود که مجموع پاداش ۱۲+ می‌شود که بیانگر بهینه بودن مدل است.

Step-size = 9, Reward = 12




0
50
100
150
200
250
300
0
100
200
300
400
500



تصویر ۳-۵: نحوه حرکت تاکسی (agent) در محیط و رسیدن مسافر به مقصد با استفاده از روش Q-Learning بهبود یافته و مبتنی بر پیمایش هوشمند

با تغییر پاداش یا جریمه چه تفاوتی در سرعت همگرایی دارند؟

با توجه به اینکه پاداش در هر مرحله فقط یک‌بار (در صورت رسیدن مسافر به مقصد) داده می‌شود، با تغییر آن صرفاً پاداش هر اپیزود به اندازه پاداش نهایی اضافه می‌شود و تغییری در سرعت همگرایی ایجاد نمی‌شود. در خصوص جریمه نیز همین موضوع برقرار است. در حقیقت زمانی تغییر پاداش یا جریمه در سرعت همگرایی می‌شود که برای مثال هر گام جریمه ای نداشته باشد و ما آن‌را به یک عدد منفی تغییر دهیم.
