به نام خدا

پروژه درس هوش مصنوعی – فاز دوم

پاییز ۱۴۰۱

امیرحسین عرب پور – علی ناظری – سارا شاه طوسی

983623021 - 983613056 – 983613038

جناب دکتر حسین کارشناس

فهرست مطالب

خواص محیط بازی

مشاهده پذیر کامل، تک عاملی، مرحله ای، گسسته، ایستا است.

همچنین محیط غیر قطعی است چون هر کنش دارای احتمالی است. همچنین جزء جدیدی

به نام سیاهچال وجود دارد که ورود به آن قطعی نیست و با ورود به آن عامل به سیاهچاله

های دیگر منتقل می شود.

Markov decision process

Value Iteration

الگوریتم مارکوف پیاده سازی شده را در سه مرحله تشریح می شود.

مرحله اول

در ابتدا زمین بازی را تحت عنوان chart دریافت می شود.

سپس با توجه به آخرین الماس خورده شده (در ابتدا با توجه به امتیازدهی اولیه برای خوردن الماس ها) ماتریس جدید ساخته می شود که Matrix نام گذاری شده است.

درواقع Matrix یک جدول تغییریافته از زمین بازی به نحوی است که عناصری که نباید به سمت خانه های آنها برویم به صورت دیوار و خانه های الماس با توجه آخرین رنگ خورده شده امتیاز گرفته اند.

پس خروجی مرحله اول یک Matrix باتوجه به شرایط آن لحظه Agent است که آماده است بر روی آن الگوریتم Markov اجرا شود.

مرحله دوم

خروجی مرحله قبل ماتریسی برای اجرای مارکوف است .

پس در این مرحله بر روی آن الگوریتم مارکوف اجرا می شود و خروجی این بخش ماتریسی است که هر خانه ی آن باتوجه به الگوریتم مارکوف امتیازی گرفته است.

چگونگی اجرای مارکوف

با استفاده از دو حلقه ی تو در تو یک بار برای هرخانه از جدول امتیاز آن را بر اساس فرمول مارکوف محاسبه می شود (هزینه هر حرکت نیز از آن کم می شود، در فرمول قرار گرفته است.)

$$V_{i+1}^{*}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s') \left[R(s,a,s') + V_{i}^{*}(s') \right]$$

این عملیات را (پیمایش همه خانه های جدول) به اندازه طول سطرهای جدول تکرار می شود تا امتیاز ها به صورت ثابت در بیایند.

دراین فرمول مارکوف rate = 0.9 در نظر گرفته شده است.

خروجی این مرحله یک Matrix است که هر خانه از آن امتیازی گرفته است.

و این امتیاز ها بر اساس آخرین رنگ خورده شده بوده است.

حالا کافیست از محلی که در آن قرار گرفته ایم با توجه به امتیازها حرکت کنیم.

مرحله سوم

دراین مرحله خروجی مرحله دوم را وارد کرده و نقطه ای که عامل در آن قرار گرفته را نیز وارد میکنیم.

کافیست از نقطه ای که عامل قرار دارد به خانه ی کناری با بیشترین مقدار حرکت کند. این کنش برای جابجایی به آن خانه همان جایی است که کافیست به سرور برگردانده شود.

اگر امتیاز خانه ای که در آن قرار گرفته ایم از همه ی خانه های اطراف آن بیشتر باشد حرکت به صورت Noap خواهد بود.

مثال

چارت

A	E	E	E	E
E	E	E	1	E
E	3	E	E	E

نتیجه مرحله اول

0	0	0	0	0
0	0	0	100	0
0	50	0	0	0

نتیجه مرحله دوم

70.16	84.58	88.42	95.18	88.09
82.32	94.10	98.00	100	84.83
74.04	50.00	93.41	96.11	88.09

بعد از خورده شدن الماس اول

E	E	E	E	E
E	E	E	A	E
E	3	E	E	E

بعد از مرحله اول با توجه به تغییر رنگ الماس خورده شده قبلی

0	0	0	0	0
0	0	0	0	0
0	150	0	0	0

بعد از مرحله دوم

115.72	127.92	138.02	132.18	106.98
140.79	145.88	144.06	131.97	116.05
134.72	150	142.82	137.99	106.98

دو نکته ی مهم

۱. هر راند بازی باید چک شود کجا قرار گرفته ایم چون به دلیل غیرقطعی بودن حرکت ها ممکن است در مرحله ی قبل از آن حرکت انتخاب شده اتفاق نیافتاده باشد.

۲. تا زمانی که عامل به الماس نرسیده است لازم نیست مرحله اول و دوم را تکرار شود و فقط با Matrix قبلی و مکان فعلی حرکت خود را تعیین می شود. اگر به الماس رسید هم رنگ الماس خورده شده تغییر می کند و هم مراحل از اول تکرار می شود تا Matrix تازه تشکیل شود.

حل مشکل کلیدها و درها

در روشی که برای پیاده سازی استفاده شده است، تا زمانی که بتوان بدون نیاز به کلید و در الماس خورد به این کار پرداخته می شود اما اگر جوابی که از سه مرحله قبل دریافت شد Noap بود بررسی می شود آیا با خوردن کلید و باز کردن در می توان این جواب را تغییر داد یا خیر.

ابتدا چک می شود به ازای هرکدام از کلیدهایی که عامل ندارد اگر آن کلید را داشته باشد و Matrix را در مرحله‌ی اول بر اساس داشتن آن ساخته شود (یعنی درهای مربوط به آن کلید، دیوار در نظر نمی شود) و مارکوفی که روی آن Matrix اعمال شد آیا جوابی جز Noap دریافت می کند (این یعنی می توان الماس را به دست آورد یا نه) اگر الماس جدیدی قابل دسترسی بود حالا چک می شود که می توان آن کلید را که فرض شد در اختیار دارد به دست آورد یا خیر.

برای این بررسی کافیست از روی chart اولیه در مرحله اول یک Matrix ساخته شود که فقط کلیدی که به دنبال آن می گردد به عنوان هدف دارای امتیاز باشد و بقیه به صورت دیوار یا 0 درنظر گرفته شوند.

سپس روی آنها مارکوف زده می شود و اگر جوابی جز Noap داشت یعنی به این کلید نیز می توان دست یافت و این Matrix را ادامه پیدا می کندتا به کلید برسد و سپس دوباره سه مرحله اول را برای یافتن الماس اجرا می شود.(این چک کردن کلیدها به شکل بازگشتی اعمال می شود پس همه ی حالت ها را پوشش می دهد.)

حل مشکل سیاهچاله

برای استفاده از سیاهچاله و همچنین استفاده از آن به عنوان آخرین امید (چون حالت های آن بسیار Random تر از قبلی ها است) کافیست آن را رنگ پنجم الماس ها با امتیازی بسیار کم تر از بقیه بدانیم. به صورتی که اگر تا شعاع قابل قبول و منطقی از موقعیت فعلی الماس وجود داشت و یا با کلید می شود به آن دست یافت به سمت آن برود و اما اگر چیزی نبود یا خیلی دور بود شانس خود را با سیاهچاله امتحان کند. پس کافیست آن را الماس پنجم در نظر بگیریم و امتیاز آن را کمتر از بقیه بدانیم.

همچنین برای اینکه اگر در موقعیتی به کلیدی نزدیک بود آن را بردارد و لازم نباشد بعدا برای برداشتن آن مسیر بیشتری را طی کند در مرحله اول (ساخت Matrix) به کلیدها نیز امتیاز پایینی اختصاص می دهد که اگر در فاصله ی بسیار نزدیک به آن بود آن را بردارد.

بخش دوم Model Free Reinforcement Learning

در بخش model free با جست و جو و کاوش در محیط ما نحوه ی رفتار در آن محیط را با تجربه ی ان ها یاد می گیریم. در این الگوریتم با حرکت کردن در یک محیط بدون شناختن حالت ها (state) و یا قوانین امتیازگیری ما مقادیری را در هر کنش یاد می گیریم تا در نهایت بتوانیم با استفاده از همگرا شدن این مقادیر در نواحی نقشه و امتیاز گیری به نتیجه برسیم.

Q-Learning

در این قسمت از Q-learning استفاده شده است. دلیل انتخاب این الگوریتم در مقایسه با SARSA به این دلیل می باشد که در مراحل ابتدایی پروژه مشاهده شد الگوریتم SARSA همگرایی کمتری دارد و برای همگرایی باید مدت زیاد تری جست و جو انجام دهد. همچنین به دلیل وجود ارزش های منفی و کنش های نامعتبر در بازی و انتخاب کنش بعدی و امتیاز دهی بر اساس آن باعث واگرایی بیشتر این الگوریتم می شد پس Q-learning انتخاب شد. برتری Q-learning به همگرایی با سرعت بیشتر و استفاده از بیشترین مقدار به جای یک مقدار تصادفی است.

الگوریتم یادگیری Q-learning به این صورت است که در هر نوبت ما یک کنش را انتخاب کرده و با توجه به آن موقعیت بعدی را در نقشه پیدا می شود و با توجه آن به هزینه حرکت و امتیاز کسب شده در موقعیت بدست می آید سپس با فرمول زیر جدول ارزش خود را بروز می کند.

فرمول زیر برای بروز رسانی ارزش های جدول در Q-learning استفاده می شود.

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

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (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

در این فرمول مقدار α مقدار یادگیری را معلوم می کند مقدار γ تاثیر مقادیر جدید را تایین می کنند که بین صفرو یک قرار دارند.

مقدار γ برابر ۰,۹ در نظر گرفته شد خواستیم مقادیر جدید تاثیر زیادی داشته باشند.

مقدار α در هر مپ از بازی با توجه به محل قرار گیری دورترین الماس از نظر فاصله منهتن (manhattan) و فضای خالی مشخص می شود.

R مقدار پاداش هر کنش و Q(S, A) مقدار ارزش حالت فعلی و Q(S’, a) مقادیر ارزش حالت بعدی می باشد.

(که برای این مورد پیاده سازی الماس خورده شده هم در نظر گرفته شده است.)

یادگیری

در بخش یادگیری عامل ده هزار(epoch) بار بازی با تعداد حداقل ۱۰۰ راند بازی می کند اگر تعداد راند ها بیشترباشد این مورد جایگزین می شود.

برای یادگیری، یک جدول ارزش که یک دیکشنری از تمام مختصات خانه های بازی است تشکیل می شود که هر خانه شامل کنش های مربوط به حالت های آخرین الماس خورده شده می باشد.

یعنی در هنگامی که الماس قرمز خورده شده باشد و یا الماس سبز خورده شده باشد برای کنش های یکسان، دو مقدار متفاوت ثبت شده است.

همچنین برای اینکه عامل کاوش مناسبی انجام دهد تعداد مراتب ورود به هر خانه از بازی ثبت می شود و بر اساس آن کنش ها انتخاب می شوند.

برای انتخاب کنش ابتدا از q-greedy استفاده شده است و سپس به آن احتمالات هم اضافه شدند. در q-greedy یک متغییر به نام epsilon وجود دارد (که در کد به نام trshld می باشد) که با توجه به مقدار آن و یک عدد رندم تولید شده بین انتخاب رندوم کنش ها یا انتخاب کنش با بیشترین وزن یک حالت را انتخاب می شود.

در صورتی که عدد رندوم تولید شده از epsilon کوچک تر باشد به صورت تصادفی کنش را انتخاب می شود که قبل از انتخاب آن به هر کنش با توجه به موقعیت نهایی که عامل می تواند به آن وارد شود وزن داده می شود در اینجا برای وزن دهی جدول تعداد ورود به خانه ها کمک گرفته شده است و در حالت دوم اگر عدد رندوم بزرگ تر باشد کنش با بیشترین مقدار در جدول ارزش، انتخاب می شود که این انتخاب با توجه به آخرین الماس

خورده شده و موقعیت عامل بستگی دارد. در صورتی که در حالتی قرار داشته باشیم که هیچ کنشی برای آن ثبت نشده باشد به صورت رندوم عمل می شود.

مقدار تصادفی به ما کمک می کند کاوش در مپ بازی را تقویت کنیم. به این منظور مقدار epsilon بر اساس بزرگی فاصله manhattan از نزدیک ترین الماس مشخص می شود.

برای ارزش دهی به درها و کلیدها در هنگام یادگیری به این صورت عمل شده است که به تعداد دیوار هایی که در اطراف یک در قرار دارد به کلید های متناظر ۱۰۰ امتیاز اختصاص می دهد و در هنگام یادگیری در صورت خوردن کلید، مقدار ارزش کلید بر روی در ها اضافه می شود.

برای سیاه چاله ها در قسمت یادگیری به این صورت عمل شده است که در ابتدای کار تمام آن ها شناسایی شده و عامل در مختصات هر یک از آن ها قرار می گیرد و عملیات یادگیری به مقدار مساوی انجام می شود (ecpoch ها تقسیم می شوند)

مقدار پاداش هر کنش با توجه به نوع کنش و موارد موجود در خانه ی بعدی تایین می شود. این موارد می تواند شامل سیم خاردار در ویا کلید و الماس ها باشند. در اینجا برای افزایش هم گرایی به الماس ها ده برابر ازرش گرفته شده پاداش داده می شود.

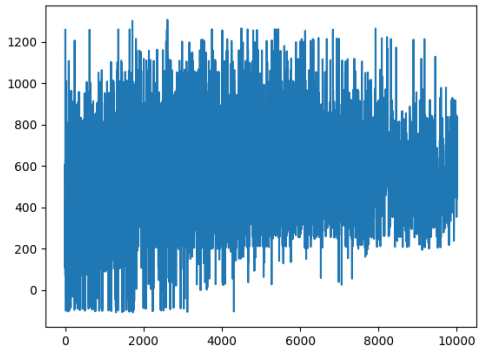
در قسمت یادگیری با توجه به اینکه باید تمامی عملیات ها روی نقشه به صورت خودکار انجام شود از تابع کمک گرفته شده که یکی از آن ها کنش را انجام می دهد و دیگری تاثیر موارد خورده شده را اعمال می کند (مانند حذف الماس ها پس از خورده شدن و ثبت آخرین الماس خورده شده)

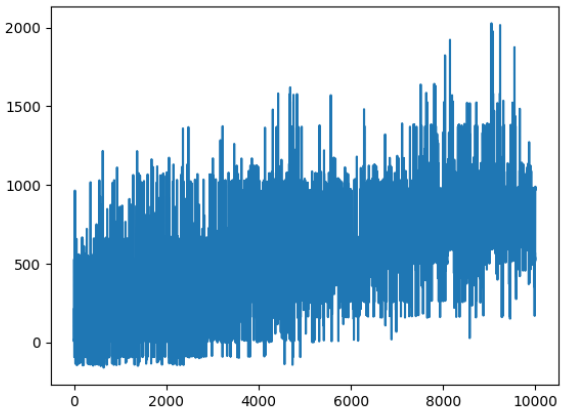
تعداد راند ها در هنگام بازی کردن وقتی به پایان می رسد که تمام الماس ها خورده شده باشند و سپس دور بعدی یادگیری شروع می شود. یادگیری وقتی پایان می یابد که تعداد دور های آن (epoch) به پایان برسد یا امتیاز گرفته شده ۵۰۰ بار افزایش یابد یا ۵۰۰ بار به حداکثر مقدار امتیاز رسیده باشد.

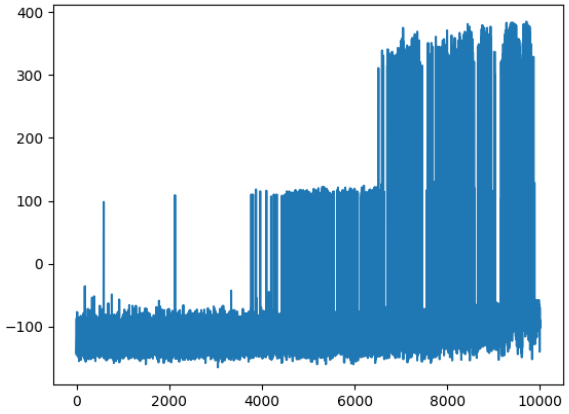
نمودارهای یادگیری

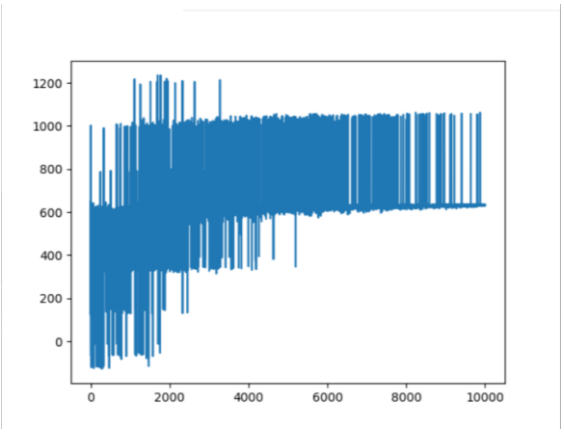
این نمودار ها در مپ هایی که الگوریتم بهتر کار می کرد بدست آمده اند.

همانطور که مشاهده می شود امتیازگیری در مپ ها در حال افزایش است.









انتخاب کنش با توجه جدول

پس از فاز یادگیری فاز بازی کردن است و با هر بار صدا زده شدن تابع do_turn کنش متناسب با جدول یادگرفته شده انتخاب می شود. در هر دور ابتدا موقعیت عامل پیدا می شود چون محیط غیر قطعی است و ممکن است عامل قطعا کنش را به درستی انجام ندهد.

همچنین در هنگام حرکت از هر ارزش ۵۰ واحد کم می شود تا اگر عامل بین چند خانه به دام بیوفتد با کم شدن ارزش آن ها در چند حرکت بتواند به حرکت خود ادامه بدهد.

در صورتی که در این فاز عامل روی سیاهچاله قرار بگیرد شروع به انجام کنش های تصادفی می کند.

همچنین یادگیری در این بخش همچنان انجام می شود.

اگر هیچ الماسی در مپ بازی نباشد عامل فقط noap می دهد.

چالش ها و مشکلات

همگرایی ارزش الماس ها با تنظیم epsilon و α تا حدود زیادی حل می شود ولی در و کلید همچنان همگرایی کم می باشد. عامل در مواجهه با سیاهچاله ها خوب عمل می کند ولی در پیدا کردن دوباره آن ها خوب عمل نمی کند. همگرایی تابع یادگیری در هنگام پیاده سازی در سرور کم شد ولی در خارج از سرور بسیار بیشتر بود و این مشکل حل نشد.

همچنین مدت زمان یادگیری بالا می باشد و با افزایش جزییات و طرح ها برای یادگیری بهتر و بهینه کردن الگوریتم مدت زمان یادگیری افزایش پیدا می کند که البته با افزایش دقت می توان دور ها را کم کرد و به آن سرعت بخشید.

تقسیم کارها

در ابتدای کار چند جلسه با هماهنگی جهت بررسی بخش های فاز گذاشته شد سپس سوالات و مشکلات بوجود آمد و پس از رفع آن ها با مطالعه و پرسش از استادیار ها حل شد و سپس تقسیم کار ها انجام شد.

در کل کار ها به دو بخش درون فاز ها تقسیم شد.

بخش اول مارکف : آقای امیرحسین عرب پور و خانم شاه طوسی

بخش دوم یادگیری تقویتی بدون مدل : آقای علی ناظری

داک نویسی : آقای علی ناظری و خانم شاه طوسی

تعهد نامه

ما دانشجویان علی ناظری، امیرحسین عرب پور، سارا شاه طوسی تعهد می نماییم که پروژه تحویل داده شده نتیجه کار ما بوده و در هیچ یک از بخس های انجام شده از کار دیگران کپی برداری نشده است. در صورتی که مشخص شود که این پروژه کار ما نبوده است، طبق ضوابط آموزشی با ما برخورد شده و حق اعتراض نخواهیم داشت.