پردیس علوم

دانشکده ریاضی، آمار، و علوم کامپیوتر

یادگیری تقویتی و بازی مین‌روب

نگارنده: علیرضا مقدس برهان

استاد راهنما: دکتر هدیه ساجدی

پایان‌نامه برای دریافت درجه کارشناسی

در رشته علوم کامپیوتر

بهمن ماه ۱۴۰۰

چکیده

یادگیری تقویتی در سال‌های اخیر توجه بسیاری به خود جلب کرده و تاثیر قابل ملاحظه‌ای روی صنعت هوش مصنوعی گذاشته است. در این روش، مدل در محیطی قرار می‌گیرد که برای تصمیمات مختلف پاداش یا جریمه‌ای در نظر گرفته می‌شود. این مدل با امتحان کردن تصمیمات مختلف، با گذشت زمان مساله را یاد می‌گیرد و پیشرفت می‌کند. به همین دلیل است که یادگیری تقویتی در بازی‌هایی به سادگی دوز تا بازی‌های پیچیده‌تری مانند شطرنج و گو نیز استفاده فراوانی داشته است. این بازی‌ها دارای محیط جذابی برای پیاده‌سازی روش‌های مختلف یادگیری تقویتی هستند. در این پروژه با یادگیری تقویتی بازی مین‌روب به عامل یاد داده می‌شود. در طول پروژه از پایتون و Tensorflow استفاده می‌شود و بازی نیز در کد پروژه پیاده‌سازی شده است.

فهرست مطالب

فصل ۱

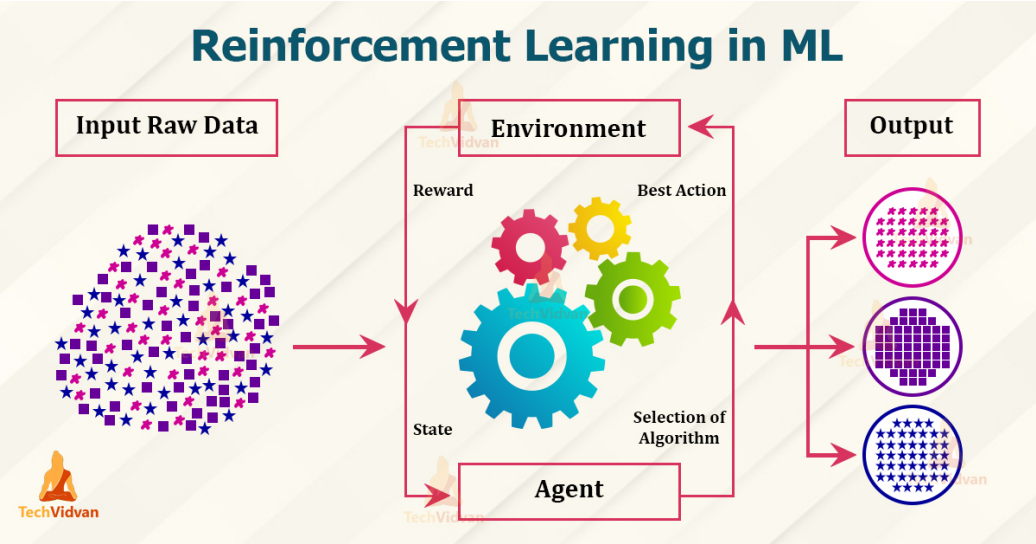
مقدمات

۱.۱ یادگیری تقویتی

یادگیری تقویتی در واقع آموزش به ماشین‌هاست تا تصمیماتی را بگیرند. عامل یاد می‌گیرد که در محیطی پیچیده و ناشناخته، به هدفی دست یابد. در یادگیری تقویتی، هوش مصنوعی با یک وضعیت بازی مانندی مواجه است و از آزمون و خطا استفاده می‌کند تا به راه‌حلی دست یابد. برای اینکه ماشین را به جایی برسانیم که می‌خواهیم، هوش مصنوعی برای اعمالش پاداش یا جریمه دریافت می‌کند. هدفش نیز این است که پاداش را حداکثر کند. با اینکه برنامه‌نویس پاداش‌ها و جریمه‌ها را مشخص می‌کند، او هیچ راهنمایی دیگری برای حل بازی ارائه نمی‌دهد. مدل باید به تنهایی بیاموزد که چگونه پاداش را حداکثر کند. او از حرکت‌های کاملا تصادفی شروع می‌کند و کم کم شرایط را می‌آموزد. برخلاف انسان‌ها، هوش مصنوعی می‌تواند از هزاران بازی هم‌زمان تجربه کسب کند و تنها پیش‌نیازش زیرساخت کامپیوتری مناسب است.

همینطور که در تصویر ۱.۱ دیده می‌شود، در یادگیری تقویتی، با دادن وضعیت فعلی به عنوان ورودی به

شکل ۱.۱: یادگیری تقویتی



عامل، هوش مصنوعی بهترین عمل را انتخاب کرده و اعمال می‌کند. سپس پاداشی دریافت می‌کند که طبق آن خود را آپدیت می‌کند. [۲]

۲.۱ یادگیری عمیق

یادگیری عمیق یک تکنیک یادگیری ماشین است که در آن کامپیوترها از مثال‌ها استفاده می‌کنند تا بیاموزند. معمولا از شبکه‌های عصبی استفاده می‌شود که لایه‌های بسیاری می‌تواند داشته باشد. [۳]

۳.۱ شبکه عصبی

شبکه‌های عصبی، شبکه‌هایی هستند که با الگو گرفتن از طرز کار مغز انسان ساخته شده‌اند. در این شبکه‌ها لایه‌هایی از نودها وجود دارند که به نودهای لایه‌های بعدی با یک وزنی وصل شده‌اند. اگر مقدار نود از یک حدی بیشتر باشد، به نودهای لایه بعدی داده را ارسال می‌کند. با گذشت زمان، شبکه‌های عصبی می‌توانند

مسائل بسیار پیچیده‌ای را حل کنند. [۳]

۴.۱ Q-Learning

الگوریتم Q-Learning یک راه ساده برای یادگیری تقویتی است. در این روش، ما جدولی از وضعیت به عمل‌هایمان داریم. برای هر وضعیت، ارزش (Q Valueها) مناسب هر عمل را مشخص و طبق آن‌ها، بهترین عمل را انتخاب می‌کنیم. پس از انتخاب هر عمل و دیدن نتیجه‌ش، Q Value را با معادله بلمن به روز رسانی می‌کنیم.

$$\mathrm{Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \Big(r_t + \gamma \cdot max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big)} \qquad (۱.۱)$$

همینطور که مشاهده می‌کنید، در معادله بلمن، دو ضریب یادگیری α و discount factor γ به چشم می‌خورد. ضریب یادگیری در واقع به ما می‌گوید که چند درصد از پاداش یا جریمه فعلی باید در یادگیری در نظر گرفته شود. Discount factor [۴] نیز به ما نشان می‌دهد که بیشترین پاداش آینده چقدر تاثیر دارد. [۵] در نهایت کاری که معادله می‌کند، جمع کردن مقدار قدیمی Q Value، اضافه کردن پاداش آینده و بهترین ارزش آینده با ضریب‌های مرتبتشان است. معادله بلمن در مرکزیت روش Q-Learning است و در آن به عنوان Q Function استفاده می‌شود. [۴]

۵.۱ Deep Q-Learning

با این حال، در صورت زیاد بودن وضعیت‌های یک بازی یا مسئله، روش کلاسیک Q-Learning پاسخگو نخواهد بود. در چنین حالتی ما از Deep Q-Learning استفاده می‌کنیم. در این روش به جای Q Function،

از شبکه‌های عصبی استفاده می‌شود. این باعث می‌شود که هوش مصنوعی بتواند حالت‌هایی را که تا به حال ندیده است نیز پاسخ دهد. [۶]

۶.۱ بازی مین‌روب

این بازی یکی از بازی‌های کلاسیک کامپیوتری است که سال‌ها در سیستم‌عامل ویندوز به صورت دیفالت وجود داشته و در سیستم‌عامل‌های دیگر به آسانی قابل نصب بوده است. بازی به این شکل است که جدولی دو بعدی، به طور پیش‌فرض ۹ در ۹، وجود دارد که در بعضی از خانه‌هایش (به طور پیش‌فرض ۱۰ تا) به شکل تصادفی، مین‌هایی قرار داده شده‌اند. مقدار خانه‌هایی که مین ندارند، عددی است به تعداد خانه‌های اطراف آن خانه که در آن مین وجود دارد. در اشکال زیر دو نمونه از این بازی را می‌توان دید. [۷]

شکل ۲.۱: نمونه بازی کوچک مین‌روب

شکل ۳.۱: نمونه بازی بزرگ مین‌روب

فصل ۲

پیاده‌سازی، مسیر، و نتایج

۱.۲ بیان مسئله

مسئله ما حل کردن بازی مین‌روب است. حقیقت این است که حتی بهترین الگوریتم هم نمی‌تواند این بازی را صد در صد مواقع حل کند، زیرا حالت‌هایی وجود دارد که در آن‌ها حرکت بعدی تنها با شانس می‌تواند منجر به باخت نشود. نمونه چنین حالتی در تصویر ۱.۲ نمایش داده شده است.

به دلیل شرایط این چنینی، هیچگاه نمی‌توان انتظار داشت که بازی در صد در صد مواقع موفق باشد. همچنین با توجه به اینکه تعداد حرکات تا رسیدن به برد نسبتا زیاد است، تعداد دفعات یادگیری باید زیاد باشد تا درصد موفقیت قابل توجهی کسب کرد.

شکل ۱.۲: نمونه بازی که حرکت منطقی و قطعی بعدی ندارد

۲.۲ ابزارهای مورد استفاده

در این پروژه از پایتون، Tensorflow، و به طور خاص APIهای Keras استفاده شده است. در کنار این‌ها، برخی توابع کتابخانه Numpy نیز به کار برده شده‌اند. برای ذخیره و بارگیری وزن‌های شبکه عصبی و همچنین مجموعه بازی‌های ساخته شده، از Pickle استفاده شده است. در کنار این‌ها، نصب کتابخانه tqdm نیز برای نمایش صحیح برخی بخش‌ها ضروری است. نسخه پایتون مورد استفاده حین نوشتن پروژه Python 3.6.9 می‌باشد. ورژن‌های بالاتر یا پایین‌تر مینور امتحان نشده‌اند ولی به احتمال بسیار زیاد مشکلی ندارند.

۳.۲ پیاده‌سازی اولیه کلاس‌ها

در ابتدای انجام پروژه، چند کار زیرساختی، بعلاوه پیاده‌سازی یک مدل ساده انجام شد. مدل‌های به کار گرفته شده با نام‌های MinesweeperGame، Model و Main حضور داشتند.

بازی مین‌روب در کلاس MinesweeperGame پیاده‌سازی شد. در ابتدا تنها کاری که این کلاس می‌کرد، ساختن یک بازی تصادفی، نگه داشتن خانه‌های نمایش داده شده، و انجام حرکت بعدی بود. با صدا کردن اولیه این کلاس (initializing) یک بازی تصادفی نیز ساخته می‌شود، در نتیجه می‌توان با ساختن

متوالی چند نمونه از این کلاس، بازی‌های متفاوتی ساخت.

در کلاس Model، قسمت یادگیری ماشین پیاده‌سازی شد. دو تابع اصلی این کلاس، train و test می‌باشد. در ابتدا روی تعدادی بازی مدلمان را train و وزن شبکه عصبی نهایی را در یک فایل ذخیره می‌کنیم. سپس می‌توانیم با صدا کردن تابع test، روی تعداد دلخواهی بازی از پیش ساخته شده مدلمان را تست کرده و میزان موفقیت را ببینیم.

کلاس Main اولین کلاسی است که صدا زده می‌شود. این کلاس در واقع کار ابتدایی و انتهایی لود کردن وزن‌های شبکه عصبی پیش از test و همچنین ذخیره کردنش پس از train را به عهده دارد.

همچنین، دستوری به نام create_games در برنامه تعبیه شده است که به تعداد دلخواه بازی می‌سازد و آن‌ها را در فایلی ذخیره می‌کند. این بازی‌ها به عنوان ورودی به دستورهای train و test داده می‌شوند.

شکل ۲.۲: نمونه اجرای دستور create_games

```
0 alireza 16:37:38 Project master → python3 main.py create_games 100000 g.P
2022-02-05 16:37:40.202639: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcuda
rt.so.11.0: cannot open shared object file: No such file or directory
2022-02-05 16:37:40.202673: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
 Ignore above cudart dlerror if you do not have a GPU set up on your machine.
 85%|                                   | 84845/100000 [00:31<00:06, 2464.68it/s]
```

در ابتدا ما تنها سه نوع پاداش در نظر گرفته بودیم: پاداش پایان موفقیت‌آمیز، پاداش منفی رویت مین، و پاداش حرکت رو به جلوی موفقیت‌آمیز. ولی این نوع پاداش‌ها نتیجه‌بخش نبودند زیرا حرکات تصادفی بسیار زیادی اتفاق می‌افتاد که پاداش مثبت می‌گرفتند.

برای حل این مشکل، پاداش منفی برای حرکت‌های تصادفی در نظر گرفتیم. حرکت تصادفی را نیز حرکتی تعریف می‌کنیم که روی خانه‌ای انجام شود که اطرافش هیچ خانه رویت شده‌ای وجود ندارد.

۴.۲ پیاده‌سازی الگوریتم

در نسخه ابتدایی، مدلمان شکل ساده‌ای داشت. تنها سه لایه dense وجود داشت که اولی ورودی و سومی خروجی بود، در نتیجه عملا تنها یک لایه میانی وجود داشت و انتظار چندانی از مدل نمی‌شد داشت. با تحقیق بیشتر، نتیجه گرفته شد که از چهار لایه Conv2D موجود در Keras به همراه سه لایه Dense استفاده شود. لایه ورودی یکی از این چهار لایه Conv2D می‌باشد که ابعاد ورودی‌اش برابر ابعاد بازی است. لایه خروجی یک لایه Dense می‌باشد که به تعداد خانه‌های بازی نود دارد. هر خانه معادل با یک عمل کلیک روی آن خانه است. مقادیر نودهای خروجی Q Valueها خواهد بود.

۵.۲ ساده‌سازی‌ها

همانطور که گفته شد، در ابتدا کلاس MinesweeperGame تنها جدول بازی را ساخته و حرکت‌ها را روی آن اعمال می‌کرده است. ولی با توجه به کم بودن سرعت یادگیری توسط الگوریتم، تصمیم گرفته شد که ساده‌سازی‌هایی در این مدل به وجود بیاید که در زیر توضیح داده می‌شوند.

۱.۵.۲ نشان دادن یک خانه قبل از اولین حرکت

در اکثر پیاده‌سازی‌های بازی مین‌روب، این موضوع در نظر گرفته شده است که قبل از اولین حرکت، هیچ بخشی از جدول بازی برای بازیکن رویت نشده است و حرکتش کاملا اتفاقی است. در نتیجه اگر در خانه کلیک شده مین وجود داشته باشد، بازی قبل از شروع شدن پایان می‌یابد. به همین دلیل معمولا جدول بازی را پس از اولین کلیک، و با توجه به آن کلیک می‌سازند. ما در این پروژه به جای این کار، یک خانه بدون مین را قبل از حرکت بازیکن نمایان می‌سازیم.

۲.۵.۲ نمایش خانه‌ها با مقدار صفر

با رویت یک خانه که مقدارش صفر باشد، به این معنی که در هیچ یک از حداکثر ۸ خانه اطرافش مین وجود نداشته باشد، خانه‌های اطرافش گزینه‌های بدیهی بعدی برای کلیک می‌شوند. همگام با اکثر پیاده‌سازی‌ها، ما نیز این حرکت‌های بدیهی را به این صورت حذف می‌کنیم که با کلیک روی یک خانه با مقدار صفر، تمام خانه‌های اطرافش رویت می‌شود و این موضوع تا جایی پیش می‌رود که به خانه‌هایی با مقدار غیر صفر برسیم و عملا این جزیره صفرها را کامل نمایش دهیم.

۳.۵.۲ شروع از خانه با مقدار صفر

با ترکیب دو ساده‌سازی بالا، می‌توانیم شروع را از حالت ذاتا تصادفی خود بیرون بیاوریم تا یادگیری نیز آسان‌تر شود. برای این کار، اولین خانه که قبل از حرکت بازیکن رویت می‌شود را از بین خانه‌های با مقدار صفر انتخاب می‌کنیم تا به جای یک خانه، مجموعه‌ای از خانه‌ها نمایش داده شوند و روند بازی کمتر به شانس وابسته باشد.

۴.۵.۲ حذف حرکت‌های تکراری

با هر حرکت، یک خانه از جدول نمایان می‌شود. این موضوع را شبکه عصبی ما می‌تواند در بلند مدت با مقادیر ورودی تشخیص دهد، در ورودی، به جای خانه‌هایی که هنوز نمایش داده نشده‌اند، عدد منفی ۲ و برای خانه‌های نمایش داده شده، مقادیر خودشان را وارد می‌کنیم. مین‌ها نیز در کد با منفی ۱ نمایش داده می‌شوند ولی هیچگاه به عنوان ورودی به شبکه عصبی خورانده نمی‌شوند (اگر مینی رویت شده باشد به معنی پایان بازی است). با این حال، زمان بسیار زیادی برای آموزش دادن این موضوع به شبکه‌مان باید سپری شود. پس به جای این که با تنظیم پاداش‌ها به شبکه عصبی‌مان بیاموزیم که از انجام حرکت تکراری خودداری کند، می‌توانیم با تنظیم Q Valueها در زمان آپدیت کردن شبکه این کار را انجام دهیم. روش ما در این پروژه، تغییر Q Value اینگونه حرکت‌های تکراری به کمترین Q Value موجود است.

۶.۲ آموزش

با حذف مرحله آموزش انجام ندادن حرکت‌های تکراری، دو مسئله دیگر برای ما باقی می‌ماند که باید به الگوریتممان آموزش داده شود: انجام ندادن حرکات تصادفی (حرکات در قسمتی از صفحه بازی که هیچ خانه کناری رویت نشده باشد) و انتخاب حرکت بعدی بدون رویت مین.

با توجه به این موضوع، پاداش‌هایمان را به شکل زیر تنظیم می‌کنیم:

پاداش اتمام موفقیت‌آمیز بازی: ۱

پاداش رویت مین: منفی ۱

پاداش حرکت عادی موفقیت‌آمیز: ۰.۳

پاداش حرکت تصادفی: منفی ۰.۴

مقدار ضریب یادگیری (learning rate) را ۷۰ درصد در نظر می‌گیریم.

تابع فعال‌سازی استفاده شده تابع ReLU می‌باشد.

۷.۲ مشکل با **discount factor**

در ابتدا برای discount factor عدد ۰.۶ در نظر گرفته شده بود. ولی با دیدن اختلالاتی در یادگیری، مانند تکرار شدن دوره‌ای اشتباه پس از حل شدنشان حین آموزش، متوجه شدیم که این عدد برای کار ما مناسب نیست. با توجه به کم‌اهمیت بودن پاداش‌های آینده، این عدد باید بسیار پایین‌تر باشد. پس discount factor را ۰.۰۵ قرار می‌دهیم، هرچند می‌توانیم به صفر کردنش نیز فکر کنیم.

۸.۲ تغییرات در طول مسیر

در چند مرحله از این پروژه، تلاش کرده شد که با دادن تغییرات در قسمت‌های مختلف مدل یا بازی، نتیجه بهتری گرفته شود. نمونه‌هایی از آن‌ها در قسمت‌های ۵ و ۷ فصل دوم نوشته شده است. برخی دیگر را در زیر بیان می‌کنیم.

۱.۸.۲ تعداد و نوع لایه‌ها

بدیهی‌ترین تغییری که در مدل می‌توان داد، تغییر تعداد و جنس لایه‌های شبکه عصبی است. در ابتدا، تنها یک لایه میانه کاملا متصل داشتیم که مشخصا پاسخگوی نیازهایمان نبود. پس از تلاش برای استفاده از دو یا سه لایه میانی، در نهایت تصمیم گرفته شد که چهار لایه Conv2D و سه لایه Dense داشته باشیم، که به معنی داشتن ۵ لایه میانی است. همچنین Conv2D بودن لایه ورودی به ما اجاره می‌داد به جای دادن ورودی‌ها به عنوان یک لیست مسطح، آن‌ها را به عنوان فضایی دو بعدی به شبکه عصبیمان وارد کنیم.

۲.۸.۲ مقداردهی اولیه

برای مقداردهی اولیه شبکه عصبی نیز از روش‌های مختلفی استفاده شد. این روش‌ها HeUniform، مقداردهی با صفر و مقداردهی با یک بودند. مقداردهی با صفر و HeUniform نتیجه بهتری داشتند که در نهایت HeUniform انتخاب شد.

۳.۸.۲ ضریب یادگیری

ضریب یادگیری ما در ابتدا ۰.۷ بود. پس از پردازش چند صد هزار بازی، شک کردیم که شاید بخاطر زیاد بودن این ضریب، تاثیر هر حرکت روی مدل بیش از حد معقول است. در نتیجه عددهای ۰.۳ و ۰.۲ نیز امتحان شد، ولی نتیجه چندان متفاوت نبود.

۴.۸.۲ یادگیری در هر **N** حرکت

برای افزایش سرعت یادگیری، پس از مدتی تصمیم گرفته شد که به جای هر حرکت، در هر N حرکت مدل ما خود را بروزرسانی کند تا سرعت یادگیری بیشتر شود. تاثیر این کار چشم‌گیر بود. عددهای امتحان شده برای N برابر بودند با ۱، ۲، ۴، ۵، و ۲۰ که در نهایت عدد ۵ انتخاب شد.

۹.۲ نتیجه

پس از یادگیری روی بیش از یک میلیون بازی، الگوریتم ما توانسته تشخیص دهد که حرکت تصادفی انجام ندهد. ولی متاسفانه همچنان توانایی تشخیص مین‌ها را ندارد و بازی به سرعت با باخت به پایان می‌رسد. در نتیجه از دو چیزی که هدف آموختنش را داشتیم، تنها یکی نتیجه‌بخش بوده است.

شکل ۳.۲: نمونه‌ای نهایی از اجرا روی بازی

```
0 0 1 1 1 0 0 1 1
2 2 3 X 2 0 1 2 -
- - - - 2 0 1 - -
- 3 2 1 1 0 1 1 1
1 1 0 0 1 1 1 0 0
0 0 0 1 2 - 1 0 0
0 0 0 1 - 2 1 0 0
0 0 0 1 1 1 0 1 1
0 0 0 0 0 0 0 1 -
Rewards: [0.3, 0.3, 0.3, 0.3, 0.3, -1]
Mean rewards: 0.08333333333333333
Median rewards: 0.3
```

۱۰.۲ کارهای احتمالی آینده

در آینده می‌توان با تغییر تعداد لایه‌ها، مقدار متغیرهای ضریب یادگیری و discount factor، و به طور خاص loss function و optimizer که به طور پیش‌فرض و بدون تغییر در طول مسیر، Huber و Adam انتخاب شده بودند، به نتیجه مطلوب‌تری دست یافت.

مراجع

[1] Jayashree P., Ramakrishnan K. (2018) Design and Evaluation of Reinforcement Learning Based AI Agent: A Case Study in Gaming. In: Abraham A., Cherukuri A., Madureira A., Muda A. (eds) Proceedings of the Eighth International Conference on Soft Computing and Pattern Recognition (SoCPaR 2016). SoCPaR 2016. Advances in Intelligent Systems and Computing, vol 614. Springer, Cham. https://doi.org/10.1007/978-3-319-60618-7_33

[2] Sutton, Richard; Barto, Andrew (1998). Reinforcement Learning: An Introduction. MIT Press. p. 127-129

[3] Schmidhuber, J. (2015). ”Deep Learning in Neural Networks: An Overview”. Neural Networks. 61: 85–117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003. PMID 25462637. S2CID 11715509

[4] Watkins, C.J.C.H., Dayan, P. Q-learning. Mach Learn 8, 279–292 (1992). https://doi.org/10.1007/BF00992698

[5] Russell, Stuart J.; Norvig, Peter (2010). Artificial Intelligence: A Modern Approach (Third ed.). Prentice Hall. p. 649. ISBN 978-0136042594.

[6] Matiisen, Tambet (December 19, 2015). ”Demystifying Deep Reinforcement Learning”. neuro.cs.ut.ee. Computational Neuroscience Lab. Retrieved 2018-04-06.

[7] Becerra, David J. 2015. Algorithmic Approaches to Playing Minesweeper. Bachelor’s thesis, Harvard College.

Abstract

In recent years, reinforcement learning has attracted great attention and had a significant effect on artificial intelligence and its industry. In this method, our model is placed in an environment where rewards and penalties are given for the agent's actions. The model learns about the environment through trial and error. That is why reinforcement learning has been used in games as simple as tic-tac-toe and as difficult as chess and go. These games have attractive environments for implementing different kinds of reinforcement learning. In this project reinforcement learning is used to teach minesweeper to our agent. We use Python and Tensorflow and the game is also implemented alongside our model.

College of Science

School of Mathematics, Statistics, and Computer Science

Reinforcement Learning and the Game of Minesweeper

Alireza Moghaddas Borhan

Supervisor: Dr. Hedieh Sajedi

A thesis submitted in partial fulfillment of

the requirements for the degree of

B.Sc. in Computer Science

February 2022