مسئله چندعامله Pac-Man

تمرین درس هوش مصنوعی – آذر ۱۴۰۰

دانشگاه صنعتی امیرکبیر

کدهای اصلی بازی در اختیار شما قرار گرفته است. وظیفه شما دو قسمت خواهد بود: تئوری و عملی. بخش تئوری جواب دادن تشریحی به برخی سوالات است و عملی نیز پیاده سازی الگوریتم به زبان پایتون خواهد بود.

فهرست

مسئله چندعامله Pac-Man ۱

تمرین درس هوش مصنوعی – آذر ۱۴۰۰ ..... ۱

مقدمه ۳

توضیحات بازی ۳

موارد ارسالی ۴

شروع بازی ۵

مسئله اول ۶

مقدمه ۶

مسئله اول-تئوری (۱۰ امتیاز) ۶

مسئله اول-عملی (۲۰ امتیاز) ۷

مسئله اول – تئوری امتیازی (۵ امتیاز) ۸

مسئله دوم – مینیمکس (۳۰ امتیاز) ۹

مسئله سوم – مینیمکس مورد انتظار ۱۰

مسئله سوم – تئوری (۱۵ امتیاز) ۱۰

مسئله سوم – عملی (۲۵ امتیاز) ۱۰

مسئله چهارم – مسائل بد سلوک (امتیازی) ۱۱

مقدمه و تعریف مسائل بدسلوک ۱۱

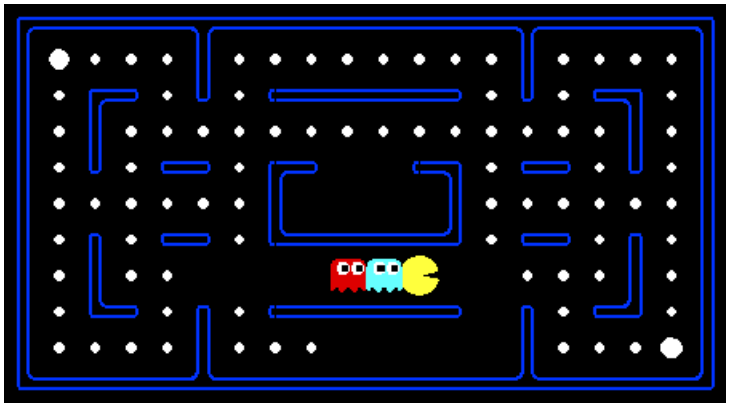
مسئله چهارم – تئوری امتیازی (۱۰ امتیاز) ۱۲

چند ویژگی از مسائل بدسلوک ۱۳

مقدمه

توضیحات بازی

پک‌من (Pac-Man) نام یک بازی قدیمی است که کاراکتر اصلی بازی (دایره زرد رنگ) باید در زمین بازی نقطه‌هایی را به عنوان امتیاز جمع کند و از کاراکترهای منفی بازی که به شکل روح هستند، فرار کند. نحوه حرکت پک‌من در زمین تو در توی بازی و همچنین عملکرد روح‌ها و همچنین تعداد آن‌ها می‌تواند به عنوان یک مسئله‌ی تخصامی چند عامله تعریف شود. در این تمرین کدهای اصلی برای شروع بازی را در اختیار دارید و باید در مراحل مختلف، الگوریتم‌های چندعامله را پیاده سازی کنید (شکل ۱).



۱- نمایی از بازی پکمن

بازی در دو حالت تمام می‌شود. اگر پکمن تمام نقطه‌های سفید را بخورد برنده می‌شود و اگر روح‌ها به پکمن برسند، پکمن میبازد و بازی تمام می‌شود. دو نقطه سفید بزرگ در گوشه‌های زمین وجود دارد که اگر پکمن آن‌ها را بخورد، در یک زمان محدود قدرت بیشتری بدست می‌آورد و میتواند روح‌ها رو بخورد. البته این قسمت مربوط به الگوریتم ما نمی‌شود. در ادامه شما باید الگوریتم‌هایی مانند مینی‌مکس برای عامل‌های بازی طراحی کنید (هم برای پکمن و هم برای روح‌ها)

موارد ارسالی

کدهای بازی که دراختیار شما قرار گرفته است که فایل‌های زیادی دارد که توضیح داده خواهد شد اما **به فهمیدن تمام فایل‌ها نیازی نیست**. تنها فایلی که شما باید تغییر دهید و ارسال کنید، submission.py است.

بخشی که شما باید کد خود را وارد کنید با کامنت زیر مشخص شده است:

```
# BEGIN_YOUR_CODE
```

لطفا نام هیچ کلاس و متغیری که از قبل وجود دارد را تغییر ندهید.

یک فایل ورد به نام pacman نیز در اختیار دارید. لطفا سوالات تشریحی را در آن وارد کنید و خروجی pdf بگیرید و به همراه فایل submission.py ارسال کنید.

بنابراین دو نوع سوال تئوری و عملی داریم. سوالات تئوری را در فایل ورد pacman وارد کنید و سوالات عملی را هم به زبان پایتون در submission.py بنویسید.

نکته مهم (حتما بخوانید): برای اجرای درست کد بازی، لطفا چک کنید که نسخه پایتون سیستم شما حداقل ۳.۷.۵ باشد. برای اینکار میتوانید دستور زیر را در ترمینال وارد کنید:

```
python --version
```

یک کد تست کننده برنامه نیز وجود دارد که دو روش سنجش دارد. لطفا فایل grader.py را ببینید. در روش اول سنجش، هیچ فشاری بر روی الگوریتم شما وارد نمی‌کند و تنها میزان صحت آن را می‌سنجد. در روش دوم سنجش پیچیده‌تر با اعمال فشار بر روی الگوریتم (مانند ورودی‌های بزرگ) انجام می‌شود. نمره‌دهی اولیه برای تمرین توسط همین فایل انجام خواهد گرفت. لطفا در مرحله آخر و پس از پیاده سازی الگوریتم خود، آن را اجرا کنید و بخش‌های آن را مطالعه کنید تا بفهمید چگونه الگوریتم شما را تست می‌کند (به فهمیدن این فایل نیازی نیست و تنها جنبه آموزشی دارد)

شروع بازی

قبل از هر کاری، یک بار بازی را اجرا کنید تا محیط آن و نحوه عملکرد بازی را ببینید. با دستور زیر میتوانید بازی را اجرا کنید:

```
python pacman.py
```

میتوانید آرگومان زیر را وارد کنید تا بازی به شکل کندتر اجرا شود:

```
python pacman.py --frameTime 1
```

دستور زیر را اجرا کنید و تفاوت عملکرد پکمن را ببینید:

```
python pacman.py -p ReflexAgent
```

این دستور، کلاس `ReflexAgent` که در فایل submission.py وجود دارد را اجرا می‌کند. حتی اگر بازی را در زمین ساده‌تری اجرا کنیم، متوجه می‌شوید که پکمن عملکرد خوبی ندارد:

```
python pacman.py -p ReflexAgent -l testClassic
```

می‌توانید بازی را با ۱ یا ۲ روح اجرا کنید:

```
python pacman.py -p ReflexAgent -k 1
python pacman.py -p ReflexAgent -k 2
```

اگر میخواهید روح ها کمی بهتر عمل کنند، آرگومان `-g DirectionalGhost` را اضافه کنید. همچنین میتوانید تعیین کنید بازی چند بار اجرا شود. ابتدا n- را وارد کنید و سپس تعداد بازی. اگر میخواهید بازی بدون محیط گرافیکی اجرا شود q- را اضافه کنید.

تمام آرگومان‌های ممکنی که میتوانید به بازی دهید در فایل pacman.py موجود است.

مهم ترین کار شما در این مرحله، مطالعه دقیق کلاس `ReflexAgent` در فایل submission.py است، سعی کنید نحوه کارکرد این کلاس را متوجه شوید.

مسئله اول

مقدمه

قبل از اینکه بازی پکمن را به عنوان یک عامل Minimax کدنویسی کنید، توجه کنید که به جای تنها یک روح، پکمن می تواند چندین روح را به عنوان حریف داشته باشد. ما الگوریتم مینیمکس را که در کتاب درسی تنها یک لایه برای یک حریف داشت، به حالت کلی تر، یعنی چند حریف گسترش می دهیم. به طور خاص، درخت مینیمکس شما چندین لایه min خواهد داشت (برای هر روح یک لایه).

به طور کلی، جستجوی مینیمکس درختی با عمق محدود را با توابع ارزیابی که در کلاس آموزش داده شد، در نظر بگیرید. فرض کنید $n+1$ عامل در زمین بازی وجود دارد ($a_0,a_1,\ldots,a_n$)، که پکمن همان a_0 و بقیه روح‌ها هستند. پکمن به عنوان عامل max عمل می کند و روح‌ها به عنوان عامل min عمل می کنند. یک عمق یعنی تمام $n+1$ عامل، حرکت خود را یک بار انجام دهند. بنابراین در عمق ۲، پکمن و هر شبح ۲ بار حرکت انجام داده‌اند. به عبارت دیگر، عمق ۲ مربوط به ارتفاع $2(n+1)$ در درخت بازی minimax است.

نکته: در واقعیت و در زمین بازی، همه عامل‌ها به طور همزمان حرکت می کنند. در فرمول ما، حرکت‌هایی که در یک عمیق انجام می‌شوند، در زمین بازی در یک لحظه انجام می‌شوند. ما در این فرمول برای ساده کردن کارها، پکمن و ارواح را به صورت متوالی پردازش می کنیم.

مسئله اول–تئوری (۱۰ امتیاز)

با توجه به مقدمه بالا مشابه تابع مینیمکس در کتاب، یک تابع ریاضی به شکل $V_{minimax}(s,d)$ برای بازی پکمن بنویسید. برای این کار و نام گذاری تابع‌های موردنیاز خود، لطفا از نام های زیر استفاده کنید:

- *IsEnd(s)*، چک می‌کند که آیا s یک حالت نهایی است یا خیر.
- *Utility(s)*، تابع سودمندی (یوتیلیتی) برای حالت s
- *Eval(s)*، تابع ارزیابی برای حالت s
- *Player(s)*، نشان می‌دهد در حالت s نوبت کدام عامل است.
- *Actions(s)*، حرکت‌هایی که میتوان در حالت s انجام داد را نشان می‌دهد.
- *Succ(s,a)*، تابع ساکسسور، که با توجه به حرکت‌ها (اکشن) a در حالت s، حالت‌های ممکن بعدی را نشان می‌دهد.

همچنین می‌توانید از n در هر جایی از راه‌حل خود استفاده کنید، بدون اینکه صریحاً آن را به عنوان آرگومان وارد کنید.

مسئله اول–عملی (۲۰ امتیاز)

حال که فرمول مسئله را بدست آورده‌اید می‌توانید در فایل submission.py، کلاس MinimaxAgent را پیدا کرده و در بخشی که مشخص شده است، کد خود را بنویسید.

به یاد داشته باشید که عامل مینیکس شما (پکمن) باید با هر تعداد روح (حریف) کار کند و درخت مینیمکس شما باید چندین لایه min (به ازای هر شبح) داشته باشد.

کد شما باید بتواند درخت بازی را تا هر عمق مشخص گسترش دهد. برگ های درخت مینیمکس خود را میتوانید با تابع self.evaluationFunction که از قبل نوشته شده است، امتیاز دهید. کلاس MinimaxAgent ، از کلاس MultiAgentSearchAgent ارث بری کرده است و میتوانید به self.depth و self.evaluationFunction دسترسی داشته باشید. هر جا که لازم است از این دو متغیر استفاده کنید.

نکات راهنما برای پیاده سازی:

- قبل از شروع به کد زدن، تمام کامنت ها در فایل submission.py را به دقت بخوانید.
- پکمن همیشه عامل شماره ۰ است و شاخص عامل ها به ترتیب افزایش می‌یابد. از self.index در پیاده سازی مینیمکس خود برای دسترسی به شاخص پکمن استفاده کنید. توجه داشته باشید که فقط پکمن می‌تواند MinimaxAgent شما را اجرا کند.
- تمام حالت‌های مینیمکس باید GameStates باشند، یا به getAction منتقل شوند یا از طریق GameState.generateSuccessor تولید شوند. در این تکلیف، شما از حالت های ساده انتزاعی استفاده نخواهید کرد.
- ممکن است توابع شرح داده شده در کامنت‌های ReflexAgent و MinimaxAgent برای شما مفید باشد.
- تابع ارزیابی این قسمت قبلاً نوشته شده است (self.evaluationFunction) و شما باید این تابع را بدون تغییر دادن آن فراخوانی کنید. در تعریف $V_{minimax}$ (در قسمت اول سوال) هر جا که از Eval(s) استفاده کردید، در کد از self.evaluationFunction میتوانید استفاده کنید. بدانید که اکنون ما به جای اعمال، حالات را ارزیابی می کنیم. عامل‌های نگاه به آینده (Look-ahead agents) *حالت های آینده* را ارزیابی می کنند در حالی که عامل‌های بازتابی (Reflex agents) اقدامات (actions) را از حالت فعلی ارزیابی می کنند.

- اگر بین چندین حرکت برای بهترین حرکت، تساوی وجود داشته باشد، ممکن است هر طور که صلاح می‌دانید، یکی را انتخاب کنید (شکست تساوی).
- مقادیر minimax حالت اولیه در طرح minimaxClassic به ترتیب ۹، ۸، ۷ و -۴۹۲ برای اعماق ۱، ۲، ۳ و ۴ است (با آرگومان "-a depth=[depth]" مقدار دهی می‌شود). می توانید از این اعداد برای بررسی درستی اجرای خود استفاده کنید. برای تأیید، می‌توانید مقدار minimax وضعیتی را که به getAction منتقل می‌شود چاپ کنید و بررسی کنید که آیا مقدار حالت اولیه (اولین مقداری که ظاهر می‌شود) برابر با مقدار ذکر شده در بالا است. توجه داشته باشید که عامل پکمن شما اغلب برنده می شود، علیرغم پیش بینی باخت جستجوی عمق ۴ minimax (دستور آن در زیر نشان داده شده است). با عمق ۴، عامل پکمن ما ۵۰-۷۰٪ مواقع برنده می شود. عمق ۲ و ۳ نرخ برد کمتری را ارائه می دهد. حتما روی تعداد زیادی بازی با استفاده از آرگومان -n و -q تست کنید.

```
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
```

- فراموش نکنید که یک عمق، شامل حرکت‌های پکمن و تمام روح‌های دیگر است.

مسئله اول – تئوری امتیازی (۵ امتیاز)

۱) در زمین‌های بزرگتر بازی مانند openClassic و mediumClassic (پیش‌فرض)، پکمن را می‌بینید که در نمردن خوب است، اما در برنده شدن بسیار بد است. او غالباً بدون اینکه پیشرفتی داشته باشد، در زمین ولچرخی میکند!! او حتی ممکن است دقیقاً در کنار یک نقطه سفید (امتیاز) بدون خوردن آن بچرخد. اگر این رفتار را دیدید نگران نباشید. چرا پکمن درست در کنار یک نقطه ولچرخی میکند؟ (حداکثر چهار خط بنویسید)

۲) بازی را با این دستور اجرا کنید:

```
python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
```

فکر می‌کنید چرا پکمن در جستجوی مینیمکس در زمین بازی trappedClassic به نزدیک‌ترین روح حمله می‌کند تا توسط روح خورده شود؟ (حداکثر چهار خط)

مسئله دوم – مینیمکس (۳۰ امتیاز)

عامل جدیدی بسازید که از هرس آلفا-بتا برای کاوش موثرتر درخت مینیمکس استفاده می‌کند (کلاس AlphaBetaAgent). توجه کنید که الگوریتم شما کمی کلی تر از شبه کد موجود در کتاب خواهد بود، بنابراین بخشی از چالش این است که منطق هرس آلفا-بتا را به طور مناسب به چندین عامل روح گسترش دهید.

احتمالا یک افزایش سرعت مشاهده کنید: شاید عمق ۳ آلفا-بتا به سرعت عمق ۲ مینیمکس عادی اجرا شود. در حالت ایده‌آل، عمق ۳ در زمین بازی mediumClassic باید فقط در چند ثانیه در هر حرکت یا سریع‌تر اجرا شود.

```
python pacman.py -p AlphaBetaAgent -a depth=3
```

مقادیر مینیمکس AlphaBetaAgent باید با مقادیر مینیمکس MinimaxAgent یکسان باشد، اگرچه عمل‌هایی (actions) که انتخاب می‌کند می‌تواند به دلیل رفتارهای مختلف شکستن-تساوی تغییر کند. مجدداً توجه داشته باشید که مقادیر مینیمکس حالت اولیه در طرح minimaxClassic به ترتیب ۹، ۸، ۷ و ۴۹۲- برای اعماق ۱، ۲، ۳ و ۴ است. با اجرای دستور داده شده در بالای این پاراگراف، که از mediumClassic پیش فرض استفاده می کند، مقادیر مینیمکس حالت اولیه باید به ترتیب ۹، ۱۸، ۲۷، و ۳۶ برای اعماق ۱، ۲، ۳ و ۴ باشد. مجدداً، می‌توانید با چاپ مقدار مینیمکس که تابع getAction میتواند به شما بدهد، این موضوع را چک کنید. توجه داشته باشید که هنگام مقایسه عملکرد زمانی AlphaBetaAgent با MinimaxAgent، مطمئن شوید که از زمین بازی یکسانی برای هر دو استفاده می‌کنید. می‌توانید با اضافه کردن

```
-l minimaxClassic
```

به دستور داده‌شده در بالای این پاراگراف، زمین را به‌صورت دستی تنظیم کنید.

مسئله سوم – مینیمکس مورد انتظار

مسئله سوم – تئوری (۱۰ امتیاز)

می‌دانیم که حریف‌های تصادفی، عامل‌های خوبی برای الگوریتم مینیمکس عادی نیستند، بنابراین مدل‌سازی آنها با جستجوی مینی‌مکس بهینه نیست. بنابراین، تابع $V_{exptmax}(s,d)$ را بنویسید، که حداکثر سودمندی مورد انتظار (maximum expected utility) در برابر حریف‌هایی تصادفی به ما می‌دهد. حریف‌هایی که که هر کدام از خط مشی تصادفی پیروی می کنند، یعنی یک حرکت قانونی را به طور یکسان و تصادفی انتخاب می کند. تابع شما باید شبیه تابع مسئله اول باشد که نوشتید ، به این معنی که باید آن را بر اساس همان توابعی بنویسید که در مسئله اول مشخص شده است.

مسئله سوم – عملی (۳۰ امتیاز)

کلاس ExpectimaxAgent را کامل کنید، حالا عامل پکمن شما دیگر فرض نمی کند که حریف (روح‌ها) عمل‌هایی را انجام دهند که سود پکمن را به حداقل برسانند. در عوض، پکمن سعی می‌کند تا تابع سودمندی موردانتظار خود را به حداکثر برساند و فرض می‌کند که در برابر چندین RandomGhost بازی می‌کند، که هر کدام از getLegalActions به‌طور یکنواخت و تصادفی انتخاب می‌کنند.

اکنون باید مشاهده کنید که پکمن رویکرد بی‌باک‌تری برای نزدیک شدن به روح‌ها دارد. به ویژه، اگر پکمن درک کند که ممکن است به دام بی‌افتد اما شاید بتواند فرار کند، برای گرفتن چند تکه غذا، حداقل تلاش خواهد کرد و خود را به خطر خواهد انداخت.

```
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3
```

ممکن است مجبور شوید این سناریو را چند بار اجرا کنید تا ببینید که این قمار پکمن به ثمر می‌رسد. پکمن به طور متوسط نیمی از بازی‌ها را برنده خواهد شد، و برای این دستور خاص که بالا نوشته شده است، امتیاز نهایی در صورت باخت پکمن 502- و در صورت برد، ۵۳۲ یا ۵۳۱ (بسته به روش شکست تساوی شما و نحوه اجرا شدن کد) خواهد بود. شما می توانید از این اعداد برای تأیید اعتبار پیاده سازی خود استفاده کنید.

این سوال فقط برای تفکر خودتان است:

چرا رفتار پکمن به عنوان یک عامل ماکسیمم مورد انتظار (expectimax) با رفتار او به عنوان یک عامل مینیمکس تفاوت دارد (یعنی چرا او مستقیماً به سمت روح‌ها نمی رود)؟ توجه کنید که فقط در مورد آن فکر کنید. نیازی به نوشتن جواب نیست.

مسئله چهارم – مسائل بد سلوک (امتیازی)

مقدمه و تعریف مسائل بدسلوک

بازی هایی مانند پکمن به خوبی فرموله شده اند تا با هوش مصنوعی حل شوند. حالت های آنها از قبل تعریف شده، مجموعه محدودی از حرکات مجاز دارند و مهمتر از همه، شرایط برد را به وضوح تعریف کرده اند. حل و تدوین مشکلات دنیای واقعی اغلب دشوارتر است.

در بخش مدل‌سازی برای پارادایم مدل‌سازی-استنتاج-یادگیری، ما یک مسئله دنیای واقعی را به عنوان یک مدل فرموله می‌کنیم و یک انتزاع دقیق ریاضی به منظور تسهیل مقابله با آن می‌سازیم. مدل‌سازی ذاتاً با خطا انجام می‌شود، اما در برخی از حوزه‌ها، حتی تعیین مفهوم و معنی آن، دشوار و بحث‌برانگیز است.

مشکلاتی که دارای اهداف متعدد و بالقوه متضاد هستند و درجه بالایی از عدم قطعیت و ریسک دارند و در مورد اینکه چه چیزی به عنوان راه حل مشکل به حساب می آید، بین نظر ذینفعان، اختلاف نظر وجود دارد. این مسائل گاهی اوقات «مسائل بدسلوک» (wicked problems) نامیده می شوند (منبع). در اینجا یک مسئله بدسلوک را با یک بازی ساده مانند پکمن مقایسه می کنیم. می‌توانید فهرست جامع‌تری از ویژگی‌های یک بدسلوک را در این بخش وجود دارد.

مثال مسئله بدسلوک: مدیریت آب در ایالت کالیفرنیا	ویژگی مسئله بدسلوک	مثال پکمن	ویژگی‌های بازی ساده
تخصیص عادلانه آب بین کشاورزان و شهرها؟ حفظ آب به گونه ای که اکوسیستم های طبیعی اطراف هنوز رشد می کنند؟ چه ترکیبی از این اولویت ها؟	هیچ توافقی برای فرمول بندی مشکل وجود ندارد	تا جایی که ممکن است نقطه سفید بخور	فرمول بندی و تعریف مسئله واضع
هر تصمیمی پیامدهایی برای مردم در زمان انجام آن دارد، حتی اگر بعداً تصمیم دیگری گرفته شود.	فقط یک بار می‌توان امتحان کرد	می توانید بازی را مجدداً بازی کنید تا زمانی که آن را درست انجام دهید	چندین بار می‌توان امتحان کرد
آیا این تعداد هکتار از محصولات آبیاری شده است؟ اکوسیستم بازسازی شد؟ تعداد خانه هایی که دیگر آب دریافت نمی کنند؟	هیچ تعریفی برای نحوه اندازه‌گیری موفقیت در حل وجود ندارد	شمردن نقطه‌های سفید خورده شده	تعریف واضح از برد

مسئله چهارم – تئوری امتیازی (۱۰ امتیاز)

برخی از مسائل بدسلوک قدیمی در تلاش برای پرداختن به مسائلی مانند فقر، بی خانمانی، جرم و جنایت، تغییرات آب و هوای جهانی، تروریسم، مراقبت های بهداشتی، سلامت زیست محیطی و بیماری های همه گیر وجود دارند. یکی از این نه حوزه مسئله بدسلوک را انتخاب کنید (به جز این حوزه‌ها یا یک حوزه جدید را انتخاب کنید که خودتان به وضوح تعریف کرده‌اید؛ یا از این منبع یک مسئله بدسلوک دیگر انتخاب کنید).

در ۴ تا ۶ جمله توضیح دهید که چرا این مسئله یک مسئله بدسلوک است. نشان دهید که حداقل دو معیار مسئله بدسلوک را دارد و توضیح دهید که چرا فرمول بندی، مدل سازی و حل آن به عنوان یک مسئله هوش مصنوعی دشوار است. تنها ذکر معیارها کافی نیست. (مثال بعدی را ببینید)

مثال: مدیریت منابع آب در کالیفرنیا یک مسئله بدسلوک است. مانند جدول بالا، ذینفعان مختلف در مورد اینکه مشکل چیست اختلاف نظر دارند. کشاورزانی که به آب بیشتری نیاز دارند ممکن است مشکل را کمبود آب برای آبیاری بدانند، در حالی که محیط بانان مشکل را این واقعیت می دانند که آب زیادی هدر می شود که اکوسیستم های اطراف دیگر سالم نیستند. بنابراین تعیین معیارهای یک راه حل در یک مسئله یادگیری هوش مصنوعی دشوار است: کدام یک از این دو باید به عنوان حالت های راه حل اولویت بندی شوند، یا چگونه باید دو فرمول مسئله را با هم ترکیب کرد؟ همانطور که در جدول بالا میبینیم، وجود یک اختلاف را می توان به چند روش مختلف حل کرد. اگر آب کمتر از حد انتظار مدل به اقیانوس آرام برسد، آیا به این دلیل است که آب بیش از حد برای استفاده انسان منحرف می‌شود، یا به این دلیل است که اکوسیستم در طول مسیر آب بیشتری از حد انتظار جذب می‌کند؟ هوش مصنوعی ما باید از این اختلاف در مقادیر آب درس بگیرد، اما بدون تحقیق بیشتر، دشوار است که بدانیم چگونه به درستی این اختلاف را مدل کنیم و بنابراین چه چیزی از آن یاد بگیریم.

چند ویژگی از مسائل بدسلوک

- هیچ فرمول قطعی برای یک مسئله بدسلوک وجود ندارد. ذینفعان مختلف در مورد مشکلی که باید حل شود، اختلاف نظر دارند.
- مسئله بدسلوک قانون توقف (stopping rule) ندارد.
- راه‌حل‌های مسئله بدسلوک، درست یا نادرست نیستند، بلکه خوب یا بد هستند. پایان به عنوان "بهتر" یا "بدتر" یا "به اندازه کافی خوب" ارزیابی می شود.
- هیچ آزمون قطعی فوری یا نهایی برای راه حل یک مسئله بدسلوک وجود ندارد.
- هر راه حلی برای یک مسئله بدسلوک یک "عملیات یکباره" است. زیرا هیچ فرصتی برای یادگیری با آزمون و خطا وجود ندارد، هر امتحان کردنی اهمیت زیادی دارد.
- مسائل بدسلوک مجموعه ای از راه حل های بالقوه قابل شمارش (یا به طور کامل قابل توصیف) ندارند، و همچنین مجموعه ای از عملیات مجاز به خوبی توصیف شده که ممکن است در طرح گنجانده شود وجود ندارد.
- هر مسئله بدسلوک اساسا منحصر به فرد است. ۸
- هر مسئله بدسلوک را می توان نشانه مسئله دیگری دانست.
- وجود یک اختلاف نشان دهنده یک مسئله بدسلوک را می توان به روش های متعددی توضیح داد. انتخاب توضیح ماهیت حل مشکل را تعیین می کند.
- برنامه ریز حق ندارد خطا یا اشتباه کند، زیرا هر اشتباه عواقبی دارد.

موفق باشید 😊