دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

پروژه سوم مبانی و کاربردهای هوش مصنوعی

نگارش
مهدیه سادات بنیس

استاد درس
دکتر مهدی جوانمردی

نیم سال دوم ۱۴۰۱

• بخش ۱: تکرار ارزش

توضیحات پیاده سازی:

برای پیاده سازی value iteration ابتدا تابع computeQValueFromValues را پیاده سازی میکنیم تا از تکرار کد بپرهیزیم زیرا میدانیم Value هر استیت در وقاع ماکسیمم Q-value های مربوط به آن استیت می باشد. خروجی دستورات: با توجه به رابطه زیر که برای محاسبه Q-value ها داریم:

پیاده سازی را میتوانیم به صورت زیر انجام دهیم یعنی درهر استیت با توجه به اکشن انجام شده که در ورودی می آید استیت های بعدی و اینکه چقدر احتمال دارد به آن استیت برویم را از شی mdp به کمک تابع getTransitionStatesAndProbs میگیریم و برای هر یک از از آن ها مقدار value ای را که میگیریم که شامل reward این transition (که به کمک تابع getReward به آن میرسیم) و value ای که اگر در ادامه بهینه حرکت کنیم بدست می آوریم است را بدست میاوریم و این مقادیر را به صورت وزن دار (که وزن برابر است با احتمال رخ داد آن استیت (T)) با هم جمع میکنیم تا Q-value مربوط به هر استیت و اکشن بدست بیاید. همچینین اگر یک استیت هیچ اکشن ممکنی نداشته باشد با توجه به مقدار دهی اولیه q_value مقدار Q آن صفر خواهد ماند.

حال برای پیاده سازی تابع runValueIteration تنها لازم است به تعداد iteration های لازم برای هر استیت value را محاسبه کنیم که value همان ماکسییم Q-value های هر استیت با تجه به اکشن های ممکن در همان استیت می باشد

لازم به ذکر است که در هنگام محاسبه value ها کافیست برای استیت های غیر ترمینال این محاسبات را انجام دهیم زیرا در استیت ترمینال بازی تمام میشود. همچنین متغیر self.values همواره آخرین مقدار value بدست آمده برای استیت ها را در خود دارد. تابع بعدی می باشد که با توجه به q_value ها بهترین اکشن را در استیت داده شده حساب میکند برای این کار کافیست اکشنی که بیشترین Q-value را میدهد پیدا کنیم.

لازم به ذکر است که چون در استیت ترمینال بازی تمام میشود اکشنی در آن استیت نداریم.

سوال : حداقل ۳ مورد از مشکلات روش value iteration را نام برده و توضیح دهید.

- تکرار ارزش باید هر حالت را در هر تکرار لمس کند، بنابراین اگر تعداد کل حالت‌ها زیاد باشد، تکرار ارزش آسیب می‌بیند.
- کند است زیرا ما باید همه اقدامات را در هر گره در نظر بگیریم و اغلب، اقدامات زیادی وجود دارد
- "حداکثر" در یک حالت به ندرت تغییر می کند. این بدان معنی است که اندازه نسبی مقادیر Q قبل از همگرا شدن مقادیر همگرا می شوند.

• بخش ۲: تجزیه و تحلیل عبور از پل

سوال: دلیل انتخاب این مقادیر را به زبان ساده و به صورت شهودی توضیح دهید.

برای حل این سوال ما مقدار noise را تغییر میدهیم و صفر میکنیم با انجام این کار قطعیت را در انجام اکشن ها و استیت های بعدی بالا میبریم یعنی وقتی یک اکشن انجام میگیرد حتما به درستی انجام میشود در نتیجه استیت بعدی به صورت قبلی مشخص است. با انجام این کار دیگر نگران value های منفی استیت های دوطرف پل نخواهیم بود و با حرکت به سمت راست ریسک گرفتن امتیاز منفی نداریم پس تا وقتی که discount باعث نشود که ارزش بدست آمده از سمت راست پل کمتر از ارش بدست آمده از سمت چپ شود بهینه آن است که به راست حرکت کنیم و با توجه به مقدار تخفیف و فاصله استیت های روی پل در همه این حالات بهینه حرکت به سمت سمت راست است زیرا انقدر ارزش زیادی دارد که با احتساب تخفیف همچنان value نهایی از حرکت به سمت چپ بهنر باشد.

سوال:چگونه به این نتیجه رسیدیم که درحل مسائل با روش ValueIteration از Discount Factor استفاده میکنیم و این فاکتور چه کمکی به ما میکند؟ (میتوانید از نتایجی که در این بخش گرفتید نیز کمک بگیرید و به کمک آنها توضیح دهید).

Discount Factor، که معمولاً به عنوان γ (گاما) نشان داده می شود، مقداری بین ۰ و ۱ است. این نشان دهنده اهمیت پاداش های آینده نسبت به پاداش های فوری است. با Discount پاداش‌های آتی، ترجیح می‌دهیم پاداش‌های فوری را نسبت به پاداش‌های دیرتر ترجیح دهیم، که نشان‌دهنده این واقعیت است که ارزش پاداش‌های آتی ذاتاً نامشخص است.

Discount Factor بر همگرایی و رفتار الگوریتم تأثیر می گذارد. وقتی γ به ۱ نزدیک‌تر است، الگوریتم توجه بیشتری به پاداش‌های آینده می‌کند که به طور بالقوه منجر به برنامه‌ریزی طولانی‌مدت می‌شود. از سوی دیگر، زمانی که γ به ۰ نزدیک‌تر است، الگوریتم بیشتر بر پاداش‌های فوری تمرکز می‌کند که منجر به رویکرد تصمیم‌گیری نزدیک‌بینانه می‌شود.

Discount Factor در حل مشکلات با روش تکرار ارزش ضروری است زیرا به ما امکان می دهد بین پاداش های فوری و آینده تعادل برقرار کنیم. با گنجاندن عامل تخفیف، می‌توانیم سناریوهایی را مدیریت کنیم که در آن تاخیر زمانی قابل توجهی بین انجام یک اقدام و مشاهده پیامدهای آن وجود دارد و می‌توانیم تصمیم‌گیری را بر اساس اهداف بلندمدت و خطرات احتمالی بهینه کنیم.

Discount Factor اساساً تعیین می‌کند که عوامل یادگیری تقویتی چقدر به پاداش‌ها در آینده دور نسبت به پاداش‌های آینده نزدیک اهمیت می‌دهند. اگر $\gamma = 0$، عامل کاملاً نزدیک بینی خواهد بود و فقط در مورد اقداماتی که پاداش فوری ایجاد می کند یاد می گیرد. اگر $\gamma = 1$ ، عامل هر یک از اقدامات خود را بر اساس مجموع کل پاداش های آینده خود ارزیابی می کند.

سوال: راه حل Value Iteration راهی زمانبر است که باید برای هر State همه حالتها را بسنجیم و گاهی ناگزیر به انجام آن هستیم. اما در این مسئله به خصوص آیا راه حل سادهتری نسبت به ValueIteration وجود دارد که تعداد حالتهای بررسی شده را کاهش دهد؟این روش را نام ببرید و توضیح دهید و سپس آن ها را از نظر پیچیدگی زمانی مقایسه کنید.

در موارد خاص، زمانی که حالت های یک فرآیند تصمیم گیری مارکوف (MDP) بزرگ یا پیوسته است، روش ت Value Iteration در واقع می تواند از نظر محاسباتی گران باشد. یک رویکرد جایگزین که تعداد حالت های بررسی شده را کاهش می دهد، Q-Learning نامیده می شود.

Q-Learning یک الگوریتم یادگیری تقویتی بدون مدل است که یک تابع ارزش عمل (Q-function) را مستقیماً بدون تخمین صریح احتمالات انتقال حالت یاد می‌گیرد. Q-Learning به جای تکرار بر روی همه حالت ها مانند تکرار ارزش، بر یادگیری مقادیر جفت حالت-عمل (Q-values) تمرکز می کند. این امر آن را به ویژه در مسائل مربوط به فضاهای حالت بزرگ یا پیوسته که در آن نمایش و تکرار صریح بر روی همه حالت ها غیرعملی است، مفید است.

توضیح مختصری از الگوریتم Q-Learning:

۱- مقداردهی اولیه: مقادیر Q را برای همه جفت‌های حالت-اقدام اولیه کنید.

۲- اکتشاف در مقابل بهره‌برداری: اقدامی را برای انجام در وضعیت فعلی با استفاده از استراتژی اکتشاف-استثمار (به عنوان مثال، epsilon-greedy) انتخاب کنید. این تعادل بین کاوش اقدامات جدید و بهره برداری از بهترین اقدامات در حال حاضر شناخته شده است.

۳- به روز رسانی Q-values: پاداش دریافتی و وضعیت جدید رسیده را مشاهده کنید. Q-values جفت وضعیت فعلی را با استفاده از معادله به روز رسانی Q-Learning به روز کنید:

$$Q(s, a) = Q(s, a) + \alpha * (r + \gamma * \max(Q(s', a')) - Q(s, a))$$

که در آن $Q(s, a)$ مقدار Q جفت حالت-عمل (s, a)، α نرخ یادگیری، r پاداش دریافتی فوری، γ ضریب تخفیف، s' حالت جدید است، و a' عملی است که مقدار Q را در حالت جدید به حداکثر می رساند.

۴- مراحل ۲ و ۳ را تا زمان همگرایی یا تعداد معینی تکرار تکرار کنید.

Q-Learning با یادگیری مستقیم مقادیر Q و به روز رسانی آنها بر اساس پاداش های مشاهده شده و انتقال حالت، بار محاسباتی را در مقایسه با Value Iteration کاهش می دهد. از نیاز به تکرار صریح بر روی همه حالت ها جلوگیری می کند و آن را برای فضاهای حالت بزرگ یا پیوسته مناسب تر می کند.

از نظر پیچیدگی زمانی، Value Iteration دارای پیچیدگی زمانی $O(S * A)$ است، که در آن S تعداد حالات و A تعداد اقدامات است. این به این دلیل است که تکرار ارزش مستلزم تکرار بر روی همه حالات و اقدامات در هر تکرار تا زمان همگرایی است.

از سوی دیگر، پیچیدگی زمانی Q-Learning معمولاً کمتر از Value Iteration است، زیرا فقط مقادیر Q را برای جفت‌های حالت-عمل مشاهده شده به‌روزرسانی می‌کند. با این حال، پیچیدگی زمانی دقیق Q-Learning به عواملی مانند استراتژی اکتشاف، اندازه فضای حالت، و تعداد تکرارهایی که برای همگرایی انجام می شود بستگی دارد. به طور کلی، Q-Learning هزینه محاسباتی کمتری نسبت به Value Iteration برای مشکلات فضاهای حالت بزرگ یا پیوسته دارد.

- **بخش ۳: سیاست‌ها**

سوال: دلیل انتخاب خود برای هریک از مقادیر پارامترهای مذکور را در هر سیاست بیان کنید.

- خروجی نزدیک را ترجیح دهید (۱+)، ریسک صخره را بپذیرید (۱۰-)

برای اینکه به سمت خروجی نزدیک برویم باید مقدار تخفیف کم باشد همچنین وقتی ریسک صخره هارا انجام میدهد که نویز کمتری داشته باشد.

```
def question3a():
    answerDiscount = 0.1
    answerNoise = 0
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

- خروجی نزدیک را ترجیح دهید (۱+)، اما از صخره اجتناب کنید (۱۰-)

برا اینکه ریسک صخره را نکند باید مقدار نویز بیشتر از قبل باشد. همچنین باید مقدار تخفیف و جریمه هر حرکت را بیشتر کنیم که با امتحان کردن مقادیر مختلف بدست آوردیم.

```
def question3b():
    answerDiscount = 0.6
    answerNoise = 0.4
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

- خروجی دور را ترجیح دهید(۱۰+)، ریسک صخره را بپذیرید(۱۰-)

این همان حالت اول است با این تفاوت که به جای رفتن به سمت خروجی نزدیک باید با افزایش تخفیف آنرا به سمت خروجی دورتر بکشانیم.

```
def question3c():
    answerDiscount = 1
    answerNoise = 0
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

- خروجی دور را ترجیح دهید (۱۰+)، اما از صخره اجتناب کنید (۱۰-)

```
def question3d():
    answerDiscount = 1
    answerNoise = 0.4
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

این همان حالت دوم است با این تفاوت که به جای رفتن به سمت خروجی نزدیک باید با افزایش تخفیف آنرا به سمت خروجی دورتر بکشانیم.

- از هر دو خروجی و صخره اجتناب کنید (بنابراین اجرای آن هرگز نباید پایان یابد)

```
def question3e():
    answerDiscount = 0.1
    answerNoise = 0
    answerLivingReward = 10
    return answerDiscount, answerNoise, answerLivingReward
```

برا اینکه از خروجی ها و صخره دور شود، مقدار پاداش به ازای هر حرکت را مقدار مثبت بزرگی گذاشته شود تا عامل به جای اینکه بخواهد خود را به خروجی ها برساند مکررا در نقشه حرکت کند.

سوال: در سیاست پنجم، همانطور که مشاهده کردید در یک لوپ بینهایت میافتادیم و عامل علاقهای به پایان بازی نداشت. برای حل این مشکل چه راه حل هایی به نظرتان میرسد. آنها را توضیح دهید.

همان طور که در سوال قبل توضیح داده شد اگر به ازای هر حرکت مقدار پاداش بزرگی(۱۰) بگذاریم و باتوجه به اینکه پاداش خروجی حداکثر ۱۰ است عامل علاقه ای به پایان بازی ندارد و دوست دارد در نقشه حرکت کند.

پس میتوانیم برای حل مشکل راه حل های زیر پیشنهاد میشود:

برای حل این مشکل، چندین رویکرد وجود دارد که می توان انجام داد:

- تغییر تابع پاداش: راه حل دیگر این است که تابع پاداش را تغییر دهید تا عامل را تشویق کند تا به سمت هدف پیشرفت کند. به عنوان مثال، تابع پاداش می تواند برای هر قدم برداشته شده به سمت هدف، پاداش کوچک و برای رسیدن به هدف، پاداش بزرگتری ارائه دهد. این می تواند عامل را تشویق کند تا اقدامات جدیدی را که ممکن است به هدف منجر شود، کشف کند.

- یک محدودیت زمانی معرفی کنیم: راه حل دیگر این است که یک محدودیت زمانی برای بازی معرفی کنید. اگر عامل در مدت زمان معینی به هدف نرسد، بازی را می توان خاتمه داد و عامل را جریمه کرد. این می تواند عامل را تشویق کند تا به جای گیر کردن در یک حلقه نامحدود، روی پیشرفت به سمت هدف تمرکز کند.

- از الگوریتم متفاوتی استفاده کنید: در نهایت، اگر راه حل های بالا جواب نداد، ممکن است به طور کلی از الگوریتم دیگری استفاده کنید. به عنوان مثال، اگر الگوریتم فعلی مبتنی بر تکرار ارزش باشد، الگوریتم متفاوتی مانند یادگیری Q یا جستجوی درخت مونت کارلو ممکن است در کاوش فضای حالت و اجتناب از حلقه‌های بی‌نهایت مؤثرتر باشد.

به طور کلی، راه حل یک حلقه بی نهایت در یک سیاست به مشکل خاص و رفتار عامل بستگی دارد. ممکن است ترکیبی از راه حل های فوق برای حل موثر مشکل و اطمینان از پیشرفت عامل به سمت هدف ضروری باشد.

سوال: آیا استفاده از الگوریتم تکرار ارزش تحت هر شرایطی به همگرایی میانجامد؟

خیر. مثلا اگر reward از مقدار ارزش ترمینا استیت ها بیشتر باشد بازی هر گز پایان نمیابد و ارزش های نیز واگرا میشوند.

• بخش ۴: تکرار ارزش ناهمزمان

سوال: روش های بروزرسانی‌ای که در بخش اول (بروزرسانی با استفاده از batch) و در این بخش (بروزرسانی به صورت تکی) پیاده کرده اید را با یکدیگر مقایسه کنید. (یک نکته مثبت و یک نکته منفی برای هرکدام)

این بار برخلاف بخش یک در هر iteration به جای اینکه ارزش تمامی استیت ها را آپدیت کنیم فقط ارزش یک استیت را تغییر میدهیم و این استیت در هر iteration نوبتی تغییر میکند. فرآیند آپدیت کردن ارزش نیز به صورت قبل است که برای استیت هایی که نهایی نباشند و اکشن ممکنی داشته باشند q-value به ازای تمامی اکشن های ممکن را محاسبه و ماکسیمم آن را به عنوان ارزش جدید در نظر میگیریم.

روش دسته ای سریع تر همگرا میشود یعنی در تعداد iteration کمتری چون از داده های بیشتری استفاده میکند اما محاسبات بیشتری دارد. روش دوم هزینه محاسباتی کمتری در هر iteration دارد اما به تعداد بیشتری iteration برای رسیدن به دقت مشابه به روش اول دارد.

مزایای بروزرسانی با استفاده از batch:

همگرایی تضمین شده: با توجه به زمان کافی و منابع محاسباتی، تکرار ارزش دسته ای تضمین می شود که به تابع مقدار بهینه یک MDP همگرا شود.

سادگی: تکرار مقدار دسته ای یک الگوریتم ساده و سرراست است که پیاده سازی و درک آن آسان است.

معایب بروزرسانی با استفاده از batch:

از نظر محاسباتی گران است: تکرار مقدار دسته ای می تواند از نظر محاسباتی گران باشد، به خصوص برای فضاهای حالت بزرگ. الگوریتم باید مقدار هر حالت را در هر تکرار به روز کند، که می تواند زمان بر و حافظه فشرده باشد.

نرخ همگرایی: تکرار مقدار دسته ای ممکن است به آرامی همگرا شود، به ویژه برای MDP هایی با فضای حالت بزرگ یا دینامیک پیچیده. الگوریتم ممکن است برای رسیدن به همگرایی به تکرارهای زیادی نیاز داشته باشد.

الزامات حافظه: تکرار مقدار دسته ای نیاز به ذخیره مقدار هر حالت در حافظه دارد، که می تواند برای MDP هایی با فضای حالت بزرگ یک چالش باشد.

مزایای تکرار ارزش ناهمزمان:

همگرایی سریعتر: تکرار مقدار ناهمزمان می تواند سریعتر از تکرار مقدار دسته ای همگرا شود زیرا زیر مجموعه ای از فضای حالت را در هر تکرار به روز می کند. این رویکرد به الگوریتم اجازه می دهد تا بر روی حالت های مربوطه تمرکز کند و از به روز رسانی های غیر ضروری جلوگیری کند.

نیازهای حافظه کمتر: تکرار مقدار ناهمزمان نیازی به ذخیره مقدار هر حالت در حافظه ندارد، که می تواند نیازهای حافظه را در مقایسه با تکرار مقدار دسته ای کاهش دهد.

مقیاس پذیری: تکرار مقدار ناهمزمان می تواند مقیاس پذیرتر از تکرار مقدار دسته ای باشد، به خصوص برای MDP هایی با فضای حالت بزرگ.

معایب تکرار ارزش ناهمزمان:

عدم تضمین همگرایی: تکرار مقدار ناهمزمان ممکن است به تابع مقدار بهینه همگرا نشود، به خصوص اگر زیرمجموعه فضای حالت که به روز می شود نماینده کل فضای حالت نباشد. این می تواند منجر به سیاست های غیربهینه و کاهش عملکرد شود.

پیچیدگی: تکرار مقدار ناهمزمان پیچیده‌تر از تکرار مقدار دسته‌ای است، زیرا برای به‌روزرسانی در هر تکرار نیاز به انتخاب دقیق زیر مجموعه فضای حالت دارد. این ممکن است نیاز به تنظیم و آزمایش اضافی برای دستیابی به نتایج خوب داشته باشد.

حساسیت به شرایط اولیه: تکرار مقدار ناهمزمان می تواند به شرایط اولیه تابع مقدار حساس باشد که می تواند بر نرخ همگرایی و کیفیت راه حل تأثیر بگذارد.

- **بخش ۵: تکرار ارزش اولویت بندی شده**

توضیح پیاده سازی:

ابتدا تمام حالت های پسین را پیدا میکنیم. برای این کار ابتدا یک دیکشنری که key ای آن استیت ها و value آن مجموعه های خالی هستند ایجاد میکنیم سپس به ازای هر استیت به کمک تابع getTransitionStatesAndProbs تمام حالت های بعدی ممکن به ازای تمام اکشن های ممکن را به همراه احتمال رفتن به آن استیت بدست می آوریم و اگر این احتمال بیشتر از صفر بود و استیت بعدی ما استیت نهایی نبود استیت اولیه را به عنوان استیت پسین استیت بعدی اضافه میکنیم. لازم به ذکر است که استیت های پسین هر استیت را در یک مجموعه میریزیم تا تکراری نشوند.

• بخش ۶: یادگیری Q

توضیح پیاده سازی:

ابتدا یک Counter برای ذخیره مقدار q-value ها تعریف میکنیم. برای پیاده سازی getQValue کافیست که از این Counter تعریف شده q-value مربوط به استیت و اکشن داده شده در ورودی را باز گردانیم.

در تابع computeValueFromQValues نیز مانند بخش های قبل کافیست از بین q-value های متناظر با اکشن های ممکن در استیت فعلی بزرگترین q-value را به عنوان ارزش این استیت انتخاب کنیم.

برای تابع بعدی یعنی computeActionFromQValues ابتدا اکشن های ممکن برای استیت فعلی را پیدا میکنیم سپس بررسی میکنیم اگر اکشن ممکنی نداشتیم ینی در استیت نهایی بودیم None باز میگردانیم در غیر این صورت value استیت فعلی را به کمک تابع computeValueFromQValues پیدا میکنیم و اکشن هایی از بین اکشن های مجاز که این مقدار را به ما میدهند را در لیست اکشن های برگزیده قرار میدهیم و در نهایت یکی از این اکشن ها را به صورت رندوم باز میگردانیم.

تابع آخر تابع update می باشد که مقدار Q(s,a) را با توجه به فرمول های زیر که در کلاس دیدیم آپدیت میکند:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)[sample]$$

سوال: توضیح دهید که اگر مقدار Q برای اقداماتی که عامل قبلا ندیده، بسیار کم یا بسیار زیاد باشد چه اتفاقی میافتد.

در این صورت تعداد تجربه بیشتری برای اصلاح آنها نیاز می باشد همچنین اگر مقدار Q-value ها بسیار کم باشد با احتمال کمتری انتخاب و سخت تر آپدیت میشوند و اگر بسیار زیاد هم باشد در تجربه های اول زیاد به آن ها برمیخوریم و به مقادیر اشتباه کاذب میرسیم.

سوال: بیان کنید Q-learning یک الگوریتم Off-policy است یا On-policy؟ Value-based است یا Policy-based ؟ توضیح دهید.

Q-learning یک الگوریتم Off-policy و Value-based برای یادگیری تقویتی است.

Off-policy به این معنی است که الگوریتم تابع Q-value بهینه را با استفاده از policy رفتاری متفاوت از policy هدف می آموزد. به طور خاص، Q-learning تابع Q-value را با استفاده از حداکثر Q-value بر روی تمام اقدامات ممکن در حالت بعدی، بدون توجه به اقدام انجام شده توسط policy رفتار، به روز می کند. این امر به یادگیری Q اجازه می دهد تا یک policy بهینه را بدون تأثیرپذیری از policy رفتار بیاموزد، که می تواند کمتر از حد مطلوب باشد.

Value-based به این معنی است که یادگیری Q تابع ارزش Q را مستقیماً، بدون نمایش صریح policy، یاد می گیرد. تابع Q-value نشان‌دهنده پاداش تجمعی مورد انتظار است که از یک وضعیت معین شروع می‌شود و یک اقدام معین را انجام می‌دهد و به طور مکرر با استفاده از معادله بلمن به‌روزرسانی می‌شود.

هنگامی که تابع Q-value آموخته شد، این policy را می توان با انتخاب عملکرد با بالاترین مقدار Q در هر حالت استخراج کرد.

در مقابل، الگوریتم‌های مبتنی بر policy مستقیماً policy را یاد می‌گیرند، بدون اینکه به صراحت تابع Q-value را نشان دهند. این الگوریتم‌ها سیاست را با حداکثر کردن مستقیم پاداش تجمعی مورد انتظار، معمولاً با استفاده از روش‌های مبتنی بر گرادیان، بهینه می‌کنند.

سوال: الگوریتم Q-learning از TD-Leaning استفاده میکند آن را با Monte Carlo مقایسه کنید و بیان کنید استفاده هر کدام چه مزایا و چه معایبی دارند.

Q-learning یک الگوریتم یادگیری temporal difference (TD) برای یادگیری تقویتی است، به این معنی که تابع Q-value را بر اساس تفاوت بین Q-value برآورد شده و Q-value هدف به روز می کند. مونت کارلو (MC) الگوریتم دیگری برای یادگیری تقویتی است که تابع مقدار را بر اساس بازده مشاهده شده از یک حالت تخمین می زند.

مزایا و معایب اصلی Q-learning و Monte Carlo عبارتند از:

- مزایای یادگیری Q:

یادگیری آنلاین: Q-learning تابع Q-value را به طور مکرر به‌روزرسانی می‌کند، زیرا عامل با محیط تعامل دارد، که به آن اجازه می‌دهد به صورت آنلاین در زمان واقعی بیاموزد.

بدون مدل: یادگیری Q نیازی به دانش دینامیک محیط یا مدلی از احتمالات و پاداش‌های انتقال ندارد و آن را به یک الگوریتم بدون مدل تبدیل می‌کند.

مناسب برای فضاهای حالت بزرگ: یادگیری Q می تواند برای MDP هایی با فضای حالت بزرگ استفاده شود، زیرا فقط باید تابع Q-value را برای وضعیت و عملکرد فعلی به روز کند.

- معایب یادگیری Q:

سوگیری: یادگیری Q می‌تواند با استفاده از روش‌های تقریب تابع، مانند شبکه‌های عصبی، به دلیل ماهیت راه‌اندازی یادگیری TD، سوگیری داشته باشد.

تخمین بیش از حد: یادگیری Q ممکن است عملکرد Q-value را در موقعیت‌های خاص، مانند زمانی که عامل با رویدادهای نادر یا پاداش‌های بزرگ مواجه می‌شود، بیش از حد برآورد کند.

همگرایی آهسته: یادگیری Q ممکن است به آرامی همگرا شود، به خصوص برای فضاهای حالت بزرگ یا پویایی های پیچیده.

- مزایای Monte Carlo:

Unbiased: مونت کارلو یک برآوردگر بی طرفانه از تابع مقدار است، به این معنی که وقتی با روش های تقریب تابع استفاده می شود، سوگیری کمتری نسبت به یادگیری Q دارد.

محیط‌های اپیزودیک را کنترل می‌کند: Monte Carlo را می‌توان برای محیط‌های اپیزودیک استفاده کرد، جایی که عامل تا رسیدن به حالت پایانی با محیط تعامل دارد و بازگشت برای هر قسمت مشاهده می‌شود.

خوب برای مشکلات با واریانس بالا: Monte Carlo می تواند برای مشکلات با واریانس بالا مفید باشد، جایی که پاداش دریافت شده در یک حالت بسیار متغیر است.

- معایب Monte Carlo:

واریانس بالا: تخمین های Monte Carlo می توانند واریانس بالایی داشته باشند، به خصوص زمانی که عامل با رویدادهای نادر یا پاداش های بزرگ روبرو می شود.

به قسمت‌های کامل نیاز دارد: Monte Carlo از نماینده می‌خواهد قبل از به‌روزرسانی تابع مقدار، یک قسمت کامل را تکمیل کند. این می تواند برای اپیزودهای طولانی یا محیط هایی با فضاهای حالت بزرگ ناکارآمد باشد.

برای یادگیری آنلاین نامناسب: Monte Carlo را نمی توان برای یادگیری آنلاین استفاده کرد زیرا به عامل نیاز دارد که کل قسمت را قبل از به روز رسانی تابع ارزش تکمیل کند.

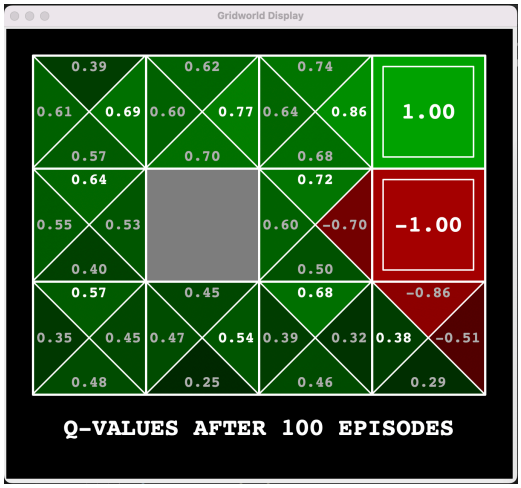
• بخش ۷: epsilon حریصانه

توضیح پیاده سازی:

ابتدا به کمک تابع flipCoin با حتمال اپسیلون به صورت رندوم انتخاب میکنیم که حرکت رندوم انجام دهیم(انتخاب یک اکشن رندوم از بین اکشن های مجاز) یا اینکه از مقادیر q-value بدست آمده تا اینجا استفاده کنیم و به کمک تابع computeActionFromQValues بهترین اکسن را بتوجه به ارزش های بدست آمده تا الان انجام دهیم.

بعد از اجرا دستور مشاهده میکنیم که:

خانه های پایین سمت راست که به مقادیر منفی بودن کمتر پیمایش و ررسی شدند چون از یک زمانی به بعد عامل جای امتیاز بیشتر را یاد میگیرد و ترجیح میدهد به آن سمت حرکت کند مگر آنکه تصادفی به سمت مخالفی برود.



سوال: هدف از استفاده از اپسیلون و به کارگیری روش اپسیلون حریصانه چیست؟

اپسیلون پارامتری است که در یادگیری تقویتی برای کنترل معاوضه اکتشاف و بهره برداری استفاده می شود. مبادله اکتشاف و بهره برداری به تصمیم گیری در مورد اینکه آیا اقدامات و حالات جدید را برای به دست آوردن اطلاعات بیشتر در مورد محیط زیست یا بهره برداری از دانش فعلی و انتخاب اقداماتی که به عنوان پاداش بالا شناخته می شوند، بررسی کنیم، اشاره دارد.

روش اپسیلون حریصانه یک استراتژی رایج برای ایجاد تعادل بین اکتشاف و بهره برداری در یادگیری تقویتی است. این روش با انتخاب عمل با بالاترین Q-value با احتمال (epsilon-1) و انتخاب یک عمل تصادفی با اپسیلون احتمال کار می کند. ارزش اپسیلون در طول زمان به تدریج کاهش می یابد تا عامل ترغیب شود تا با کسب اطلاعات بیشتر در مورد محیط، کمتر کاوش کند.

هدف از استفاده از اپسیلون و به کارگیری روش اپسیلون حریصانه، تشویق عامل به کشف اقدامات و حالات جدید برای جلوگیری از گیر افتادن در یک policy غیربهینه است. اگر عامل همیشه اقدامی را با بالاترین Q-value انتخاب کند، ممکن است اقدامات بالقوه بهتری را که هنوز کاوش نشده اند از دست بدهد. از سوی دیگر، اگر عامل همیشه یک اقدام تصادفی را انتخاب کند، ممکن است زمان را برای بررسی اقداماتی که بعید است منجر به پاداش بالا شود، تلف کند.

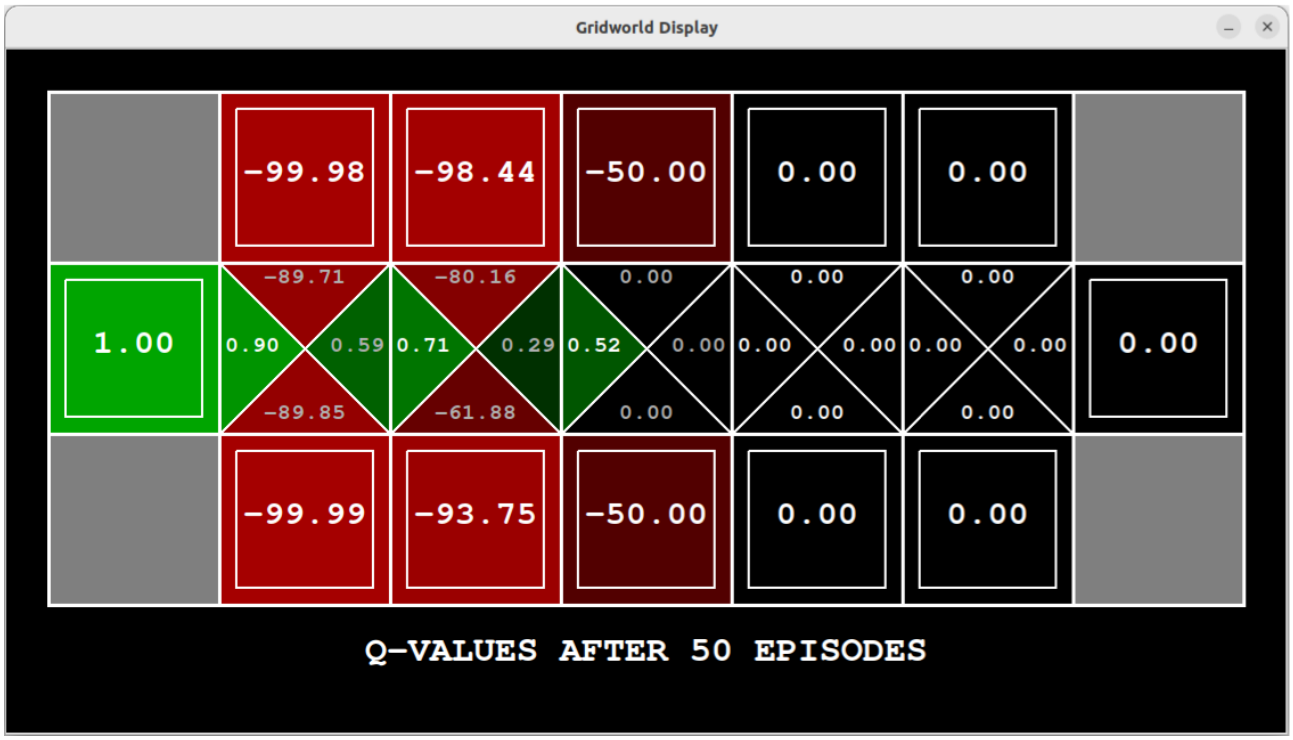
با کاهش تدریجی اپسیلون در طول زمان، عامل به دانش خود از محیط اطمینان بیشتری پیدا می کند و شروع به بهره برداری بیشتر از دانش فعلی می کند. این امر تعادل exploration-exploitation را متعادل می کند و به عامل اجازه می دهد تا یک policy بهینه را بیاموزد و در عین حال به کشف اقدامات و حالات جدید ادامه دهد.

به طور خلاصه، هدف از استفاده از اپسیلون و به کارگیری روش اپسیلون حریصانه، ایجاد تعادل بین مبادلهexploration, exploitation در یادگیری تقویتی است. اپسیلون عامل را تشویق می کند تا اقدامات و حالات جدید را بررسی کند، در حالی که روش اپسیلون حریص عملی را با بالاترین Q-value با احتمال زیاد و یک عمل تصادفی با احتمال کم انتخاب می کند. کاهش

تدریجی اپسیلون در طول زمان به عامل این امکان را می‌دهد تا بین exploration exploitation, تعادل برقرار کند و policy بهینه را بیاموزد و در عین حال به کشف اقدامات و حالات جدید ادامه دهد.

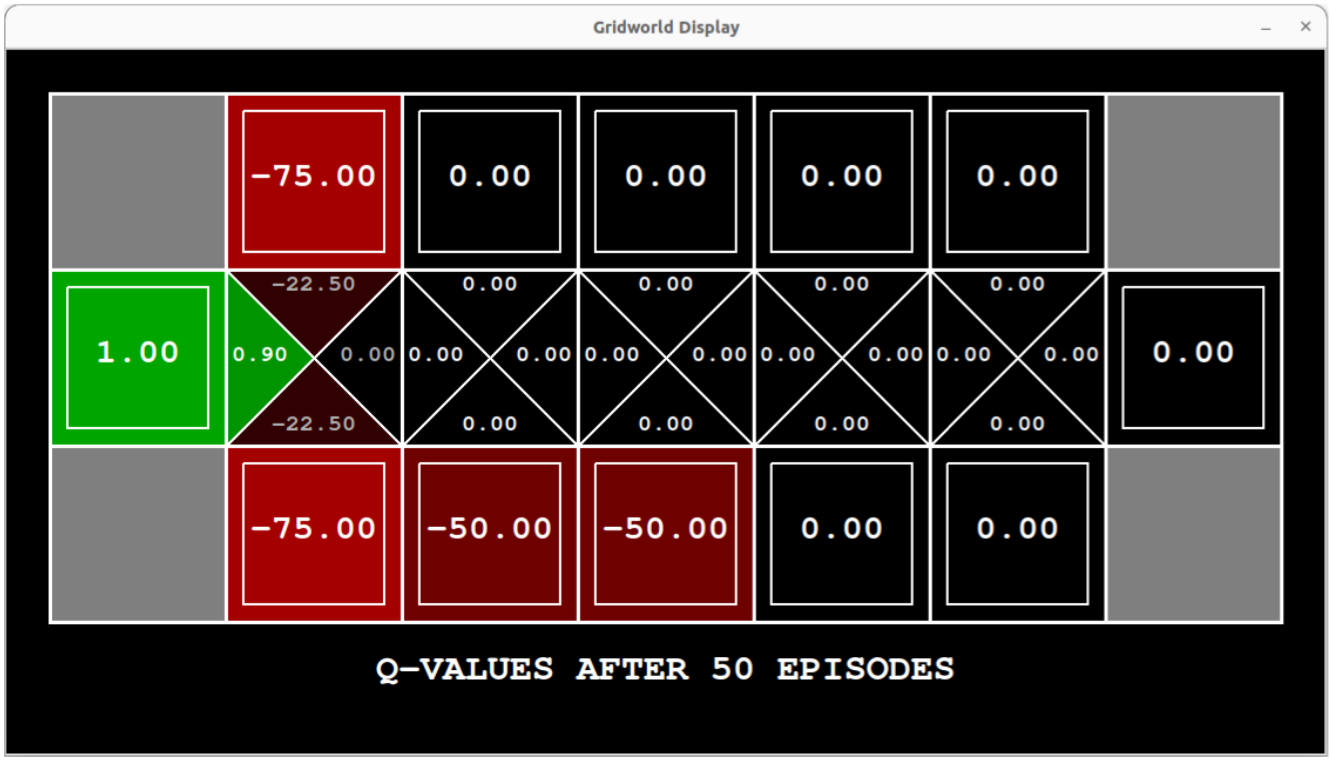
- **بخش ۸: بررسی دوباره عبور از پل**

در حالت اول که اپسیلون یک است خروجی به صورت زیر میشود و سیاست بهینه در این تعداد iteration پیدا نمیشود زیرا عامل هماره رندوم عمل میکند و به دانش بدست آمده برای بهبود روند کشف خود توجهی نمیکند:



سوال: حال، همین کار را با اپسیلون ۰ دوباره تکرار کنید. آیا مقدار اپسیلون و ضریب یادگیریای وجود دارد که با استفاده از آنها، سیاست بهینه با احتمال خیلی بالا (بیشتر ۹۹ درصد) بعد از ۵۰ بار تکرار یاد گرفته شود؟

درحالت بعدی یعنی زمانی که اپسیلون صفر باشد عامل باز هم نمیتواند سیاست بهینه را پیدا کند زیرا با چند حرکت اول استیت نهایی نزدیک به خود با ارزش ۱ را پیدا میکند و چون از هیچ شانسی در تصمیمی گیری برای امتشاف استفاده نمیکند پس از پیدا کردن استیت نهایی با ارزش ۱ همواره ترجیح میدهد به سمت آن حرکت کند و تجربه کند زیرا فقط بر اساس ارزش ها و Q-value هایی که بدست آورده عمل میکند.خروجی این حالت به صورت زیر می باشد:



- آیا اپسیلونی وجود دارد که با احتما بالا پس از iteration ۵۰ سیاست بهینه پیدا شود؟ خیر

ایده ای که در بخش دوم استفاده شد این بود که نویز را کم کنیم تا حرکت تصادفی عامل به کمترین مقدار خود برسد.اما در اینجا با توجه به اینکه عامل در نیمی از حالت ها به صورت تصادفی انتخاب میکند عملا با آن ایده متناقض است چون مثل این است که نویز زیادی در انتخاب اکشن عامل وجود دارد. پس مقدار Not Possible را برمیگردانیم.

سوال: به صورت ساده و شهودی توضیح دهید که با کم یا زیاد کردن مقدار epsilon روند یادگیری عامل چگونه تغییر میکند.

همان طور که در بالا دیدیم با زیاد شدن اپسیلون در روند اکتشاف حرکت های تصادفی زیادی داریم و به اطلاعات بدست آمده کمتر رجوع میکنیم پس دیتر تر سیاست بهینه را پیدا میکنیم اما بیشتر میگردیم و فضای بیشتری را بررسی میکنیم که ممکن است جواب های بهتری از نردیکترین جواب پیدا کنیم. اما با کم کردن اپسیلون اعتماد ما به دانش بدست آمده زیاد میشود و تجربه تصادفی کمتر میشود پس زمانی که امتایز مثبتی را میبینیم تمایل داریم در تجربه های بعدی به همان سمت بیشتر حرکت کنیم و تجربه جدید تصادفی در محل های دیده نشده کمتر شود انگار به نوعی حریصانه عمل میکنیم و با احتمال کمتری به جستجوی مکان های جدید تر که ممکن است ارزش خوبی هم در آن های نهفته باشد میپردازیم.

• بخش ۹: پک من و Q-Learning

سوال: تغییرات و فعالیتهایی که در این بخش انجام دادهاید را توضیح دهید.

با توجه به پیاده سازی های قبلی این بخش بدون تغییر خاصی توانست نمره لازم را بدست بیاورد.

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l
smallGrid
```

```
Training Done (turning off epsilon and alpha)
-----------------------------------------
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Average Score: 499.0
Scores:        503.0, 495.0, 503.0, 503.0, 495.0, 499.0, 503.0, 495.0, 495.0, 49
9.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
(base) mahdieh@mahdiehs-MacBook-Pro AI_P3_SP23 %
```

```
python autograder.py -q q9
```

```
*** PASS: test_cases/q9/grade-agent.test (1 of 1 points)
***     Grading agent using command:  python pacman.py -p PacmanQAgent -x 2000 -
n 2100 -l smallGrid -q -f --fixRandomSeed
***     100 wins (1 of 1 points)
***         Grading scheme:
***          < 70:  0 points
***         >= 70:  1 points

### Question q9: 1/1 ###


Finished at 21:05:10

Provisional grades
==================
Question q9: 1/1
------------------
Total: 1/1

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a
numTraining=10
```

```
(base) mahdieh@mahdiehs-MacBook-Pro AI_P3_SP23 % python pacman.py -p PacmanQAgent -n
10 -l smallGrid -a numTraining=10
Beginning 10 episodes of Training
Pacman died! Score: -506
Pacman died! Score: -507
Pacman died! Score: -516
Pacman died! Score: -511
Pacman died! Score: -509
Pacman died! Score: -505
Pacman died! Score: -508
Pacman died! Score: -508
Pacman died! Score: -508
Pacman died! Score: -514
Training Done (turning off epsilon and alpha)
-----------------------------------------
Average Score: -509.2
Scores:        -506.0, -507.0, -516.0, -511.0, -509.0, -505.0, -508.0, -508.0, -508.0
, -514.0
Win Rate:      0/10 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
```

• بخش ۱۰: یادگیری تقریبی Q

برای پیاده سازی تابع update از دو فرمول زیر استفاده میکنیم:

$$w_i \leftarrow w_i + \alpha \,.\, difference \,.\, f_i(s,a)$$
$$difference = (r + \gamma max_{a'} Q(s',a')) - Q(s,a)$$

ابتدا feature ها که همان f در فرمول بالا باشند را با توجه به استیت و اکشن فعلی پیدا میکنیم سپس مقدار difference را پیدا میکنیم و برای هر یک از درایه های weights یعنی برای هر یک از وزن های با توجه به مقدار difference بدست آمده آن ها را آپدیت میکنیم.

تابع دیگر difference است که مقدار q-value را از روی وزن و feature های مربوط به استیت و اکشن داده شده با توجه به فرمول زیر بدست می آورد.

$$Q(s,a) = \sum_{i=1}^{n} f_i(s,a) w_i$$

```
python autograder.py -q q10
```

```
Question q10
============

*** PASS: test_cases/q10/1-tinygrid.test
*** PASS: test_cases/q10/2-tinygrid-noisy.test
*** PASS: test_cases/q10/3-bridge.test
*** PASS: test_cases/q10/4-discountgrid.test
*** PASS: test_cases/q10/5-coord-extractor.test

### Question q10: 3/3 ###


Finished at 22:21:04

Provisional grades
==================
Question q10: 3/3
------------------
Total: 3/3

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

سوال: درباره Deep Q-leaning تحقیق کنید و بیان کنید در چه مواردی این الگوریتم به جای الگوریتم Q-learning عادی استفاده میشود. هر کدام از این الگوریتمها، Approximate Q learning- و Deep Q-learning چه مشکلی را از Q-learning حل میکنند.

Deep Q-learning گونه ای از Q-learning است که از یک شبکه عصبی عمیق برای تقریب تابع Q-value استفاده می کند. شبکه عصبی حالت را به عنوان ورودی می گیرد و برای هر اقدام ممکن Q-value را خروجی می کند. یادگیری Q عمیق با استفاده از تقریب تابع، محدودیت‌های یادگیری Q در مدیریت فضاهای حالت با ابعاد بالا را برطرف می‌کند.

یادگیری عمیق Q معمولاً زمانی استفاده می شود که فضای حالت بزرگ باشد و استفاده از تابع Q-value جدولی غیرممکن باشد. به عنوان مثال، در محیط های مبتنی بر تصویر مانند بازی های آتاری، فضای حالت شامل مقادیر پیکسل صفحه نمایش است که می تواند ابعاد بالایی داشته باشد. در چنین مواردی، Deep Q-learning را می توان برای تقریب تابع Q-value با استفاده از یک شبکه عصبی عمیق، که می تواند ورودی های با ابعاد بالا را مدیریت کند، استفاده کرد.

از سوی دیگر، یادگیری Q تقریبی یک اصطلاح کلی است که به هر نوع یادگیری Q اشاره می کند که از تقریب تابع برای تخمین تابع Q-value استفاده می کند. این می تواند شامل تقریب تابع خطی یا روش های دیگر مانند روش های مبتنی بر هسته یا درخت های تصمیم باشد. یادگیری تقریبی Q زمانی استفاده می‌شود که فضای حالت برای استفاده از تابع Q-value جدولی بسیار بزرگ باشد، اما آنقدر بزرگ نیست که نیاز به استفاده از شبکه‌های عصبی عمیق داشته باشد.

هدف هر دو یادگیری Q تقریبی و یادگیری عمیق Q حل مسئله مقیاس پذیری در یادگیری Q است، که زمانی رخ می دهد که فضای حالت برای نمایش تابع Q-value با استفاده از روش جدولی بسیار بزرگ باشد. با استفاده از تقریب تابع، یادگیری تقریبی Q و یادگیری عمیق Q می توانند تابع Q-value را با استفاده از تعداد کمتری از پارامترها تخمین بزنند، که می تواند فضاهای حالت با ابعاد بالا را کارآمدتر از یادگیری Q جدولی اداره کند.

مزیت اصلی Deep Q-Learning نسبت به Approximate Q-Learning توانایی آن در مدیریت موثرتر فضاهای حالت پیچیده و با ابعاد بالا است. Deep Q-learning می تواند یاد بگیرد که فضای حالت را به صورت فشرده و معنی دار نشان دهد و به آن اجازه تعمیم به حالت های جدیدی را می دهد که در طول آموزش دیده نشده اند. این باعث می‌شود که Deep Q-Learning برای محیط‌های مبتنی بر تصویر مناسب باشد، جایی که فضای حالت با یک تصویر یا مجموعه‌ای از تصاویر نشان داده می‌شود.

با این حال، Deep Q-Learning همچنین دارای محدودیت هایی است، از جمله تمایل به بیش از حد برآورد کردن مقادیر Q در موقعیت های خاص، مانند زمانی که عامل با رویدادهای نادر یا پاداش های بزرگ روبرو می شود. این می تواند منجر به سیاست های غیربهینه و کاهش عملکرد شود. علاوه بر این، Deep Q-learning می تواند از نظر محاسباتی گران باشد، به خصوص در هنگام آموزش شبکه های عصبی بزرگ.

به طور خلاصه، Deep Q-Learning نوعی از یادگیری Q است که از یک شبکه عصبی عمیق برای تقریب تابع Q-value استفاده می کند و زمانی استفاده می شود که فضای حالت بزرگ و پیچیده باشد. یادگیری Q تقریبی یک اصطلاح کلی است که به هر گونه ای از یادگیری Q اشاره دارد که از تقریب تابع برای تخمین تابع Q-value استفاده می کند و زمانی استفاده می شود که فضای حالت برای استفاده از یک تابع Q-value جدولی بسیار بزرگ باشد، اما نه به اندازه ای بزرگ است که نیاز به استفاده از شبکه های عصبی عمیق داشته باشد. هدف هر دو یادگیری Q تقریبی و یادگیری عمیق Q حل مسئله مقیاس پذیری در یادگیری Q و بهبود کارایی یادگیری در فضاهای حالت با ابعاد بالا است.