سوال : حداقل ۳ مورد از مشکلات روش Value Iteration را نام برده و توضیح دهید.

از مشکلات آن میتوان به:

1.حافظه مصرفی بالا Value Iteration :برای حل مسئله نیاز به نگهداری تابع ارزیابی برای تمام حالت‌ها دارد. این ممکن است در مسائل با فضای حالت بزرگ منجر به مصرف بیش از حد حافظه شود. این امر محدودیت‌هایی را در استفاده از این الگوریتم در مسائل عملی ایجاد می‌کند . 2.مشکل تجزیه‌پذیری Value Iteration (Decomposability): فرض می‌کند که عملکرد برای هر حالت فقط به عملکرد حالت بعدی وابسته است. این فرضیه تجزیه‌پذیری را ایجاد می‌کند. در برخی از مسائل، این فرضیه ممکن است برقرار نباشد و تعامل بین حالت‌ها منجر به پیچیدگی بیشتر مسئله بشود 3سرعت همگرایی Value Iteration :به طور تئوری به جواب بهینه می‌رسد، اما سرعت همگرایی آن می‌تواند بسیار کند باشد. الگوریتم باید تا زمانی که تابع ارزش به تغییرات قابل توجهی نرسد، تکرار شود. این ممکن است به معنی نیاز به تعداد بالا از تکرارها و محاسبات باشد تا به همگرایی برسیم. این امر معمولاً در مسائل با ابعاد بزرگ، مانند مسائل با فضای حالت بزرگ، به چالش کشیده می‌شود . اشاره کرد

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

۲- برای بهبود سیاست و تشویق عامل به عبور از پل، می‌توانیم مقدار تخفیف یا نویز را تغییر دهیم. با صفر کردن مقدار نویز، عامل بهترین تلاش را برای عبور از پل انجام می‌دهد و مسئله به حالت قطعی تبدیل می‌شود.

سوال: چگونه به این نتیجه رسیدیم که در حل مسائل با روش Value Iteration 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).

در روش Value Iteration درMDP ، از Discount Factor برای مدل‌کردن تأثیر آینده در ارزیابی و تخمین ارزش یک وضعیت استفاده می‌شود. این فاکتور با یک عدد بین صفر و یک نمایش داده می‌شود و نشان می‌دهد که میزان تمایل ما برای دستیابی زودتر به نتیجه چقدر است .با استفاده از Discount Factor، می‌توانیم تأثیر تعویض بین پاداش فوری و پاداش آینده را کنترل کنیم. وقتی که مقدار Discount Factorبرابر با یک است، تمامی پاداش‌ها در آینده با اهمیت یکسان در نظر گرفته می‌شوند و هیچ گونه تخفیفی اعمال نمی‌شود. اما اگر Discount Factor به شدت کوچک شود)نزدیک به صفر)، تنها پاداش‌های فوری در نظر گرفته می‌شوند و پاداش‌های آینده کمتر اهمیت دارند. این باعث می‌شود الگوریتم به سرعت به جواب همگرا شود .با تنظیم مناسب Discount Factor، الگوریتم Value Iteration قادر است تخمین بهتری از ارزش وضعیت‌ها ارائه دهد و به طور کلی عملکرد تصمیم‌گیری را از طریق مدل‌کردن تأثیر آینده بهبود بخشیده و به سرعت همگرا می‌شود .خلاصه: در الگوریتم Value Iteration درMDP ، از Discount Factor برای تأثیر آینده در ارزیابی و تخمین ارزش استفاده می‌شود. با تنظیم مناسب این فاکتور، الگوریتم می‌تواند بهترین تخمین از ارزش وضعیت‌ها را ارائه دهد و عملکرد تصمیم‌گیری را بهبود بخشیده و به سرعت همگرا شود.

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 Value Iteration 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.

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

الف) هدف در این سیاست، انتخاب خروجی نزدیک به +۱ و پذیرش ریسک دریافت پاداش منفی است. با تنظیم تخفیف به ۰/۱، عامل بیشتر به پاداش‌های فعلی توجه می‌کند و پاداش‌های آینده را کمتر در نظر می‌گیرد. عامل همچنین بدون تلنگرهای تصادفی عمل می‌کند و تمایلی به کاوش مسیرهای جدید ندارد. پاداش زندگی نیز برای تشویق پایبندی به سیاست انتخاب شده است.

ب)هدف در این سیاست نیز انتخاب خروجی نزدیک به +۱ است، اما با تنظیم نویز به ۰/۱، عامل به تلنگرهای تصادفی بیشتری پاسخ می‌دهد و احتمال بررسی مسیرهای جدید بیشتر می‌شود. مقدار تخفیف نیز کمتر از ۱ است تا عامل به پاداش‌های فعلی بیشتر توجه کند. هدف پاداش زندگی نیز تشویق عامل به پایبندی به سیاست است.

ج)هدف در این سیاست همچنان انتخاب خروجی نزدیک به +۱ است و عامل بدون تلنگرهای تصادفی عمل می‌کند. با تنظیم تخفیف به ۱، عامل به صورت کاملاً بی‌تفاوت نسبت به پاداش‌های آینده عمل می‌کند و تنها به پاداش فعلی توجه می‌کند. هدف پاداش زندگی نیز تشویق پایبندی به سیاست انتخاب شده است.

د)هدف در این سیاست ترجیح دادن خروجی دور (+۱۰) است. با تنظیم نویز به ۰/۱، عامل به تلنگرهای تصادفی بیشتری پاسخ می‌دهد و احتمال بررسی مسیرهای جدید بیشتر می‌شود. مقدار تخفیف را نیز برابر ۱ قرار می‌دهیم تا عامل به صورت کاملاً بی‌تفاوت نسبت به پاداش‌های آینده عمل کند و فقط به پاداش فعلی توجه کند. هدف پاداش زندگی نیز تشویق پایبندی به سیاست انتخاب شده است.

ه) هدف در این سیاست همچنان ترجیح دادن خروجی دور (+۱۰) است، اما با تنظیم نویز به ۰/۱. عامل به تلنگرهای تصادفی بیشتری پاسخ می‌دهد و احتمال بررسی مسیرهای جدید بیشتر می‌شود. مقدار تخفیف را نیز برابر ۱ قرار می‌دهیم تا عامل به صورت کاملاً بی‌تفاوت نسبت به پاداش‌های آینده عمل کند و فقط به پاداش فعلی توجه کند. اما این سیاست غیرممکن است به دلیل بزرگ بودن مقدار پاداش زندگی که عامل هرگز به پایان نمی‌رسد

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.

میتوان از روش های نظیر :

اعمال تابع جریمه به پاداش‌های منفی به منظور تشویق عامل به خروج از لوپ بینهایت و انتخاب بهترین مسیر

- تعیین حد زمانی حداکثر برای اجرای الگوریتم به منظور جلوگیری از لوپ بینهایت و return -1

- استفاده از روش‌های تقریبی مانند Q-Learning بجای الگوریتم‌های کاملاً دقیق، که امکان بهبود تجربه و یادگیری بهترین سیاست را به عامل می‌دهند..

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

الگوریتم Value Iteration، در شرایط خاصی به همگرایی می‌رسد، اما در شرایط دیگر ممکن است نتواند به همگرایی برسد. این الگوریتم برای مسائل MDP با حالت‌ها و عمل‌های متناهی قابل اجراست و تضمین بهبود یا همگرایی به سیاست بهینه را در این شرایط دارد. اما در مسائلی مانند مسائل با فضای حالت یا عمل بی‌نهایت، الگوریتم تکرار ارزش به همگرایی نمی‌رسد و به لوپ بینهایت گیر می‌کند. در این حالت‌ها، استفاده از روش‌های تقریبی مبتنی بر تقریب مانند Q-Learning معمولاً منجر به حل مسئله می‌شود. این روش‌ها با تجربه و تعامل مستقیم با محیط، به سیاست بهینه نزدیک می‌شوند بدون اینکه در لوپ بینهایت گیر کنند.

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

روش بروزرسانی با استفاده از batch در هر مرحله تمام حالت‌ها را همزمان بروزرسانی می‌کند، که اطلاعات بهتری درباره وضعیت‌های مختلف محیط را فراهم می‌کند. اما هزینه محاسباتی آن بیشتر است، زیرا نیاز به محاسبه و به‌روزرسانی تمام حالت‌ها دارد. از سمت دیگر روش بروزرسانی به صورت تکی هزینه محاسباتی کمتری دارد، زیرا فقط یک حالت در هر مرحله بروزرسانی می‌شود. اما ترتیب انتخاب حالت‌ها می‌تواند تأثیری در عملکرد الگوریتم داشته باشد و احتمال مرور کامل فضای حالت کاهش می‌یابد

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

مقدار Q-value برای یک اکشن تازه، اثر زیادی بر رفتار عامل دارد. اگر این مقدار بالا باشد، عامل به سمت اکتشاف محیط حرکت می‌کند و تلاش می‌کند اکشن‌های جدیدی انجام دهد. اما اگر مقدار آن کم باشد، عامل بیشتر به سمت استفاده بهینه میرود و اکشن‌های قبلی خود را تکرار

می‌کند. این رفتارها به دلیل این است که عامل در حالت اول به سمت اکشن‌های تازه می‌رود و در حالت دوم اکشن تازه را انجام نمی‌دهد و به جای آن، اکشن‌های قبلی را انجام می‌دهد.

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

Q-learningیک الگوریتم Off-policy و Value-based است. درOff-policy ، عامل یاد می‌گیرد که بهترین عملکرد را در محیط داشته باشد، بدون اهمیت دادن به سیاست فعلی. در Q-learning، عامل از تجربه‌هایی که از سیاست‌های مختلف به دست آمده استفاده می‌کند و به‌روزرسانی Q-value را براساس بهترین عمل در حالت بعدی انجام می‌دهد.

همچنین، Q-learningیک الگوریتم Value-based است که با به‌روزرسانی تابعQ-value ، تلاش می‌کند تا بهبودی در عملکرد خود ایجاد کند و به جواب بهینه نزدیک شود.

با اینکه Q-learning به عنوان یک الگوریتم Off-policy شناخته می‌شود، عامل قادر است از تجربه‌هایی که از سیاست‌های مختلف به دست آمده استفاده کند و در هر مرحله بهترین عمل را براساس Q-value انتخاب کند، بدون وابستگی به سیاست فعلی.

سوال: الگوریتم Q-leaning 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-learningمستقل از سیاست است و می‌تواند از تجربه‌هایی که توسط سیاست‌های مختلف به دست آمده استفاده کند. این به عامل این امکان را می‌دهد که به صورت مستقل از سیاست فعلی خود به‌روزرسانی انجام دهد و عمل بهتری را انتخاب کند. همچنین، از TD-Learning برای به‌روزرسانی تابع ارزش استفاده می‌کند، که باعث می‌شود سریع‌تر و بهینه‌تر عمل کند.اما Q-learning نیز دارای معایب است. به‌روزرسانی آن به صورت آفلاین انجام می‌شود، به این معنی که در حین اجرا ارزش‌های Q را به‌روز نمی‌کند و از عملکرد فعلی خود برای بهبود استفاده نمی‌کند. همچنین، نیازمند تعیین تابع پاداش مناسب است که عامل را به سمت هدف هدایت کند.

در مقابل، Monte Carloاز تجربه کامل تا زمان خروج از محیط برای به‌روزرسانی تابع ارزش استفاده می‌کند و بهترین عمل‌ها را بر اساس تجربه واقعی دریافت می‌کند. این الگوریتم در محیط‌های تغییراتی با عملکرد خوبی همراه است.با این حال، Monte Carloنیز دارای معایب است. برای به‌روزرسانی تابع ارزش باید منتظر پایان هر مسیر باشیم، که ممکن است زمان‌بر باشد. همچنین، معماری پیچیده‌تری دارد و نیازمند ذخیره و مدیریت تمام تجربه‌های یک مسیر است.

به‌طور کلی، Q-learningبا استفاده از تجربه فعلی و TD-Learning سریع‌تر و بهینه‌تر عمل می‌کند و مزیت استفاده از تجربه تازه را دارد، در حالی که Monte Carlo با استفاده از تجربه کامل عمل می‌کند و بهترین عمل‌ها را بر اساس تجربه واقعی دریافت می‌کند، اما ممکن است زمان بیشتری برای بهبود سیاست نیاز داشته باشد. همچنین، Q-learning

مستقل از سیاست و حساسیت به تابع پاداش است، در حالی که Monte Carlo تا حدی معماری پیچیده‌تری دارد و برای بهبود عملکرد نیازمند تجربه کامل است.

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

ستفاده از اپسیلون و روش اپسیلون حریصانه در الگوریتم‌های تقویت‌شده به منظور تعادل بین بهره‌وری و کاوش استفاده می‌شود. با استفاده از اپسیلون، عامل می‌تواند بهترین عمل‌ها را بر اساس تجربه فعلی انتخاب کند (بهره‌وری)، در عین حال امکان کاوش و بررسی عملکرد عمل‌های جدید نیز را دارد. استفاده از اپسیلون در الگوریتم‌های تقویت‌شده اهمیت بهره‌وری و کاوش را به تعادل می‌رساند و به عامل این امکان را می‌دهد که با بهره‌وری از عملکرد بهترین عمل‌ها، در عین حال به کاوش و بررسی عملکرد عمل‌های جدید نیز بپردازد.

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؟

در بخش بعد از ۵۰ ایتریشن، به دنبال رسیدن به سیاست بهینه با احتمال بالا برای مقادیر اپسیلون و نرخ یادگیری هستیم. اما به طور عملی این کار به آسانی امکان پذیر نیست. با استفاده از متد بخش ۸، ما نمی‌توانیم با احتمال بیشتر از ۹۹ درصد به یک سیاست بهینه برسیم، زیرا برای اینکار به بیش از ۵۰ اپیزود نیاز داریم. بنابراین، رسیدن به سیاست بهینه با احتمال بالا نیازمند اپیزودهای بیشتری است.

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

افزایش مقدار اپسیلون باعث می‌شود عامل بیشتر به صورت تصادفی عمل کند و به کاوش تمایل داشته باشد. این ممکن است باعث کند شدن روند یادگیری شود، اما در عوض، به عامل کمک می‌کند تا در مراحل اولیه یادگیری با استفاده از تجربه‌های تصادفی بهبود سیاست خود را پیدا کند

.

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-Learningیک توسعه از الگوریتم Q-Learning است که از شبکه‌های عصبی عمیق به عنوان تقریب‌گر تابع Q استفاده می‌کند. این الگوریتم برای مسائلی با فضای وضعیت بزرگتر و پیچیده‌تر

مناسب است و مشکلات ذخیره‌سازی تابع Q در جداول ساده را حل می‌کند-Approximate Q Learning نیز یک روش تقریبی است که تابع Q را با استفاده از تقریب‌گرهای پیچیده‌تر تقریب می‌زند Deep Q-Learning. از این روش بهبود داده شده است و با استفاده از شبکه‌های عصبی عمیق می‌تواند تابع Q را به دقت بالا تقریب بزند. این الگوریتم‌ها برای مسائلی با تابع Q پیچیده و نیاز به تقریب زدن دقیق آن استفاده می‌شوند و مشکلات ذخیره‌سازی تابع Q را حل می‌کنند.

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

```
def question3a():
    answerDiscount = 0.5
    answerNoise = 0.0
    answerLivingReward = -5.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'


def question3b():
    answerDiscount = 0.5
    answerNoise = 0.1
    answerLivingReward = -1.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'


def question3c():
    answerDiscount = 1.0
    answerNoise = 0.0
    answerLivingReward = -1.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'


def question3d():
    answerDiscount = 1.0
    answerNoise = 0.1
    answerLivingReward = -0.1
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'


def question3e():
    answerDiscount = 1.0
    answerNoise = 0.1
    answerLivingReward = 9999.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

در بخش a، پاداش زندگی عددی منفی و قابل توجه است تا عامل تلاش کند هر چه سریع‌تر بازی را به پایان برساند، پس به سراغ خروجی نزدیک‌تر می‌رود. ضمنا نویز نیز صفر است و عامل خطر صخره را می‌پذیرد.

در بخش b، مقدار نویز را افزایش می‌دهیم تا عامل به ریسک صخره اهمیت دهد و مسیر طولانی‌تر اما کم‌خطرتر را انتخاب کند.

در بخش c، برعکس بخش a مقدار پاداش زندگی طوری انتخاب می‌شود که عامل اهداف دورتر اما پرارزش‌تر را ترجیح دهد. همچنین مشابه بخش a نویز را صفر در نظر می‌گیریم تا عامل خطر صخره را بپذیرد.

در بخش d، مشابه بخش b و با افزایش مقدار نویز به عامل درباره ریسک صخره هشدار می‌دهیم و تغییر تخفیف به ۱/۰ عامل را تشویق به انتخاب اهداف دورتر اما پرارزش‌تر می‌کند.

در بخش e، پاداش زندگی را عددی بسیار بزرگ قرار می‌دهیم تا عامل همواره بازی را ادامه دهد و از پایان بازی در خروجی‌ها یا صخره‌ها بپرهیزد.