Department of
Computer Engineering

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

به نام خالق زیبایی ها

پروژه عملی سوم

درس مبانی هوش مصنوعی و کاربردها

امیرفاضل کوزه گر کالجی

9931099

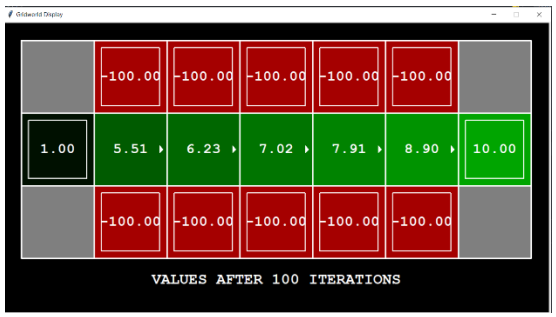
بخش اول)

- پیچیدگی در محاسبات

  در value iteration، ما به صورت iterative مقادیر را بروزرسانی می‌کنیم. این مورد با افزایش حالت ها و عمل هایمان باعث استفاده بیشتر و بیشتر از حافظه و رم شده، و پیچیدگی محاسبات را بالا میبرد.
- در این روش ما افزایش نمایی میزان تخفیف را داریم. میدانیم که با افزایش بعد های مسئله، مجبور میشویم از منابع بیشتری نیز استفاده کنیم. پس بزرگتر شدن حالات مسئله، کارآمدی الگوریتم را کمتر می کند.
- در این روش با هر ایتریشن، هر بار کل فضای حالت را به ازای یک جفت حالت، عمل بررسی می کنیم این بدین معنی می باشد که نمیتوان این راه حل را به مسائلی با فضای ناشناخته تعمیم داد.

بخش دوم)

در این مورد، با استفاده از سعی و خطا رفتار برنامه را در تغییر مقادیر تخفیف و نویز، بررسی می‌کنیم. مشاهده می‌شود که با کاهش دادن هر دو مورد، اعداد درون مربع ها مثبت تر شده و رنگ آنها به سمت سبز بودن می‌رود. نکته دوم مشاهده شده این است که با کاهش دادن مقدار نویز، میزان بیشتری از بهتر شدن را مشاهده می کنیم. نویز را به 0 نزدیک تر می‌کنیم و مقدار تخفیف را همان 0.9 نگه خواهیم داشت. مقدار نویز را برابر با 0.001 قرار می‌دهیم. نتیجه نهایی به مانند شکل زیر خواهد بود:



```
def question2():
    answerDiscount = 0.9
    answerNoise = 0.001
    return answerDiscount, answerNoise
```

تخفیف، یک بازه عمری را برای عامل مان تعریف می‌کند. که به آن عامل ها، یک تعدادی قدم می دهد تا بتوانند تا آنجایی که می توانند، جایزه جمع کنند تا قبل از اینکه به طور اتومات حذف شوند. پیاده سازی آنها به صورت امتیاز هایی جایزه ای است که مقدارشان به طور نمایی چند برابر می شود. و نکته مهم درباره آنها این است که مقدار اولیه شان بین 0 و 1 است. پس میزان جایزه پس از یک تعداد گام معینی، به 0 میل خواهد کرد.

یک از روش هایی که میتواند جایگزین روش value iteration شود، روش Q-learning می باشد که به طور مستقیم بر روی تابع سیاست عملیات انجام می‌دهد.

در این الگوریتم یک جدول Q در نظر میگیریم برای ذخیره ارزش اعمال در حالت های متفاوت. پس از هر گام عامل، یک سری عمل برای انتخاب دارد و پس از انتخاب یکی از آنها، به حالت جدید می رود و جایزه می گیرد به همین صورت. و در ادامه مقدار Q برای یک جفت حالت، عمل بروز رسانی می شود.

بخش سوم)

```
def question3a():
    answerDiscount = 0.6
    answerNoise = 0.1
    answerLivingReward = -2.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

چون خروجی نزدیک است، جایزه را منفی در نظر می گیریم

```
33
34  def question3b():
35      answerDiscount = 0.3
36      answerNoise = 0.2
37      answerLivingReward = -1
38      return answerDiscount, answerNoise, answerLivingReward
39      # If not possible, return 'NOT POSSIBLE'
40
```

اینجا نیز میزان جایزه را منفی در نظر می گیریم اما مقدار تخفیف را کمتر می کنیم.

```
def question3c():
    answerDiscount = 1
    answerNoise = 0
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

در این مورد نویز را به صفر می رسانیم. تخفیف را به یک نزدیک میکنیم تا به صخره نزدیک شویم.

```
def question3d():
    answerDiscount = .9
    answerNoise = .2
    answerLivingReward = 0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

پاداش را 0 قرار می دهیم و میزان تخفیف را به یک نزدیک میکنیم.

```python
def question3e():
    answerDiscount = 0.01
    answerNoise = 0.4
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

جایزه را کم و منفی در نظر میگیریم. میزان تخفیف را نیز کم و نزدیک به 0 در نظر خواهیم گرفت.

برای حل این مشکل می توان از راهکار های زیر بهره برد:

افزودن محدودیت: می توان با افزودن برخی محدودیت هایی از گیرکردن در حلقه بینهایت جلوگیری کرد.

شرط پایان: می توانیم یک سری شروط پایان را اضافه کنیم تا سیاستمان وقتی به حالت های خاصی رسید، سیاست به پایان برسد.

زمانی که میزان تخفیف در مقایسه با میزان نویز و جایزه مناسب نباشد و نتیجه خوبی را تضمین نکند، به حلقه بینهایت برخورد خواهیم کرد و همگرایی را نتیجه نخواهد داد.

بخش چهارم)

در روش batch، فقط با یکبار انجام شدن محاسبه تمامی اعضای Q آپدیت می‌شود. این مورد باعث تسریع کارایی و بهبود عملکرد می شود.

عملیات بروزرسانی در batch، بدین گونه عمل می کند که تمام داده ها را در حافظه ذخیره کرده و همه را همزمان پردازش می کنیم.

اگر تک به تک داده ها را بررسی کنیم، می توانیم به بهینگی و کارآمدی بهتری برسیم. اما این روش، مشکلاتی نیز دارد برای مثال ممکن است زمان زیادی ببرد.

بخش ششم)

اگر مقدار Q برای اقداماتی که قبلا ندیده، فاصله بسیاری داشته باشد چه دور چه نزدیک، سیاستی که انتخاب می شود، برای انجام یک عمل در یک حالت ممکن است تا اخر فرایند یادگیری، عامل به اشتباه بیفتد.

Q-learning یک روش off-policy است و value-based میباشد.

TD-learning:

- بروز رسانی مقادیر را بر اساس هر قدم انجام میدهد
- می تواند از اپیزود های غیرکامل یاد بگیرد و در هر قدم بروز رسانی کند

Montecarlo:

- بروز رسانی مقادیر پس از پیمایش کل حالات و اپیزود رخ میدهد.
- نیازمند تمامی اپیزودها می باشد تا بتواند مقدار صحیح را برگرداند.

بخش هفتم)

این روش به عامل این اختیار را میدهد که در حین یادگیری، به صورت تصادفی اقدامات جدید را بررسی کند و در عین حال به اقداماتی که در حال حاضر بهترین عملکرد را دارند، بهره برداری کند.

اگر مقدار اپسیلون بزرگ باشد، احتمال بررسی اقدامات تصادفی بالاتر می رود و عامل بیشتر از جستجوی عملکردهای جدید استفاده میکند.

اگر مقدارش کوچک باشد، احتمال بهره برداری از عملکردهای بهتر بالا میرود و عامل بیشتر به تلاش برای بهینه سازی استفاده میکند.

بخش هشتم)

با نزدیک به 0 بودن اپسیلون، عامل تلاش میکند تا از عملکرد های بهتر بهره برداری کند. یعنی عامل بیشترین احتمال را برای انتخاب عملی دارد که در حال حاضر بهترین عملکرد حساب میشود.

با نزدیک به یک بودن اپسیلون، عامل تلاش بر کاوش و کشف عملکرد های جدید می کند. یعنی عامل بیشترین احتمال را برای انتخاب عمل تصادفی دارد و تلاش میکند تا بیشترین اطلاعات ممکن را از فضای عمل جمع آوری کند.

بخش نهم)

بخش دهم)

Q-learning یک نسخه تقریبی از Deep Q-learning است که به حل مشکل فضای وضعیت های بزرگ و محیط های پیچیده می‌پردازد. در این روش، از یک شبکه عصبی عمیق، به عنوان تقریب گر استفاده می‌شود. این روش به معضل حجم بزرگ حافظه پاسخ می‌دهد. با استفاده از Q-vlaue، توابع را برای فضای وضعیت پیچیده و با ابعاد بالا تخمین Q-value قادر است توابع DQN شبکه عصبی عمیق و شبکه هدف بزند. همچنین، با استفاده از تکنیک هایی مانند تکرار تجربه عملکرد و استفاده از حافظه را بهبود می‌بخشد.