علی بازشوشتری ۹۹۲۳۰۰۸

گزارش پروژه‌ی سوم مبانی و کاربردهای هوش مصنوعی

بخش ۱)

پیاده سازی)

- Recursive definition of value:

$$V^*(s) = \max_a Q^*(s,a)$$

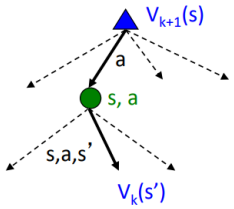
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

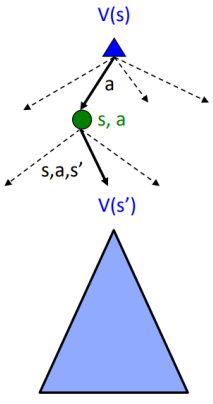


- Repeat until convergence

Value Iteration

- Bellman equations characterize the optimal values:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$



- Value iteration computes them:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values

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

۱- سرعت همگرا شدن بسیار کند است. در این روش value states آنقدر آپدیت می‌شوند تا همگرا شوند و سرعت این همگرا شدن بسیار کند است بخصوص اگر MDP بزرگ باشد. همچنین قابل ذکر است که policyها زودتر از مقادیر همگرا می‌شوند.

۲- حجم و پیچیدگی محاسباتی در آن زیاد است، بخصوص هنگامی که تعداد Stateها زیاد می‌شود. زیرا در هر iteration باید برای همه‌ی stateها محاسبات لازم را انجام دهیم.

۳- ممکن است به برخی از اطلاعات نظیر احتمالات (Transition Function) ویا rewardها دسترسی نداشته باشیم (که در آن صورت با مسائل Reinforcement Learning مواجه می‌شویم)

۴- این روش معمولا برای MDPها با تعداد گسسته حالات و actionها استفاده می‌شود درحالی که بسیار از مسائل دنیای واقعی پیوسته هستند.

۵- روش value iteration با فرض مارکوف کار میکند یعنی حالت آینده تنها به حالت فعلی و action وابسته است و به حالت های قبلی وابسته نیست که در برخی مسائل این شرط برقرار نیست (مثلا هنگامی که تاثیرات با تاخیر داریم)

بخش ۲)

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

```
def question2():
    answerDiscount = 0.9
    answerNoise = 0.01
    return answerDiscount, answerNoise
```

با کاهش نویز به ۰.۰۱ محیط را قطعی تر کردیم و این کار باعث می‌شود که عامل با سیاست بهینه از روی پل عبور کند و در حالت‌های با پاداش منفی نیفتد.

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).

Discount Factor در اینکه رسیدن به rewardهای نزدیک تر و زودتر تاثیرگذار است. به اینصورت که rewardای که خیلی دور است و باید time stepهای زیادی طی شود تا به آن برسیم، با استفاده از discount factor، ارزش آن را در ارزیابی کاهش می‌دهیم و کم می‌کنیم تا عامل تمایلی برای رسیدن به آن نداشته باشد و برنامه ریزی نکند و به جایش برای رسیدن به rewardهای نزدیک‌تر تلاش کند.

استفاده از Discount Factor هنگامی که در محیط نویز داریم نیز بسیار تاثیر مشهودتر و مهم‌تری دارد. زیرا وقتی در محیط نویز وجود دارد، احتمال رسیدن به rewardهای دورتر کمتر و کمتر می‌شود. به عبارتی برای رسیدن به reward دورتر نیاز به انجام actionهای بیشتری است و در نتیجه احتمال خطا افزایش می‌یابد.

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.

روش Policy Iteration. در این روش ابتدا یک fixed policy ارائه می‌دهیم و سپس آن را ارزیابی می‌کنیم و بهبودش می‌دهیم. در این مسئله خاص policyای که به نظر خوب و مطلوب می‌آید، حرکت به سمت راست (شرق) در تمامی خانه‌ها و حالت‌هاست. با تنظیم این policy و قرار دادن این actionها برای این stateها زودتر و راحت‌تر می‌توان به راه حل رسید.

همگرایی در این روش سریعتر است پس پیچیدگی زمانی آن کمتر است. همچنین واضح است که چون در fixed policy یک action داریم پس محاسبات کمتر است و سرعت بیشتر است.

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)
- Both are dynamic programs for solving MDPs

بخش ۳)

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

- خروجی نزدیک را ترجیح دهید (**+1**)، ریسک صخره را بپذیرید (**-10**)

```
def question3a():
    answerDiscount = 0.01
    answerNoise = 0
    answerLivingReward = -5
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

Discounting Factor: مقدارش را بسیار کم میگذاریم تا ارزش خروجی دورتر به میزان قابل توجهی کاهش یابد و عامل تمایلی برای رفتن به سمت آن نداشته باشد.

Answer Noise: مقدار آن را برابر ۰ قرار میدهیم تا عامل ترجیح دهد از مسیر نزدیک‌تر که کنار صخره است عبور کند و ریسک را بپذیرد.

Answer Living Reward: مقدار آن را برابر با یک مقدار منفی قرار می‌دهیم تا هم عامل تمایل داشته باشد از مسیر کوتاه‌تر است عبور کند و ریسک صخره را بپذیرد هم اینکه بین خروجی نزدیک‌تر و دورتر، خروجی نزدیک‌تر را انتخاب کند زیرا زنده ماندن برای آن هزینه دارد.

- خروجی نزدیک را ترجیح دهید (**+1**)، اما از صخره اجتناب کنید (**-10**)

```
def question3b():
    answerDiscount = 0.1
    answerNoise = 0.1
    answerLivingReward = -2
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

Discounting Factor: مقدارش را کم می‌گذاریم تا عامل خروجی نزدیک‌تر را به خروجی دورتر ترجیح دهد. (ارزش خروجی دورتر کاهش یابد)

Answer Noise: با ایجاد noise سبب می‌شویم که عامل از صخره‌ها اجتناب کند و از مسیر دورتر عبور کند.

Answer Living Reward: به گونه‌ای تنظیم می‌شود که عامل ترجیح دهد به سمت خروجی نزدیک‌تر برود زیرا زنده ماندن برای آن هزینه دارد اما در عین حال باید به گونه‌ای باشد که عامل نخواهد ریسک کند و از کنار صخره‌ها عبور کند

- خروجی دور را ترجیح دهید (**+10**)، ریسک صخره را بپذیرید (**-10**)

```
def question3c():
    answerDiscount = 1
    answerNoise = 0
    answerLivingReward = -2
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

Discounting Factor: مقدارش را برابر ۱ قرار می‌دهیم تا از ارزش خروجی دورتر چیزی کم نشود و در نتیجه عامل ترجیح دهد به سمت خروجی دورتر که با ارزش‌تر است برود.

Answer Noise: برابر ۰ قرارش میدهیم تا عامل ریسک عبور از کنار صخره‌ها را بپذیرد.

Answer Living Reward: یک مقدار منفی برای آن قرار می‌دهیم تا عامل بخواهد از مسیر نزدیک‌تر عبور کند و زودتر به هدف برسد زیرا زنده ماندن برایش هزینه دارد. در نتیجه با این کار عامل ریسک صخره‌ها را می‌پذیرد و از کنار آنها عبور می‌کند.

- خروجی دور را ترجیح دهید (**+10**)، اما از صخره اجتناب کنید (**-10**)

```
def question3d():
    answerDiscount = 1
    answerNoise = 0.1  # the more answerNoise, the more interested to take the long road
    answerLivingReward = -0.5
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

Discounting Factor: مقدارش را برابر ۱ قرار می‌دهیم تا از ارزش خروجی دورتر چیزی کم نشود و در نتیجه عامل ترجیح دهد به سمت خروجی دورتر که با ارزش‌تر است برود.

Answer Noise: یک مقدار مثبت برای نویز در نظر میگیریم تا عامل ترجیح دهد از مسیر امن‌تر عبور کند و از کنار صخره‌ها عبور نکند.

Answer Living Reward: یک مقدار منفی کوچک برایش در نظر می‌گیریم تا عامل اولا نخواهد دائما زنده بماند و دوما تفاوت چشمگیری میان خروجی نزدیک‌تر و دورتر برایش وجود نداشته باشد زیرا برای رفتن به سمت خروجی دورتر باید time stepهای بیشتری داشته باشیم و بیشتر زنده بمانیم و زنده ماندن هزینه دارد. اگر این هزینه زیاد و چشمگیر باشد ممکن است عامل ترجیح دهد به سمت خروجی نزدیک‌تر برود یا حتی خودکشی کند اصلا!

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

```
def question3e():
    answerDiscount = 1
    answerNoise = 0.1
    answerLivingReward = 100
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

یک مقدار بسیار بزرگ برای Answer Living Reward درنظر می‌گیریم تا ترجیح دهد دائما زنده بماند و امتیاز دریافت کند؛ با اینکار وارد خروجی‌ها نمی‌شویم. همچنین مقداری را برای نویز درنظر می‌گیریم تا از صخره‌ها اجتناب شود.

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.

مقدار living reward را کاهش دهیم به گونه‌ای که رفتن به خروجی و جایزه آن، بیشتر از جایزه زنده ماندن باشد.

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

خیر. در صورتی مطمئنیم همگرا می‌شود که مقدار گاما کوچکتر از ۱ باشد زیرا با افزایش تعداد iterationها و حرکت به سمت بی‌نهایت، این مقدار گاما دائما در ارزش ضرب می‌شود و حاصل به ۰ میل می‌کند.

بخش ۴)

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 batch:

مزیت: همه‌ی حالات بررسی می‌شوند و value stateها زودتر به همگرایی می‌رسند.

عیب: میزان محاسبات و پردازش در آن بیشتر است و در نتیجه زمان بیشتری صرف می‌شود.

روش برروزرسانی به صورت تکی:

مزیت: میزان محاسبات و زمان کمتر است و قرار نیست در هر Iteration برای تمام stateها محاسبات را انجام دهیم.

عیب: value stateها دیرتر همگرا می‌شوند.

بخش ۶)

Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

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

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

اگر مقدار Q بسیار زیاد باشد عامل تمایل دارد بیشتر به سمت اقدام متناظر با آن Q-value برود و کمتر به سمت دیگر اقدامات برود و به عبارتی می‌توان گفت بیشتر به سمت exploration میرویم چون به سمت اقداماتی که تا حالا انجام نداده می‌رود.

اگر مقدار Q بسیار کم باشد عامل از انتخاب اقدام متناظر خودداری می‌کند و دیگر اقدامات را امتحان می‌کند و به عبارتی می‌توان گفت بیشتر به سمت exploitation می‌رویم چون به سمت اقدامی که تا حالا انجام نداده نمی‌رود و اقدامات قبلی خود را تکرار می‌کند.

پس در هر حال اقدامات و حالاتی هستند که کمتر انتخاب و امتحان کند و یا اینکه اصلا امتحان نکند.

همچنین در این حالت مدت زمان بسیار زیادی طول می‌کشد تا مقدار Q به مقدار واقعی آن نزدیک شود و نیازمند تعداد تکرارهای زیادی است.

به عنوان مثال اگر مقدار Q بسیار زیاد باشد و action متناظر با آن در واقع خوب و مناسب نباشد و به ما امتیاز کمی بدهد یا حتی منفی باشد (به عبارتی مقدار واقعی Q بسیار کمتر و یا منفی باشد) آنگاه عامل به تعداد دفعات زیادی آن action متناظر را انتخاب می‌کند و مدت زمان زیادی طول می‌کشد تا مقدار Q به مقدار واقعی آن برسد. همچنین درتمام این مدت عامل دائما در حال دریافت امتیاز کم و یا منفی است که مطلوب نمی‌باشد.

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 است زیرا به سمت policy بهینه همگرا می‌شود حتی اگر دارد به صورت suboptimal عمل می‌کند. Value function و policy را با داده‌های بدست آمده از دنبال کردن **سیاست‌های مختلف** یاد می‌گیرد. به عبارتی عامل در حال پیدا کردن و بهبود سیاست هدف است.

Q-learning یک الگوریتم value-based است زیرا در حال آپدیت کردن و استفاده از Q-values هستیم و policy را بر اساس این Q-values انتخاب و مشخص می‌کنیم و مستقیما در حال یادگیری و بهینه policy نیستیم.

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.

Temporal Difference learning از یک step استفاده می‌کند در حالی که Monte Carlo از یک اپیزود کامل از تجربه استفاده می‌کند؛ به عبارتی تا انتهای اپیزود صبر می‌کند و در پس از به انتها رسیدن اپیزود یادگیری را انجام می‌دهد.

مزیت Monte Carlo در سادگی فهم آن است و اما عیب آن در این است که بعد از کل اپیزود می‌توانیم آپدیت را انجام دهیم و زمان بیشتری طول می‌کشد تا یادگیری انجام شود؛ به علاوه در محیط‌هایی که اپیزودها طولانی هستند یا نیازمند feedback آنی هستیم این مورد دردسرساز می‌شود.

در TD-learning از آنجا که پس از هر گام آپدیت انجام می‌شود پس سرعت همگرایی بیشتری نسبت به Monte Carlo دارد. همچنین از دیگر مزایا آن این است که چون پس از هر گام آپدیت انجام می‌شود، برای online learning بخصوص در محیط‌های با اقدامات پیوسته مناسب است زیرا ما به صورت آنی feedback می‌گیریم. از معیاب TD-learning داشتن trade-off میان exploration و exploitation است.
همچنین هزینه آن بیشتر است و پیاده سازی پیچیده تری دارد.

بخش ۷)

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

استفاده از اپسیلون سبب می‌شود تا عامل برخی اوقات (با احتمال اپسیلون) به صورت رندوم عمل کند و با این کار exploration را انجام می‌دهیم و عامل حالات و اقدامات جدید را امتحان می‌کند (با رندوم عمل کردن به این هدف دست پیدا می‌کنیم)

بخش ۸)

ابتدا، یک **Q-learner** کاملا تصادفی را با ضریب یادگیری پیش‌فرض بر روی **BridgeGrid** بدون نویز، با **۵۰** اپیزود آموزش دهید و بررسی کنید که آیا سیاست بهینه در این حالت یافت می‌شود یا خیر.

```
python gridworld.py -a q -k 50 -n 0 -g BridgeGrid -e 1
```

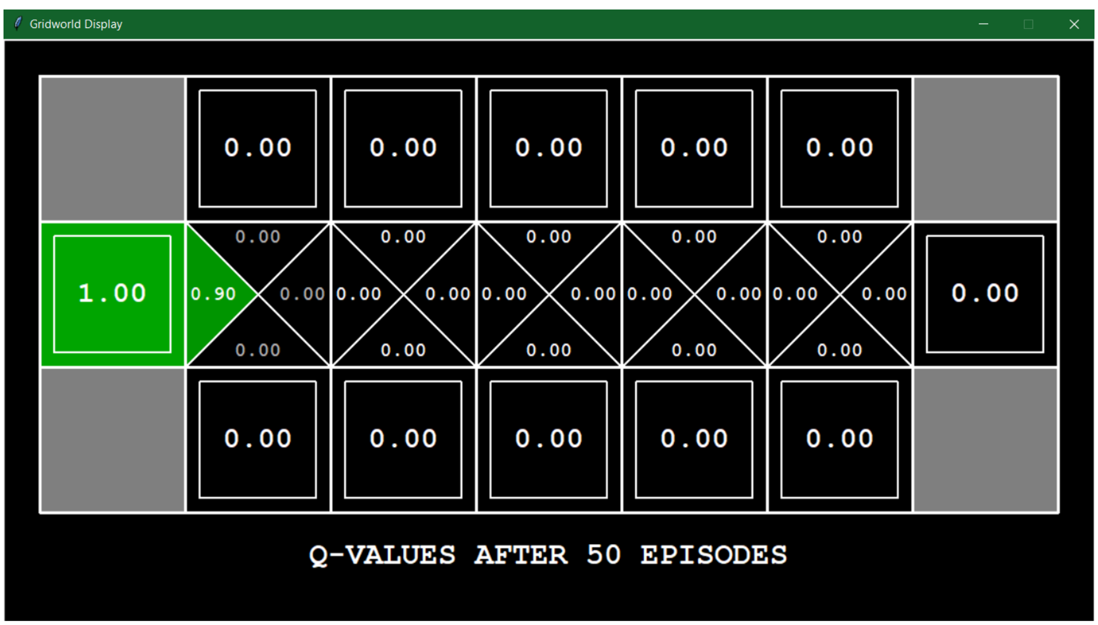
مشاهده می‌شود که سیاست بهینه در این حالت یافت نمی‌شود.

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؟

خیر وجود ندارد. در این مثال خاص سیاست بهینه این است که عامل ۴ بار به سمت راست حرکت کند. اگر اپسیلون را برابر ۱ درنظر بگیریم یعنی عامل دائما در حال exploration و امتحان حالات جدید است. پس اولا احتمال اینکه به سیاست بهینه دست پیدا کند و از پل عبور کند کم است و ثانیا اگر هم به این سیاست دست پیدا کند، از آنجا که دائما در حال exploration هستیم از آن سیاست بهینه استفاده نمی‌کند و سیاست‌های مختلف و متفاوت را امتحان می‌کند.

همچنین اگر اپسیلون را برابر ۰ در نظر بگیریم نیز عامل دائما در حال exploitation است و آن سیاستی که قبلا به آن دست پیدا کرده را پیش می‌گیرد و سیاست جدید را امتحان نمی‌کند و در نتیجه به سیاست بهینه دست پیدا نمی‌کند. به عبارتی یادگیری نمی‌کند و از معلومات خود استفاده می‌کند.

پس به طور کلی برای دستیابی به سیاست بهینه، ۵۰ اپیزود کم است و به اپیزودهای بیشتری نیاز داریم.



شکل) Q-Values به ازای اپسیلون ۰

شکل) Q-Values به ازای اپسیلون ۱

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

اگر اپسیلون را زیاد کنیم، عامل بیشتر یادگیری می‌کند و اقدامات و حالات مختلف را امتحان می‌کند. وقتی اپسیلون زیاد باشد بیشتر در حال exploration هستیم.

اما اگر اپسیلون را کم کنیم، عامل بیشتر از معلومات خود و اقداماتی که می‌داند سودمندی بیشتری دارند استفاده می‌کند و بیشتر در حال exploitation است.

بخش ۹)

اجرای دستور python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid:

```
Reinforcement Learning Status:
        Completed 1500 out of 2000 training episodes
        Average Rewards over all training: -139.52
        Average Rewards for last 100 episodes: 236.00
        Episode took 0.84 seconds
Reinforcement Learning Status:
        Completed 1600 out of 2000 training episodes
        Average Rewards over all training: -116.01
        Average Rewards for last 100 episodes: 236.74
        Episode took 0.84 seconds
Reinforcement Learning Status:
        Completed 1700 out of 2000 training episodes
        Average Rewards over all training: -98.87
        Average Rewards for last 100 episodes: 175.34
        Episode took 0.85 seconds
Reinforcement Learning Status:
        Completed 1800 out of 2000 training episodes
        Average Rewards over all training: -81.41
        Average Rewards for last 100 episodes: 215.49
        Episode took 0.84 seconds
Reinforcement Learning Status:
        Completed 1900 out of 2000 training episodes
        Average Rewards over all training: -63.11
        Average Rewards for last 100 episodes: 266.26
        Episode took 0.84 seconds
Reinforcement Learning Status:
        Completed 2000 out of 2000 training episodes
        Average Rewards over all training: -48.63
        Average Rewards for last 100 episodes: 226.44
        Episode took 0.86 seconds
Training Done (turning off epsilon and alpha)
-----------------------------------------
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 501
Pacman emerges victorious! Score: 499
Average Score: 498.4
Scores:        499.0, 495.0, 495.0, 503.0, 503.0, 499.0, 495.0, 495.0, 501.0, 499.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

همانطور که مشاهده می‌شود پس از ۲۰۰۰ بار آموزش، عامل در فاز آزمون توانست هر ۱۰ بازی را برنده شود.

اجرای دستور python autograder.py -q q9:

```
Reinforcement Learning Status:
        Completed 500 out of 2000 training episodes
        Average Rewards over all training: -362.12
        Average Rewards for last 100 episodes: -218.28
        Episode took 0.78 seconds
Reinforcement Learning Status:
        Completed 600 out of 2000 training episodes
        Average Rewards over all training: -333.37
        Average Rewards for last 100 episodes: -189.60
        Episode took 0.81 seconds
Reinforcement Learning Status:
        Completed 700 out of 2000 training episodes
        Average Rewards over all training: -305.58
        Average Rewards for last 100 episodes: -138.87
        Episode took 0.85 seconds
Reinforcement Learning Status:
        Completed 800 out of 2000 training episodes
        Average Rewards over all training: -286.05
        Average Rewards for last 100 episodes: -149.29
        Episode took 0.80 seconds
Reinforcement Learning Status:
        Completed 900 out of 2000 training episodes
        Average Rewards over all training: -266.25
        Average Rewards for last 100 episodes: -107.87
        Episode took 0.80 seconds
Reinforcement Learning Status:
        Completed 1000 out of 2000 training episodes
        Average Rewards over all training: -237.24
        Average Rewards for last 100 episodes: 23.85
        Episode took 0.89 seconds
Reinforcement Learning Status:
        Completed 1100 out of 2000 training episodes
        Average Rewards over all training: -204.48
        Average Rewards for last 100 episodes: 123.16
        Episode took 0.92 seconds
Reinforcement Learning Status:
        Completed 1200 out of 2000 training episodes
        Average Rewards over all training: -171.06
        Average Rewards for last 100 episodes: 196.54
        Episode took 0.84 seconds
Reinforcement Learning Status:
        Completed 1300 out of 2000 training episodes
        Average Rewards over all training: -147.40
        Average Rewards for last 100 episodes: 136.44
        Episode took 0.78 seconds
Reinforcement Learning Status:
        Completed 1400 out of 2000 training episodes
        Average Rewards over all training: -126.48
        Average Rewards for last 100 episodes: 145.47
```

همانطور که در شکل بالا مشخص است، اولین بار پس از ۱۰۰۰ بار بازی و آموزش, عامل به طور میانگین به یک امتیاز مثبت دست پیدا می‌کند.

```
Reinforcement Learning Status:
        Completed 1500 out of 2000 training episodes
        Average Rewards over all training: -102.95
        Average Rewards for last 100 episodes: 226.52
        Episode took 0.81 seconds
Reinforcement Learning Status:
        Completed 1600 out of 2000 training episodes
        Average Rewards over all training: -83.61
        Average Rewards for last 100 episodes: 206.49
        Episode took 0.80 seconds
Reinforcement Learning Status:
        Completed 1700 out of 2000 training episodes
        Average Rewards over all training: -61.81
        Average Rewards for last 100 episodes: 286.92
        Episode took 0.97 seconds
Reinforcement Learning Status:
        Completed 1800 out of 2000 training episodes
        Average Rewards over all training: -44.62
        Average Rewards for last 100 episodes: 247.62
        Episode took 0.88 seconds
Reinforcement Learning Status:
        Completed 1900 out of 2000 training episodes
        Average Rewards over all training: -30.85
        Average Rewards for last 100 episodes: 217.10
        Episode took 0.80 seconds
Reinforcement Learning Status:
        Completed 2000 out of 2000 training episodes
        Average Rewards over all training: -17.97
        Average Rewards for last 100 episodes: 226.73
        Episode took 0.82 seconds
Training Done (turning off epsilon and alpha)
----------------------------------------------
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
```

پس از آن نیز مشاهده می‌شود که در پایان آموزش امتیاز همچنان مثبت است و نسبتا نیز بالاس (بین ۱۰۰ ت ا ۳۵۰). در شروع آزمون‌ها نیز به دلیل ۰ شدن اپسیلون و آلفا مشاهده می‌شود که دیگر عامل تنها exploitation می‌کند و از معلومات خود و سیاست‌های نسبتا بهینه‌ای که پیدا کرده استفاده می‌کند. به همین دلیل امتیاز مثبت مانده و مقدار آن بسیار بیشتر است (حدود ۵۰۰)

```
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Reinforcement Learning Status:
        Completed 100 test episodes
        Average Rewards over testing: 500.20
        Average Rewards for last 100 episodes: 500.20
        Episode took 0.86 seconds
Average Score: 500.2
Scores:         499.0, 499.0, 495.0, 503.0, 503.0, 495.0, 495.0, 495.0, 503.0, 503.0, 495.0, 503.0, 503.0, 503.0, 495.0, 503.0, 503.0, 503.0, 499.0, 503.0, 503.0, 5
503.0, 503.0, 495.0, 503.0, 495.0, 499.0, 503.0, 499.0, 499.0, 499.0, 503.0, 495.0, 495.0, 503.0, 503.0, 503.0, 495.0, 503.0, 495.0, 499.0, 503.0, 495.0, 499.0, 50
99.0, 499.0, 499.0, 503.0, 503.0, 499.0, 503.0, 503.0, 503.0, 503.0, 495.0, 499.0, 495.0, 495.0, 503.0, 503.0, 495.0, 503.0, 503.0, 503.0, 495.0, 503.0, 503.0, 499
3.0, 495.0, 503.0, 503.0, 499.0, 499.0, 503.0, 503.0, 503.0, 499.0, 499.0, 495.0
Win Rate:       100/100 (1.00)
Record:         Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
in, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
n, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** 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 ###
```

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

تغییری انجام نشده است.

بخش ۱۰)

پیاده سازی:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear Q-functions:

transition $= (s, a, r, s')$

difference $= \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha$ [difference] Exact Q's

$w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$ Approximate Q's

مطابق با فرمول‌های بالا عمل می‌کنیم و کدهای آنها را پیاده سازی می‌کنیم.

```
def getQValue(self, state, action):
    """
      Should return Q(state,action) = w * featureVector
      where * is the dotProduct operator
    """
    "*** YOUR CODE HERE ***"
    features = self.featExtractor.getFeatures(state, action)
    return sum([self.weights[feature] * features[feature] for feature in features])
    # util.raiseNotDefined()
```

در قطعه کد بالا مثل فرمول موجود در کادر آبی، مقدار Q-value را بر اساس feature functions و وزن آنها محاسبه می‌کنیم و با یکدیگر جمع می‌کنیم.

```
def update(self, state, action, nextState, reward):
    """
       Should update your weights based on transition
    """
    "*** YOUR CODE HERE ***"
    difference = (reward + self.discount * self.getValue(nextState)) - self.getQValue(state, action)
    features = self.featExtractor.getFeatures(state, action)
    for feature in features:
        self.weights[feature] += self.alpha * difference * features[feature]
    # util.raiseNotDefined()
```

در این قطعه کد نیز باقی فرمول‌ها را پیاده سازی می‌کنیم و مقدار وزن‌ها را بر اساس آلفا و تفاضل آپدیت می‌کنیم.

اجرای دستور **python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid:**

```
Beginning 2000 episodes of Training
Reinforcement Learning Status:
        Completed 100 out of 2000 training episodes
        Average Rewards over all training: -510.11
        Average Rewards for last 100 episodes: -510.11
        Episode took 0.62 seconds
Reinforcement Learning Status:
        Completed 200 out of 2000 training episodes
        Average Rewards over all training: -510.77
        Average Rewards for last 100 episodes: -511.43
        Episode took 0.79 seconds
Reinforcement Learning Status:
        Completed 300 out of 2000 training episodes
        Average Rewards over all training: -508.22
        Average Rewards for last 100 episodes: -503.13
        Episode took 0.95 seconds
Reinforcement Learning Status:
        Completed 400 out of 2000 training episodes
        Average Rewards over all training: -461.49
        Average Rewards for last 100 episodes: -321.30
        Episode took 1.02 seconds
Reinforcement Learning Status:
        Completed 500 out of 2000 training episodes
        Average Rewards over all training: -427.50
        Average Rewards for last 100 episodes: -291.54
        Episode took 1.13 seconds
Reinforcement Learning Status:
        Completed 600 out of 2000 training episodes
        Average Rewards over all training: -398.08
        Average Rewards for last 100 episodes: -250.98
        Episode took 1.32 seconds
Reinforcement Learning Status:
        Completed 700 out of 2000 training episodes
        Average Rewards over all training: -365.31
        Average Rewards for last 100 episodes: -168.70
        Episode took 1.13 seconds
Reinforcement Learning Status:
        Completed 800 out of 2000 training episodes
        Average Rewards over all training: -325.52
        Average Rewards for last 100 episodes: -46.99
        Episode took 1.07 seconds
Reinforcement Learning Status:
        Completed 900 out of 2000 training episodes
        Average Rewards over all training: -294.50
        Average Rewards for last 100 episodes: -46.34
        Episode took 1.03 seconds
Reinforcement Learning Status:
        Completed 1000 out of 2000 training episodes
        Average Rewards over all training: -261.75
        Average Rewards for last 100 episodes: 33.02
```

```
Reinforcement Learning Status:
        Completed 1500 out of 2000 training episodes
        Average Rewards over all training: -115.06
        Average Rewards for last 100 episodes: 186.26
        Episode took 1.10 seconds
Reinforcement Learning Status:
        Completed 1600 out of 2000 training episodes
        Average Rewards over all training: -96.78
        Average Rewards for last 100 episodes: 177.45
        Episode took 1.05 seconds
Reinforcement Learning Status:
        Completed 1700 out of 2000 training episodes
        Average Rewards over all training: -74.18
        Average Rewards for last 100 episodes: 287.35
        Episode took 1.08 seconds
Reinforcement Learning Status:
        Completed 1800 out of 2000 training episodes
        Average Rewards over all training: -57.43
        Average Rewards for last 100 episodes: 227.42
        Episode took 1.05 seconds
Reinforcement Learning Status:
        Completed 1900 out of 2000 training episodes
        Average Rewards over all training: -42.98
        Average Rewards for last 100 episodes: 217.07
        Episode took 1.07 seconds
Reinforcement Learning Status:
        Completed 2000 out of 2000 training episodes
        Average Rewards over all training: -33.53
        Average Rewards for last 100 episodes: 145.92
        Episode took 1.09 seconds
Training Done (turning off epsilon and alpha)
-----------------------------------------
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Average Score: 500.6
Scores:        503.0, 499.0, 499.0, 499.0, 495.0, 503.0, 503.0, 499.0, 503.0, 503.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

اجرای دستور

:python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid

```
Beginning 50 episodes of Training
Training Done (turning off epsilon and alpha)
-----------------------------------------
Pacman emerges victorious! Score: 527
Pacman emerges victorious! Score: 527
Pacman emerges victorious! Score: 527
Pacman emerges victorious! Score: 527
Pacman emerges victorious! Score: 527
Pacman emerges victorious! Score: 529
Pacman emerges victorious! Score: 529
Pacman emerges victorious! Score: 527
Pacman emerges victorious! Score: 529
Pacman emerges victorious! Score: 529
Average Score: 527.8
Scores:        527.0, 527.0, 527.0, 527.0, 527.0, 529.0, 529.0, 527.0, 529.0, 529.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

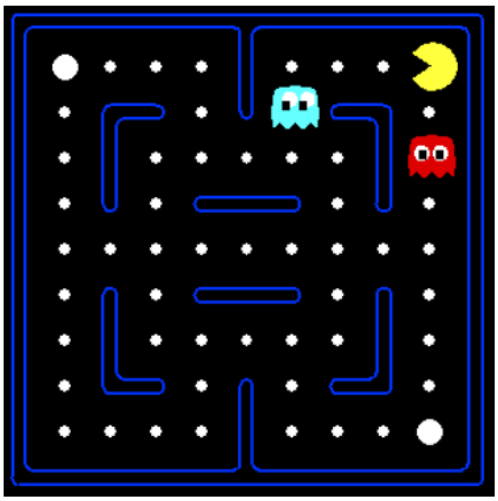
سوال: درباره Deep Q-leaning تحقیق کنید و بیان کنید در چه مواردی این الگوریتم به جای الگوریتم Q-learning عادی استفاده می‌شود. هر کدام از این الگوریتم‌ها، Approximate Q-learning و Deep Q-learning چه مشکلی را از Q-learning حل می‌کنند.

Deep Q-learning نوعی از الگوریتم‌های reinforcement learning است که از شبکه عصبی عمیق (deep neural network) برای تخمین Q-functions جهت تشخیص action بهینه در هر state استفاده می‌کند.

مزیت‌ها و مواردی که Deep Q-learning بجای Q-learning استفاده می‌شود: هنگامی که فضای حالات و اقدامات بزرگ‌تر و پیچیدتر هستند زیرا شبکه عصبی قادر است ویژگی‌های پیچیده‌تر در محیط را تشخیص دهد و دقت بیشتری دارد. همچنین در فضای action پیوسته نیز deep Q-learning کارآمد است.

Q-learning جدولی از تمام Q-values نگه می‌دارد که در محیط‌های واقعی همچین چیزی امکان پذیر نیست زیرا هم تعداد حالاتی که باید در یادگیری ویزیت شوند زیاد است و هم حافظه زیادی لازم داریم.

Q-learning میان حالاتی که در برخی ویژگی‌ها شبیه به یکدیگر هستند تمایز و تفاوت قائل می‌شود.



به عنوان مثال Q-learning میان دو حالت بالا تمایز قائل می‌شود! و اگر به عنوان مثال بفهمد که حالت سمت چپ بد است، اما هیچ اطلاعاتی در ارتباط با حالت سمت راست ندارد.

Approximate Q-learning این مشکل را حل می‌کند و تعدادی feature ارائه و معرفی می‌کند و به آنها وزن می‌دهد. در نتیجه تجربه‌مان را می‌توانیم در این featureها خلاصه کنیم.

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Deep Q-learning مشکل مقیاس پذیری و اجرا در محیط‌های بزرگ‌تر و پیچیده‌تر (از جمله محیط‌های پیوسته) را حل می‌کند. هنگامی که محیط بزرگ‌تر و پیچیده‌تر می‌شود، تعداد فضای حالات و اقدامات بسیار زیاد می‌شوند و Q-table می‌تواند به صورت نمایی رشد کند که ذخیره و آپدیت آن را غیرممکن می‌سازد. اما Deep Q-learning با استفاده از شبکه‌های عصبی عمیق این مشکل را حل می‌کند.