به نام خدا

دانشگاه تهران

پردیس دانشکده های فنی

دانشکده مهندسی برق و کامپیوتر

تمرین شماره‌ی 6

بخش عملی

استاد درس:

دکتر فدائی

دکتر یعقوب‌زاده

نگارش:

فاطمه محمدی 810199489

اهداف:

آشنایی با یادگیری تقویتی.

آشنایی و استفاده از Qlearning.

مقدمه:

یادگیری تقویتی یکی از شاخه های یادگیری ماشین است و با توجه به گستردگی آم، در زمینه های گوناگون مانند نظریه بازی ها، نظریه کنترل و غیره استفاده میشود.

در یادگیری تقویتی Agent هوشمند با جوستجو و اکتشاف در محیط قابل تعامل کورد نظر، در ابتدا باید دانش و تجربه ای از محیط جمع اوری کند و سپس براساس دانش کسب شده، باید عملا و رفتار هایی را در ان محیط انجام دهد تا مجموع پاداشی که از آن محیط میگرد بیشنه شود.

تعریف پروژه:

پیاده سازی بازی دونفره قدیمی Snake که در آن مار با خورد سیب امتیاز مثبت میگرد. بازی در حالتی تمام می شود که کله یکی از مارها

به بدن مار دیگر برخورد کند)در این حالت ماری که کله اش به بدن مار دیگر برخورد کرده می بازد، در این حالت توجه کنید که اگر سر مار به بدن خودش نیز برخورد کند نیز مار می بازد(و یا کله دو مار با یکدیگر برخورد کند که در این حالت ماری که طول بیشتری داشته باشد می برد، در حالتی که کله دو مار به یکدیگر برخورد کند و طول برابر داشته باشند نیز هیچکدام از مارها برنده نمی شود. صفحه بازی یک جدول 20 در 20 می باشد و حالت ابتدایی بازی به این شکل می باشد که هر دو مار به صورت تصادفی در نقطه ای از جدول ظاهر می شوند.

قوانین بازی کمی متفاوت از قوانین بازی اصلی می باشد، مثلا مار نمی تواند از صفحه خارج شده و در صورتی که تلاش کند خارج شود بازنده می شود. مانند بازی اصلی به طور تصادفی سیب هایی در زمین بازی ظاهر می شوند و باعث افزایش طول مار می شوند.

پیاده سازی پروژه:

در ادامه 3 فایل مورد بررسی قرار و پیاده سازی قرار گرفته اند را توضیح میدهیم:

در . snake.pyبه پیاده سازی agent هوشمند باید پرداخته شود:

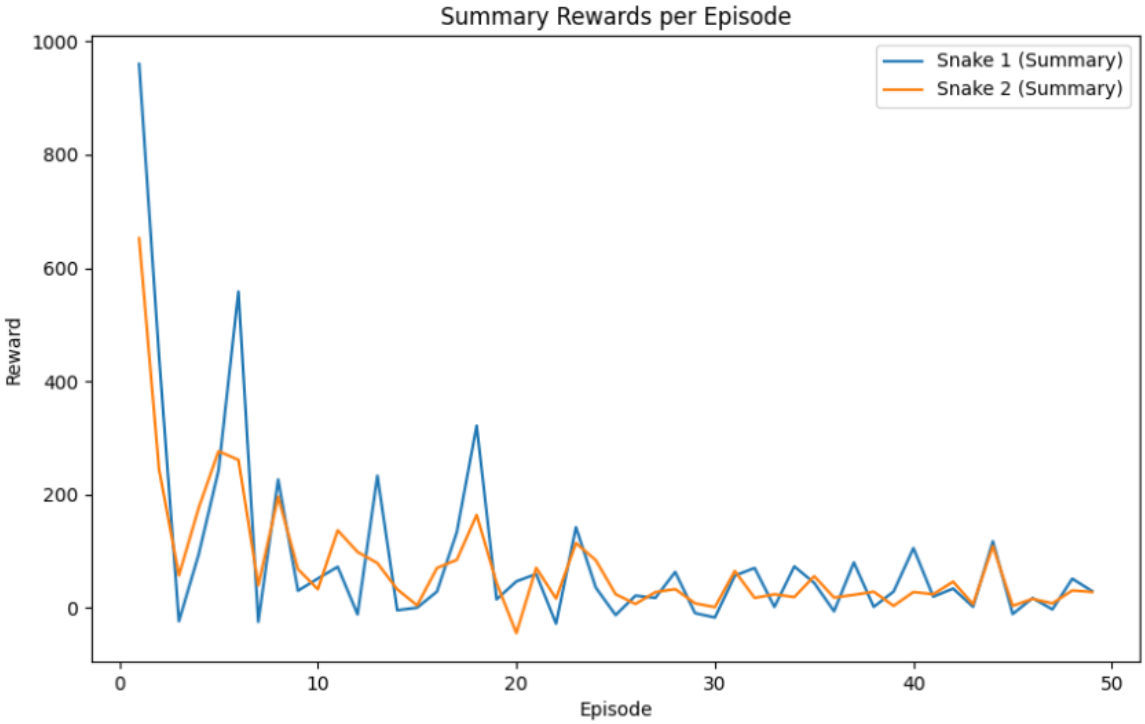
برای بررسی بهترین پارامتر ها در main.py از دو Sanke متقاوت استفاده شده و پارامتر ها مورد بررسی و تعویض قرار گرفتند.

لازم به ذکر است، برای پارامتر های اصلی برای Snake ، از مقادیر که به طور معمول استفاده میشود در ابتدا استفاده شده و سپس مقادیر را تغیر داده و در نهایت از همان مقادیر اولیه که معمول تر هستند استفاده کرده ایم.

مهم ترین تغیرات در snake.py:

get_optimal_policy را به دو صورت simple و غیر simple پیاده سازی کرده ایم، که در مورد اول agent از دانش کمتری استفاده کرده است و حالت ایده آل تعریف این تابع میباشد (Snake_3.py) اما به طور کلی از حالت غیر Simple استفاده میکنیم

که agent هوشمندی بیشتری در باره خود و درباره محیط دارد و از آنها در تعیین optimal_policy استفاده میکند و سعی میکند به دیوار یا به خود برخورد نکند و به این صورت عمل میکند که اگر حرکتی موجب باخت میشود، امتیازی منفی زیادی بگیرد و اگر به سمت سیب میرود امتیاز بیشتری بگیرد و این امتیازها را به انچه تا کنون یادگرفته است، اضافه میکند و سپس بهترین تصمیم را میگیرد.
(باتوجه به اینکه اینکار لازم به دانش نسبی محیط دارد، حالت simple هم پیاده سازی شده است که نتیجه مقایسه این دو حالت در پایین اماده است:)




```
Snake 1 Wins: 2
Snake 2 Wins: 298

```

همانطور که انتظار میرفت عملکرد مدل غیر simple (مار شماره 2) بسیار بهتر است.

اما به طور کلی با episode های بیشتر رفتار هر دو مار به یک سمت در جهت بقا (و گرفتن امتیاز) پیش میرود.

- توجه شود مدل ما هدف برد ندارد و در جهت بقا و گرفتن امتیاز بیشتر تنها پیش میرود
اما میتوانستیم با دادن مختصات سر مار رقیب و طول آن تلاش کنیم اگر طولمان بلند تر از آن است، با ان برخورد کنیم که استراتژی برد است.

در ادامه موارد دیگری از جمله پارامتر های مار مقایسه شده اند که با توجه به خواسته پروژه که تنها سه مدل کفایت میکند، در مدل سوم به تغیر learning rate پرداخته ایم و با دو برابر کردن ان برد بیشتری داشتیم (البته این استراتژی کوتاه مدت تست شده است)

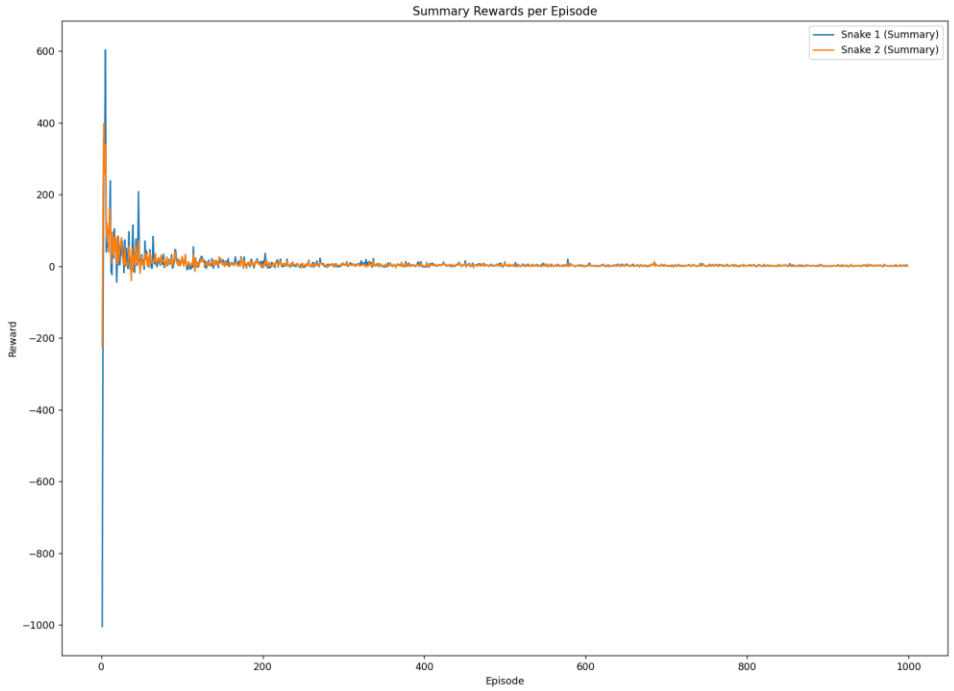
تابع مهم دیگری که در snake داشتیم، calc_rewardمیباشد که باید برای حرکات که موجب به باخت میشود عدد منفی و برای برد عددی مثبت به reward اضافه شود

توجه شود برای حرکت مار به سمت سیب نیز یک امتیاز در نظر گرفته ایم اما چون در تابع get_optimal_policy به این موضوع ارزش داده بودیم، ارزش کمی دادیم که مار به ازای باخت به سمت سیب حرکت نکند.

علاوه بر این توابع دیگری از جملهreset تغیر کردند تا مار های برنده مکان خود را از دست ندهد و به این صورت مشاهده شد، مار ها بسیار بهتر عمل کردند.

علاوه بر این موارد فایل main تغیرات جزئی داشت که جهت ذخیره و نشان دادن نتایج بود، مهم ترین تغیر اضافه کردن episode بود که به این صورت بتوانیم عملکرد مار ها را بهتر plot کرده و نشان دهیم.

در اخر یک فایل plot_rewards.py نوشته شده است تا عملکرد مار ها بهتر نشان داده شوند.



در بالا عملکرد دو مار
یکسان با get_optimal_policy غیر simple امده است که نشان دهنده تلاش مار برای بقای بیشتر میباشد و همچنین بازی در تعداد بالای episode دیگر به صورت شانسی نیست و همچنین اثر کمتر شدن epsilon نیز دیده میشود که مار ها پس از شناخت کافی از محیط کمتر به اکتشاف میپردازند.

راهکار برای توسعه و بهبود پروژه:

- افزایش تعداد مار ها
- اضافه کردن option بازی با انسان برای بهتر تحلیل کردن حرکات agent
- دادن اطلاعات در مورد مار های رقیب (که البته فضای حالت زیادی استفاده خواهد شد.)