بسم الله الرحمن الرحیم

پروژه ۶ درس هوش مصنوعی

دکتر فدایی و دکتر یعقوب زاده

مهدی وجهی

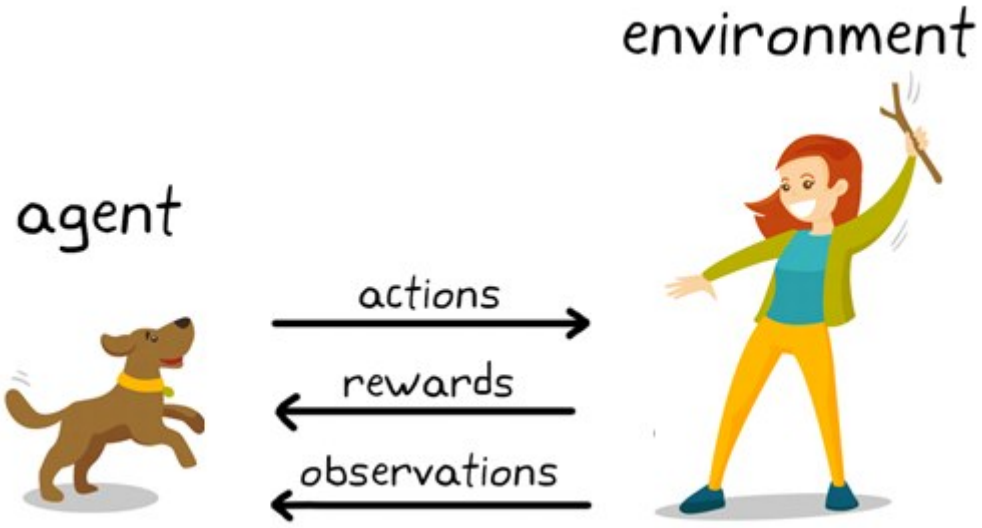
۸۱۰۱۰۱۵۵۸

فهرست

مقدمه 2
پیاده سازی 3
تعریف state ها 3
وضعیت مربع ۳×۳ اطراف 3
جهت سیب 4
تابع پاداش 5
Q-learning 5
انتخاب policy 5
انتخاب مسیر 5
بروزرسانی جدول Q 5
اجرا مدل 6
تلاش اول (کد ۶) 6
تلاش دوم (کد ۵) 6
تلاش سوم (کد ۷) 7

مقدمه

در این پروژه ما با یادگیری تقویتی آشنا می شویم. در این روش مدل در زمان تعامل با محیط می آموزد که چطور و چگونه باید عمل کند و با توجه به تابع پاداشی که برای آن نوشتیم متوجه کار های خوب و بد خود می شود. در این پروژه با استفاده از این روش می خواهیم مدلی را آموزش دهیم که بتواند در یک مار بازی دو نفره و رقابتی پیروز شود.



پیاده سازی

ابتدا نگاهی گذرا به مراحل پیاده سازی مدل می اندازیم.

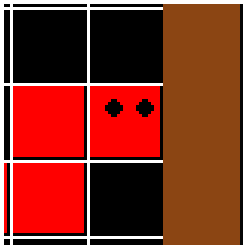
تعریف state ها

با توجه به این که اگر ما بخواهیم تمامی خانه ها را در وضعیت خود دخیل کنیم بسیار مدل بزرگ و ناکارآمدی داریم باید این وضعیت ها را کاهش دهیم. برای بخش غیر رقابتی با توجه به این که عملکرد مدل باید مشهود باشد از بازی تدافعی یا وحشیانه صرف نظر می کنیم و مدل را به صورتی آموزش می دهیم که صرفا سعی در خوردن سیب داشته باشد و از خشونت به دور باشد.
با توجه به موارد گفته شده، وضعیت های ما شامل ۲ مورد است:

1. وضعیت مربع ۳×۳ اطراف
2. جهت سیب

وضعیت مربع ۳×۳ اطراف

برای این موضوع ما مربع ۳×۳ به مرکزیت کله مار را در نظر میگیریم سپس اگر مانعی در خانه باشد آن را با ۰ علامت میزنیم و در غیر این صورت با ۱، مانند شکل:



۱	۱	۰
۰	۰	۰
۰	۱	۰

برای این موضوع ابتدا باید موقعیت مربع های ۳ در ۳ را پیدا کنیم برای این کار یک آرایه به صورت زیر درست می کنیم.

```
array([-1,  0,  1])
```

که این کار با قطعه کد زیر انجام می شود.

```
        distance = (size - 1) // 2
        tmp = np.array(range(distance + 1))
        tmp = np.union1d(tmp, -tmp)
```

سپس با جمع زدن مختصات کله مار در ۲ راستای x,y می توانیم مختصات مربوطه را به دست بیاوریم. حال مختصات رو چک می کنیم که خارج محدوده نباشه و سپس می بینیم بدن مار خود یا حریف در خانه هست یا نه در غیر این صورت خانه را خالی در نظر می گیریم.

```
        output = []
        for i in tmp + self.head.pos[0]:
            for j in tmp + self.head.pos[1]:
                if i < 1 or i >= ROWS - 1 or j < 1 or j >= ROWS - 1:
                    output.append(0)
                elif (i, j) in list(map(lambda z: z.pos, self.body)):
                    output.append(0)
                elif (i, j) in list(map(lambda z: z.pos, other_snake.body)):
                    output.append(0)
                elif (i, j) == other_snake.head.pos:
                    output.append(0)
                else:
                    output.append(1)
```

جهت سیب

برای این که مار به سمت سیب حرکت کند در موقعیت های جهت سیب را به مار می دهیم برای این کار می بینیم که مار در کدام راستا فاصله بیشتری دارد و جهت مربوطه را برمی گردانیم.

```
    def calc_snake_side(self, snack):
        # calc snake side in relation to the snake
        if abs(snack.pos[0] - self.head.pos[0]) > abs(snack.pos[1] -
self.head.pos[1]):
            if snack.pos[0] < self.head.pos[0]:
                return 0
            if snack.pos[0] > self.head.pos[0]:
                return 1
        else:
            if snack.pos[1] < self.head.pos[1]:
                return 2
            if snack.pos[1] > self.head.pos[1]:
                return 3
        return -1
```

تابع پاداش

تابع پاداش را به صورت زیر تعریف می کنیم.

برد	۵
باخت به هر روش	-۲۰۰۰
نزدیک شدن به سیب	۱۵
خوردن سیب	۵۰۰

Q-learning

انتخاب policy

برای انتخاب بهترین مسیر بین انتخاب های مختلف بالاترین q را انتخاب می کنیم که همان argmax آن state می شود.

انتخاب مسیر

برای این کار با احتمالی شناسی حرکت می کنیم و یا با توجه به جدول تصمیم می گیریم. اگر این وضعیت در استیت ها موجود نبود به صورت تصادفی انتخاب می کنیم.

بروزرسانی جدول Q

برای این کار همان فرمول کلاس را پیاده می کنیم.

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

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

فقط وقتی استیتی نبود آن را به صفر تعریف می کنیم و سپس با q جهت سیب را ۱ می دهیم.

```
        sample = reward + self.discount_factor *
np.max(self.q_table[next_state])
        self.q_table[state][action] += self.lr * (sample -
self.q_table[state][action])
```

اجرا مدل

تلاش اول (کد ۶)

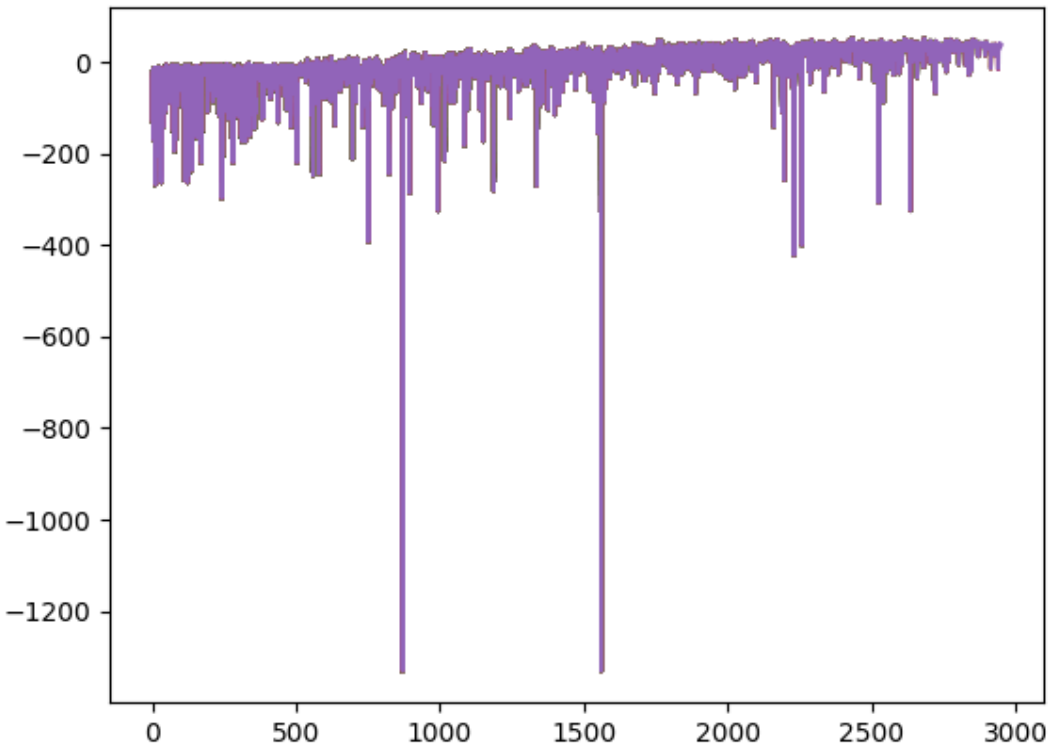
مدل را با پارامتر های زیر اجرا می کنیم.

```
self.lr = 0.6
self.discount_factor = 0.7
self.epsilon = 1
```

```
EAT_REWARD = 500
LOSE_REWARD = -2000
WIN_REWARD = 5
SNACK_RATE = 15
```

```
if len(self.hist) % 50 == 49:
    self.lr *= 0.97
    self.epsilon *= 0.92
    print(self.lr, self.epsilon)
```

نمودار میانگین امتیاز مدل در هر اجرا به صورت زیر است.



تلاش دوم (کد ۵)

این مدل دقیقا با همان پارامتر های مدل اول اجرا شده فقط ضریب یادگیری و اپسیلون ثابت است و به صورت دستی تغییر می کند.

این مدل عملکرد بهتری دارد.

تلاش سوم (کد ۷)

در این قسمت مقداری پارامتر ها را تغییر می دهیم اما این دفعه مدل را روی مدل کد ۶ اجرا می کنیم. مدل را با پارامتر های زیر اجرا می کنیم.

```
self.lr = 0.3
self.discount_factor = 0.8
self.epsilon = 0.5
```

```
EAT_REWARD = 500
LOSE_REWARD = -2000
WIN_REWARD = 1000
SNACK_RATE = 15
```

```
if len(self.hist) % 100 == 99:
    self.lr *= 0.95
    self.epsilon *= 0.92
    print(self.lr, self.epsilon)
```

نمودار میانگین امتیاز مدل در هر اجرا به صورت زیر است.