HW5

ML | spring 1401 | Dr.Abdi

Student name : **Amin Fathi**

Student id : **400722102**

CartPole-v1 یکی از محیط های OpenAI است که متن باز است. در آن یک میله به یک قطب فلزی در محیط بدون اصطحکاک متصل است و قطب در تلاش است با به چپ و راست رفتن میله را ثابت نگه دارد. تنها نیرو های وارده 1+ و 1- هستند که به ترتیب به حرکت به چپ یا راست اطلاق می شوند .اگر چکش (همان پاندول) بیشتر از ۲.۴ واحد از مرکز محیط حرکت کند اپیزود ها تمام می شود. و یا حتی اگر زاویه پاندول بیش از ۱۲ درجه باشد اپیزود تمام می شود. اگر اپیزودی تمام نشود مقدار پاداش برای هر دوره زمانی 1+ است.

همانطور که میدانیم model free qlearning است و صرفا بر اساس حالت های آموخته قبلی حرکت های آینده را بررسی می کند، این اطلاعات در جدول Q ذخیره می شوند . برای هر اقدامی که از یک وضعیت انجام می‌شود، جدول باید شامل یک پاداش مثبت یا منفی باشد. مدل با یک اپسیلون ثابت شروع می‌شود که نشان‌دهنده‌ی تصادفی‌سازی حرکات است. با گذشت زمان، تصادفی‌سازی براساس مقدار اپسیلون کاهش می‌یابد.

این شکل از یادگیری زمانی خوب است که تعداد حرکات محدودی وجود داشته باشد یا محیط پیچیدگی نداشته باشد؛ زیرا عامل حرکات گذشته را به خاطر می‌آورد و به راحتی آن‌ها را تکرار می‌کند.

برای محیط‌های پیچیده‌تر به دلیل زیاد بودن حالات، جدول Q به سرعت پر می‌شود. این می‌تواند زمان آموزش را افزایش دهد. Q-Learning پیش‌بینی نمی‌کند، بلکه در اکثر مواقع به‌صورت مطلق انجام می‌شود. معادله آن به شکل زیر است :

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

مقدار Q نشان‌دهنده‌ی کیفیت یک مقدار است، یا اینکه عملکرد در الگویتم چقدر خوب است. هر چه مقدار کیفیت بالاتر باشد، احتمال بیشتری وجود دارد که همان عمل دوباره انجام شود. کیفیت عمل با Qnew(st, at) نشان داده می‌شود، جایی که st نشان دهنده‌ی حالت و at نشان‌دهنده‌ی عمل است. این مدل با استفاده از گاما مقادیر جدید را کاهش می‌دهد و فرآیند عمل، هر مرحله را بر اساس نرخ یادگیری یا همان لرنینگ ریت تنظیم می‌کند.

متن بالا ترجمه ای بود از این دو لینک :

GitHub - RJBrooker/Q-learning-demo-Cartpole-V1

Using Q-Learning for OpenAI’s CartPole-v1 | by Ali Fakhry | The Startup | Medium

حال برای حل مسئله طبق لینک بالا

ابتدا کتابخانه های مد نظر را آپلود می کنیم :

```
import numpy as np # used for arrays
import gym # pull the environment
import time # to get the time
import math # needed for calculations
```

سپس یک محیط cartpole می سازیم :

```
env = gym.make('CartPole-v1')
```

با دستور زیر میفهمیم ۲ اکشن داریم .

```
print(env.action_space.n)
```

```
2
```

ما چهار نوع مشاهدات در این مساله داریم که رنج آن ها به شکل زیر است :

```
Observation:
    Type: Box(4)
    Num     Observation               Min                     Max
    0       Cart Position             -4.8                    4.8
    1       Cart Velocity             -Inf                    Inf
    2       Pole Angle                -0.418 rad (-24 deg)    0.418 rad (24 deg)
    3       Pole Angular Velocity     -Inf                    Inf
```

اکشن ها در دسترس هم همانطور که توضیح داده شد برابر است با :

```
Actions:
    Type: Discrete(2)
    Num   Action
    0     Push cart to the left
    1     Push cart to the right
```

حال به سراغ تعریف متغیر ها می رویم ، rate learning در واقع بیان میکند چقدر اطلاعات جدید جانشین اطلاعات قبلی می شود که صفر به معنای آن است که اطلاعات جدید یاد گرفته نشده است و یک به معنای آن است که اطلاعات قدیمی دور انداخته شده و اطلاعات جدید جانشین آن ها شده اند

مقدار DISCOUNT بین صفر و ۱ است و نشان دهنده میزان اهمیت دادن به پاداش آینده دار و طولانی مدت نسبت به پاداش فوری و آنی است. RUN تعداد کل تکرارها است . SHOW_EVERY نشان می‌دهد که راه‌حل فعلی چندبار اجرا شود.
UPDATE_EVERY هم نشان می دهد که هر چند وقت یکبار progress مان ثبت شود

```
LEARNING_RATE = 0.1
DISCOUNT = 0.95
RUNS = 1000
SHOW_EVERY = 200
UPDATE_EVERY = 100
```

حال به سراغ ساخت بین ها و جدول q می رویم ، متغیر numBin که مشخص‌کننده‌ی طول هر بازه است را ۲۰ ست میکنیم.

حال observation هایی که بالا توضیح داده شد را با توجه به پیوسته بودنشان به ۲۰ تکه تقسیم کرده و گسسته سازی می کنیم.و درادامه هم جدول Q را میسازیم با سایزی که در زیر مشاهده می شود .

```
# Get the size of each bucket
bins = [np.linspace(-4.8, 4.8, numBins),
    np.linspace(-4, 4, numBins),
    np.linspace(-.418, .418, numBins),
    np.linspace(-4, 4, numBins)]
qTable = np.random.uniform(low=-2, high=0, size=([numBins] * obsSpaceSize + [env.action_space.n]))
```

با توجه به اطلاعات بالا حالا در تابع زیر گسسته سازی را تکمیل میکنیم :

```
def get_discrete_state(state, bins, obsSpaceSize):
    stateIndex = []
    for i in range(obsSpaceSize):
        stateIndex.append(np.digitize(state[i], bins[i]) - 1) # -1 will turn bin into index
    return tuple(stateIndex)
```

حال معیار های ارزیابی را تعریف کرده :

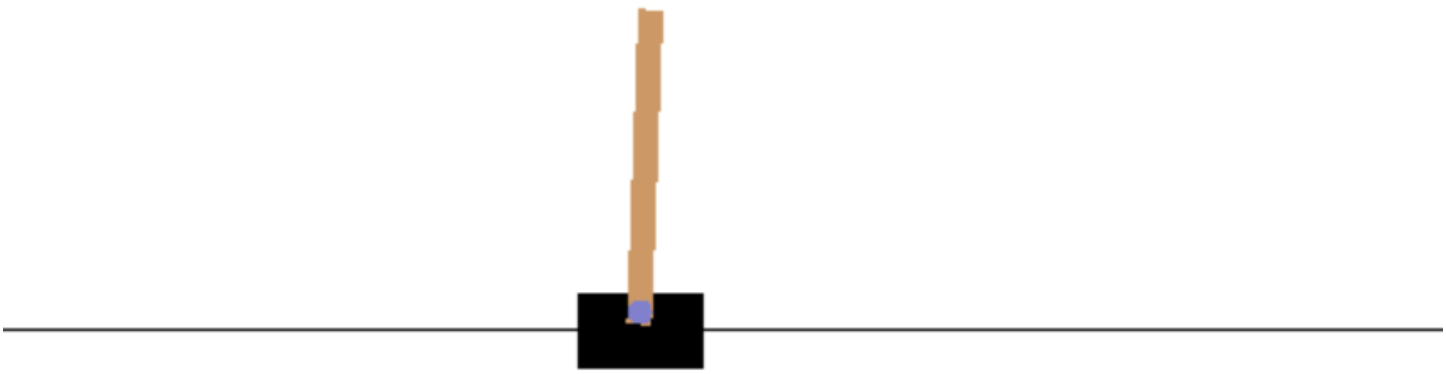
```
previousCnt = []  # array of all scores over runs
metrics = {'ep': [], 'avg': [], 'min': [], 'max': []}  # metrics recorded for graph
```

در ادامه حلقه اصلی را ران میکنیم که به تعداد اجرای های ست کرده در ابتدای کد اجرا می شود و هر بار ابتدا محیط و مشاهدات را گسسته سازی می کند و سپس done flag را false می کند تا نشان بدهد اپیزود هنوز اتمام نشده . برای ادامه کار و انتخا کردن اکشن از جدولQ استفاده می کنیم و در ابتدا که یک حالت رندم انتخاب می کنیم وگرنه با argmax بهترین اکشن انتخاب می شود و سپس اکشن را روی محیط اعمال می کنیم . در انتهای اپیزود ، done flag را تغییر میدهیم و و جایزه یا reward را مشخص می کنیم و محیط گسسته جدید را هم به دست می آوریم برای اکشن بعدی . و این چرخه ادامه پیدا میکند و مقادیر جدول متعاقبا آپدیت می شوند

```python
for run in range(RUNS):
    discreteState = get_discrete_state(env.reset(), bins, obsSpaceSize)
    done = False  # has the enviroment finished?
    cnt = 0  # how may movements cart has made

    while not done:
        if run % SHOW_EVERY == 0:
            env.render()  #if running RL comment this out
        cnt += 1
        # Get action from Q table
        if np.random.random() > epsilon:
            action = np.argmax(qTable[discreteState])
        # Get random action
        else:
            action = np.random.randint(0, env.action_space.n)
        newState, reward, done, _ = env.step(action)  # perform action on enviroment

        newDiscreteState = get_discrete_state(newState, bins, obsSpaceSize)

        maxFutureQ = np.max(qTable[newDiscreteState])  # estimate of optiomal future value
        currentQ = qTable[discreteState + (action, )]  # old value

        # pole fell over / went out of bounds, negative reward
        if done and cnt < 200:
            reward = -375

        # formula to caculate all Q values
        newQ = (1 - LEARNING_RATE) * currentQ + LEARNING_RATE * (reward + DISCOUNT * maxFutureQ)
        qTable[discreteState + (action, )] = newQ  # Update qTable with new Q value

        discreteState = newDiscreteState
    previousCnt.append(cnt)
```

```python
# Decaying is being done every run if run number is within decaying range
if END_EPSILON_DECAYING >= run >= START_EPSILON_DECAYING:
    epsilon -= epsilon_decay_value
# Add new metrics for graph
if run % UPDATE_EVERY == 0:
    latestRuns = previousCnt[-UPDATE_EVERY:]
    averageCnt = sum(latestRuns) / len(latestRuns)
    metrics['ep'].append(run)
    metrics['avg'].append(averageCnt)
    metrics['min'].append(min(latestRuns))
    metrics['max'].append(max(latestRuns))
    print("Run:", run, "Average:", averageCnt, "Min:", min(latestRuns), "Max:", max(latestRuns))
```

خروجی گرافیکی به شکل زیر است :



```
Run: 300 Average: 41.01 Min: 12 Max: 109
Run: 400 Average: 58.88 Min: 16 Max: 126
Run: 500 Average: 83.65 Min: 27 Max: 130
Run: 600 Average: 101.09 Min: 56 Max: 135
Run: 700 Average: 110.89 Min: 68 Max: 179
```

```
Run: 0 Average: 18.0 Min: 18 Max: 18
Run: 100 Average: 22.67 Min: 9 Max: 61
Run: 200 Average: 34.39 Min: 9 Max: 104
Run: 300 Average: 41.01 Min: 12 Max: 109
Run: 400 Average: 58.88 Min: 16 Max: 126
Run: 500 Average: 83.65 Min: 27 Max: 130
Run: 600 Average: 101.09 Min: 56 Max: 135
Run: 700 Average: 110.89 Min: 68 Max: 179
Run: 800 Average: 110.76 Min: 59 Max: 192
```