دانشگاه زنجان

پروژه:
پیاده‌سازی الگوریتم Hysteretic Q-Learning برای کار ball-balancing

دانشجو: سینا حیدری

تابستان ۱۴۰۱

مقدمه

در این گزارش، رویکرد Q-Learning hysteric برای یادگیری تقویتی سیستم‌های چندعامله توضیح داده شده است. ما یک آزمایش برای کار ball-balancing پیاده‌سازی کرده‌ایم. در این کار دو عامل که دو سر یک سطح صاف را کنترل می‌کنند باید با action های خود تعادل توپ را بر روی این سطح صاف حفظ کنند. برای روش Q-Learning باید یک فضای حالت گسسته در اختیار داشته باشیم. فضای حالت با توجه به محیط مشخص شده در خود مقاله گسسته شده است. برای مثال، موقعیت توپ بر روی سطح صاف به صد مقدار در بازه ۱- تا ۱ گسسته شده است. در بخش اول این گزارش، آموزش عامل‌ها و در بخش دوم، آزمایش عامل‌های آموزش داده شده بررسی شده است. در بخش سوم نیز روش اجرای پیاده‌سازی و دریافت خروجی‌ها را توضیح داده‌ایم.

۱ آموزش عامل‌ها

فضای حالت و فضای action به شکل زیر گسسته شده است:

- Sampling time: ۰.۰۳ seconds
- State space: برای موقعیت صد مقدار بین ۱- و ۱. برای سرعت پنجاه مقدار بین ۳- و ۳.
- برای action ها پانزده مقدار بین ۱- و ۱.

با این تفاسیر ابعاد Q-table ها به صورت ۱۰۰ × ۵۰ × ۱۵ خواهد بود.
منطق مورد نیاز برای آموزش عامل‌ها در تابع trainHysteretic قرار داده شده است. محتوای این تابع را در پایین مشاهده می‌کنید:

```
def trainHysteretic():
  # create q-Table
  qTable1 = create_table()
  qTable2 = create_table()
  qTables = [qTable1, qTable2]

  iterationRewards = []
  for i in range(5):
    rewardSumInTrial = []
    for trial in range(trials):
      progress(trial, trials, prefix='Iteration: ' + str(i))
      # Initialize states (x,xbar)
      states = (0.495, 1.041)
      rewardSum = 0

      for t in np.arange(0, 20, samplingTime):
```

```
17
18          new_actions = nextAction(states, actions, qTables, trial,
    numOfEps=40, trials=trials)
19
20          x, v = nextState(h1=new_actions[0], h2=new_actions[1], v=states
    [1], t=samplingTime, x_0=states[0],
21  v_0=states[1])
22
23          # deal with velocities more than 3 and less than -3
24          if v > 3: v = 3
25          if v < -3: v = -3
26
27          if np.abs(x) > 1: break
28
29          thisReward = reward(states[0], states[1])
30          rewardSum = rewardSum + thisReward
31
32          new_states = (np.round(x, decimals=decimals), np.round(v,
    decimals=decimals))
33          # check if the new states are in discretized states
34          new_states = isValidState(new_states)
35
36          qTables = hysteretic(qTables, states, new_actions, alpha, beta,
     thisReward, gamma, new_states)
37          states = new_states
38
39       rewardSumInTrial.append(rewardSum)
40
41     iterationRewards.append(rewardSumInTrial)
42   mean_output = np.mean(iterationRewards, axis=0)
43   plt.plot(list(range(trials)), mean_output, '-')
44   plt.title('Hysteretic')
45   plt.savefig('./Plots/Hysteretic.png')
46   plt.clf()
47
48   pkl.dump(qTables[0], open('QTables/qT1_Hysteretic.p', 'wb'))
49   pkl.dump(qTables[1], open('QTables/qT2_Hysteretic.p', 'wb'))
50
```

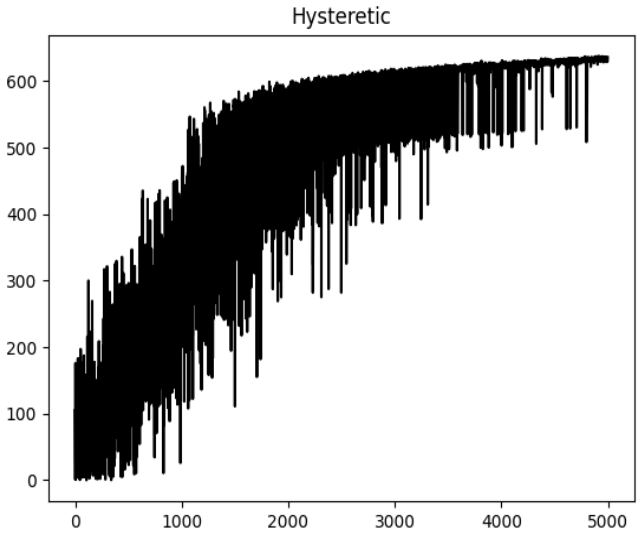
در این تابع در ابتدا دو نمونه Q-table (یکی برای هر عامل) ساخته شده است. در حلقه‌ای که در خط ۱۰ مشاهده می‌شود، منطق مربوط به هر trial جا داده شده است. در هر trial که بیست ثانیه طول می‌کشد (sampling time=0.03) عامل‌ها یک action انجام می‌دهند و آن action توپ را به حالت بعدی می‌برد. انتخاب action با استفاده از روش ε-greedy انجام شده است. دینامیک مورد نیاز برای به دست آوردن حالت بعدی با استفاده از معادله زیر به دست می‌آید:

$$m\ddot{x} = -c\dot{x} + mg\left(\frac{h_1 - h_2}{l}\right)$$

سپس با توجه حالت جدید (موقعیت و سرعت) عامل‌ها reward دریافت می‌کنند. reward برای هر حالت با توجه به فرمول زیر به دست می‌آید:
سپس با توجه به reward دریافت شده و الگوریتم Q-Learning hysteretic جدول Q-table برای

$$r = 0.8e^{-\frac{x^2}{0.25^2}} + 0.2^{-\frac{x^2}{0.25^2}}$$

هر عامل به روزرسانی می‌شود. reward در هر واحد زمانی محاسبه شده و مجموع آن برای هر trial ذخیره می‌شود. تعداد trial ها مثل مقاله ۵۰۰۰ در نظر گرفته شده است. ما پنج بار این تعداد trial ها را تکرار می‌کنیم و از آن میانگیری می‌کنیم (دقیقا مثل مقاله با این تفاوت که در مقاله بیست بار تکرار شده است). نمودار تشویق‌ها بعد از آزمایش را در تصویر زیر مشاهده می‌کنید:



همانطور که در تصویر بالا مشاهده می‌کنید در trial های بعدی reward بیشینه شده است. این یعنی الگوریتم Q-Learning hysteretic کار اصلی‌اش که maximization reward است را درست انجام داده است.

۲ آزمایش عامل‌ها

منطق تست عامل‌ها در تابع test قرار داده شده است. کد این تابع را در پایین مشاهده می‌کنید:

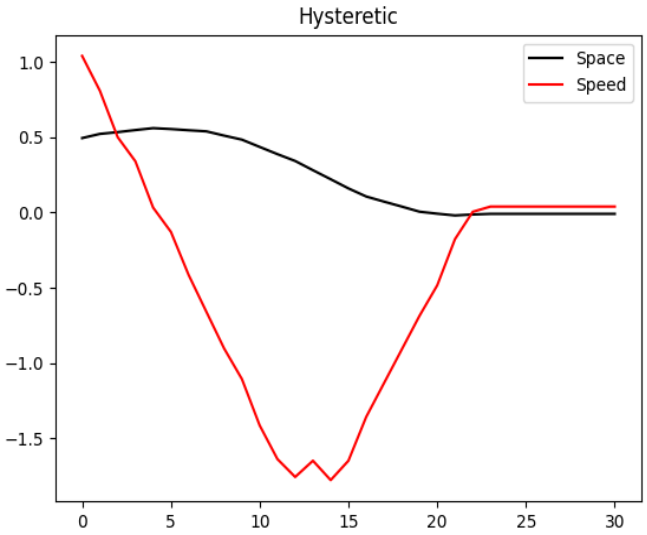
```python
def test(qTables):
  spaces, velocities = [], []
  states = (0.495, 1.041)
  spaces.append(states[0])
  velocities.append(states[1])

  for i in range(30):
    actions = nextAction(states, None, qTables, None)
    x, v = nextState(h1=actions[0], h2=actions[1], v=states[1], t=0.03,
     x_0=states[0], v_0=states[1])
    spaces.append(x)
    velocities.append(v)
    # if the ball falls from the flat surface
    if np.abs(x) > 1:
    break
    new_states = (np.round(x, decimals=3), np.round(v, decimals=3))
    new_states = isValidState(new_states)
    states = new_states

  plt.plot(spaces, '-', label="Space", color='black')
  plt.plot(velocities, '-', label="Speed", color='red')
  plt.legend()
  plt.title("Hysteretic")
  plt.savefig('./Plots/' + "Hysteretic" + '_test.png')
  plt.clf()

```

برای آزمایش عامل‌ها همانند مقاله ابتدا از یک وضعیت اولیه شروع کرده و در ۳۰ قدم عامل با توجه حالت یک تصمیم می‌گیرد. با توجه به تصمیم عامل وضعیت توپ تغییر می‌کند. نمودار مربوط به آزمایش عامل‌ها را در تصویر زیر مشاهده می‌کنید:



همانطور که مشاهده می‌کنید، به مرور زمان توپ در وسط سطح صاف و با سرعت پایین balance شده

است.

۳ اجرای پیاده‌سازی

برای اجرای پیاده‌سازی ابتدا در پوشه code دستور زیر را اجرا کنید تا پکیج‌های مورد نیاز نصب شوند:

```
pip install -r requirements.txt
```

سپس برای آموزش عامل‌ها دستور زیر وارد شود:

```
python training.py
```

و برای آزمایش عامل‌ها دستور زیر وارد شود:

```
python test.py
```