로봇인지및지능제어 Final-Exam

1970030 박근아

1. Multi-Armed Bandit

MAB 환경에서 노래 장르 추천 서비스를 구현하고 실험하였다.
총 클래식, 팝, 재즈, R&B, 힙합 다섯 가지의 장르 중에서 음원 스트리밍 플랫폼 사용자가 어떤 장르의 노래를 듣는지 파악하고, 음원 스트리밍 플랫폼에서 이 정보를 활용하여 사용자가 즐겨듣는 취향인 장르의 노래들을 추천하는 서비스를 만들 수 있다.

우선 클래식, 팝, 재즈, R&B, 힙합 다섯 장르를 0, 1, 2, 3, 4 로 지정하고 사용자의 음원 청취 기록을 나타낸 표는 다음과 같다. 이때 0은 사용자가 해당 장르의 음원을 실행하지 않은 경우, 1은 실행한 경우의 보상이다.

	classic	pop	jazz	r&b	hiphop
0	1	1	1	1	1
1	0	1	0	1	0
2	1	0	1	0	1
3	0	0	1	1	0
4	1	1	1	0	1
5	0	1	0	1	0
6	0	0	1	0	0
7	1	0	0	0	1
8	1	0	1	0	1
9	0	0	1	0	1

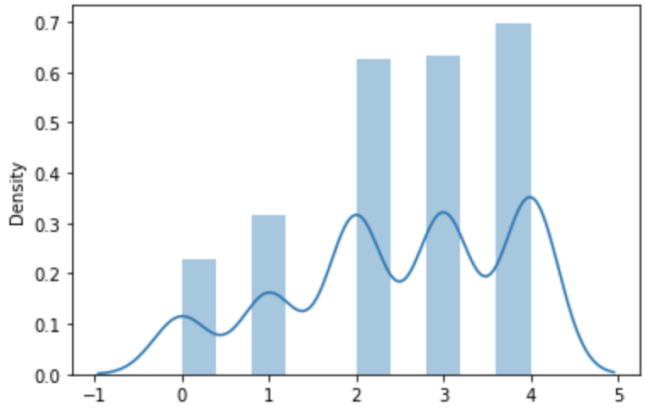
어떤 경우의 reward가 가장 높았는지 확인하기 위해 df.sum() 함수를 이용해 열별 합을 구해주었고, 결과는 다음과 같이 pop이 가장 높았다.

```
classic    5077
pop        5098
jazz       5036
r&b        5006
hiphop     4987
dtype: int64
```

Epsilon-greedy 정책을 이용하여 장르를 선택하고자하였고 이때 1000 번 정안 for문을 돌려주며 ε이 0.5 일때의 탐색결과는 다음과 같다.

Q = [0.45054945 0.48031496 0.504 0.45059289 0.51971326]
count = [91. 127. 250. 253. 279.]
sum_rewards = [41. 61. 126. 114. 145.]
사용자에게 추천할 최적의 장르는 hiphop이다.

우선 아래 그래프이미로 확인할 수 있듯 가장 많이 선택된 장르는 4번인 hiphop 이고 선택된 수 대비 reward가 높았던 것 역시 hiphop 임을 Q 값을 통해 확인할 수 있다.



따라서 프로그램 종료시에 사용자에게 추천할 최적의 장르를 Q값을 이용해 출력해줄 수 있다.

하지만 이는 초반에 확인한 reward 결과와 같기 때문에 for문에서 확인하는 범위인 0~999행까지의 reward 합을 다시 구해보았다.

```
In [26]: df2 =df.iloc[0:1000]

In [27]: df2.sum()

Out[27]: classic     516
         pop         519
         jazz        500
         r&b         491
         hiphop      501
         dtype: int64
```

이때도 최적의 추천장르인 hiphop과 reward 합의 최대인 pop이 일치하지는 않지만 이는 epsilon-greedy 정책에서 장르를 선택함에 있어 나타난 랜덤성으로 인해 나타난 결과로 해석할 수 있다.

결론적으로 해당플랫폼에서는 사용자에게 hiphop 장르의 음악을 추천하여 사용자 맞춤 서비스를 구축할 수 있다.

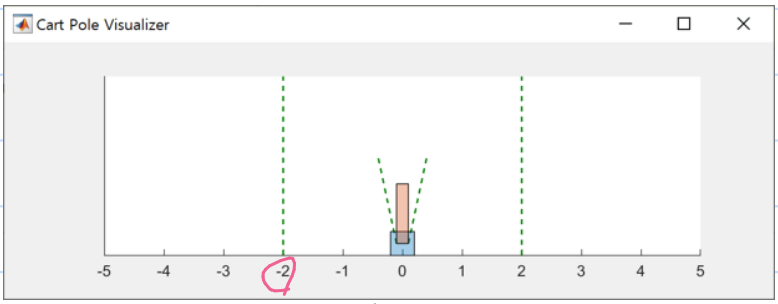
2. Actor-Critic

A3C 알고리즘을 이용하여 Matlab에서 Cart Pole 환경의 training을 진행해보았다.
이때 조금 더 극단적인 환경에서 training을 진행하기 위해
XThreshold 값은 2.4 → 2 로, PenaltyForFalling 값은 -5 → -15 로
변경해주었다.

```
env =

  CartPoleDiscreteAction - 속성 있음:

                 Gravity: 9.8000
                MassCart: 1
                MassPole: 0.1000
                  Length: 0.5000
                MaxForce: 10
                      Ts: 0.0200
    ThetaThresholdRadians: 0.2094
               XThreshold: 2
      RewardForNotFalling: 1
       PenaltyForFalling: -15
                   State: [4×1 double]
```



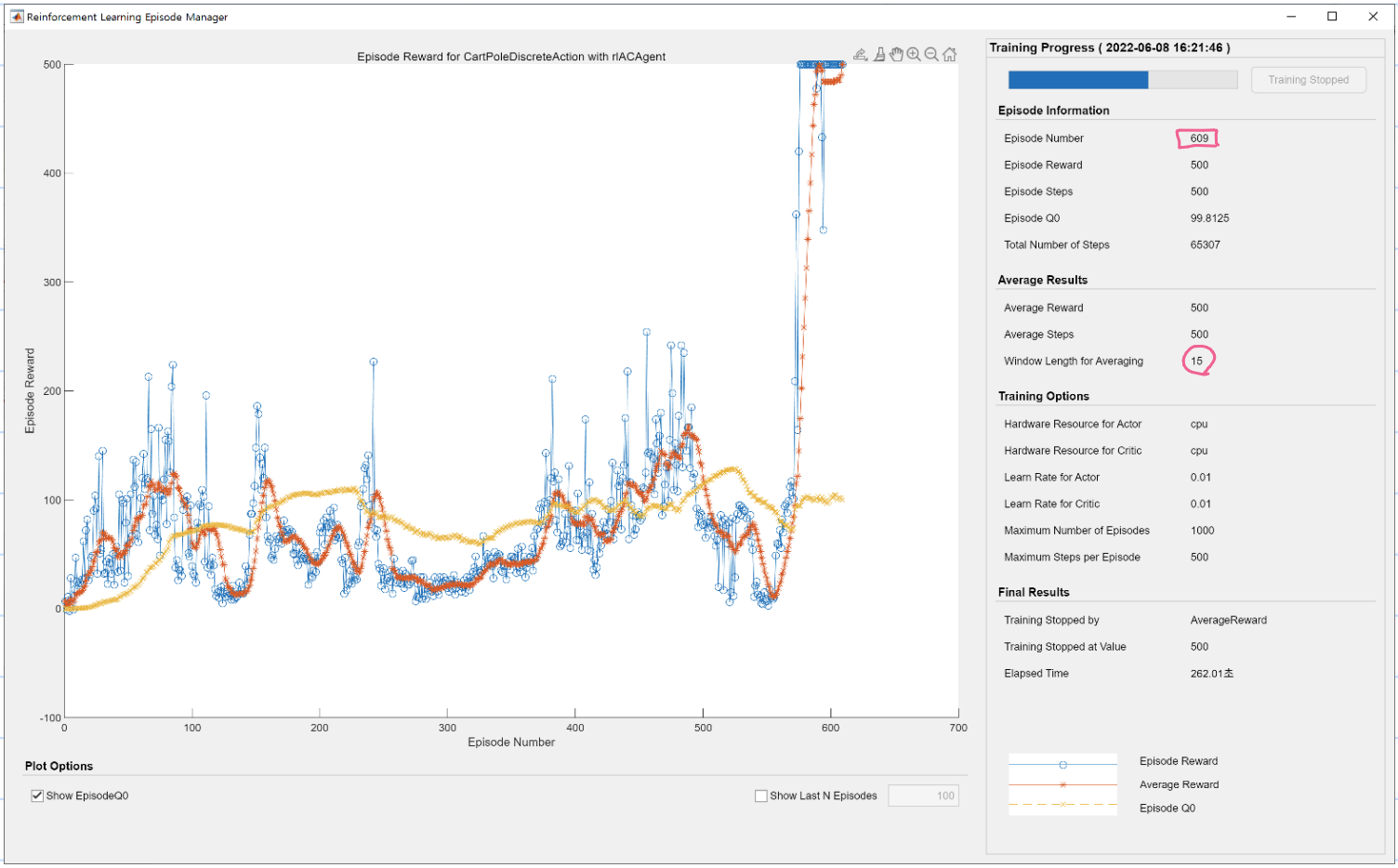
* threshold가 바뀐모습

또한 traing option 부분에서 최대 reward인 500이 10번이상 → 15번 까지
나와야 training을 마치는 것으로 수정했다.

```
%% Parallel Training Options

trainOpts = rlTrainingOptions(...
    'MaxEpisodes',1000,...
    'MaxStepsPerEpisode', 500,...
    'Verbose',true,...
    'Plots','training-progress',...
    'StopTrainingCriteria','AverageReward',...
    'StopTrainingValue',500,...
    'ScoreAveragingWindowLength',15);
```

Training을 진행한 과정은 다음과 같다.



앞서 수정해준 training option에 따라 max reward인 500이 15번 나온이후 train을 멈추었고 이에 대한 출력 결과는 다음과 같다.

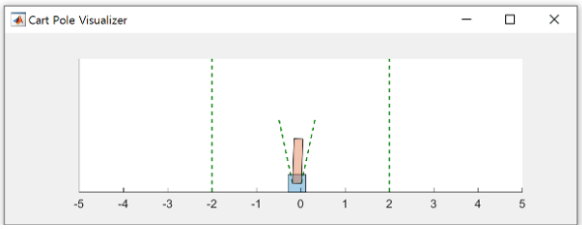
```
Episode: 595/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 58307 | Episode Q0:   98.68
Episode: 596/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 58807 | Episode Q0:   98.50
Episode: 597/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 59307 | Episode Q0:   97.05
Episode: 598/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 59807 | Episode Q0:   97.59
Episode: 599/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 60307 | Episode Q0:   99.08
Episode: 600/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 60807 | Episode Q0:  103.54
Episode: 601/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 61307 | Episode Q0:  102.35
Episode: 602/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 61807 | Episode Q0:  102.34
Episode: 603/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  483.93 | Step Count: 62307 | Episode Q0:  104.50
Episode: 604/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  485.40 | Step Count: 62807 | Episode Q0:  104.19
Episode: 605/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  485.40 | Step Count: 63307 | Episode Q0:  100.22
Episode: 606/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  485.40 | Step Count: 63807 | Episode Q0:  101.33
Episode: 607/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  485.40 | Step Count: 64307 | Episode Q0:  102.38
Episode: 608/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  489.87 | Step Count: 64807 | Episode Q0:  101.26
Episode: 609/1000 | Episode Reward:  500.00 | Episode Steps:  500 | Average Reward:  500.00 | Step Count: 65307 | Episode Q0:   99.81
```

Training이 잘 되어서 simulation에서는 max step을 1000으로 바꾸어 실행해보았고 결과적으로 1000 step까지 Cart pole이 잘 버티는 것을 확인할 수 있었다.

```
>> plot(env)
%%
simOptions = rlSimulationOptions('MaxSteps',1000);
experience = sim(env,agent,simOptions);
totalReward = sum(experience.Reward)

totalReward =

        1000
```



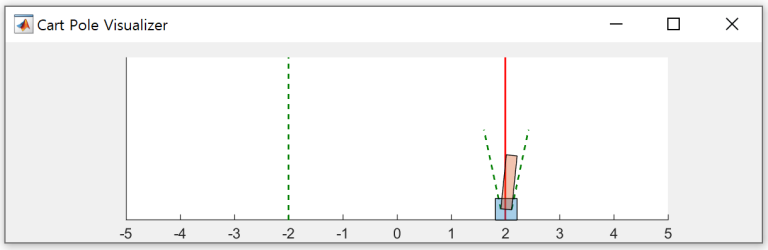
A3C 에서의 training 한 agent 성능이 좋아서 같은 환경에서 AC 알고리즘을 이용해 training 하고 그 결과를 서로 비교해보고자 한다.

A3C 알고리즘을 수행한 환경에서 AC training을 수행한 결과이다.
우선 training 이 1000번 안에 수렴하지 못한것을 확인할 수 있다. 또한 앞서 training이 262초 정도가 걸렸던 A3C에 비해 그 시간이 10배이상 필요한 것을 확인할 수 있었다. 그리고 AC로 train된 agent로 시뮬레이션을 실행한 결과 그 성능이 좋지않은 것을 알 수 있다.

```
>> plot(env)
simOptions = rlSimulationOptions('MaxSteps',1000);
experience = sim(env,agent,simOptions);
totalReward = sum(experience.Reward)

totalReward =

   146
```



두 알고리즘을 같은 환경에서 train 시켜본 결과 A3C 알고리즘의 장점이 확연하게 드러나는 것을 알수 있었다. 우선 한개가 아닌 4개의 워커를 이용하여 병렬연산을 수행하기 때문에 적은 계산량과 학습시간이 소요되고 알고리즘의 강화도 역시 높은 것을 확인할 수 있다.