Team 5

Data Science & Reinforcement Learning Final Report

권혁진, 서성민, 이성영

0. Abstract

본 레포트에서는 강화학습 수업 최종 프로젝트에 대한 Team 5의 시행착오들에 대해 다루었다. [Task1: Chain MDP]와 [Task2: Lava] 모두 Tabular Method 대신 Function Approximation Method만을 시도하였다. 결론적으로 Chain MDP는 3개의 Q-principle & Q-target 쌍을 이용한 **Ensemble-DQN**을 사용하였으며 세부적으로는 Q-function을 업데이트 과정에서 'soft update' 방법을 사용함과 동시에 'decaying ε-greedy' 방법으로 agent 학습을 진행하였다. Lava는 학습시간을 크게 감소시킬 수 있었던 **Coarse / Fine control PPO** 방법을 사용하였으며 자세한 내용은 후술하겠다.

구성은 크게 1. Chain MDP와 2. Lava에 대하여 각 문제에 대한 *1) 알고리즘의 변천과정* 과, *2) 최종 알고리즘에 대한 설명* 으로 이루어져 있다. 최종적인 3. 결론을 도출하고, 4. 부록 부분에 *코드 실행 결과* 를 정리하였다.

1. Chain MDP

1) Developing the Algorithm

Chain MDP 문제의 경우 좌우로만 움직일 수 있었다는 점이 'LAB 3'에서 다뤘던 Cartpole 예제와 매우 유사하다고 생각했고, 이에 **일반적인 Deep Q Network**(simple-DQN)을 적용하여 해결을 시도하였다. 이 경우에 대한 결과는 성능이 좋은 듯 보였으나 seed에 따라 최종 학습된 agent가 S_N에 도달하지 않는 경우가 있었다. 이를 exploration 과정에서 S_N에 충분히 도달한 sample의 수가 부족하다고 판단하였고 여러 parameter들을 수정해가며 학습을 시도하였다. 하지만 다양한 sample을 확보하기에는 무리가 있어 다른 모델을 구상하였다. 간단한 변형으로는 simple-DQN에 **decaying ε-greedy**와 **Q-function soft update** 기법을 추가여 시도해보았는데, 기존 방법에 비해 성능 발전이 조금 있었던 것으로 보였지만 큰 성능 향상으로는 이어지지 못했다.

이에 decaying ε-greedy와 Q-function soft update 방법은 유지하면서도 DQN에서 사용하는 Q-principle & Q-target을 여러 쌍 사용하는 **Ensemble-DQN**을 도입하게 되었다. 이 알고리즘을 활용해서 S_N에 도달하는 sample을 안정적으로 확보할 수 있었고, head의 개수와 masking 확률(Bernoulli probability)을 수정하여 optimal policy를 도출하였다.

2) Final Algorithm Explanation

Ensemble-DQN에서 가장 학습 결과에 직접적으로 영향을 주는 요인들은 ensemble number와 Bernoulli probability였다. Ensemble number를 통해 학습에 사용할 Q-function의 수 (head 수)를 조절하고 Bernoulli probability를 통해 update할 Q-function 선택의 랜덤성을 조절한다. Bernoulli probability의 경우 Ensemble number에 따라서 최적의 성능을 보

이는 값이 달라졌으며, Ensemble number의 경우도 무조건적으로 많기 보다는 적절한 값이 있었다. 초기에 찾아낸 Ensemble number는 5개였으나 학습 시간의 제한성 문제로 인해 그 이하의 값으로 최대한 도출하고자 하였다. 여러 시도를 통해 최종적으로 구한, S_N에 확실하게 도달하며 1000 episode에 1시간 전후로 학습이 완료되는 모델의 구조는 ***Ensemble number : 3개*** 와 ***Bernoulli probability : 0.5*** 였다.

알고리즘을 도입해 발전시키는 과정에서는 ε-greedy로 우선 학습한 후, 최종적으로 agent의 action 선택이 greedy하게끔 진행을 했었다. 그러나 실제 상황에서는 학습과 테스트 사이의 경계를 명확히 할 수 없으며, 출제자에 따라 학습할 episode를 공개하지 않을 경우도 있기 때문에 action 선택의 함수를 학습과 테스트에 대해서 동일하게 추출하도록 수정하였다. 이러한 경우에 발생하는 문제는 epsilon 값이 정확히 0이 아니라면 최종 performance가 완벽한 값(이 경우 10점)이 나오지 않을 가능성이 있었으며, sample efficiency 계산에 있어서도 최적의 policy가 도출된 이후에 같은 문제가 발생할 수 있었다. 따라서, **특정 조건에 따라 epsilon을 아예 0으로 낮추는 시도가 필요했다.**

최고의 효율을 얻어내기 위해서는 epsilon을 빠르게 감소시키며 Q-function 또한 최대한 이른 시점부터 update를 진행할 수 있도록 해야 한다. 따라서 학습 과정에서 S_N에 도달한 경우가 있는 순간부터 epsilon decay를 시작하고 decay 비율을 늘리며 initialized 값과 τ(tau) 값을 줄여서 학습의 빠른 시작과 학습 시간을 단축하며 최종 reward 값을 안정적으로 최대화하였다. 추가적으로 변화된 환경에 대한 적응력(adaptability)을 향상시키기 위해 수많은 시도를 했다. 우선 chain MDP의 경우 변화시킬 수 있는 environment의 가능성이 *chain 시작 위치* 와 *chain length* 뿐이라고 판단하였고, 자체적으로 환경을 변화시켜보며 그 성능을 검증해보았다. 시작 위치의 경우, 이미 기존 모델(simple-DQN)로도 해결이 가능했고, length의 경우는 확률적으로 **initialized 값**과 **tau**에 의해 학습 성공 여부가 좌우된다고 생각되어 이를 **chain length 값에 비례하여 조절할 수 있도록** 수정하였다.

하지만 현재의 알고리즘은 S_N 에 도달하는 방식을 random action에 의한 확률에 의존하고 있기 때문에 여전히 chain length가 길어지면 optimal policy를 도출하지 못하는 경우가 많았다. 따라서 exploration에 대한 bonus reward를 통해 새로운 state에 도달하는 action을 가치 있게 평가하도록 수정하였다. UCB와 비슷하게 각 state에 도달한 횟수를 누적한 값의 exponential 값에 반비례하도록 exploration bonus를 지급하여 학습에 사용하였다. 결과적으로 chain length의 2배 정도 되는 episode 수 만에 S_N에 도달하는 sample들을 확보할 수 있었고, 모든 state에 대한 학습이 가능해져서 optimal policy를 항상 도출할 수 있게 되었다.

2. Lava

1) Developing the Algorithm

Lava 문제도 chain MDP 문제와 같이 **simple-DQN**으로 처음 시도를 진행하였는데, 결과는 기대 이하였다. 여러 Hyperparameter들을 튜닝해가며 수 백 번 이상의 학습과정을 거쳤지만 최종 Goal에 안정적으로 도달하지는 못했고 초기 initialized 된 값에 큰 의존을 하

였다. 무엇보다 학습 과정에서 map에 존재하는 두 개의 라바를 우회할 때, 두 번째 라바에 넘지 못하는 모습을 보여주었고 최종 결과값에서는 *그림 DQN_result_1* 과 같이 epoch마다의 variation이 컸다.

최소 300 에포크 (대략 15-20분) 이상이 소요되는 긴 학습 시간과 더불어 경우에 따라 아예 학습이 되지 않는 경우도 존재했고, *그림 DQN_result_2* (10번의 trial 중 에피소드가 끝났을 때의 위치를 나타낸 배열이다. 가령 Iter830의 경우 10번 중 6번만 Goal에서 종료됨)와 같이 학습이 꽤나 진행되었음에도 불구하고 Goal에 안정적으로 도달하지는 못하였다.

이러한 과정들을 겪으며 좋지 못한 성능을 보이는 simple-DQN 대신 PPO 알고리즘을 선택하기로 했다. **일반적인 PPO**(simple-PPO)가 잘 될 것이라 예상한 이유는 바로 trust-region을 바탕으로 한 특성 때문이었다. lava를 탈출해 나가는 과정 속에서, 잘 나아가던 policy를 일부 얻어내면 이를 유지하며 나아갈 것이라 기대되었기 때문이다

하지만 실제 simple-PPO 알고리즘만을 갖고 training을 진행했을 때에는 학습이 완료될 때까지 소모되는 시간이 매우 길었다. 또한 최적의 policy가 학습되지 않는 경우가 생겼는데, 구체적으로 보자면 lava에는 빠지지 않지만 안전한 벽에 붙어 같은 자리를 무한 반복하도록 학습되었다. 따라서 더 효율적인 방법론을 고민하게 되었고, 그에 따라 생각해낸 방법이 바로 *"골 도달 전과 후의 학습 reward"* 에 차이를 둔 **Coarse/Fine Control PPO** 방법이다.

2) Final Algorithm Explanation

agent는 한 에포크 내에서 lava나 goal에 도달하지 않는 이상 max_step(이 경우 100)만큼 움직이도록 설정했다(따라서 한 에포크 내에 여러 개의 episode가 존재할 수 있다). 앞서 언급했듯 agent가 lava는 피하면서 안전한 자리를 반복해 도는 문제(ex: 아래로 3번 움직인 후 왼쪽으로 97번 이동해 제자리)가 발생하였고, 이를 해결하기 위해 "**agent가 특정 지점에 여러 번 도달하는 것을 방지하는**" 새로운 메커니즘을 추가했다.

우선 episode별로 agent의 모든 trajectory를 저장한다. 해당 episode를 모두 진행한 뒤 trajectory를 봤을 때, 임의의 점을 한 번 방문하면 **미약한 '+'** reward를, 2번 방문하면 **미약한 '-'** reward를, 그보다 더 많이 방문하면 **큰 '-'** reward를 추가하여 buffer에 저장하였다. 이때, 가해주는 변화는 환경으로부터 받는 실제 reward의 최소값으로 정해줌으로써 실제 scale을 반영해줄 수 있도록 하였다.

이후 환경을 reset시켜 다음 episode가 실행되게끔 했는데 이를 통해 같은 학습시간동안 더 유의미한 데이터를 바탕으로 학습을 진행할 수 있었다. 즉, **일종의 데이터 선별 과정이다**.

(Coarse Control)

Trajectory상 2번의 방문까지 일부 허용한 이유는, 1번의 방문만 허용한 경우, 두 번째 라바를 따라 올라가다 다시 왼쪽 아래로 쭉 내려가 terminate되는 경우가 있었기 때문이다. 그 후 PPO 알고리즘의 특성상 비약적인 trajectory 발전이 이루어지지 않고 local optimum에 빠져 골에 도달할 수 없는 경우가 생기는 것을 확인하였다. 2번의 허용만으로도 이런 *local-optima* 문제가 해결되었고, 3번 이상 같은 곳을 지날 시에는 큰 불이익을 주도록 구성했다(다만 오직 1번의 이동을 허용했을 때가 학습 속도 자체는 더 빨랐다).

에포크 내에서 여러 episode가 진행될 때, 목표 지점 도달 후에는 '*goal_reached*' 변수를 통해 학습모드를 전환하였다. 모드가 전환되면 데이터 선별 과정을 종료하고, environment로부터 받는 reward만으로 학습을 진행해 최적의 경로를 학습할 수 있도록 하였다.

(fine control)

Coarse control만으로는 trajectory상 한 번만 방문한 곳이 최대 개수가 되도록 학습이 진행되므로 높은 확률로 최적 경로로 골 위치에 도달하지 못한다. 이를 보완하기 위해 한 epoch 내에 골을 도달하면 임의로 buffer에 저장하는 reward를 조작하는 과정을 멈추고 본래 environment의 reward만을 받도록 한 것이다. 이 때의 에포크는 agent가 100 step을 모두 움직이지 않아도 goal에 도달하는 경우라면 terminate 되도록 설정하였다.

최종 정리를 하자면, [1]우선 목표 지점에 도달하기 전까지는 한 에포크 안에 agent의 100 step을 확정적으로 보장한 뒤 다음 에포크로 넘어가게 하였다(*AUC=epoch에서의 reward 총합 ÷ episode 수*). [2]그 후 에포크 내에서 한 번이라도 goal에 도달한 경우가 생기면, [2-a)]그 뒤 진행되는 episode가 목표 지점에 도달하는지 확인하고 [2-b)]그렇지 않다면 100 step을 다 움직이거나 혹은 목표에 도달할 때까지 해당 에포크를 진행한다(4000 스텝마다 *goal_reached* 는 0으로 초기화).

즉, **한 epoch는 100 step보다 같거나 작은 size로 구성**되고, 완전한 학습이 이루어진 후에는(보통 3~400 에포크 안에 끝남) 매우 빠르게 골지점에 도달하게 되므로, 에포크의 길이가 매우 짧아진다. 따라서 simple-PPO를 사용하는 경우보다 전반적인 학습 자체가 빠르게 진행될 것으로 기대된다. 본 팀에서 차용한 알고리즘이 안정적으로 목표 지점에 도달하면서 학습도 빠르기 때문에 Lava 문제를 해결하는데 simple-PPO보다 훨씬 유용하다는 결론을 지었고 이를 차용하였다.

3. 결론

Chain MDP의 경우 Lava문제와 달리 PPO 알고리즘을 사용하지 않았는데 그 이유는 ***"Deep Exploration"*** 이 핵심이라고 생각하였기 때문이다. PPO는 policy를 update할 때 기존에 학습하던 policy로부터 큰 차이가 나지 않도록 update가 진행되기 때문에, chain MDP의 경우 local optimum에 빠질 확률이 매우 클 것이라 생각했다. 실제로 다른 특성을 가진 Ensemble-DQN이 Chain MDP에서 매우 잘 동작하는 것을 확인할 수 있었다.

Chain MDP와 Lava에서 알고리즘에 대한 개선 방향도 다양하게 제시하고자 한다. 우선 Chain MDP의 경우 Ensemble-DQN에 soft update와 decaying ε-greedy 기법만을 활용했는데, 이 외에도 rainbow DQN에서 사용하는 **prioritized replay buffer**, **Double-DQN**, **Dueling-DQN** 등 세부적으로 변형을 시도해볼 수 있는 부분들이 있다. Lava의 경우 주어진 max step에 제한을 두지 않았을 때 성능 발전을 경험하였다. 한 epoch 내에서 episode 수에 상관없이 agent가 무조건 4000 step을 움직이도록 하고, 해당 epoch가 끝날 때마다 actor와 critic을 update하는 형식을 사용하였을 때, 최적 경로로 매우 안정적이게 수렴하는 것을 다양한 시도를 통해 발견할 수 있었다. 따라서, 오히려 **epoch의 수를 줄이고 max step의 크기를 키워 학습을 진행한다면**, 성능적으로 유리할 것이라 기대한다.

4. 부록

<Chain MDP>

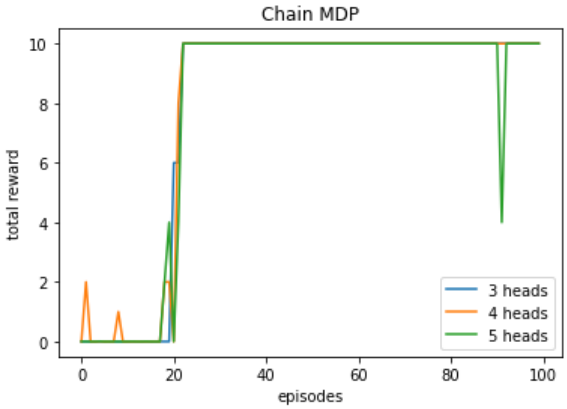

그림 1 head 수에 따른 100 ep의 reward 총합

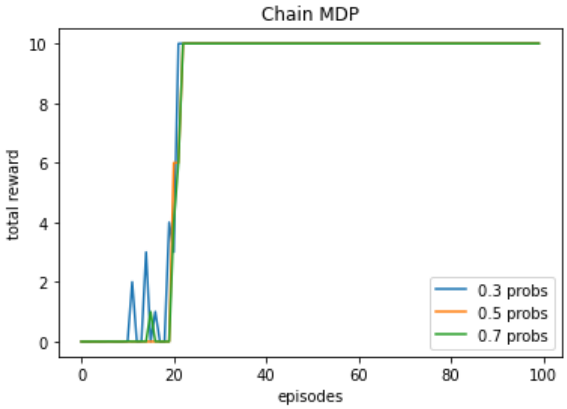

그림 2 Bernoulli prob에 따른 100 ep의 reward 총합

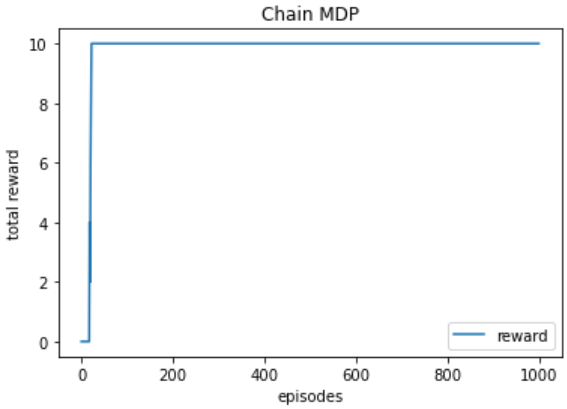

그림 3.1 최종 알고리즘의 1000 ep 성능
(seed=0, chain length=10)

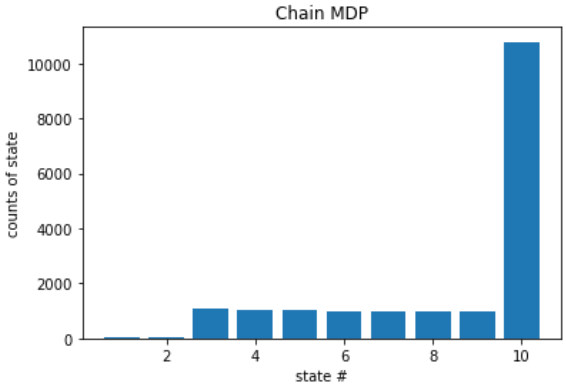

그림 3.2 (같은 조건) 각 state 별 지나는 횟수

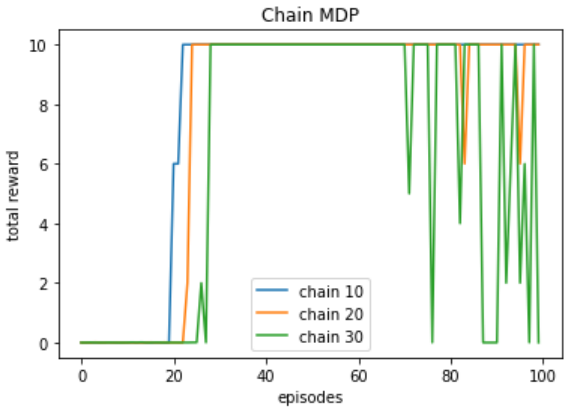

그림 4 chain 길이에 따른 최종 알고리즘의 성능

<Lava>

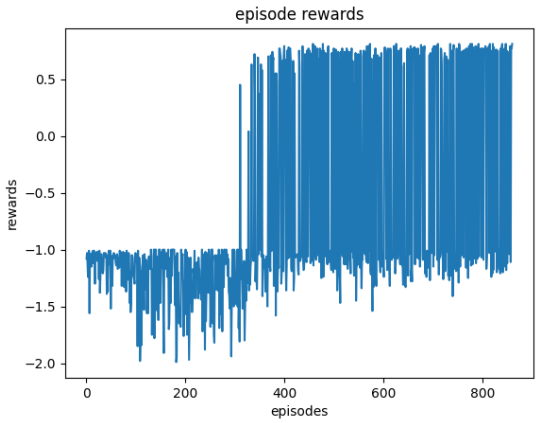

그림 5 DQN_result_1 : simple DQN의 성능

```
iteration 810 avg reward 0.196 / avg step 21.4
final state :
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 7.]]

iteration 820 avg reward 0.15699999999999995 / avg step 25.3
final state :
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 2. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 7.]]

iteration 830 avg reward -0.02300000000000004 / avg step 23.3
final state :
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 2. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 2. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 6.]]

iteration 840 avg reward -0.5610000000000002 / avg step 17.1
final state :
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 2. 2. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 3.]]
```

그림 6 DQN_result_2 : simple DQN의 불안정성

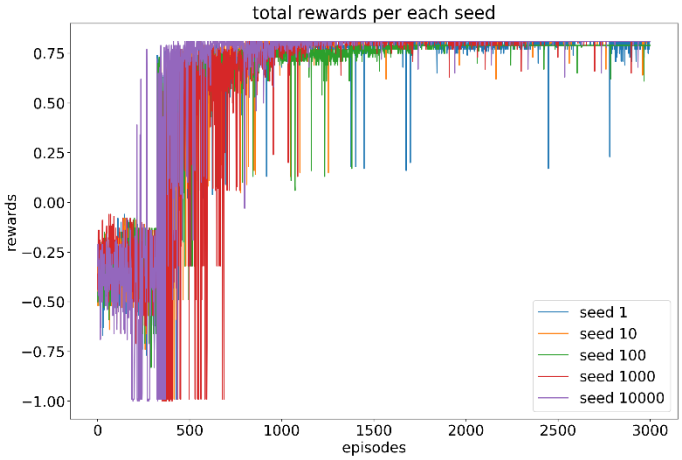

그림 7 seed에 따른 최종 알고리즘의 성능

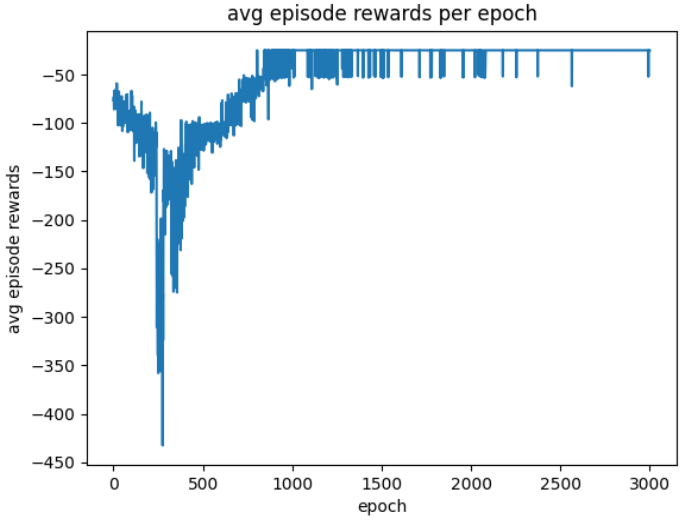

그림 8 Dense reward를 가했을 때 성능