고급전산학특강 1

과제 #04 보고서

이름	정재민
학번	M19522
소속 학과/대학	일반대학원 컴퓨터 공학과
분반	01 (담당교수: 김태운)

<주의사항>

- 개별 과제 입니다. (팀으로 진행하는 과제가 아니며, 모든 학생이 보고서를 제출해야 함)
- 각각의 문제 바로 아래에 답을 작성 후 제출해 주세요.
    - 소스코드/스크립트 등을 작성 한 경우, 해당 파일의 이름도 적어주세요.
- 스마트캠퍼스 제출 데드라인: 2020. 04. 28. ~ 2020. 05. 11. (월요일) 23:59 // 2 주 과제
    - 데드라인을 지나서 제출하면 24 시간 단위로 20%씩 감점(5 일 경과 시 0 점)
    - 주말/휴일/학교행사 등 모든 날짜 카운트 함
    - 부정행위 적발 시, 원본(보여준 사람)과 복사본(베낀 사람) 모두 0 점 처리함
    - 예외 없음
- 스마트캠퍼스에 아래의 파일을 제출 해 주세요
    - 보고서(PDF 파일로 변환 후 제출)
    - 보고서 파일명에 이름과 학번을 입력 해 주세요.
    - 소스코드, 스크립트, Makefile 등을 작성해야 하는 경우, 모든 파일 제출

<개요>

이번 과제는

- Monte Carlo Prediction 및 Monte Carlo RL 을 구현하고,
- TD Prediction 및 TD RL 을 구현하는 내용입니다.

[MC RL] 업로드된 소스코드를 다운받아서 3 곳의 빈칸에 들어갈 코드를 입력하면 됩니다.

[TD RL] mc_agent.py 코드를 일부 변형해서 구현하면 됩니다.

** MC RL 과 TD RL 모두 동일한 environment.py 를 사용합니다.

이번 과제는 총점 120 점 만점입니다.

<과제>

[Q 1] 1 번 빈칸, update_EveryVisit [배점: 20]

"1) 여기에 들어갈 코드를 작성하세요." 부분에 들어갈 코드를 작성하고, 아래에 해당 코드를 복사해서 붙여 넣으세요.

답변 (코드):

```
def update_EveryVisit(self):
        g = 0

        for s, r, _ in self.samples[::-1]:
            state_key = f'{s}'

            v = self.value_table[state_key]
            g = r + self.discount_factor * g

            self.value_table[state_key] = v + self.learning_rate * (g - v)
```

[Q 2] 2 번 빈칸, `update_FirstVisit` [배점: 20]

"2) 여기에 들어갈 코드를 작성하세요." 부분에 들어갈 코드를 작성하고, 아래에 해당 코드를 복사해서 붙여 넣으세요.

답변 (코드):

```
def update_FirstVisit(self):
        g = 0
        is_visit = []

        for s, r, _ in self.samples[::-1]:
            state_key = f'{s}'

            if state_key not in is_visit:
                is_visit.append(state_key)

                v = self.value_table[state_key]
                g = r + self.discount_factor * g

                self.value_table[state_key] = v + self.learning_rate * (g - v)
```

```python
def update_FirstVisit(self):
    g = 0
    is_visit = []

    for s, r, _ in self.samples[::-1]:
        state_key = f'{s}'

        if state_key not in is_visit:
            is_visit.append(state_key)

            v = self.value_table[state_key]
            g = r + self.discount_factor * g

            self.value_table[state_key] = v + self.learning_rate * (g - v)
```

[Q 3] 3 번 빈칸, `get_action` [배점: 20]

"3) 여기에 들어갈 코드를 작성하세요." 부분에 들어갈 코드를 작성하고, 아래에 해당 코드를 복사해서 붙여 넣으세요.

답변 (코드):

```
def get_action(self, state_):
        random_value = np.random.rand()

        if self.epsilon > random_value:
            action = random.choice(self.actions)
        else:
            next_state = self.possible_next_state(state_)
            action = self.arg_max(next_state)

        return action
```


```
def get_action(self, state_):
    random_value = np.random.rand()

    if self.epsilon > random_value:
        action = random.choice(self.actions)
    else:
        next_state = self.possible_next_state(state_)
        action = self.arg_max(next_state)

    return action
```

[Q 4] First-visit vs every-visit [배점: 10]

First-visit 으로 구현한 Agent 와 every-visit 으로 구현한 Agent 의 성능을 비교하세요.

- 어떤 기준을 사용해서 결과를 비교 했나요?
- 두 Agent 간 성능상 차이가 있나요? 아니면 차이점이 없나요? 만약 차이점이 있다면 어째서 이와 같은 차이가 발생한다고 생각하나요?

답변 (설명: 성능 비교 기준):

성능차이의 기준으로 **시간, 수렴, 적절한 보상을 받을 확률**을 계산한다.

시간은 최적의 경로를 잘 찾았는지에 대한 성능평가 기준이 될 수 있다. 보상을 받는 경로지만 덜 최적화된 길로 가는 경우가 존재하기 때문에 agent 가 움직인 step 수로 성능을 비교한다.

GridWorld 에서 최적의 길로 가는 가장 빠른 길의 step 수는 6 칸이다. 그리고 실패하는 길로 가는 가장 빠른 길의 step 수는 3 칸이다. 둘 다 환경의 조건은 같다.

적절한 보상을 받을 확률은 episode 가 원하는 곳으로 도착할 확률을 나타낸다. 총 에피소드의 횟수에서 몇 번 도달하였는지를 알면 계산할 수 있다.

그리고 랜덤에 의존하는 경우가 많기 때문에 총 5 번 실행하여 평균으로 비교한다.

마지막으로 얼마나 빠르게 학습을 하였는지에 대해서 알기 위해 처음으로 수렴한 episode 를 구한다. 수렴한 episode 는 연속적으로 성공을 5 번 하는 처음 경우다.

Every'

성공 : 942 + 938 + 939 + 935 + 937 = 4691 / 5 = 938.2 성공 확률 : 93.82%

실패 : 58 + 62 + 61 + 65 + 63 = 309 / 5 = 61.8 실패 확률 : 6.18%

총 Step 수 : 18906 + 13614 + 6887 + 28828 + 12521 = 80756 / 5 = 16151

처음으로 수렴한 episode 평균: 20

First

성공 : 940 + 937 + 963 + 964 + 944 = 4748 / 5 = 949.6 성공 확률 : 94.96%

실패 : 60 + 63 + 37 + 36 + 56 = 252 / 5 = 50.4 실패 확률 : 5.04%

총 Step 수 : 6712 + 10949 + 10172 + 7225 + 12570 = 47628 / 5 = 9525.6

처음으로 수렴한 episode 평균: 12

Everyvisit 과 Firstvisit 둘 다 내가 원하는 곳에 도착할 확률이 높다. Firstvisit 이 약간 더 좋다. 하지만 총 step 수를 보았을 때 everyvisit 같은 경우는 firstvisit 의 2 배 가량의 차이가 난다. 즉, firstvisit 이 agent 가 목표지점에 도착할 확률이 높으면서 최적의 경로를 찾을 확률이 높다는 것(탐험할 확률이 적다.)을 의미한다.

이 의미를 해석하면 everyvisit 의 경우 방문한 모든 step 을 업데이트 하기 때문에 여러 번 같은 상태에 도달할 경우 감가율이 적용 되어서 남은 상태를 업데이트 하는데 매우 작은 값이 업데이트 된다. 그래서 탐험을 할 확률이 더 높아진다.

하지만 firstvisit 은 업데이트를 한번만 진행하기 때문에 최적의 길로 빠르게 수렴이 잘 될 것이다. 하지만 탐험을 할 확률은 everyvisit 보다 작다.

[Q 5] TD Learning 구현 [배점: 40]

Monte Carlo Learning 코드를 약간 변형하면 TD Learning 을 구현할 수 있습니다. TD Learning (= TD Prediction + ε –Greedy 기반의 행동 결정) 을 구현하고 소스코드를 제출하세요.

답변 : (본 문제는 답을 할 필요가 없습니다. 소스코드를 제출하면 됩니다)

[Q 6] MC Learning vs TD Learning [배점: 10]

Monte Carlo Learning 과 TD Learning 을 모두 실행해 보고, 성능을 비교하세요.

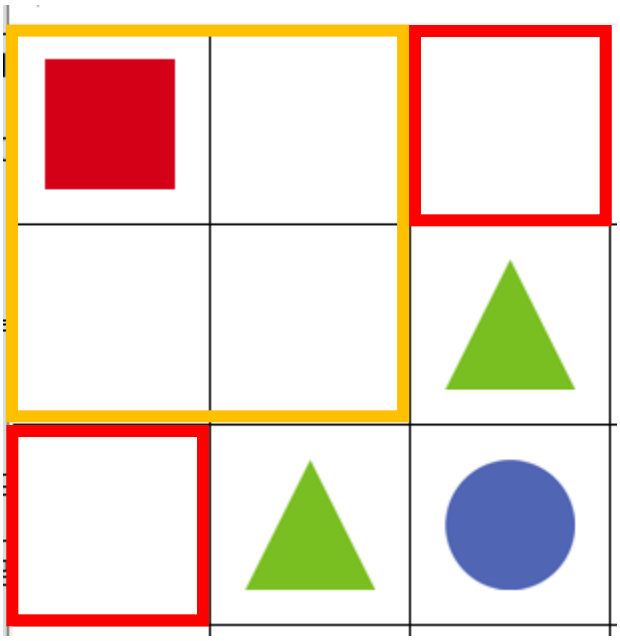
- 어떤 기준으로 성능을 비교했나요?
- 두 알고리즘 간 성능상 차이점이 있나요? 아니면 차이점이 없나요? 만약 차이점이 있다면 어째서 이와 같은 차이가 발생한다고 생각하나요?

소스코드(td_agent.py)를 스마트캠퍼스에 업로드 하세요.

답변 (설명: 성능 비교 기준):

성능차이의 기준으로 **시간, 수렴, 적절한 보상을 받을 확률**을 계산한다. 위에서 MC Learning 의 Everyvisit Firstvisit 을 비교한 것과 동일하다.

답변 (설명: 성능 비교):



TD 는 환경의 구성상 [0, 0], [1, 0], [0, 1], [1, 1] 주황색 영역에 갇히게 된다. 왜냐하면 현재 상태를 다음 상태의 가치 함수를 이용하여 업데이트를 한다. [2, 0]에서 함정 [2, 1]로 가는 경우 [2, 0]은 가치 함수가 음수가 된다. 그렇게 [0, 2]에서 함정 [1, 2]로 가는 경우 또한 발생한다면 agent 는 [2, 0], [0, 2] 상태를 가지 못하게 된다. 즉, [0, 0], [1, 0], [0, 1], [1, 1] 영역에 갇히게 된다. 이러한 문제 때문에 TD Learning 과 MC Learning 의 성능을 비교하기가 매우 어렵다. 결론적으로 확률적으로 결과를 확인하기 어려운 TD Learning 같은 경우는 여러 번 실행하기에 어려움이 있다. 그래서 한번만 실행시켜서 MC Learning 과 비교한다.

TD Learning

1 – 200 episode 의 경우 많은 step 을 가진다. (영역에 갇혔기 때문) : 평균 2500 step

200 – 1000 episode 의 경우 수렴하여 최적의 길로 계속 도달하게 된다.

성공 : 942 => 94.2%

실패 : 58 => 5.8%

처음 수렴한 episode : 11

TD Learning 의 경우 MC Learning 과 비교하여 매우 많은 수의 step 을 가지고 수렴하기 때문에 시간이 매우 오래 걸리고 갇히는 증상이 발생하여 정확한 성능 평가가 어렵다.

성공확률 실패확률은 MC Learning 과 유사하다.

끝! 수고하셨습니다 ☺