바닥부터 배우는 강화학습

Chapter 5.
MDP를 모를 때 밸류 평가하기

Jiyong Ahn

Recap

- **Bellman Expectation Equation**

$$V_\pi(S_t) = E[G_t] \longrightarrow \textbf{Monte Carlo Method}$$

$$= E[r_{t+1} + \gamma v_\pi(s_{t+1})] \longrightarrow \textbf{Temporal Difference}$$

Why we use online methods?

“MDP를 모른다. (model-free)”
(보상 함수 r_s^a 혹은 전이 확률 $P_{ss'}^a$를 모른다.)

“실제 시행을 해보고 정책을 평가”

“Sampling”

“Large-scale MDP”
(기존의 offline method 혹은 가능한 모든 정책들을 평가하기에는 계산량이 너무 많다.)

“최적일 가능성이 큰 것들부터 우선적으로 시행”

Monte Carlo 학습

"MDP를 모르니 planning 불가" → "실제 경험을 통해 학습"

"대수의 법칙" → $v_\pi(s_t) = \mathbb{E}_\pi[G_t]$

① 초기화
- $N(s_t)$: 상태 s를 몇 번 방문했는지
- $V(s_t)$: 해당 상태에서 경험했던 리턴의 총합

② 경험 쌓기: 상태를 sampling한 후 에피소드가 종료될 때까지 실행하기

③ 업데이트: 실행 결과를 토대로 N,V 업데이트

$$N(s_t) \leftarrow N(s_t) + 1$$

$$V(s_t) \leftarrow V(s_t) + G_t$$

④ 밸류계산: 시행한 에피소드들을 종합하여 밸류를 계산

$$v_\pi(s_t) \cong \frac{V(s_t)}{N(s_t)}$$

Monte Carlo 학습

- **조금씩 업데이트 하는 버전**

$$V(s_t) \leftarrow (1-\alpha) * V(s_t) + \alpha * G_t \qquad V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

α가 0.1이라면 — 원래 값 90%와 — 새로운 값 10%를 섞음

- Exponentially weighted average를 이용하여 에피소드가 끝날 때 마다 업데이트 하는 방법
- N의 값을 저장해둘 필요 없음(메모리 이득)

Cf. Exponentially weighted average

- Moving average를 구할 때, 오래된 data의 영향을 지수적으로 감쇠(exponential decay)하도록 계산 $(\lim_{\varepsilon \to 0}(1-\varepsilon)^{\frac{1}{\varepsilon}} = \frac{1}{e})$
- 근사적으로 $\frac{1}{\alpha}$개의 data를 이용해 평균을 취한 결과
- 메모리에 최근 값만 저장하고 계속 업데이트->메모리,연산에서 효율->ML에 많이 쓰임
  Ex. Adam Optimizer

Monte Carlo 학습

- **몬테카를로 학습 구현하기**

GridWorld.py

```python
import random

class GridWorld():
    def __init__(self):
        self.x = 0
        self.y = 0

    def step(self,a):
        """
        Args: action - from Agent
        Return: next state, reward, done
        """

        if a==0:
            self.move_right()
        elif a==1:
            self.move_left()
        elif a==2:
            self.move_up()
        elif a==3:
            self.move_down()

        reward = -1
        done=self.is_done()

        return (self.x,self.y), reward, done

    def move_right(self):
        self.y+=1
        if self.y>3:
            self.y = 3

    def move_left(self):
        self.y-=1
        if self.y<0:
            self.y = 0

    def move_up(self):
        self.x-=1
        if self.x<0:
            self.x = 0

    def move_down(self):
        self.x+=1
        if self.x>3:
            self.x = 3

    def is_done(self):
        if self.x==3 and self.y==3:
            return True
        else:
            return False

    def get_state(self):
        return (self.x, self.y)

    def reset(self):
        self.x = 0
        self.y = 0
        return (self.x, self.y)
```

- **환경** : 에이전트의 액션을 받아 상태변이를 일으키고, 보상을 줌
- **에이전트** : 4방향 랜덤 정책을 이용해 움직임
- **경험 쌓는 부분** : 에이전트가 환경과 상호작용하며 데이터를 축적
- **학습하는 부분** : 쌓인 경험을 통해 테이블을 업데이트

```python
class Agent():
    def __init__(self):
        pass

    def select_action(self):
        """
        select action - policy: uniformly random
        """

        coin=random.random()
        if coin<0.25:
            action = 0
        elif coin<0.5:
            action = 1
        elif coin<0.75:
            action = 2
        else:
            action = 3

        return action
```

Monte Carlo 학습

- **몬테카를로 학습 구현하기**

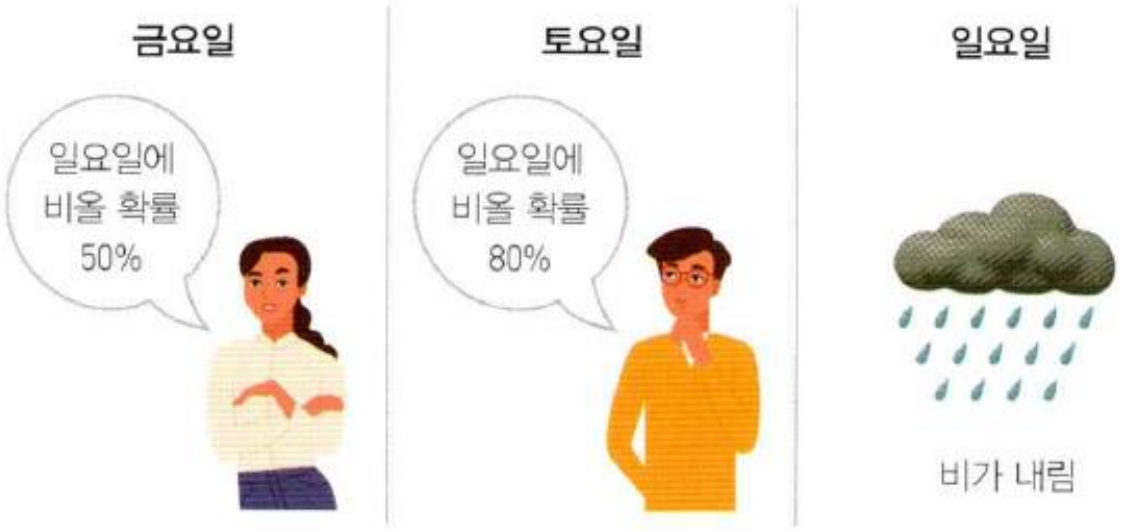
GridWorld_MC.py

```python
import numpy as np
from GridWorld import GridWorld, Agent

def main():
    env = GridWorld()
    agent = Agent()
    gamma = 1.0
    alpha = 0.0001
    num_samples = 50000
    data = np.zeros((4,4)) # 각 state의 value

    for k in range(num_samples): # 에피소드 진행
        done = False
        history = []
        while not done:
            action = agent.select_action()
            (x,y),reward,done = env.step(action)
            history.append((x,y,reward))
        env.reset()

        # 매 에피소드가 끝나고 바로 해당 데이터를 이용해 업데이트
        cum_reward = 0 # 각 에피소드에서의 리턴 값을 임시 저장
        for transition in history[::-1]:
            # 방문했던 state들을 뒤에서부터 보며 차례차례 리턴을 계산
            x, y, reward = transition
            # exponentially weighted average
            data[x][y] = data[x][y]+alpha*(cum_reward-data[x][y])
            cum_reward = reward + gamma*cum_reward # 책 오류?

    # 학습이 끝나고 난 후 value 데이터 출력
    print(data)

if __name__=="__main__":
    main()
```

```
[[-58.63807809 -56.87647539 -54.75944227 -52.59719093]
 [-56.45753032 -53.967608   -49.9276436  -46.45242858]
 [-54.26952399 -50.39870106 -41.26789606 -30.70693709]
 [-51.7026206  -45.19180377 -29.31696691   0.        ]]
```

Temporal Difference 학습

MC는 에피소드가 끝나야 업데이트 가능(terminating MDP에서만 사용 가능)
-〉 “에피소드가 끝나기 전에 업데이트 할 수는 없을까?”
(non-terminating MDP에서도 학습 가능)
-〉 “추측을 추측으로 업데이트하자 "



| 그림 5-7 | 일요일에 비가 올 지 추측하는 상황

토요일이 금요일보다 정보가 더 많으므로 일요일의 날씨를 더 잘 예측할 가능성이 큼
-〉 토요일의 예측 값을 이용해서 금요일의 예측 값을 업데이트하자. (시간적인 차이(TD)를 이용)

Temporal Difference 학습

- **이론적 배경**

[Monte Carlo]

$$v_\pi(s_t) = \mathbb{E}_\pi[G_t]$$

unbiased estimate

[Temporal Difference]

$$v_\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]$$

TD target

biased estimate

$$r_{t+1} + \gamma V(s_{t+1})$$

Temporal Difference 학습

- Temporal Difference 학습 알고리즘

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_7 \rightarrow S_6 \rightarrow S_{10} \rightarrow S_{11} \rightarrow \text{종료}$$

$$\text{MC}: V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$ 1번 업데이트

$$\text{TD}: V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$ 8번 업데이트

$$V(s_0) \leftarrow V(s_0) + 0.01 * (-1 + V(s_1) - V(s_0))$$

$$V(s_1) \leftarrow V(s_1) + 0.01 * (-1 + V(s_2) - V(s_1))$$

...

일반적으로 TD에서 MC보다 큰 alpha값을 사용

$$V(s_{11}) \leftarrow V(s_{11}) + 0.01 * (-1 + V(s_{15}) - V(s_{11}))$$

Temporal Difference 학습

- Temporal Difference 학습 구현하기 GridWorld_TD.py

```python
import numpy as np
from GridWorld import GridWorld, Agent

def main():
    env = GridWorld()
    agent = Agent()
    gamma = 1.0
    alpha = 0.01 # MC에 비해 큰 값을 사용
    num_samples = 50000
    data = np.zeros((4,4)) # 각 state의 value

    for k in range(num_samples): # 에피소드 진행
        done = False
        history = []
        while not done:
            x,y = env.get_state()
            action = agent.select_action()
            (x_prime,y_prime),reward,done = env.step(action)
            x_prime,y_prime = env.get_state()

            # 한 번의 step이 진행되자마자 바로 테이블의 데이터를 업데이트
            data[x][y] = data[x][y]+alpha*(reward+gamma*data[x_prime][y_prime]-data[x][y])
        env.reset()

    # 학습이 끝나고 난 후 value 데이터 출력
    print(data)

if __name__=="__main__":
    main()
```

```
[[-60.6903466  -59.04715912 -55.88801149 -53.45967306]
 [-58.57192473 -56.17040219 -51.8620463  -46.90914954]
 [-55.94522554 -51.71042959 -43.61039712 -32.09705258]
 [-53.39183096 -47.90907469 -31.49833977   0.        ]]
```

MC vs TD

	[Monte Carlo]	[Temporal Difference]
학습 시점	Episodic MDP에서만	상관x
편향성 (bias)	unbiased	biased $V(s_{t+1}) \neq v_\pi(s_{t+1})$
분산 (variance)	여러 번의 확률적 결과 종합 -> 분산이 큼	1번의 확률적 결과 ->분산이 작음

MC와 TD의 중간?

- Temporal Difference 학습 구현하기

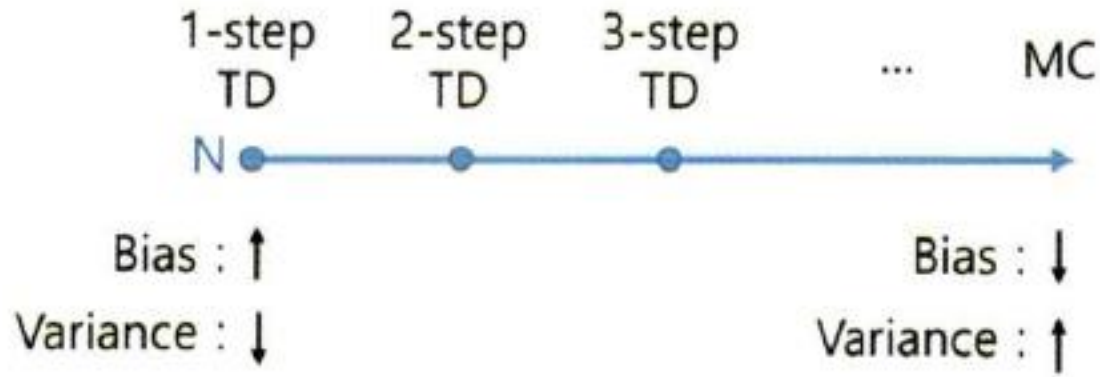
$$N = 1 : r_{t+1} + \gamma V(s_{t+1})$$

$$N = 2 : r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2})$$

$$N = 3 : r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V(s_{t+3})$$

...

$$N = n : r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^n V(s_{t+n})$$



| 그림 5-9 | TD와 MC의 스펙트럼

Summary

MDP를 모를 때
Large-scale

sampling

대수의 법칙

MC
TD

$$v_\pi(s_t) = \mathbb{E}_\pi[G_t]$$

$$v_\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]$$

- **Reference**
  - 노승은, 바닥부터 배우는 강화학습, 영진닷컴, 2020.
  - Andrew Ng, deeplearning.ai, coursera


Q & A