MC VS SARSA VS Q-learning

IT경영
2018316014
신형찬

GridWorld

환경소개

Action = ⬅ ➡

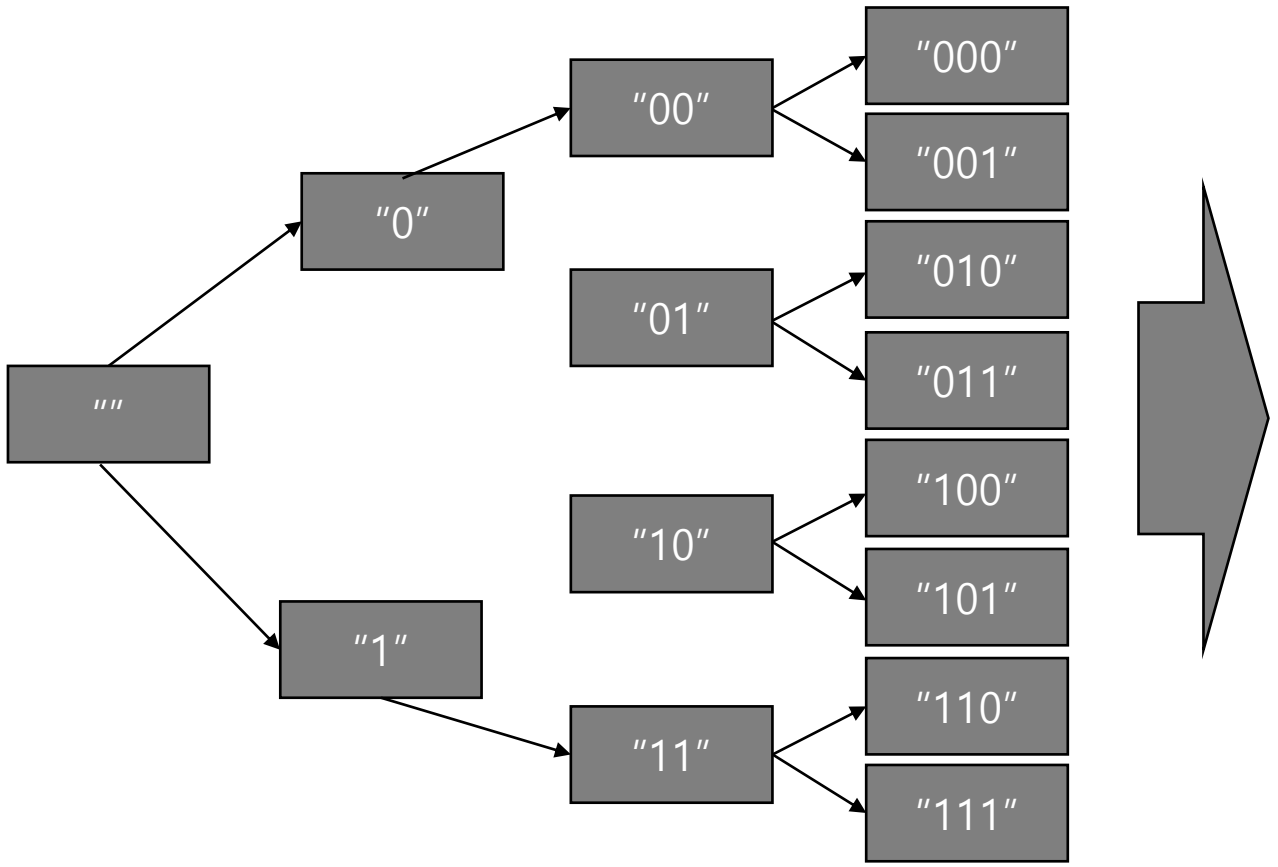
Reward = ⬅ -1 ➡ +1

IF State= ⬅ ➡ ⬅ ➡ ⬅ Action = ⬅ Reward = 1000

- 액션을 6번 실행하면 에피소드 종료
- 좌우로 움직이는 두가지 액션 존재
- 일반적으로 좌로 움직이면 -1의 보상을 얻고 우로 움직이면 +1의 보상을 얻음
- 단 좌우좌우좌의 상태에서 좌로 움직이면 +1000의 보상을 얻음

GridWorld

스테이트 정의



- 총 127개의 state 존재
- 0은 좌로 움직인 것
- 1은 우로 움직인 것

환경소개

- 초기 상태는 빈칸으로 지정
- 0은 좌로 이동하고 -1의 보상을 얻음
- 1을 우로 이동하고 +1의 보상을 얻음
- 이전 상태가 "01010"일 경우 0을 선택 시 +1000의 보상을 얻음
- 종료 상태인지 확인
- 상태 s와 보상, 종료상태를 리턴함

```
class GridWorld():
    def __init__(self):
        self.s= ""

    def step(self, a):
        if a==0:
            if self.s == "01010":
                reward = +1000
            else:
                reward = -1
            self.move_left()
        elif a==1:
            self.move_right()
            reward = +1
        done = self.is_done()
        return self.s, reward, done
```

환경소개

- 좌로 이동시 시퀀스에 0을 더함
- 우로 이동시 시퀀스에 1을 더함
- 시퀀스의 길이가 6이 되면 에피소드 종료
  - 6번의 액션수행을 의미
- 상태를 다시 리셋함

```
def move_left(self):
    self.s = self.s+"0"

def move_right(self):
    self.s = self.s+"1"

def is_done(self):
    if len(self.s) == 6:
        return True
    else:
        return False

def reset(self):
    self.s = ""
    return self.s
```

MC

환경소개

- State와 Action수에 맞게 q테이블 생성
- 초기 eps를 0.9로 설정
- get_state:
  - 시퀀스가 몇 번째 스테이트인지를 반환
  - 자릿수에 따라 2의 제곱을 해서 더함
  - 1일 경우에는 2를 더 곱함
  - 0101
  - 2+2+8+8

```
class QAgent():
    def __init__(self):
        self.q_table = np.zeros((127,2))
        self.eps = 0.9
        self.alpha = 0.1

    def get_state(self, s):
        state = 0
        for i in range(len(s)):
            if s[-i-1]=="0":
                state = state + 2**(i)
            else:
                state = state + 2 * 2**(i)
        return state
```

MC

환경소개

- Select_action
- Coin을 통해 eps보다 낮을 경우 랜덤한 액션 선택
- 높을 경우 q_table에서 가장 높은 값 사용
- Seledt_action2
- 무조건 q_table에서 가장 높은 값을 사용
- 학습이 완료된 후 정책대로 액션을 수행한 경우를 보기 위한 함수

```
def select_action(self, s):
    coin = random.random()
    k = self.state(s)
    if coin < self.eps:
        action = random.randint(0,1)
    else:
        action_val = self.q_table[k,:]
        action = np.argmax(action_val)
    return action

def select_action2(self, s):
    k = self.get_state(s)
    action_val = self.q_table[k,:]
    action = np.argmax(action_val)
    return action
```

MC

환경소개

- Update_table:
  - History에서 하나씩 뽑아서 사용
  - MC식을 이용해서 업데이트
- Anneal_eps:
  - Eps를 0.001씩 줄이면서 0.1보다는 낮아지지않게 유지함
- Show:
  - Q테이블의 모습을 보여줌

```
def update_table(self, history):
    cum_reward = 0
    for transition in history[::-1]:
        s, a, r, s_prime = transition
        k = self.get_fitness(s)
        self.q_table[k,a] = self.q_table[k,a] + self.alpha *
        (cum_reward - self.q_table[k,a])
        cum_reward = cum_reward+r

def anneal_eps(self):
    self.eps -=0.001
    self.eps = max(self.eps, 0.1)

def show(self):
    print(self.q_table.tolist())
```

MC

환경소개

- main:
  - Episode를 10000번 진행
  - 상태를 초기화 해줌
  - episode가 끝날 때 까지 history의 저장
  - Episode가 종료되면 테이블 업데이트
  - Eps 조절
  - 학습이 완료된 q테이블을 보여줌

```
def main():
    env = GridWorld()
    agent = QAgent()

    for n_epi in range(10000):
        done = False
        history=[]

        s = env.reset()
        while not done:
            a = agent.select_action(s)
            s_prime, r, done = env.step(a)
            history.append((s,a,r,s_prime))
            s = s_prime
        agent.update_table(history)
        agent.anneal_eps()
    #agent.show()
```

MC

환경소개

- 정책을 따라갔을 때 보상을 확인하는 단계
- 초기화 진행
- 해당 스테이트의 가장 좋은 액션을 선택하며 episode 끝까지 진행
- 보상을 return

```
done=False
s=env.reset()
r_sum=0
while not done:
    a = agent.select_action2(s)
    s_prime, r, done = env.step(a)
    r_sum= r_sum+r
    s = s_prime

return r_sum
```

SARSA

환경소개

- Transition이 진행 될 때 마다 업데이트
- SARSA 공식을 이용

```
def update_table(self, transition):
    s, a, r, s_prime = transition
    k = self.get_fitness(s)
    next_k = s_prime
    a_prime = self.select_action(s_prime)
    next_k=self.get_fitness(next_k)
    self.q_table[k,a] = self.q_table[k,a] + self.alpha *
    (r + self.q_table[next_k, a_prime] - self.q_table[k,a])
```

SARSA

환경소개

- Episode가 종료되지 않아도 단계가 끝날 때마다 테이블을 업데이트 함
- Episode가 끝날 때마다 eps를 조정함

```
def main():
    env = GridWorld()
    agent = QAgent()

    for n_epi in range(10000):
        done = False

        s = env.reset()
        while not done:
            a = agent.select_action(s)
            s_prime, r, done = env.step(a)
            agent.update_table((s,a,r,s_prime))
            s = s_prime
        agent.anneal_eps()
    #agent.show()
```

Q-learning

환경소개

- 벨만 최적방정식을 활용
- 다음 테이블에서 가장 좋은 액션을 선택함

```
def update_table(self, transition):
    #"",1,1,1,False
    s, a, r, s_prime = transition
    k = self.get_fitness(s)
    #k=0
    next_k = s_prime
    next_k=self.get_fitness(next_k)
    #SARSA 업데이트 식을 이용
    self.q_table[k,a] = self.q_table[k,a] + self.alpha *
    (r + np.amax(self.q_table[next_k, :]) - self.q_table[k,a])
```

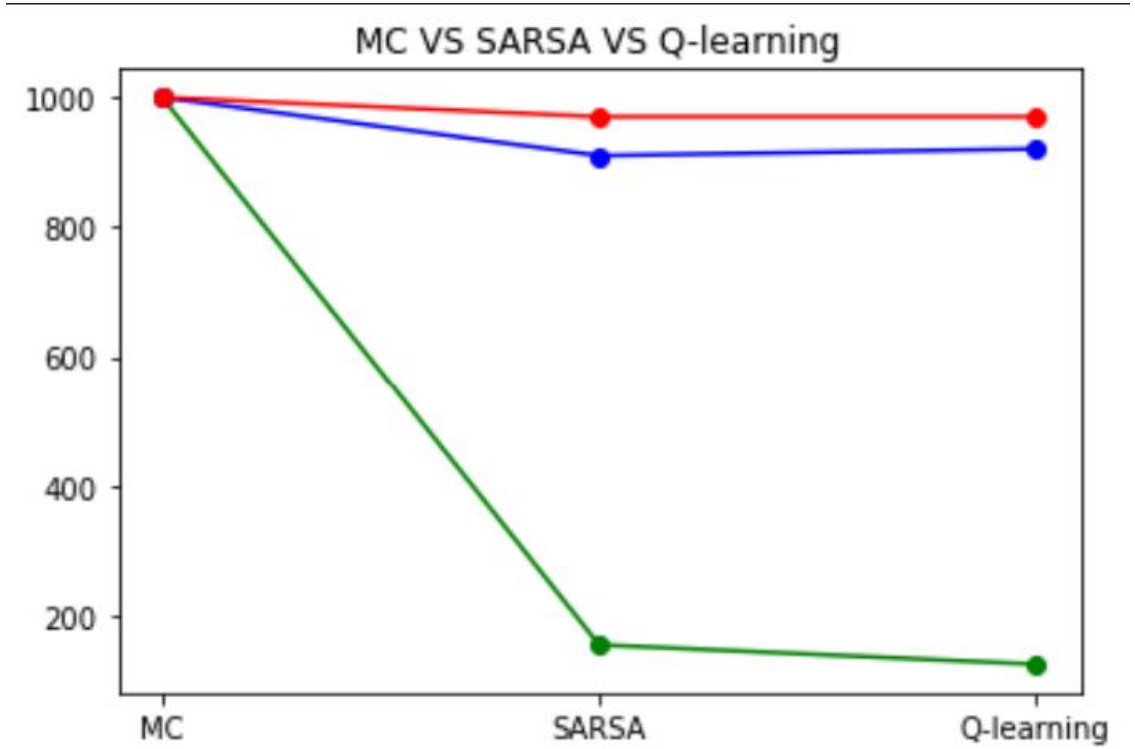
실행결과

사용한 파라미터

- Eps = 0.9
  Alpha = 0.1
  Anneal eps = -0.01
  Min eps = 0.2
  Ep =1000

- Eps = 0.9
  Alpha = 0.1
  Anneal eps = -0.001
  Min eps = 0.2
  Ep=1000

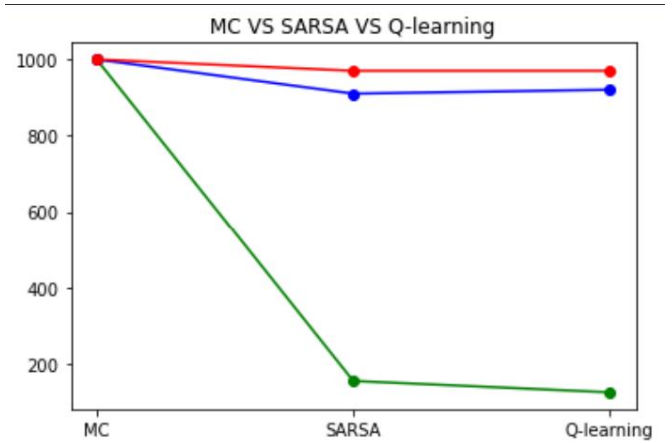
- Eps = 0.9
  Alpha = 0.1
  Anneal eps = -0.001
  Min eps = 0.2
  Ep=10000



실행결과

학습 평가

- SARSA와 Q-learning은 eps가 빠르게 줄어들면서 탐험을 적절히 하지 못해 설익은 수렴을 함
- Eps의 값이 천천히 줄어들게 하자 평균 reward가 크게 증가 한 것을 알 수 있음
- Episode를 증가시켜 학습을 더 많이 한 결과 최적정책을 더 잘 찾게 됨



Eps = 0.9
Alpha = 0.1
Anneal eps = -0.01
Min eps = 0.2
Ep =1000

Eps = 0.9
Alpha = 0.1
Anneal eps = -0.001
Min eps = 0.2
Ep=10000

Eps = 0.9
Alpha = 0.1
Anneal eps = -0.001
Min eps = 0.2
Ep=1000

부록: MC와 TD비교

그래서 왜 MC가 TD보다 좋은 정책을 찾았을까

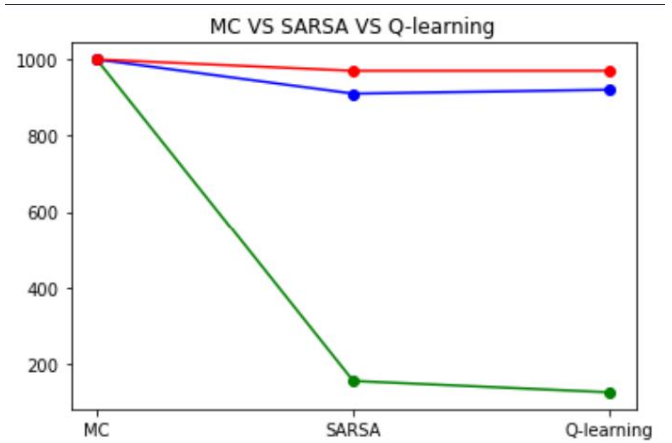
- TD 업데이트식 복기
- 반면에 TD는 한 스텝을 움직일 때마다 가치함수를 업데이트해줌
- 따라서 reward 1000을 받는데 성공해도 그 전 상태인 '좌우좌우좌 ' 의 가치함수만 크게 증가함
- MC와 다르게 첫 선택에서 좌측으로 이동하고 나머지 액션이 모두 우측이더라도 좌측으로 가는 정책은 좋지 않은 정책이라고 판단함
- '좌우좌우좌좌' 상태에 도달 하기 위해서는 앱실론의 영향을 많이 받아야 함

```
def update_table(self, transition):
    s, a, r, s_prime = transition
    k = self.get_fitness(s)
    next_k = s_prime
    a_prime = self.select_action(s_prime)
    next_k=self.get_fitness(next_k)
    self.q_table[k,a] = self.q_table[k,a] + self.alpha *
    (r + self.q_table[next_k, a_prime] - self.q_table[k,a])
```

실행결과

학습 평가

- MC가 꾸준하게 좋은 이유
  - 100번 중 하나의 에피소드라도 64개 중 리워드가 +1000이 되는 스테이트에 도달할 경우 무조건 적으로 최적정책을 찾을 수 있음



Eps = 0.9
Alpha = 0.1
Anneal eps = -0.01
Min eps = 0.2
Ep =1000

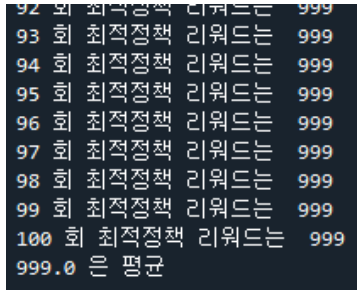
Eps = 0.9
Alpha = 0.1
Anneal eps = -0.001
Min eps = 0.2
Ep=10000

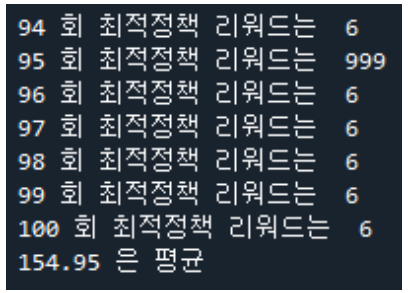
Eps = 0.9
Alpha = 0.1
Anneal eps = -0.001
Min eps = 0.2
Ep=1000

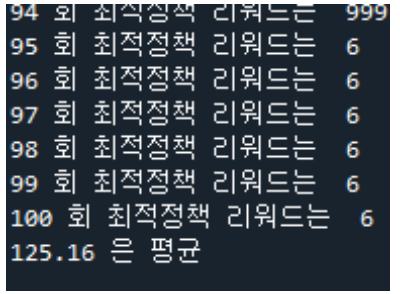
Model Free Control

사용한 파라미터

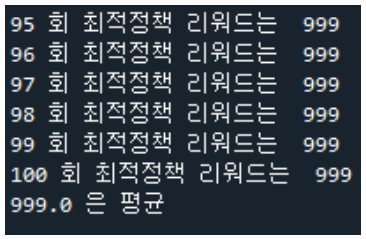
Eps = 0.9
Alpha = 0.1
Anneal eps = -0.01
Min eps = 0.2
Ep =1000

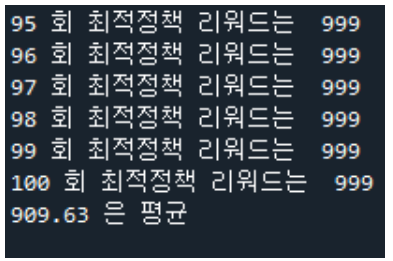


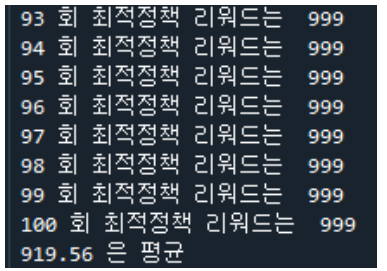




Eps = 0.9
Alpha = 0.1
Anneal eps = -0.001
Min eps = 0.2
Ep=1000







Eps = 0.9
Alpha = 0.1
Anneal eps = -0.001
Min eps = 0.2
Ep=10000