

Reinforcement Learning Basic

Week2. MDP & Cross Entropy Method

박준영

Hanyang University
Department of Computer Science

지난 시간

- 강화학습은 환경(Environment)과 에이전트(Agent)가 서로 상호작용하면서 최적의 행동을 학습하는 방법이다.
 - 에이전트에게는 보상(Reward)만 주어진다.
 - 정답이나 지침이 주어지지 않는다.
- OpenAI Gym 사용법

1 Markov Decision Process

2 Cross Entropy Method

Markov Process(Markov Chain)

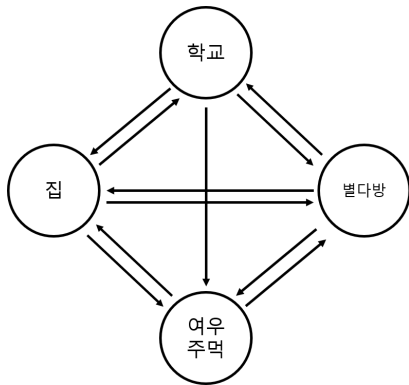
Definition

Stochastic process $X(t)$ is **Markov** if for any $t_1 < \dots < t_n < t$ and any sets $A; A_1, \dots, A_n$

$$\begin{aligned} P[X(t) \in A | X(t_1) \in A_1, \dots, X(t_n) \in A_n] \\ = P[X(t) \in A | X(t_n) \in A_n] \end{aligned}$$

$$P[\text{미래} \mid \text{과거, 현재}] = P[\text{미래} \mid \text{현재}]$$

Markov Process(Markov Chain)



Markov Decision Process

강화학습 문제를 풀 때 MDP를 정의해서 푼다.

Definitions

Markov Decision Process는 $\langle S, A, P, R, \gamma \rangle$ 의 튜플이다.


- S : 상태(State)
- A : 행동(Action)
- P : 상태 전이 확률(State Transition Probability)
- R : 보상 함수(Reward Function)
- γ : 감가율(Discount Factor)

상태(State)

에이전트가 관찰 가능한 상태의 집합 : \mathbf{S}

시간 t 에서의 상태 S_t 가 어떤 상태 s 임은 다음과 같이 표현한다.

$$S_t = s$$

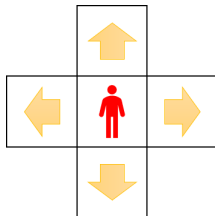
S	F	F	F
	H	F	H
F	F	F	H
H	F	F	G

행동(Action)

에이전트가 어떤 상태에서 할 수 있는 행동의 집합 : \mathbf{A}

시간 t 에서 에이전트가 어떤 행동 a 를 함은 다음과 같이 표현한다.

$$A_t = a$$



상태 전이 확률(State Transition Probability)

상태의 변화에는 확률적인 요인이 들어간다.

→ 이를 수치적으로 표현한 것이 상태 전이 확률 (각 상태로 변할 확률)

$$P_{s \rightarrow s'}^a = P[S_{t+1} = s' | S_t = s, A_t = a]$$

(예시)

- 미로 찾기에서 벽을 뚫을 순 없다. 이때의 $P_{s \rightarrow s'}^a = 0$
- 어떤 문이 $\frac{1}{3}$ 의 확률로 열릴 때 이때의 $P_{s \rightarrow s'}^a = \frac{1}{3}$

보상 함수(Reward Function)

에이전트가 학습할 수 있는 **유일한 정보**

$$R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

- 시간 t 에서 상태가 $S_t = s$ 일 때 행동 $A_t = a$ 를 했을 때 받을 보상에 대한 **기댓값**.
- 보상 함수에 따라 에이전트의 행동 양상이 달라질 수 있다.

반환값(Return)

강화학습의 정확한 목표는 **보상을 최대로 하는** 행동을 찾는 것.

- 당장의 보상(reward)만을 최대화 하는 건 바람직하지 않음.
- 한 에피소드 전체의 보상을 고려해야 한다.

반환값(Return)

Definition

반환값(Return)은 다음과 같이 정의된다.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (\gamma \in [0, 1])$$

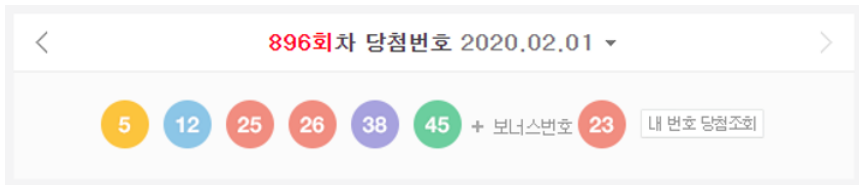
- γ 가 0에 가까울수록 에이전트가 가까운 보상만을 고려한다.
- γ 가 1에 가까울수록 에이전트가 먼 미래까지 고려한다.
- $\gamma \geq 1$ 이면 급수가 발산한다.

감가율(Discount Factor)

- 에이전트는 항상 현재 시점에서 판단을 내리기 때문에 현재에 가까운 보상일수록 더 큰 가치를 갖는다.
- 에이전트는 그 보상을 얼마나 시간이 지나서 받는지를 고려해 감가시켜 **현재의 가치**로 따진다.

→ 보상의 크기가 100일 때, 에이전트가 지금 바로 보상을 받을 땐 100 그대로 받아들이지만 현재로부터 일정 시간이 지나서 보상을 받을 경우엔 크기가 100이라 생각하지 않는다.

감가율(Discount Factor)



감가율(Discount Factor)

A은행

"당첨금 1억 원을 지금 당장 드리겠습니다."

\neq

B은행

"지금 당장 받으면 막쓰다가 탕진할 가능성이 크니 10년 후에 당첨금 1억 원을 드리겠습니다."

감가율(Discount Factor)

A은행

"당첨금 1억 원을 지금 당장 드리겠습니다."

=

B은행

"지금 당장 받으면 막쓰다가 탕진할 가능성이 크니 10년 후에 당첨금 1억 원에
이자까지 드리겠습니다."

감가율(Discount Factor)

Definition

감가율은 같은 크기의 보상이 시간이 지날수록 가치가 줄어드는 것을 표현한 것이다.

현재의 시간 t 로부터 시간 k 가 지난 이후 보상 R_{t+k} 는 $\gamma^{k-1}R_{t+k}$ 가 된다.

(예시)

$\gamma = 0.9$, R_{t+1} , R_{t+2} , $R_{t+3} = 100$ 이라 하면 각각의 현재 시점에서의 가치

- R_{t+1} 의 현재 가치: $R_{t+1} = 100$
- R_{t+2} 의 현재 가치: $\gamma R_{t+2} = 0.9 \times 100 = 90$
- R_{t+3} 의 현재 가치: $\gamma^2 R_{t+3} = 0.9^2 \times 100 = 81$

정책(Policy)

Definition

정책(policy, π)이란 에이전트의 행동을 제어하는 일련의 규칙이다. 상태가 입력으로 들어오면 행동을 출력하는 일종의 함수이다.

보통 확률분포로 정의한다.

시간 t 에 에이전트가 $S_t = s$ 에 있을 때, 가능한 행동 중 $A_t = a$ 를 다음과 같이 표현한다.

$$\pi(a|s) = P[A_t = a | S_t = s]$$

가치 함수(Value Function)

에이전트는 에피소드가 끝난 후에야 반환값을 알 수 있다.

하지만, 때로는 정확한 값을 얻기 위해 끝까지 기다리는 것보다 정확하지 않더라도 현재의 정보를 토대로 행동하는 것이 나을 때가 있다.

Definition

정책 π 를 따랐을 때, 어떤 상태 s 에서의 반환값의 기댓값을 가치함수(value function) 혹은 상태 가치함수(state-value function)이라 한다.

$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s]$$

가치 함수(Value Function)

Definition

어떤 상태 s 에서 정책 π 에 따라 어떤 행동 a 를 했을 때 반환값의 기댓값을 행동 가치함수(action-value function) 또는 Q함수라고 한다.

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

상태 가치함수와 행동 가치함수 사이에는 다음과 같은 등식이 성립된다.

$$v_{\pi}(s) = \sum_a \pi(a|s) q_{\pi}(s, a)$$

강화학습 알고리즘은 다음 세 가지 기준을 통해 분류 할 수 있다.

- Model-free vs Model-based
- Value-based vs Policy-based
- On-policy vs Off-policy

Model-free vs Model-based

Definition

환경 모델은 현재 상태와 그에 따라 취해지는 행동으로부터 다음 상태와 보상을 예측한다.

모델을 통해 미래의 상황을 실제로 경험하기 전에 가능성만을 고려하여 일련의 행동을 **계획**할 수 있다.

- Model-free: 모델을 만들지 않고, 관측만으로 바로 행동을 도출한다.
- Model-based: 모델을 만들어 상태와 보상을 예측해 최선의 행동을 도출한다.

Value-based vs Policy-based

- Value-based: 가치를 예측해서 가치를 최대로 하는 행동을 선택한다.

$$\pi(s) = \operatorname{argmax}_a q(s, a)$$

- Policy-based: 바로 정책을 근사해 행동을 선택한다.

On-policy vs Off-policy

- On-policy: 하나의 정책 π 에 대해서 π 를 사용해서 얻은 결과만 사용해서 작동.
(ex) $Q(s, a) \leftarrow Q(s, a) + \alpha (R_s + \gamma Q(s', a') - Q(s, a))$
- Off-policy: 탐험 정책과 최종 정책이 서로 독립이다.
(ex) $Q(s, a) \leftarrow Q(s, a) + \alpha (R_s + \gamma \max_{a'} Q(s', a') - Q(s, a))$

Cross Entropy Method

Cross Entropy Method는 model-free, policy-based, on-policy 이다.


- 보상 함수와 상태를 예측할 필요가 없다.
- 정책을 바로 근사한다.
- 항상 최신 정책으로 만든 데이터만 가지고 학습한다.

Algorithm

- 1 현재 정책을 이용해 N 개의 에피소드를 진행한다.
- 2 각 에피소드마다 보상의 총합을 계산한다.
- 3 학습에 사용할 보상의 총합의 하한을 계산한다. (50th or 70th percentile)
- 4 3번에서 계산한 경계보다 높은 elite 에피소드를 가지고 학습을 진행한다.
- 5 만족스러울 때까지 1번으로 돌아가 위 과정을 반복한다.

구현하기

필요한 모듈을 로드한다.



```
import gym
from collections import namedtuple
import numpy as np

import torch
import torch.nn.functional as F
from torch import nn, optim
```

구현하기

학습에 쓰일 자료형을 만들고, 설정값을 지정한다.

```
HIDDEN_SIZE = 128
BATCH_SIZE = 100
PERCENTILE = 30
GAMMA = 0.9

Episode = namedtuple('Episode', field_names=['reward', 'steps'])
EpisodeStep = namedtuple('EpisodeStep', field_names=['observation', 'action'])
```


구현하기

Fronzen Lake의 observation을 신경망에 넣을 수 있도록 one-hot 벡터로 가공

```
class DiscreteOneHotWrapper(gym.ObservationWrapper):  
    def __init__(self, env):  
        super(DiscreteOneHotWrapper, self).__init__(env)  
        self.observation_space = gym.spaces.Box(  
            0.0, 1.0, (env.observation_space.n, ), dtype=np.float32)  
  
    def observation(self, observation):  
        res = np.copy(self.observation_space.low)  
        res[observation] = 1.0  
        return res
```

구현하기

학습 데이터를 만드는 함수.



```
def iterate_batches(env, net, batch_size):  
    batch = []  
    episode_reward = 0.0  
    episode_steps = []
```

구현하기

현재 정책 π 를 이용해 에피소드를 진행하며 학습 데이터를 만든다.

```
obs = env.reset()
while True:
    obs_v = torch.FloatTensor([obs])
    act_probs_v = F.softmax(net(obs_v), dim=1)
    act_probs = act_probs_v.data.numpy()[0]
    action = np.random.choice(len(act_probs), p=act_probs)

    next_obs, reward, done, _ = env.step(action)
```

구현하기

현재 정책 π 를 이용해 에피소드를 진행하며 학습 데이터를 만든다.

```
episode_reward += reward
episode_steps.append(EpisodeStep(observation=obs, action=action))

if done:
    batch.append(Episode(reward=episode_reward, steps=episode_steps))

    episode_reward = 0.0
    episode_steps = []
    next_obs = env.reset()

    if len(batch) == batch_size:
        yield batch
        batch = []

obs = next_obs
```


구현하기

모인 학습 데이터 중 elite episode만 추려내는 함수.

```
def filter_batch(batch, percentile):  
    disc_rewards = list(  
        map(lambda s: s.reward * (GAMMA ** len(s.steps)), batch))  
    reward_bound = np.percentile(disc_rewards, percentile)  
  
    train_obs = []  
    train_act = []  
    elite_batch = []  
    for example, discounted_reward in zip(batch, disc_rewards):  
        if discounted_reward > reward_bound:  
            train_obs.extend(map(lambda step: step.observation, example.steps))  
            train_act.extend(map(lambda step: step.action, example.steps))  
            elite_batch.append(example)  
  
    return elite_batch, train_obs, train_act, reward_bound
```

구현하기

정책을 근사할 신경망과 학습 도구 생성

```
if __name__ == "__main__":  
    env = DiscreteOneHotWrapper(gym.make('FrozenLake-v0', is_slippery=False))  
  
    obs_size = env.observation_space.shape[0]  
    n_actions = env.action_space.n  
  
    net = nn.Sequential(  
        nn.Linear(obs_size, HIDDEN_SIZE),  
        nn.ReLU(),  
        nn.Linear(HIDDEN_SIZE, n_actions)  
    )  
  
    objective = nn.CrossEntropyLoss()  
    optimizer = optim.Adam(params=net.parameters(), lr=0.001)
```

구현하기

학습 데이터 만들기

```
full_batch = []  
for iter_no, batch in enumerate(iterate_batches(env, net, BATCH_SIZE)):  
    reward_mean = float(np.mean(list(map(lambda s: s.reward, batch))))  
    full_batch, obs, acts, reward_bound = filter_batch(  
        full_batch + batch, PERCENTILE)  
  
    if not full_batch:  
        continue
```

구현하기

신경망 학습하기

```
obs_v = torch.FloatTensor(obs)
acts_v = torch.LongTensor(acts)
full_batch = full_batch[-500:]

optimizer.zero_grad()
action_scores_v = net(obs_v)

loss_v = objective(action_scores_v, acts_v)
loss_v.backward()

optimizer.step()

print("%d: loss=%.3f, reward_mean=%.3f, reward_bound=%.3f, batch=%d" % (
    iter_no, loss_v.item(), reward_mean, reward_bound, len(full_batch)))

if reward_mean > 0.8:
    print("Solved!")
    break
```