김범수

gym 튜토리얼

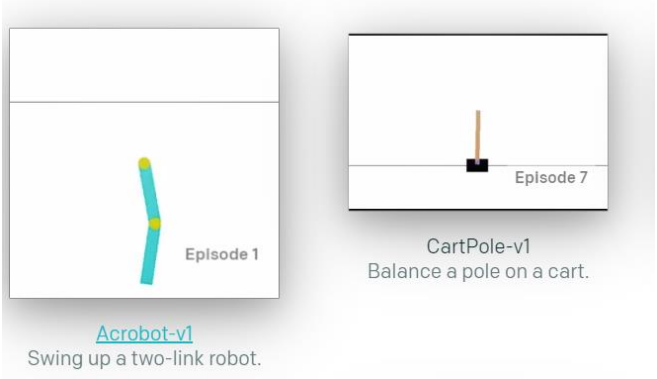
gym tutorial

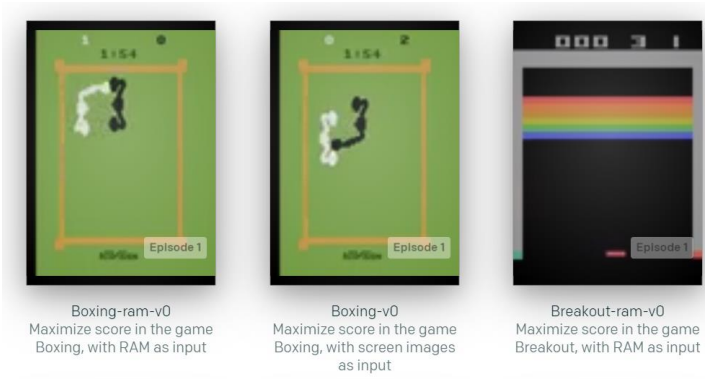
By POSCAT

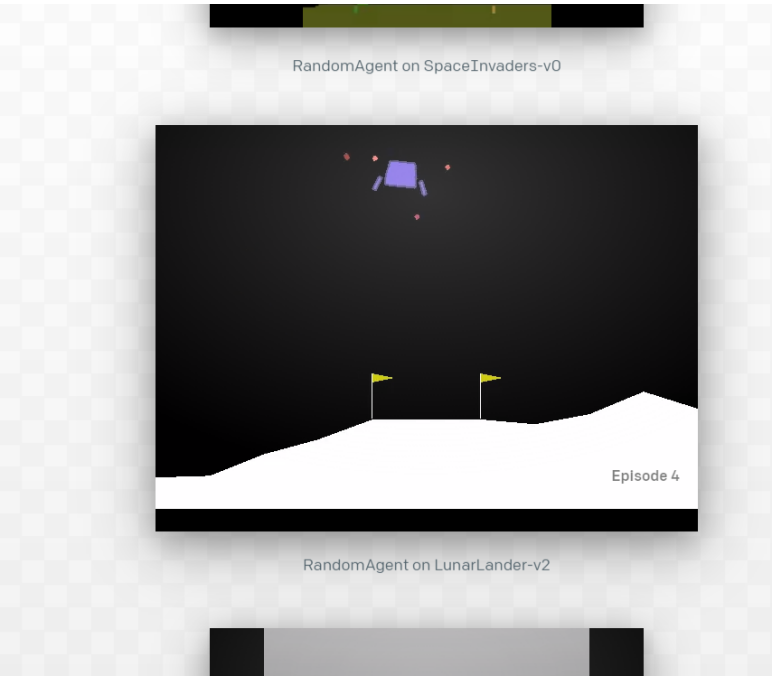
Contents

gym 설치

- Gym은 강화학습 알고리즘의 개발과 객관적인 비교를 위해 개발된 툴킷입니다. gym에서는 다양한 환경을 제공하고 있습니다. 간단한 조정부터 atari game까지 다양한 환경이 존재합니다.







gym 설치

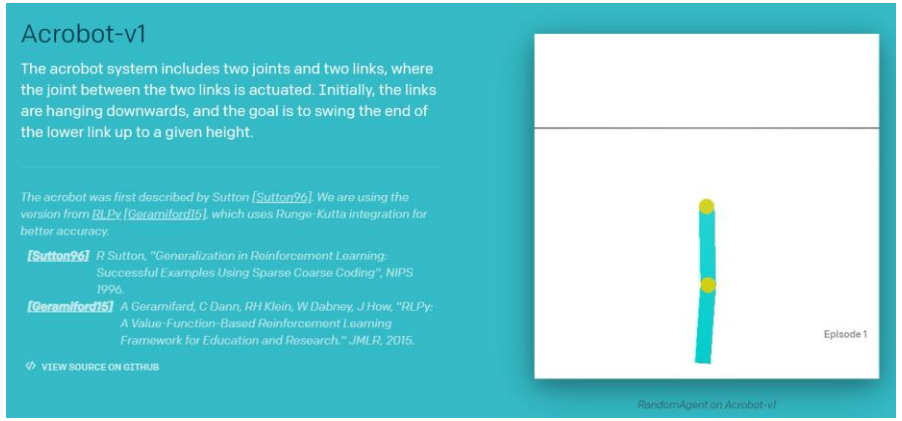
- 이처럼 gym은 강화학습의 객관적인 기준을 마련하기 위해 등장하였고, 현재까지도 많은 강화학습 논문에서는 gym을 이용하여 강화학습 모델의 성능을 수치화하고 있습니다.
- gym은 python에서 사용할 수 있으며, gym 모듈을 이용해 gym 환경을 사용할 수 있습니다.
- gym은 python 모듈 설치하듯이, pip install gym 명령어로 설치할 수 있습니다.

```
C:₩Users₩qjatn>pip install gym
```

- 이번 세미나 시간에는 gym의 기본적인 사용법을 알아보고, gym의 기본적인 문제들을 풀면서 사용법을 익힙니다.
- (이번 세미나에는 연습문제가 없습니다. 따라서 이번에는 세미나에서 푸는 문제들의 소스코드를 깃헙에 올려두었습니다.)

gym의 기본적인 함수들

- gym의 공식 사이트에서는 대표적인 환경들을 소개하고 있습니다.
- https://gym.openai.com/envs/ 해당 사이트를 들어가보면, 다양한 환경을 분야 별로 나누어 소개하고 있습니다.
- 각 환경에는 이름이 있습니다. Acrobot-v1, CartPole-v1 등의 이름이 존재합니다.
- 이름을 누르면, 오른쪽 화면처럼 해당 환경에 대한 자세한 설명이 나오고, VIEW SOURCE ON GITHUB를 통해 환경의 구현 방식을 알 수 있습니다. 대부분의 경우, 소스코드를 통해 해당 환경의 작동 방식을 이해합니다.



gym의 기본적인 함수들

- 먼저, GuessingGame-v0 문제를 풀어보면서 gym의 기초에 대해 배워봅시다. 아래 주소에서 해당 환경에 대한 설명을 볼 수 있습니다.
- https://gym.openai.com/envs/GuessingGame-v0/
- GuessingGame-v0는 흔히 업다운 문제라고 알려진 문제입니다. 게임에서 숨겨져 있는 숫자를 맞추는 문제입니다. 만약 우리가 제시한 숫자보다 숨겨진 숫자가 크면 up, 작으면 down이라고 알려주고, 범위를 좁혀가며 숫자를 맞추면 됩니다.
- 1% 오차 범위 안에서 숫자를 맞추면 정답으로 처리합니다. 예를 들어, 랜덤한 숫자의 범위가 1000이고, 숨겨진 숫자가 620이면, 1000의 1%인 10의 오차는 인정하여, 610~630의 숫자는 정답으로 처리합니다.
- 우리의 목표는 최소한의 질문으로 숨겨진 숫자를 맞추는 것입니다.

gym의 기본적인 함수들

- 먼저 gym 모듈을 가져옵니다.

```
1   import gym
```

- 그 후에는 gym의 make 함수를 이용하여 환경을 불러옵니다. gym.make("환경 이름") 형식으로 불러올 수 있습니다. 아래 코드에서는 GuessingGame 환경을 불러오고 이 환경을 env에 저장하고 있습니다.

```
3   env = gym.make("GuessingGame-v0")
```

- gym의 환경 env는 크게 2개의 함수로 시뮬레이션을 진행합니다.
- reset() 함수를 이용해 환경을 먼저 초기화합니다. reset 함수는 초기 state를 반환합니다.
- 그 후 step(action) 함수를 이용해 action을 취합니다. 환경이 종료될 때까지 step 함수로 계속 action을 취합니다.
- 이는 GuessingGame이외에도 모든 환경에서 적용되는 사항입니다.

gym의 기본적인 함수들

- 이 때 step(action) 함수는 observation, reward, done, info의 네 가지를 반환합니다.
- observation은 action을 취한 후의 state를 의미합니다. 환경의 종류에 따라 정수, 실수, 튜플 등 다양합니다.
- reward는 해당 action을 취한 후의 보상을 의미합니다. 모든 환경에서 reward는 실수(float)형 입니다.
- done은 환경이 종료되었는 지 여부입니다. True 또는 False의 값만 가지므로, 모든 환경에서 bool형 입니다.
- info는 디버깅을 할 때 추가적으로 주어지는 정보들입니다. 하지만, 문제를 풀 때, info를 사용하는 것은 금지되어 있으므로 우리는 사용하지 않는다고 생각해도 됩니다.
- 또한, env에는 render()함수가 존재하는 데, 자동차 조종 등의 환경에서 실제로 시뮬레이션 되는 모습을 보고 싶을 때, 이 함수를 사용하면 환경의 모습을 새 창에 띄워서 보여줍니다.

gym의 기본적인 함수들

- step 함수는 보통 아래와 같은 형태로 사용됩니다.

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

- state, reward는 어떤 값인지 등은 보통 VIEW SOURCE ON GITHUB를 통해 해당 환경의 코드를 보면 주석으로 설명하고 있습니다.
- GuessingGame 코드의 주석에서도 이러한 내용을 볼 수 있습니다.
- GuessingGame을 직접 풀어보며 아까 전의 함수들을 직접 사용해봅시다.

gym의 기본적인 함수들

- 먼저 GuessingGame의 state가 무엇인지 알아봅시다.
- 오른쪽의 주석을 보면, 초기 state는 0이고, action을 취해 숫자를 전달했을 때 목표 숫자와 동일하면 2, 만약 목표 숫자보다 작으면 1, 목표 숫자보다 크면 3이 된다는 것을 알 수 있습니다.

```
0 - No guess yet submitted (only after reset)
1 - Guess is lower than the target
2 - Guess is equal to the target
3 - Guess is higher than the target
```

- GuessingGame의 reward는 1% 오차범위 이내로 값을 맞추면 1, 아니면 0임을 알 수 있습니다.

```
The rewards are:
0 if the agent's guess is outside of 1% of the target
1 if the agent's guess is inside 1% of the target
```

- action은 우리가 추측할 숫자를 정수로 전달해주면 됩니다.

```
41          self.range = 1000  # Randomly selected number is within +/- this value
```

- 코드를 보면 주어지는 숫자는 -1000~1000 사이의 값 임을 알 수 있습니다.
- GuessingGame이는 render 함수가 존재하지 않음을 알 수 있습니다.

gym의 기본적인 함수들

- 먼저 GuessingGame을 불러오고, 환경을 초기화합니다. 그 후, 숫자의 범위 MIN~MAX를 MIN, MAX 변수에 저장합니다.

```
1   import gym
2
3   env = gym.make("GuessingGame-v0")
4
5   state = env.reset()
6   MIN = -1000
7   MAX = 1000
```

- GuessingGame의 풀이는 간단합니다. 중간값을 물어보고, 값의 범위를 절반으로 줄여나가는 방법으로 문제를 풀 수 있습니다. 따라서 action으로는 MIN과 MAX값의 평균으로 지정합니다.

```
10      action = (MIN + MAX) // 2
```

- 그 후 action을 취합니다.

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

gym의 기본적인 함수들

- state가 1이라면, 숨겨진 숫자는 action보다 크다는 뜻이 됩니다. 따라서 MIN이 action+1이 됩니다. 반면에 state가 3이라면, 숨겨진 숫자는 action보다 작다는 뜻이 됩니다. 따라서 MAX값이 action-1이 됩니다.

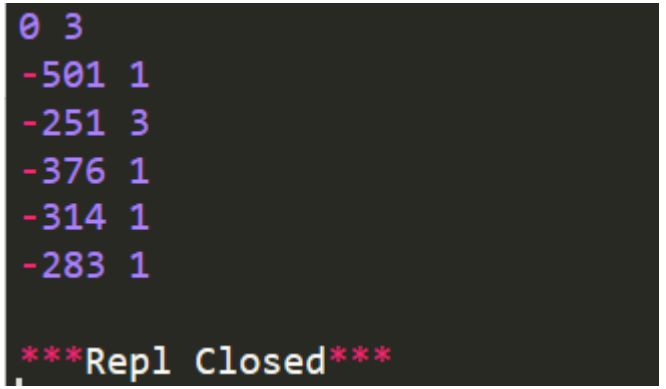
```
14          if state == 1:
15              MIN = action + 1
16          elif state == 3:
17              MAX = action - 1
```

- 만약 게임이 끝났다면, done은 True가 됩니다. 하지만, 게임이 끝나지 않았다면 done은 False가 됩니다. 따라서 done이 True가 될 때까지 게임을 반복합니다. 또한 실제로 게임이 잘 되는 지를 확인하기 위해 중간에 action과 state를 출력합니다.

```
 9 while 1:
10     action = (MIN + MAX) // 2
11     state, reward, done, _ = env.step(action)
12     print(action, state)
13
14     if state == 1:
15         MIN = action + 1
16     elif state == 3:
17         MAX = action - 1
18
19     if done:
20         break
```

gym의 기본적인 함수들

- 실제로 코드를 실행해보면 아래처럼 숨겨진 값을 잘 찾아, 6~7번만에 숨겨진 값을 찾는 것을 알 수 있습니다.



- 이번 문제는 간단해서 바로 풀이를 알 수 있었지만, 풀이를 몰라, Q-learning 등을 통해 풀이를 학습해야 하는 문제는 여러 번의 게임을 통해 학습을 진행할 수 있습니다.

Q-learning으로 frozen-lake 풀어보기

- 이번에는 4강에서 배웠던 Q-learning을 이용하여 frozen-lake 문제를 풀어봅니다. frozen game은 평범한 미로 문제와 거의 똑같지만, 바닥이 미끄러워 가끔 원하는 방향이 아니라 다른 방향으로 미끄러진다는 차이점이 있습니다. 따라서 일정 확률로 랜덤한 방향으로 이동하게 됩니다.
- 미로의 바닥은 총 4종류가 존재합니다. 시작점, 도착점, 얼음판, 구멍으로 우리의 목표는 구멍에 빠지지 않고, 얼음판 위를 잘 미끄러져서 시작점에서 도착점으로 최대한 빨리 가는 것입니다.
- frozen lake에서 action은 각 방향으로 한 칸 움직이는 것입니다. 각 방향은 오른쪽처럼 하나의 숫자에 대응합니다.

```
LEFT = 0
DOWN = 1
RIGHT = 2
UP = 3
```

- state는 정수로, 현재 위치가 어디인지를 의미합니다.
- reward는 goal에 도착하면 1, 아니면 0을 얻게 됩니다.
- Q-learning이 어떤 알고리즘인지는 다음 슬라이드의 수도코드를 보면서 떠올려봅시다.

Q-learning으로 frozen-lake 풀어보기

Code Explanation

```
Initialize Q(s, a) for all s∈ 𝒮 and a∈ 𝒜, arbitrarily and Q(terminal state, ) = 0
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e-greedy)
        Take action A, observe R, S'
        Q(S, A) <- Q(S, A) + alpha * [R + gamma * maxa Q(S', a)) – Q(S, A)]
        S <- S'
    until S is terminal
```

Q-learning으로 frozen-lake 풀어보기

- env에서는 action_space 변수와 observation_space 변수를 통해 action_space를 통해 state와 action에 대한 특징을 알 수 있습니다. 특히, action_space.n과 observation_space.n을 통해 내가 할 수 있는 action이 몇 종류인지, 그리고 가능한 state는 몇 종류인지 등을 알 수 있습니다.
- frozen-lake에는 총 4개의 action이 가능하고, 4x4의 미로이기 때문에 state는 총 16종류가 가능합니다.

```python
import gym

env = gym.make("FrozenLake-v0")

action_size = env.action_space.n
state_size = env.observation_space.n
print(action_size)
print(state_size)
```

```
4
16

***Repl Closed***
```

Q-learning으로 frozen-lake 풀어보기

- 먼저 frozen-lake 환경을 가져오고, action value를 저장할 Q table을 생성합니다.

```
1   import gym
2   import random
3
4   env = gym.make("FrozenLake-v0")
5
6   action_size = env.action_space.n
7   state_size = env.observation_space.n
8
9   Qtable = [[0] * action_size] * state_size
```

- Q table의 초기 모양은 아래와 같습니다. (Qtable[state][action]의 형태로 사용할 수 있습니다.)

```
[[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, 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, 0, 0]]
```

Q-learning으로 frozen-lake 풀어보기

- 먼저 학습률(learning rate), 게임 시행 횟수(max_episode) 등의 상수를 정의합니다.
- 이와 같이 중요한 상수들을 미리 정의해서 사용하면, 후에 상수 값을 바꾸고 싶을 때 간편하게 바꿀 수 있습니다.

```
11   alpha = 0.01 # learning rate
12   gamma = 0.9 # discount factor
13   epsilon = 0.05
14   max_episode = 10000
```

- 이제 본격적으로 학습을 시행합니다.

Q-learning으로 frozen-lake 풀어보기

- 먼저 environment를 초기화합니다.

```
16    for i_episode in range(1, max_episode + 1):
17        state = env.reset()
```

- 그 후 게임이 끝날 때까지 학습을 진행합니다.
- 먼저 epsilon greedy를 이용하여 action을 선택합니다.
- 1-epsilon의 확률로 action value가 가장 높은 action을 선택합니다.

```
25            action = -1
26
27            if random.random() > epsilon:
28                max_action_value = -100000
29                for i in range(4):
30                    if Qtable[state][i] > max_action_value:
31                        max_action_value = Qtable[state][i]
32                        action = i
```

- epsilon의 확률로 랜덤한 action을 선택합니다.

```
27        else:
28            action = random.choice([0, 1, 2, 3])
```

Q-learning으로 frozen-lake 풀어보기

- action을 취하고, next state, reward를 관찰합니다

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

- action value를 update합니다.

```
32      Qtable[state][action] += alpah * (reward + gamma * max(Qtable[next_state]) - Qtable[state][action])
33      state = next_state
```

- 만약 게임이 끝났다면 while문에서 빠져나옵니다.

```
35      if done:
36          break
```

- 이와 같이 하면 학습은 종료됩니다.

Q-learning으로 frozen-lake 풀어보기

- Frozen-lake는 미끄러지는 경우 때문에 최적의 답으로도 골인까지 가지 못하고 구멍에 빠질 확률이 존재합니다. 만약 미끄러지는 옵션을 끄고 싶다면, 아래 코드처럼 is_slippery 옵션을 끄면 됩니다. 또한 8x8을 붙이면, 8x8 크기의 미로에서 게임을 할 수 있습니다.

```
4   env = gym.make("FrozenLake8x8-v0", is_slippery = False)
```

- epsilon값이 너무 작으면, 처음에 도착 지점을 찾지 못하는 경우가 발생합니다. 따라서 epsilon을 상수로 잡지 않고, 처음에는 1로 잡은 후 점점 줄어들게 하는 방식을 사용하면, 더욱 빠르게 학습이 가능합니다.

- 주어진 문제에서는 도착 시에 1의 보상을 주는 방식으로 구현되어 있습니다. 하지만 이 경우, 위험을 감수하고 도착지점으로 가는 action보다 그냥 제자리에서 hole을 피하는 action을 하는 경향이 있어 학습이 비효율적으로 진행됩니다.따라서 도착 시에 1의 보상이 아닌, 이동할 때마다 -1의 보상을, 구멍에 떨어졌을 때에는 -100의 보상을 주는 방식으로 변경하면 효율적으로 학습을 진행할 수 있습니다.