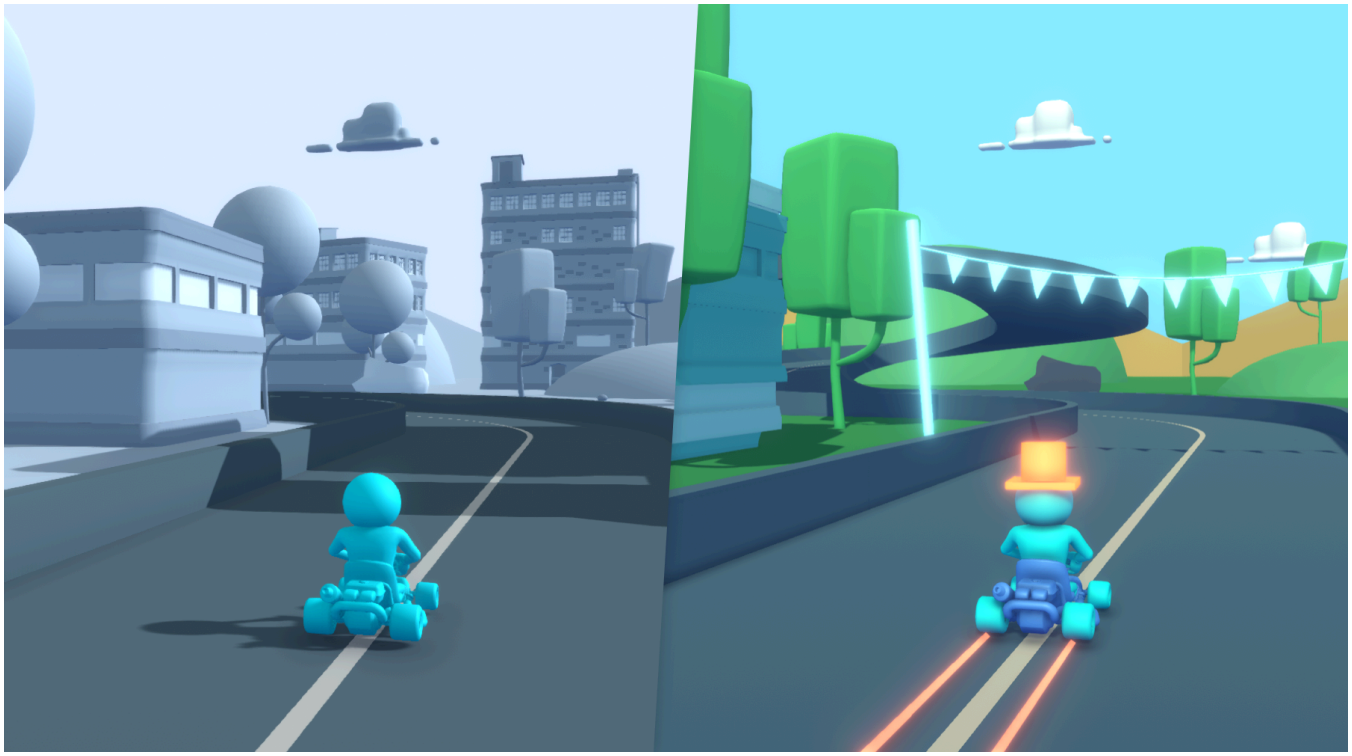

무작정 따라하는 강화학습 프로젝트

1. Introduction

강화학습은 **환경**(**Environment**)과 **에이전트**(**Agent**)의 상호작용이다.

이번 챕터에서는 이러한 *환경*으로 파이썬의 gym 모듈들 사용하고 *Deep q-learning*(*DQN*)을 학습 알고리즘으로 사용하여, 간단한 모델을 생성하여 학습하고 결과를 보겠다.
하지만, 알고리즘을 이해하기 위해서는 강화학습과 *DQN* 알고리즘을 이해할 필요가 있다.
따라서 강화학습의 기본 개념을 시작으로 *DQN*알고리즘을 알아보고, 배운 개념을 Python 코드로 짜보려고 한다.
마지막으로는 학습의 결과를 확인하고, 배운 개념을 정리하면서 끝맺도록 하겠다.

2. 강화학습의 기본 개념

1) 강화학습의 목표와 Bellman Equation

Reward Hypothesis에 의하면 에이전트의 목표는 보상 축적값의 최대화이다.

보상의 최대화가 아닌 보상 축적값의 최대화가 목표인 이유가 무엇인지 생각해 볼 필요가 있다.

> 예를 들어, 기업의 이윤을 최대화하는 에이전트를 생각해보자. 이때 나는 에이전트를 기업의 경영자 A라 가정하겠다.A의 목표는 1년동안의 이윤을 최대화 하기이다. 하지만 A는 여름 계절상품을 기획하였고, 기업의 여름 이윤은 급 상승 하였지만 다른 계절에는 좋은 성과를 만들지 못하였다. 이로 인해, 기업의 1년간 이윤은 평균적으로 낮아졌고 A의 목표를 달성하지 못했다. 여기서 A의 실패 요인은 *근시안적*인 목표 달성에 중점을 두었다는 것이다. 따라서 A가 성공하기 위해서는 최소 1년동안 이윤을 보장하는 기획이 필요하다.

위의 예를 보면 알 수 있듯이, 바로 얻을 수 있는 *보상(여름 동안의 기업이윤)*만을 고려하게 되면 *보상의 축적값(1년 동안 축적된 기업이윤)*이라는 목표를 달성할 수 없다. 따라서, 에이전트는 *근시안적*인 단일 보상이 아닌, 보상의 축적값을 최대화 할 수 있는 방법을 찾아야한다.

이러한 축적된 보상을 Return이라 하고, 다음과 같이 정의한다.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- t : *time-step* t는 에이전트와 환경의 상호작용을 순서대로 구분할 때 사용하는 인덱스이다.
- T : *Terminal time-step* T는 에이전트와 환경의 상호작용이 자연스럽게 종료되는 *time-step* 인덱스이다.
- R_t : *time-step* t에서의 보상(**Reward**)
- G_t : *time-step* t에서부터 T까지의 **축적된 보상**(**Return**)

에이전트의 목표인 *Return*의 최대화를 바탕으로, 강화학습의 목표를 다음곽 같이 설명할 수 있다.

에이전트가 ***Return***을 최대화 하는 행동을 선택할 때, 강화학습의 목표가 달성된다.

즉, 강화학습은 에이전트가 최적의 행동을 하도록 학습시킨다.

위같은 조건을 만족하기 위해서, 특정 상태의 가치와 그때 선택되는 행동의 가치를 판단할 필요가 있는데, 이를 결정하는 함수를 **Value Function**이라 하고 다음과 같이 정의한다.

State-value Function : $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right] \ , \forall s \in \mathcal{S}$

Action-value Function : $q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \ , \forall s \in \mathcal{S}, a \in \mathcal{A}$

- π : 에이전트의 *Policy*
- s : 환경에서 얻어진 *상태(State)* s
- γ : *step-size* γ는 미래 보상의 반영도를 결정한다. e.g. γ가 0이면 $\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} = R_{t+1}$이므로 즉각적인 보상만을 고려하게 되고, 1이면 $\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} = \sum_{k=0}^{\infty} R_{t+k+1}$이므로 먼 미래의 보상과 즉각적인 보상을 같은 정도로 고려하게 된다.
- $v_\pi(s)$: *Policy* π를 따라갈 때, *State* s에서의 가치
- $q_\pi(s,a)$: *Policy* π를 따라갈 때, *State* s에서 *Action* a를 선택하면 얻을 수 있는 가치

2) Q_learning 알고리즘이란?

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

Q-learning의 의사코드
출처: Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.

off-policy TD(Temporal Difference) control이란?

우선 **off-policy**와**on-policy**는 다음과 같은 방법이다.

- *on-policy*는 학습에 필요한 데이터를 얻은 *Policy*와 실제로 학습되는 *Policy*로 같은 *Policy*를 사용하는 방법이다.
- *off-policy*는 *on-policy*와 달리 데이터 수집과 실제 학습에 다른 *Policy*를 사용한다.

> 예를 들어, 자동차 에이전트를 학습시킨다고 생각하자. *on-policy*의 경우, 항상 가치가 높은 행동을 선택하게 되면 새로운 경로를 탐색할 수 없어진다. 하지만 *off-policy*를 사용하여 학습에 필요한 데이터를 얻기 위해 랜덤하게 행동을 선택하는 *Policy*를 사용하면 여러 경로를 탐색할 수 있게되고, 가치가 높은 행동을 선택하는 *Policy*를 학습시키면 모든 경로를 탐색할 수 있게 된다.
> 하지만 *on-policy*도 여러 경로를 탐색 못하는 것은 아니다. 예를 들어, *Q-learning* 의사코드에도 제시된 **ϵ-greedy Policy**의 경우 ϵ의 확률로 랜덤한 행동을 선택하고, $1-\epsilon$의 확률로 가치가 가장 높은 행동을 선택한다.

강화학습을 하기 위해 환경으로부터 데이터를 수집하고 가치를 수정하는 방법은 여러가지 있지만 간단히 3개로 나뉘어 진다.

1. 하나의 *Episode*를 진행한 후, 얻어진 데이터로 가치를 수정한다.
2. T번의 *time-step*후, 얻어진 데이터로 가치를 수정한다.
3. 한번의 *time-step*후, 얻어진 데이터로 가치를 수정한다.

여기서 **Temporal Diffrence (TD)** 는 이름에서도 예상할 수 있듯이 한번의 *time-step*후 가치를 수정한다. 즉, 에이전트가 행동을 선택하고 보상을 얻을 때마다 가치를 수정한다.

마지막으로 **control**은 얻어진 정보를 바탕으로 *Policy*를 수정함을 의미한다.

한 줄씩 알고리즘 해석하기

위의 개념을 바탕으로 한 줄씩 알고리즘을 해석하자.

> Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

1. *step-size* α와 ϵ-*greedy Policy*를 위한 ϵ을 선택한다.

> Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

2. 임의의 방법으로 모든 s와 행동 a에 대하여 $Q(s,a)$를 초기화한다.
   s가 *Episode*의 마지막(*Terminal*)이면 $Q(s,a) = 0$

- $\mathcal{S}$: *Terminal*을 제외한 모든 상태들의 집합
- $\mathcal{S}^+$: *Terminal*을 포함한 모든 상태들의 집합
- $\mathcal{A}(s)$: 상태 s에서 선택 가능한 모든 행동의 집합

> Loop for each episode:

3. *Episode*마다 안의 내용을 반복한다.

    - 여기서 '안'이란 파이썬 코드와 비슷하게 들여쓰기 한 부분을 의미한다.

> Initialize S

4. 임의의 방법으로 처음 상태 S를 선택한다.

> Loop for each step of episode:

5. 각 *step*마다 안에 있는 내용을 반복한다.

- step: 에이전트가 상태를 바탕으로 행동하고 보상을 받고 다음 상태를 확인하는 일련의 과정

> Choose A from S using policy derived from Q (e.g., ε-greedy)

6. Q를 사용한 *Policy*에 대하여 행동 *A*를 선택한다.

    - 여기서 *off-policy*가 보장되는 이유는 학습되는 *Policy*는 Q에 대한 ϵ-*greedy Policy*가 아닌 *greedy Policy*를 선택하기 때문이다.

> Take action A, observe R, S'

7. 에이전트는 선택된 행동을 실행하고 결과를 관찰한다.

> $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

8. 이는 $Q(S,A)$를 업데이트 하는 과정이다.

보면 *Gradient Descent*와 비슷한 모양을 하고 있는데,
$R + \gamma \max_a Q(S', a)$는 타켓이고,
$[R + \gamma \max_a Q(S', a) - Q(S, A)]$는에러로 해석할 수 있다.

$S \leftarrow S'$

9. 다음 상태를 지금 상태에 저장한다.

until S is terminal

10. 위 과정을 *Terminal* 상태에 도달할 때(*Episode*가 끝날 때) 까지 진행한다.

3) Deep q-learning이란?

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Deep Q-learning의 의사코드
출처: Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (Dec 2013). Playing Atari with deep reinforcement learning. Technical Report arXiv:1312.5602 [cs.LG], Deepmind Technologies.

위에서 **Q-learning** 알고리즘을 살펴보았다. 이를 뉴럴 네트워크, 그리고 **Gradient Descent**와 어떻게 결합했는지 살펴보도록 하겠다.

위에서도 알 수 있듯이 *Action Value*는 다음과 같이 정의된다.

$$v(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}R_{t+k+1}|S_t = s, A_t = a\right] \ , \forall s \in \mathcal{S}, a \in \mathcal{A}$$

하지만 ***Deep q-learning***에서는 이러한 가치함수와 뉴럴 네트워크를 결합한 함수를 다음과 같이 표현한다.

$$Q(\phi, a; \theta)$$

이 함수는 가중치가 θ인 뉴럴 네트워크에 대한 행동의 가치함수이다. 하지만 상태 s가 보이지 않는다. 대신에 ϕ가 보이는데, 이 ϕ는 다음과 같다.

ϕ는 상태를 전처리하는 함수이다.

> - 예를 들어, 논문에서는 아타리 게임을 학습시키는데 $\phi(s)$는 게임 화면의 이미지인 s를 전처리 하는 함수이고 다음과 같은 과정을 거친다.
>
> 1. 이미지를 흑백(gray-scale)로 변경한다.
> 2. 이미지 사이즈를 110x84 픽셀로 줄인다. (**down-sampling**)
> 3. 실제 플레이과 관련 있는 부분을 84x84 픽셀로 잘라낸다. (**cropping**)
> 4. 이러한 방법으로 얻어진 이미지를 4개 가중시켜 $\phi(s)$에 저장한다.

이제 ϕ를 이해했다. 다음으로 뉴럴 네트워크는 어떻게 결합되었는지 생각할 수 있는데, 이는 다음 문장을 보면 알 수 있다.

Deep q-learning* 에서는 ***Policy***가 뉴럴 네트워크이다. 즉, **Policy**에 환경의 상태값을 인풋하면 에이전트 행동의 확률을 리턴한다.

지금까지 *Deep q-learning* 알고리즘 이해를 위한 바탕을 모두 공부하였다.

이제부터 알고리즘을 한줄 씩 분석해 보겠다.

Initialize replay memory $\mathcal{D}$ to capacity N

1. 사이즈가 N인 *Replay Memory* $\mathcal{D}$를 초기화한다.

Initialize action-value function Q with random weights

2. 랜덤한 가중치인 θ에 대하여 행동 가치함수 Q를 초기화한다.

for episode $= 1, M$ **do**

3. M개의 *Episode*에 대하여 안의 내용을 반복한다.

Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

4. 데이터열 $s_1 = x_1$과 이를 전처리 한 $\phi_1 = \phi(s_1)$를 초기화 한다.

- x_1 : 환경에서 얻어진 상태이다.

> - 전 예문에서도 봤듯이 아타리 게임의 경우 x_1은 처음 얻어진 아타리 게임의 화면 이미지이다.

for $t = 1, T$ **do**

5. *step*마다 안의 내용을 반복한다.

- T : *Terminal State*

With probability ϵ select a random action a_t

6. ϵ의 확률로 랜덤한 행동 $a_t \in \mathcal{A}(s_t)$을 선택한다.

otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

7. $1 - \epsilon$의 확률로 *greedy*한 행동을 선택한다.

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

8. 행동 a_t를 취한 후, 환경(*emulator*)에서 보상 r_t와 상태(*image*) x_{t+1}를 가져온다.

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

9. s_{t+1}에 데이터열 s_t, a_t, x_{t+1}을 저장하고, 다음과 같은 방식으로 전처리 한다.
$\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$

10. 0*Replay Memory*에 $(\phi_t, a_t, r_t, \phi_{t+1})$를 저장한다.

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$

11. *Replay Memory* $\mathcal{D}$에서 랜덤한 *minibatch*를 추출한다.

> • 이러한 방법을 사용하는 이유는 데이터의 독립성을 보장하기 때문이다.

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

12. ϕ_{j+1}가 *Terminal*이면 $\gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) = 0$ 이므로 y_j는 위와 같이 정의된다.

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

Loss Function은 다음과 같이 정의한다.
$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$

이러한 관점에서 위의 **equation 3**은 다음과 같다.
$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\varepsilon}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$

13. 이제 다음과 같이 *Gradient Descent*를 실행한다.
$\forall i,\ \theta_i = \theta_i - \alpha \nabla_{\theta_i} L_i(\theta_i)$

- α : *learning rate* 또는 *step-size* (같은 표현이다.)

end for

14. $t = T$이면 안의 반복을 종료한다.

end for

15. 모든 *Episode*에 대하여 학습을 완료한 상태이다.

즉, 에이전트가 학습을 완료한 상태가 된다.

---

3. Summary

지금까지 *Bellman Equation*에서 시작하여, 가치함수, *Q-learning*, 그리고 *Deep Q-learning*의 기본개념을 살펴보았다. 다음 챕터에서는 **PyTorch**와 OpenAI에서 제작한 모듈인 **gym**을 사용하여 *Deep Q-learning* 알고리즘을 바탕으로 실제로 에이전트를 학습 해보겠다.