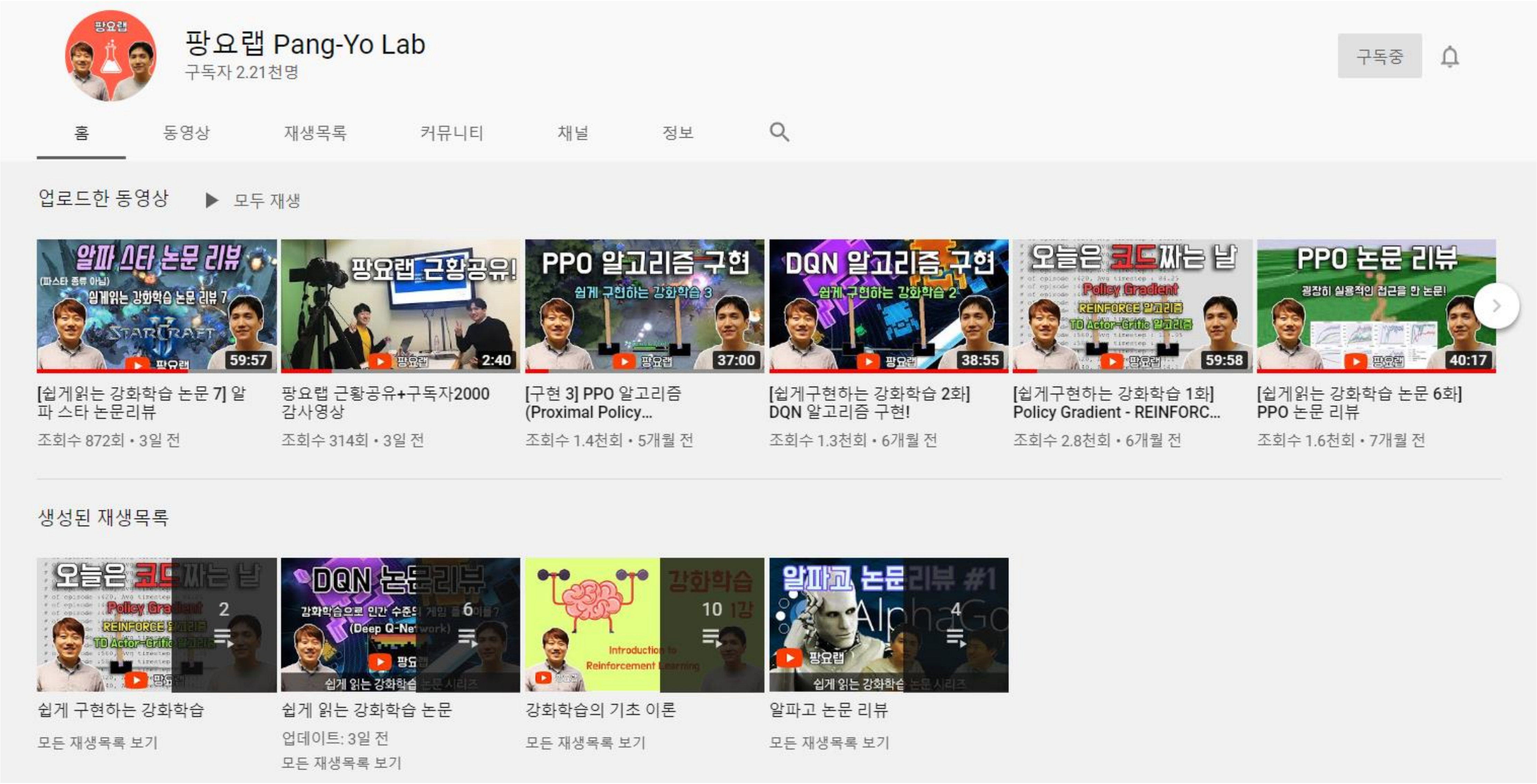
Policy Gradient

노승은
엔씨소프트 Game AI 랩
2019.11.21

Slides reference : Pieter Abbeel, David Silver


팡요랩 Pang-Yo Lab
구독자 2.21천명
구독중
홈
동영상
재생목록
커뮤니티
채널
정보
업로드한 동영상
모두 재생
[쉽게읽는 강화학습 논문 7] 알파 스타 논문리뷰
조회수 872회 • 3일 전
팡요랩 근황공유+구독자2000 감사영상
조회수 314회 • 3일 전
[구현 3] PPO 알고리즘 (Proximal Policy...
조회수 1.4천회 • 5개월 전
[쉽게구현하는 강화학습 2화] DQN 알고리즘 구현!
조회수 1.3천회 • 6개월 전
[쉽게구현하는 강화학습 1화] Policy Gradient - REINFORC...
조회수 2.8천회 • 6개월 전
[쉽게읽는 강화학습 논문 6화] PPO 논문 리뷰
조회수 1.6천회 • 7개월 전
생성된 재생목록
쉽게 구현하는 강화학습
모든 재생목록 보기
쉽게 읽는 강화학습 논문
업데이트: 3일 전
모든 재생목록 보기
강화학습의 기초 이론
모든 재생목록 보기
알파고 논문 리뷰
모든 재생목록 보기

1. 강화 학습 기초 복습

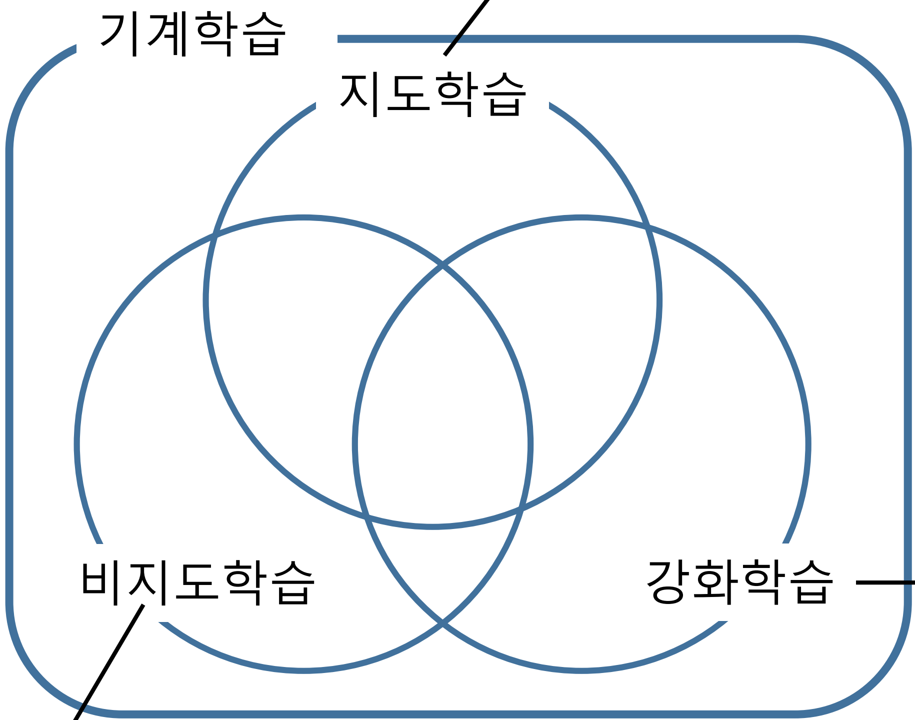
(1) 지도 학습과 강화 학습
(2) 보상
(3) MDP
(4) Model Free Prediction - MC와 TD

자전거 배우기

지도를 통한 학습

Trial & Error를 통한 학습

기계 학습의 분류



강화 학습이란

쉬운 버전	*"시행 착오를 통해 보상이 좋았던 행동은 더 하고, 보상이 적었던 행동은 덜 하며 발전하는 과정"*
정확한 버전	*"순차적 의사 결정 문제에서 누적 보상을 최적화 하기 위해 시행 착오를 통해 행동을 교정하며 학습하는 과정"*

보상의 특징

어떻게 X
얼마나 O

Scalar

스칼라

희소하고 지연된 보상

Reward Hypothesis

강화학습은 Reward Hypothesis 에 기반

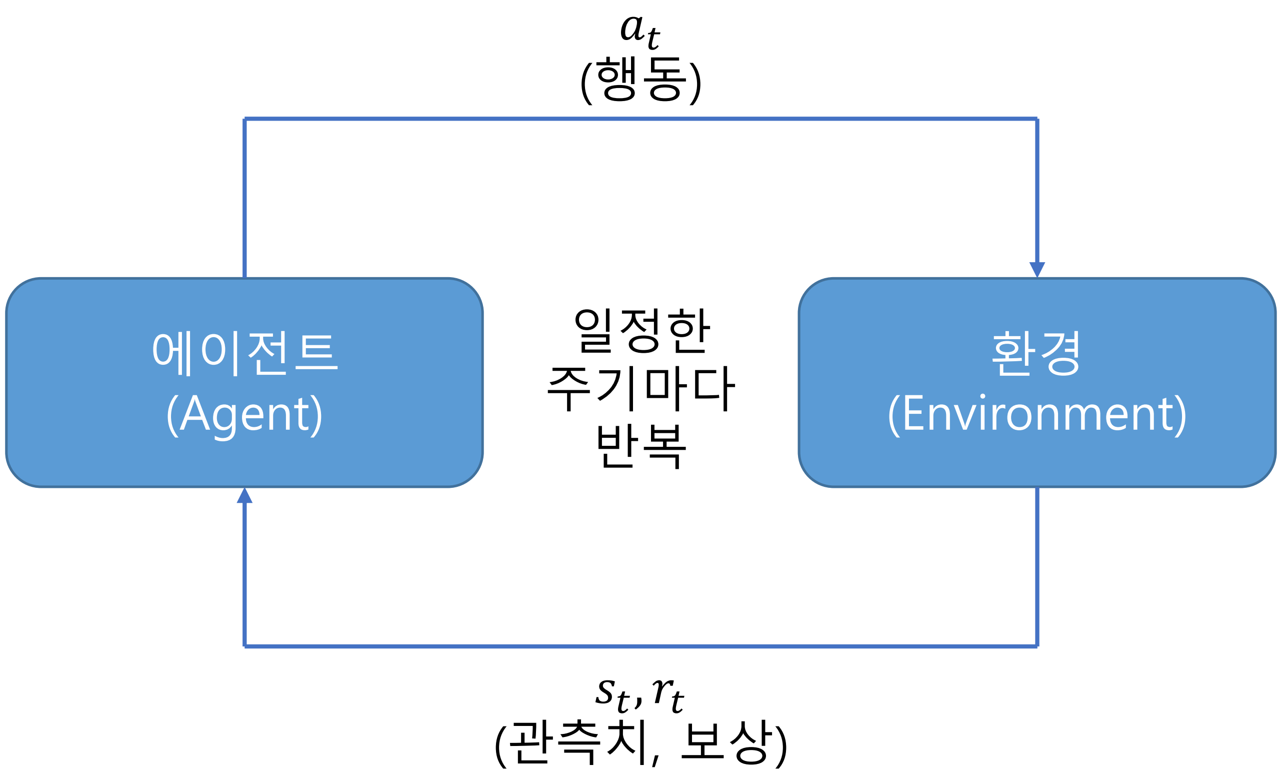
Definition (Reward Hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Reward 설계의 예시

- 바둑을 잘 둔다?
- 운전을 잘 한다?
- 로봇을 걷게 한다?
- 스타크래프트를 잘 한다?
- …

에이전트와 환경



- 에이전트
  - 환경으로부터 현재 시점 t에서의 환경에 대한 정보 s_t와 보상 r_t 를 받음
  - s_t를 바탕으로 어떤 행동을 해야 할지 결정.
  - 결정된 행동 a_t를 환경으로 보냄.
- 환경
  - 에이전트로부터 받은 행동 a_t를 통해서 상태 변화를 일으킴.
  - 그 결과 상태는 s_t -> s_{t+1}로 바뀜.
  - 에이전트에게 줄 보상 r_{t+1}도 함께 계산
  - s_{t+1}과 r_{t+1}을 에이전트에게 전달.

MDP - 정의

$$\mathbf{MDP} \;\equiv (\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \boldsymbol{\gamma})$$

- 상태의 집합 S
- 전이 확률 행렬 P

2. 액션 a를 선택 했을때

$$P^{a}_{\underline{s}\,\underline{s'}}$$

1. 상태 s에서

3. 다음 상태가 s′이 될 확률

$$P^{자장가}_{s_2, s_3} = 0.7,\; P^{자장가}_{s_2, s_4} = 0.3$$

- 액션의 집합 S
- 보상 함수 R

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

- 감쇠 인자 γ

Policy, Value

정의 1.

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

상태 s에서 액션 a를 선택할 확률

정의 2.

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

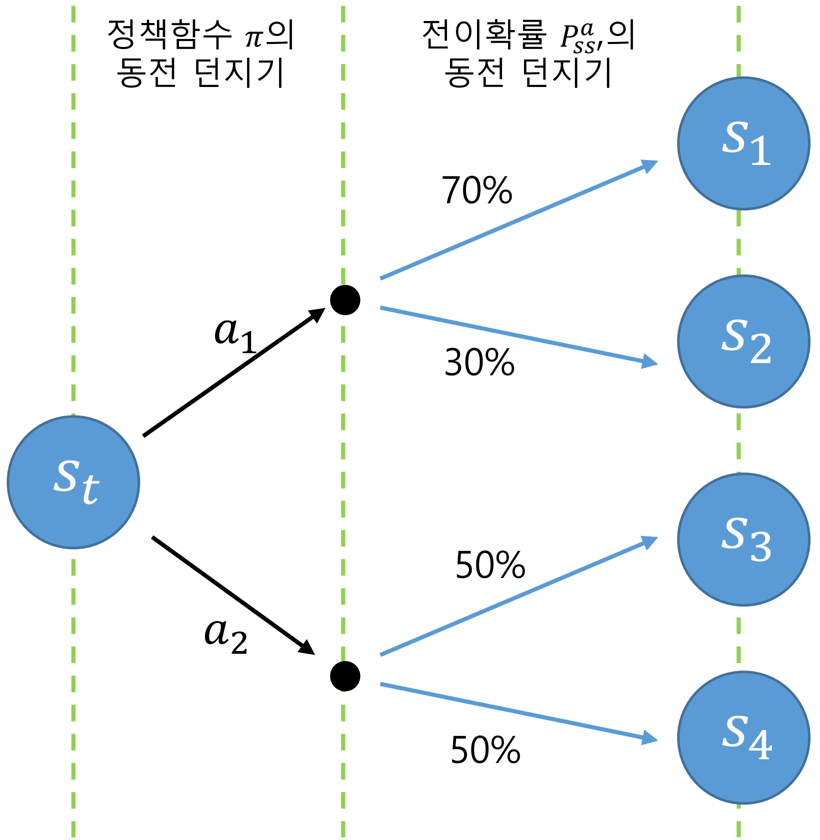
상태 s부터 π를 따라서 움직일 때 리턴의 기댓값

정의 3.

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

s에서 a를 선택하고, 그 이후에는 π를 따라서 움직일 때 얻는
리턴의 기댓값

상태 전이를 위한 두 번의 동전 던지기



벨만 기대 방정식

- 가치 함수는 두 파트로 나눠 생각할 수 있다.
  - 즉각적인 보상 R_{t+1}
  - 다음 상태의 가치 $\gamma v_\pi(s_{t+1})$

$$
\begin{aligned}
v(s) &= \boxed{\mathbb{E}\left[G_t \mid S_t = s\right]} \quad \text{기억하세요} \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \boxed{\mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]} \quad \text{기억하세요}
\end{aligned}
$$

- 액션-가치 함수도 마찬가지로 생각 가능.

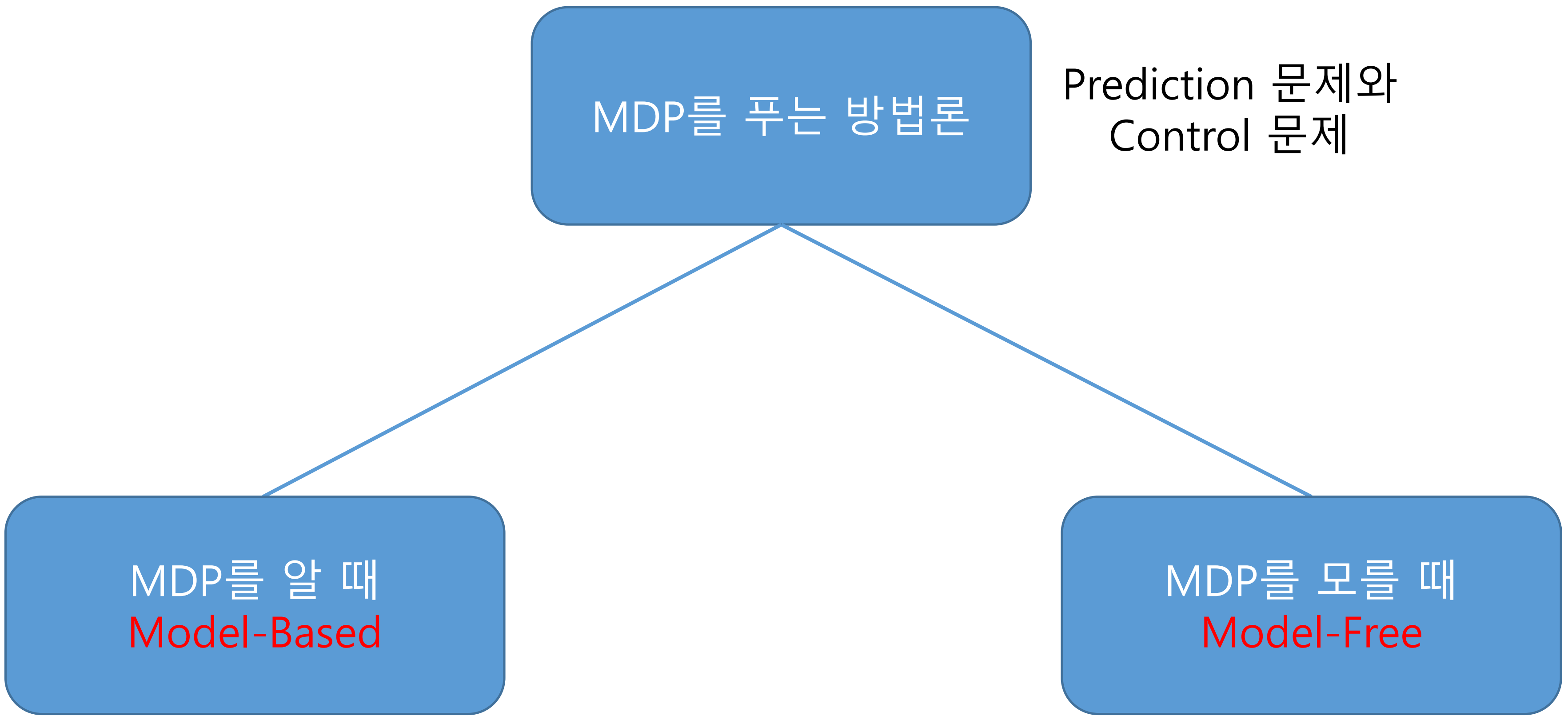
$$
q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]
$$

OX 퀴즈!

① $v_\pi(s_t) \;=\; r_{t+1} + \gamma v_\pi(s_{t+1})$ 가 성립한다. X

② $v_\pi(s_t) \;=\; \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+1} + \gamma^2 v_\pi(s_{t+2})]$ 가 성립한다. O

Model-Free Prediction



몬테 카를로 방법론

- MC는 경험으로부터 직접 배우는 방법론
- MC는 model-free 방법론
  - MDP의 상태 전이나 보상 함수에 관한 지식이 전혀 필요 없음
- MC는 **완전한** 에피소드로부터 배움
  - 에피소드가 끝나야 배울 수 있다는 뜻
- MC는 세상에서 가장 간단한 아이디어를 쓴다
  - 가치 = 평균 리턴
- 참고 : MC는 에피소드 단위로 끝나는 MDP에서만 사용할 수 있다.
  - 안 끝나는 MDP에서는 사용 불가.

몬테 카를로 방법론

- 목표 : 정책 π를 이용해 얻은 에피소드들로 부터 가치 함수 v_π 학습하기

$$S_{1,} A_{1,} R_{2,} \ldots, S_{k,} \sim \pi$$

- 리턴이 누적된 보상의 합임을 기억

$$\boldsymbol{G_t} = \boldsymbol{R_t} + \boldsymbol{\gamma R_{t+1}} + \boldsymbol{\gamma}^2 \boldsymbol{R_{t+2}} + \cdots + \boldsymbol{\gamma}^{\mathrm{T}-1} \boldsymbol{R_T}$$

- 가치 함수는 리턴의 기댓값임을 기억

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

- Monte-Carlo policy evaluation은 기댓값 대신에 실제 리턴의 평균을 사용

조금씩 업데이트 하는 버전

$$V(S_t) \leftarrow (1-\alpha) * \underbrace{V(S_t)}_{\text{원래 값 90\%랑}} + \alpha * \underbrace{G_t}_{\text{새로운 값 10\%를 섞음.}}$$

α가 0.1이라면 : 원래 값 90%랑 새로운 값 10%를 섞음.

Temporal-Difference 학습

- TD 방법론은 경험으로부터 직접 학습한다
- TD는 model-free 방법론. MDP에 대한 정보를 필요로 하지 않는다.
- TD는 에피소드가 끝나지 않아도 학습할 수 있다.
- TD는 추측을 추측으로 업데이트 하는 방법론이다.

Key Idea
모레에 비가 오는지 알고싶어?
오늘 추측하는 것 보다는 내일 추측하는게 더 정확하겠지!

MC 와 TD

MC

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

α가 0.1이라면 : 원래 값 90%랑 새로운 값 10%를 섞음.

TD

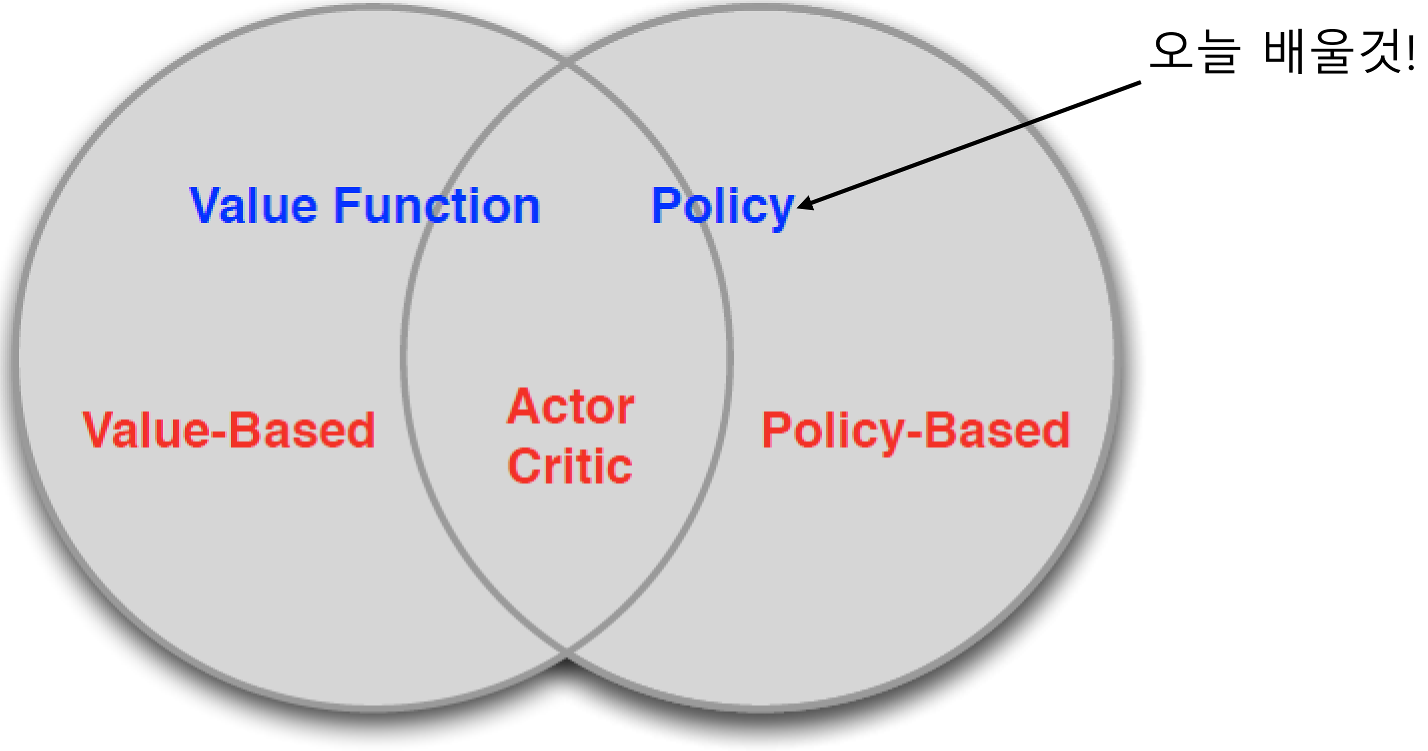
$$V(S_t) \leftarrow (1-\alpha) * V(S_t) + \alpha * (R_{t+1} + \gamma V(S_{t+1}))$$

α가 0.1이라면 : 원래 값 90%랑 새로운 값 10%를 섞음.
TD Target이라고 부름.

2. Policy Gradient

(1) Neural Network
(2) Policy Gradient의 계산

RL Agent의 카테고리

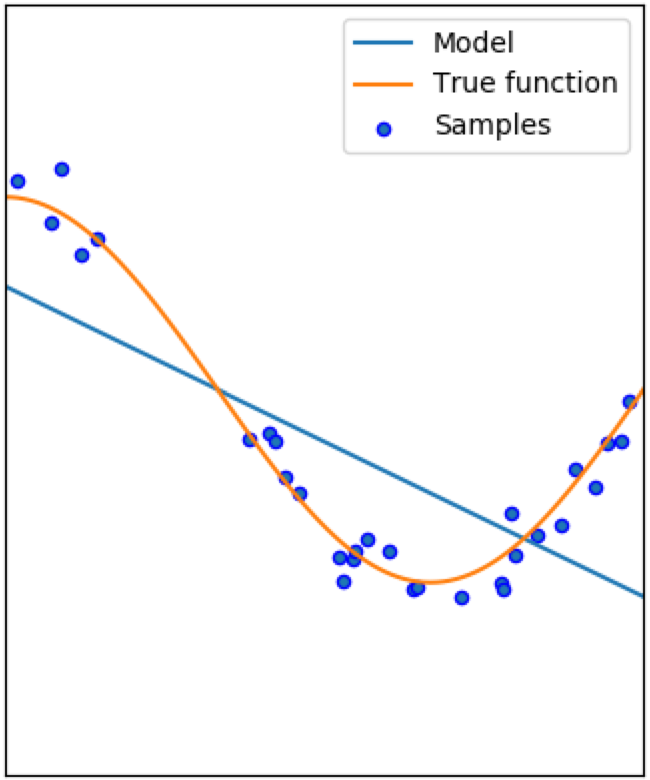


Policy-Based RL의 장점

- 수렴이 잘 됨
- 액션의 차원이 높거나 continuous action space면 Value-Based 방법론이 고전함
- Stochastic Policy를 배울 수 있음

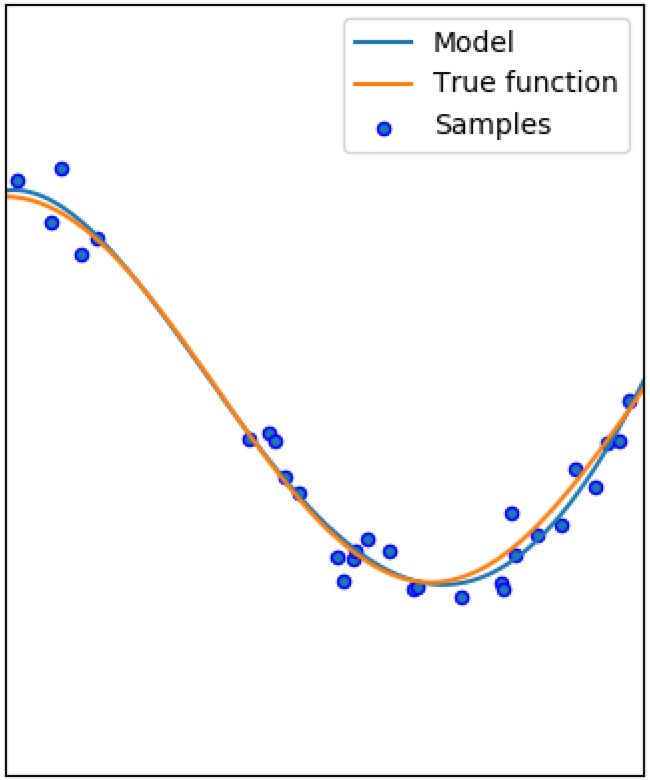
데이터와 모델링

1차 모델



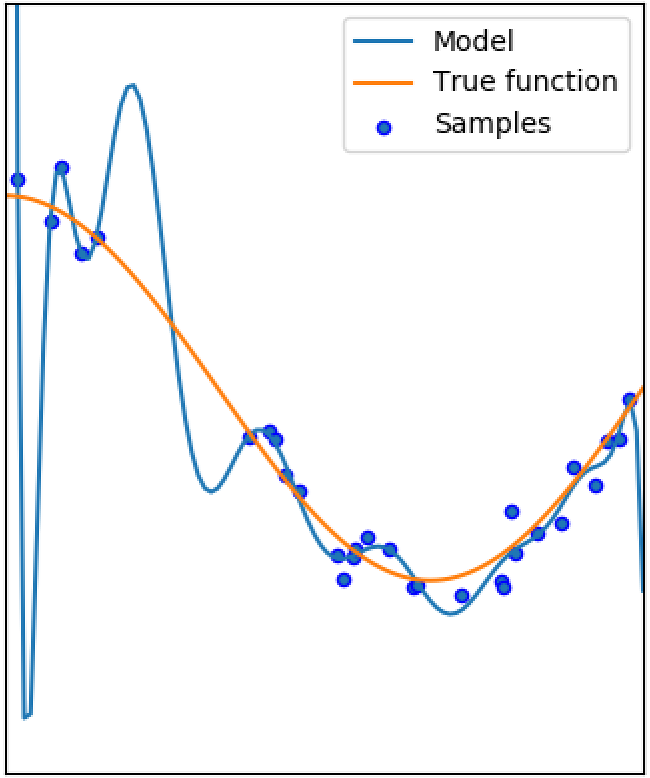
$$y = a_1 x + a_0$$

4차 모델



$$y = a_4 x^4 + a_3 x^3 + \cdots + a_1 x + a_0$$

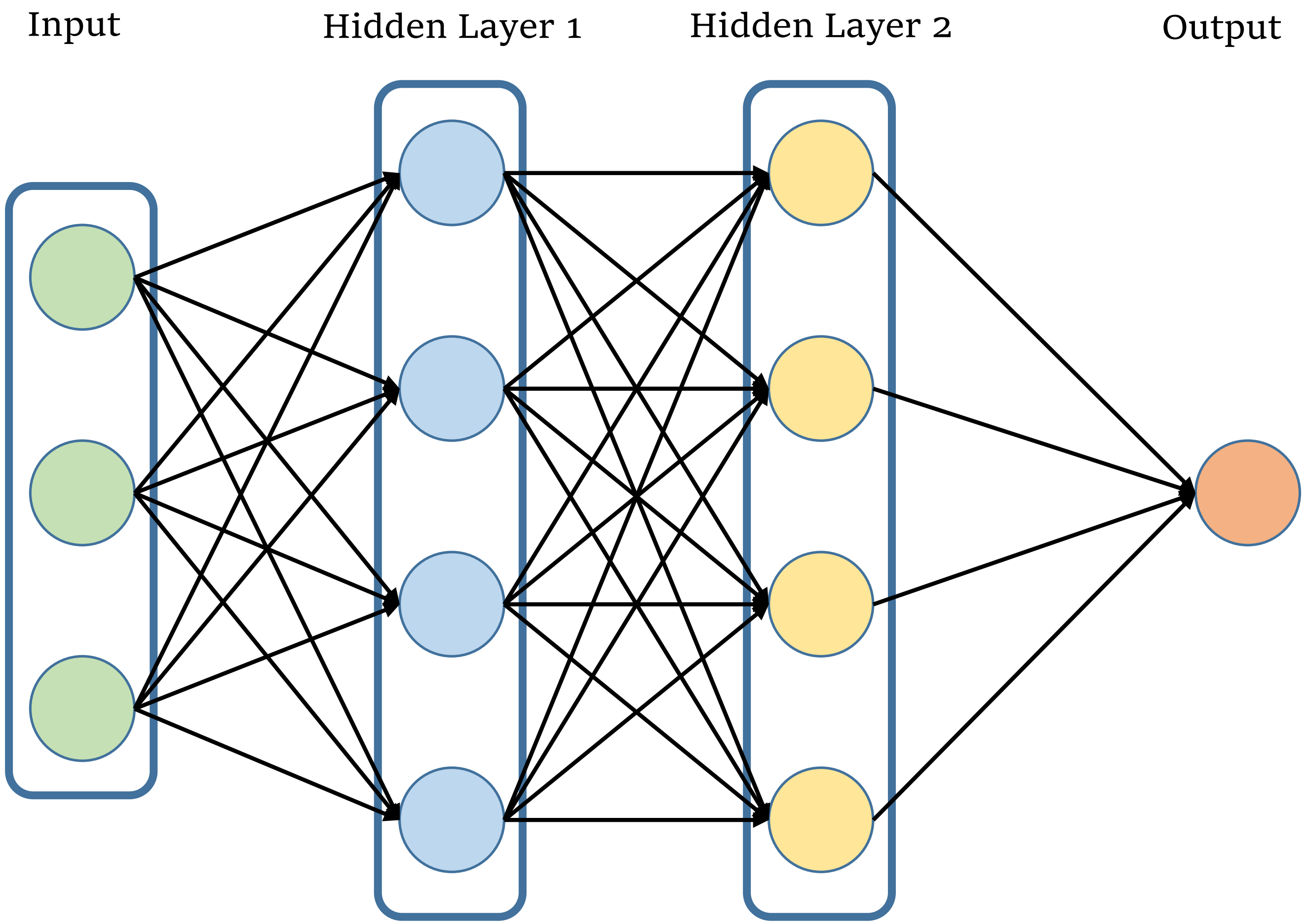
15차 모델



$$y = a_{15} x^{15} + a_{14} x^{14} + \cdots + a_1 x + a_0$$

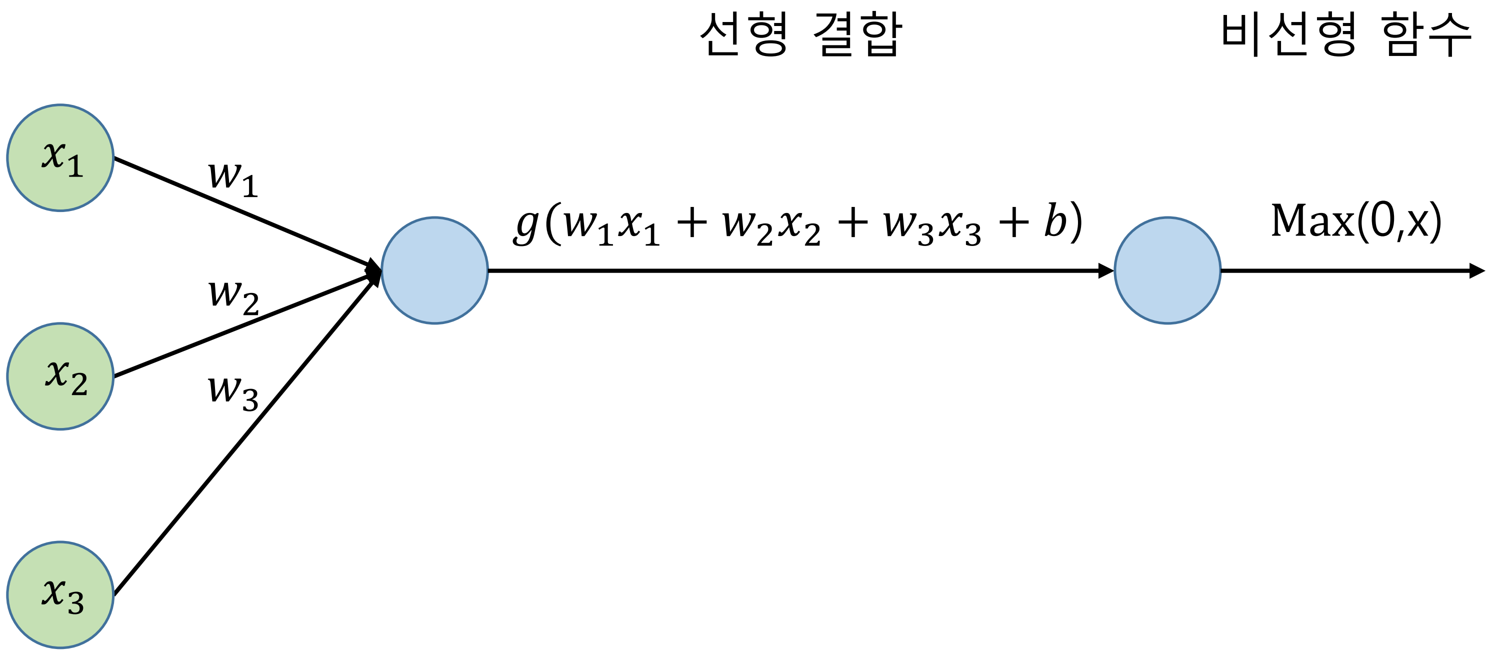
https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

Neural Network



여기 아주 유연한 함수가 하나 있다.

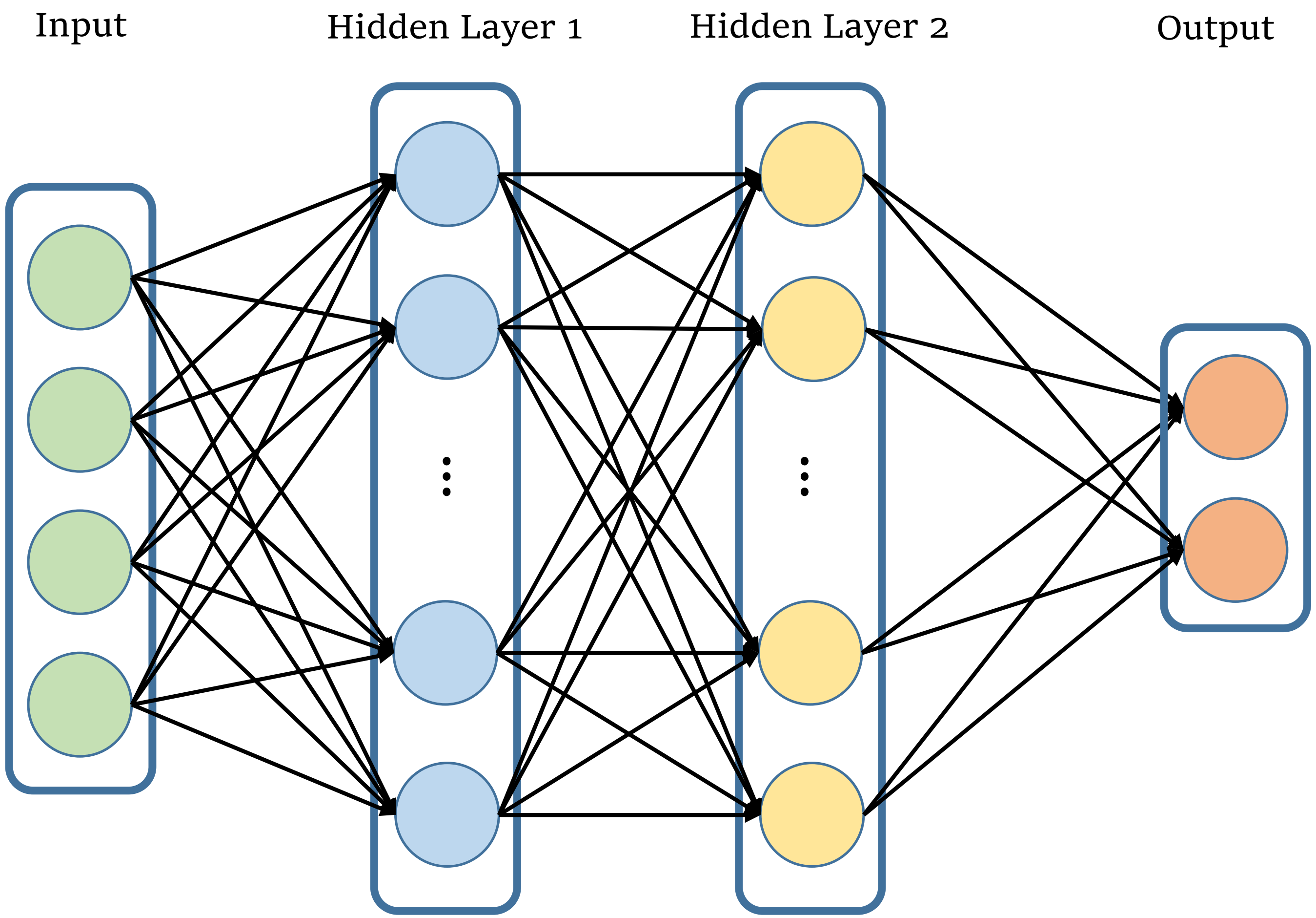
Neural Network



Neural Network을 통한 Value function 학습

- 학습 과정 : 뉴럴 넷을 수정해 주는 과정
- 정답 보다 뉴럴넷 아웃풋이 작으면 -> 아웃풋이 커지도록
  정답 보다 뉴럴넷 아웃풋이 크면 -> 아웃풋이 작아지도록
- 어떻게?
  - Gradient Descent(경사 하강법)!
- 결국 뉴럴넷에 정답을 새겨 넣는 것. 마치 테이블과 비슷함.
- 대신 좀 더 generalization이 잘 될 뿐.

Neural Net을 이용한 Policy의 표현



$$\pi_\theta(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

π_θ는 어떻게 수정하지…?

- Gradient Descent (or Ascent)를 써 보자!
- 뭐에 대한 gradient…?
- π_θ의 성능을 나타내는 함수 $U(\theta)$ 가 필요하겠군!
- $U(\theta)$는 어떻게 정의하지…?

목적함수 $U(\theta)$ 정의하기

$$U(\theta) = E[\sum_{t=0}^{T} R(s_t, a_t) \,|\pi_\theta]$$

- 세상 간단한 아이디어 : π_θ를 따랐을 때 평균 얼만큼의 보상을 받는가
- 미분을 하려면 기댓값 연산자가 없어야 하는데…

목적함수 $U(\theta)$ 정의하기

$$U(\theta) = E[\sum_{t=0}^{T} R(s_t, a_t) \,|\pi_\theta]$$

- 기댓값 연산자를 없애보자
- $s_0, a_0, s_1, a_1, \ldots, s_T, a_T$를 τ라 하자
- $\mathrm{R}(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$

$$U(\theta) = E[\textstyle\sum_{t=0}^{T} R(s_t, a_t) \left|\pi_\theta\right] = \displaystyle\sum_{\tau} P(\tau|\theta)\, \mathrm{R}(\tau)$$

- 우리의 목표는 $U(\theta)$를 최대로 하는 θ^*를 찾는 것

미분 해보자!

$$U(\theta) = \sum_{\tau} P(\tau|\theta)\, \mathrm{R}(\tau)$$

- θ에 대해 gradient를 취하면

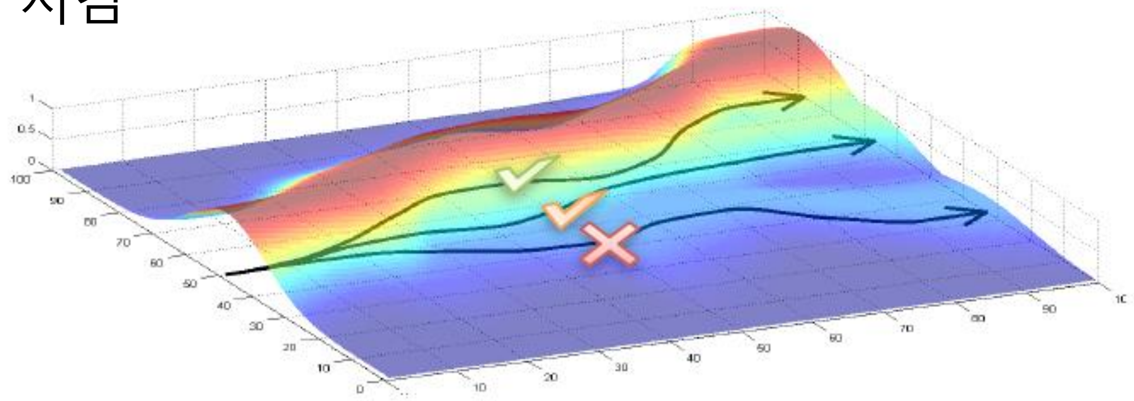
$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau|\theta)\, \mathrm{R}(\tau) \\
&= \sum_{\tau} \mathrm{R}(\tau) \nabla_\theta P(\tau|\theta) \\
&= \sum_{\tau} \frac{P(\tau|\theta)}{P(\tau|\theta)} \mathrm{R}(\tau) \nabla_\theta P(\tau|\theta) \\
&= \sum_{\tau} P(\tau|\theta) \mathrm{R}(\tau) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} \\
&= \sum_{\tau} P(\tau|\theta)\, \mathrm{R}(\tau) \nabla_\theta \mathrm{log} P(\tau|\theta)
\end{aligned}$$

직관적 이해

$$\nabla_\theta U(\theta) = \sum_\tau P(\tau|\theta)\,\mathrm{R}(\tau)\nabla_\theta \log P(\tau|\theta)$$

$$= \mathrm{E}_\tau[\mathrm{R}(\tau)\nabla_\theta \log P(\tau|\theta)]$$

- Gradient 를 통해서
  - $\mathrm{R}(\tau)$이 +면 그 경로의 로그 확률을 증가 시킴
  - $\mathrm{R}(\tau)$이 -면 그 경로의 로그 확률을 감소 시킴



τ 를 없애고 싶은데…

$$\nabla_\theta U(\theta) = \mathrm{E}_\tau[\mathrm{R}(\tau)\underline{\nabla_\theta \log P(\tau|\theta)}]$$

- $P(\tau|\theta)$에서부터 τ 를 없애보자

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \log \left[\prod_{t=0}^{T} \underbrace{Pr(s_{t+1}|s_t, a_t)}_{\text{Dynamics model}} * \underbrace{\pi_\theta(a_t|s_t)}_{\text{Policy}}\right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{T} \log Pr(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t)\right]$$

$$= \nabla_\theta \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t)$$

$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

τ 를 없애고 싶은데…

- R(τ)에서도 τ를 없애보자

$$\nabla_\theta U(\theta) = \mathrm{E}_\tau[\mathrm{R}(\tau)\nabla_\theta \mathrm{log} P(\tau|\theta)]$$

$$= \mathrm{E}_\tau\left[\mathrm{R}(\tau)\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\right]$$

$$= \mathrm{E}_\tau\left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=0}^{T} R(s_t, a_t)\right)\right]$$

$$= \mathrm{E}_\tau\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{t-1} R(s_k, a_k) + \sum_{k=t}^{T} R(s_k, a_k)\right)\right]$$

τ 를 없애고 싶은데...

$$\nabla_\theta U(\theta) = \mathrm{E}_\tau\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(\underbrace{\sum_{k=0}^{t-1} R(s_k,a_k)}_{a_t\text{보다 과거에 일어난 일}} + \sum_{k=t}^{T} R(s_k,a_k)\right)\right]$$

$$= \mathrm{E}_\tau\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{k=t}^{T} R(s_k,a_k)\right]$$

$$= \mathrm{E}_\tau\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t\right]$$

Policy Gradient 완성!

- 기댓값 E_τ는 어떻게 계산하지…!?

$$\nabla_\theta U(\theta) = \mathrm{E}_\tau\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t\right]$$

- Sample Mean을 이용하자 !!

$$\nabla_\theta U(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t$$

3. Policy Gradient - REINFORCE 실습

AutoDiff 이용하기

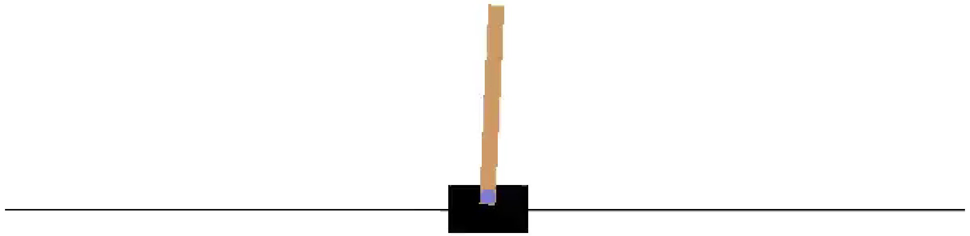
- 구해야 하는 것 : $\nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t \;=\; G_t \,*\, \nabla_\theta \log \pi_\theta(a_t|s_t)$

- 누구를 미분하면 위의 식이 되지!? : $G_t \,*\, \log \pi_\theta(a_t|s_t)$

- 그러면 loss는 어떻게!? : $-\,G_t \,*\, \log \pi_\theta(a_t|s_t)$

- –는 왜 붙지!? : loss 는 자동으로 minimize 시키는데 우리는 maximize 하고 싶으니까!

문제 - CartPole



- 카트를 잘 밀어서 균형을 잡는 문제
- 카트를 왼쪽이나 오른쪽으로 밀 수 있음
- 매 스텝마다 +1의 보상을 받음
- 막대가 수직으로부터 15도 이상 기울어지거나 화면 끝으로 나가면 종료

Import & Hyperparameter setting

```
import gym
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical

#Hyperparameters
learning_rate =      ?
gamma         =
```

- Learning rate와 gamma는 몇으로 해야 할까요?

Main

```python
def main():
    env = gym.make('CartPole-v1')
    pi = Policy()
    score = 0.0
    print_interval = 20

    for n_epi in range(10000):
        s = env.reset()
        for t in range(501): # CartPole-v1 forced to terminates at 500 step.
            prob = pi(torch.from_numpy(s).float())
            m = Categorical(prob)
            a = m.sample()
            s_prime, r, done, info = env.step(a.item())
            pi.put_data(        ?        )

            s =        ?
            score += r
            if done:
                break
```

Tensor(0.0255, -0.0159, -0.04898, -0.0408)

Tensor(0)

Policy

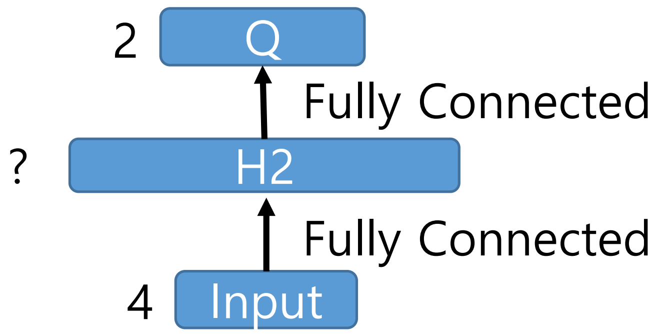
```python
class Policy(nn.Module):
    def __init__(self):
        super(Policy, self).__init__()
        self.data = []

        self.fc1 = nn.Linear(4,  ?  )
        self.fc2 = nn.Linear( ?  2)
        self.optimizer = optim.Adam(self.parameters(), lr=learning_rate)

    def forward(self, x):
        x =     ?    (self.fc1(x))
        x =      ?      (self.fc2(x), dim=0)
        return x

    def put_data(self, item):
        self.data.append(item)

    def train_net(self):
        R = 0
        for r, prob in self.data[::-1]:
            R = r + gamma * R
            loss =            ?
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
        self.data = []
```



$-\log \pi_\theta(a_t|s_t)\, G_t$

학습 결과

```
# of episode :20, avg score : 19.1
# of episode :40, avg score : 25.85
# of episode :60, avg score : 26.85
# of episode :80, avg score : 28.2
# of episode :100, avg score : 39.85
# of episode :120, avg score : 51.55
# of episode :140, avg score : 46.15
# of episode :160, avg score : 65.5
# of episode :180, avg score : 73.9
# of episode :200, avg score : 69.7
# of episode :220, avg score : 86.2
# of episode :240, avg score : 93.35
# of episode :260, avg score : 126.55
# of episode :280, avg score : 160.3
# of episode :300, avg score : 104.55
# of episode :320, avg score : 249.3
# of episode :340, avg score : 249.55
# of episode :360, avg score : 276.05
# of episode :380, avg score : 184.75
# of episode :400, avg score : 119.5
# of episode :420, avg score : 184.4
# of episode :440, avg score : 288.8
# of episode :460, avg score : 121.9
# of episode :480, avg score : 68.1
# of episode :500, avg score : 78.65
# of episode :520, avg score : 95.65
# of episode :540, avg score : 191.9
# of episode :560, avg score : 166.9
# of episode :580, avg score : 132.0
# of episode :600, avg score : 365.5
```

변형

- 네트워크 사이즈 – Layer 수, Node 수
- Gamma
- Reward 스케일
- Learning Rate
- 배치 처리?

$$\nabla_\theta U(\theta) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t | s_t)\, G_t}_{\text{다 더한 값을 한번에 BAAM 업데이트}}$$

4. Vanilla Actor-Critic

Critic의 등장

- 조금 다른 형태로 부터 출발

$$\nabla_\theta U(\theta) = \mathrm{E}_\tau\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t\right]$$

$$= E_{\pi_\theta}\left[\nabla_\theta\, log\, \pi_\theta(a|s) Q_{\pi_\theta}(s,a)\right]$$

- 실제 Q 함수를 모르기 때문에 또다른 뉴럴넷으로 모사

$$\boldsymbol{\nabla_\theta U(\theta) \approx E_{\pi_\theta}[\nabla_\theta\, log\, \pi_\theta(a|s) Q_w(s,a)]}$$

Variance 줄이기

- State s 에대한 임의의 함수 B(S)에 대해 다음이 성립

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)B(s)\right] &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s)\sum_a \nabla_\theta\pi_\theta(s,a)B(s)\\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}B(s)\nabla_\theta\sum_{a\in\mathcal{A}}\pi_\theta(s,a)\\ &= 0\end{aligned}$$

- Policy Gradient 수식에서 B(s)를 빼줌으로서 variance를 줄일 수 있다.

- State에 대한 대표적인 함수로는 v(s)가 있음!

$$\begin{aligned}\nabla_\theta U(\theta) &= E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)Q_{\pi_\theta}(s,a)\right]\\ &= E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)(Q_{\pi_\theta}(s,a)-V_{\pi_\theta}(s))\right]\\ &= \boldsymbol{E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)A_{\pi_\theta}(s,a)\right]}\end{aligned}$$

단점

$$\nabla_\theta U(\theta) = E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))\right]$$

- θ ,w, v 세 벌의 파라미터가 필요

$$Q_{\pi_\theta}(s,a) \approx Q_w(s,a)$$

$$V_{\pi_\theta}(s) \approx V_v(s)$$

$$A_{\pi_\theta}(s,a) \approx Q_w(s,a) - V_v(s)$$

- TD 러닝과 같은 방법으로 Q,V 두개의 가치 함수 모두 학습시켜줘야 함

해결책 : TD 에러

- True value function $V_{\pi_\theta}(s)$ 에 대한 TD 에러 δ_{π_θ}를 생각해보자.

$$\delta_{\pi_\theta} = r + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)$$

- δ_{π_θ}는 advantage 함수의 unbiased estimate 이다!

$$\begin{aligned} E_{\pi_\theta}\left[\delta_{\pi_\theta} | s, a\right] &= E_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s') | s, a\right] - V_{\pi_\theta}(s) \\ &= Q_{\pi_\theta}(s, a) - V_{\pi_\theta}(s) \\ &= A_{\pi_\theta}(s, a) \end{aligned}$$

- 따라서 TD에러를 이용해 policy gradient를 계산할 수 있따

$$\nabla_\theta U(\theta) = E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \delta_{\pi_\theta}\right]$$

해결책 : TD 에러

- 실전에서는 approximate TD 에러를 사용하면 됨

$$\delta_V = r + \gamma V_V(s') - V_V(s)$$

- 이렇게 되면 state-value function v(s)만 학습하면 된다

다양한 policy gradient

- The policy gradient has many equivalent forms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q^w(s, a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ A^w(s, a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ \delta\right] && \text{TD Actor-Critic}
\end{aligned}
$$

5. Vanilla Actor-Critic 실습

Main

```
def main():
    env = gym.make('CartPole-v1')
    model = ActorCritic()
    n_rollout = ?
    print_interval = 20
    score = 0.0

    for n_epi in range(10000):
        done = False
        s = env.reset()
        while not done:
            for t in range(n_rollout):
                prob = model.pi(torch.from_numpy(s).float())
                m = Categorical(prob)
                a = m.sample().item()
                s_prime, r, done, info = env.step(a)
                model.put_data(                ?                )

                s = s_prime
                score += r

                if done:
                    break

            model.train_net()
```

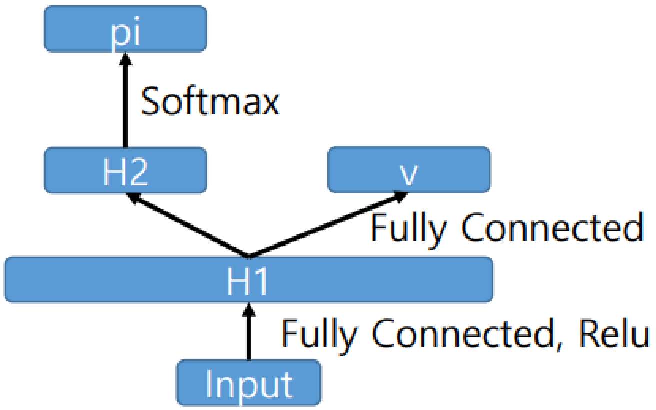
ActorCritic Class - Initialize

```python
class ActorCritic(nn.Module):
    def __init__(self):
        super(ActorCritic, self).__init__()
        self.data = []

        self.fc1 = nn.Linear(4,256)
        self.fc_pi = nn.Linear(256,2)
        self.fc_v = nn.Linear(256,1)
        self.optimizer = optim.Adam(self.parameters(), lr=learning_rate)

    def pi(self, x, softmax_dim = 0):
        x = F.relu(self.fc1(x))
        x = self.fc_pi(x)
        prob = F.softmax(x, dim=softmax_dim)
        return prob

    def v(self, x):
        x = F.relu(self.fc1(x))
        v =           Softmax?
        return v
```



ActorCritic Class – Make Batch

```
def make_batch(self):
    s_lst, a_lst, r_lst, s_prime_lst, done_lst = [], [], [], [], []
    for transition in self.data:
        s,a,r,s_prime,done = transition
        s_lst.append(s)
        a_lst.append([a])
        r_lst.append([r/100.0])
        s_prime_lst.append(s_prime)
        done_mask = 0.0 if done else 1.0
        done_lst.append([done_mask])

    s_batch, a_batch, r_batch, s_prime_batch, done_batch = torch.tensor(s_lst, dtype=torch.float), torc
                                                           torch.tensor(r_lst, dtype=torch.float), torc
                                                           torch.tensor(done_lst, dtype=torch.float)
    self.data = []
    return s_batch, a_batch, r_batch, s_prime_batch, done_batch
```

ActorCritic Class - Train

```
def train_net(self):
    s, a, r, s_prime, done = self.make_batch()
    td_target = r +                ?
    delta =                ?

    pi = self.pi(s, softmax_dim=1)
    pi_a = pi.gather(1,a)
    loss = -torch.log(pi_a) * delta.detach() +                ?

    self.optimizer.zero_grad()
    loss.mean().backward()
    self.optimizer.step()
```

$\delta_V = r + \gamma V_V(s') - V_V(s)$

$log\, \pi_\theta(a|s)\delta_{\pi_\theta}$

$value\ loss$

학습 결과

```
# of episode :20, avg score : 22.6
# of episode :40, avg score : 26.2
# of episode :60, avg score : 29.2
# of episode :80, avg score : 32.6
# of episode :100, avg score : 42.1
# of episode :120, avg score : 63.0
# of episode :140, avg score : 59.0
# of episode :160, avg score : 34.8
# of episode :180, avg score : 98.5
# of episode :200, avg score : 90.7
# of episode :220, avg score : 91.0
# of episode :240, avg score : 106.9
# of episode :260, avg score : 159.8
# of episode :280, avg score : 288.4
# of episode :300, avg score : 215.4
# of episode :320, avg score : 160.6
# of episode :340, avg score : 220.2
# of episode :360, avg score : 283.7
# of episode :380, avg score : 237.4
# of episode :400, avg score : 408.9
# of episode :420, avg score : 371.2
```

기타 코드

https://github.com/seungeunrho/minimalRL

감사합니다

seung_eun07@naver.com