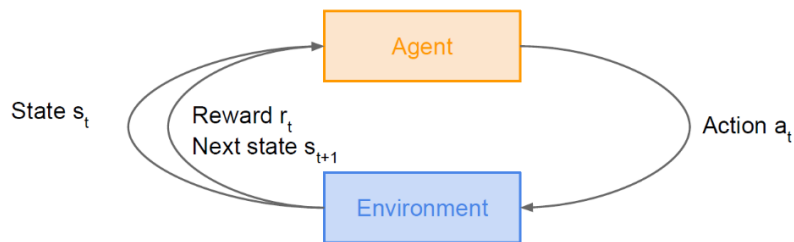


Lecture 14



-> 강화학습. 환경에서 agent로 보상을 해주는 형태. 그리고 다음 state부여.

강화학습 가능 풀이:

카트 위에서 균형 잡기.

로봇 앞으로 가게 하기.

게임 높은 점수로 끝내기

바둑게임.

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

\mathcal{S} : set of possible states

\mathcal{A} : set of possible actions

\mathcal{R} : distribution of reward given (state, action) pair

\mathbb{P} : transition probability i.e. distribution over next state given (state, action) pair

γ : discount factor

-> 'Markov Decision Process'(MDP) : 강화학습법을 수식화 한 것.

S: 가능한 상태의 집합.

A: 가능한 액션 집합

R: 보상에 대한 분포

P: 다음 상태에 대한 분포(전이 확률)

r: 보상 받는 시간의 중요성.

#강화 학습은 누적 보상을 최대화하는 것.

Formally: $\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi \right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$

-> MDP를 수식화 한 것. 미래 보상들의 합에 기댓값을 최대화하는 식.

$$V^{\pi}(s) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi \right]$$


-> Value function: 기댓값 수식화.

$$Q^{\pi}(s, a) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$

-> Q-Value function: 어떤 행동을 해야 가장 좋은지 알려주는 함수.

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$


 function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning!**

-> Deep Q-Learning. NN써서 $Q(s, a)$ 에 근사하게 하는 것.(ex. Atari game)

상태: 게임의 픽셀 전부.

행동: 게임의 방향키.

보상: 게임 점수

목표: 가장 높은 점수.

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

Policy Gradients: 함수 복잡한 Q-learning에서의 문제 보완.

상태를 학습하는 것이 아닌, policy 자체를 학습.

Intuition

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$

Interpretation:

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

-> Policy Gradients의 최종 식.

#구체적인 값 몰라도 gradient를 구해 최적의 정책 찾기 가능.

하지만 이 방식은 분산이 높다는 문제 발생.

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

-> 이 분산값 줄이는 첫번째 방식.

현재 스텝에서 종료 시점까지 얻을 수 있는 보상의 합 고려.

Second idea: Use discount factor γ to ignore delayed effects

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

-> 분산값 줄이는 두번째 방식.

지연된 보상에 '할인률'을 적용한다고 하는데, 미래에 받을 보상과 지금 바로 받을 보상을 구분하는 역할.

Variance reduction: Baseline

Problem: The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

What is important then? Whether a reward is better or worse than what you expect to get

Idea: Introduce a baseline function dependent on the state.
Concretely, estimator is now:

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

-> 분산값 줄이는 세번째 방식.

우리가 얻은 보상이 앞으로 얻을 것이라고 예상했던 것 보다 좋은건지 나쁜건지 판단하는게 중요한데,

Baseline function은 우리가 얼마만큼의 보상을 원하는지 설명해주는 함수.

Actor-Critic Algorithm

Problem: we don't know Q and V. Can we learn them?

Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay
- **Remark:** we can define by the **advantage function** how much an action was better than expected

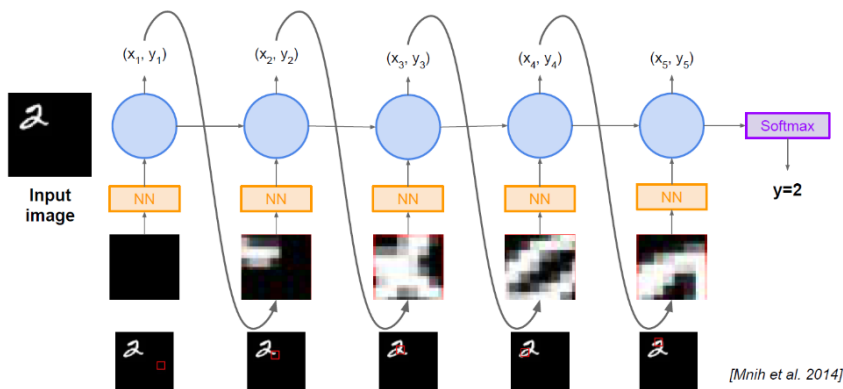
$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

-> Actor-Critic Algorithm: Policy gradient랑 Q-learning 조합한 알고리즘.

Actor: Policy로 어떤 상태 결정할지 정해주는 것.

Critic: Q-function으로 좋은지 나쁜지 판별하고, 어떤 식으로 조절해 나가야 하는지 결정.

REINFORCE in action: Recurrent Attention Model (RAM)



-> RNN을 강화학습에 적용.