

# 11월 3일 수업 강의 정리

## 1. 중간고사 이전 강의내용 Review

1) MC, TD 사용 이유

- Value function 추정
- Optimal action 찾기(by policy iteration): greedy 또는  $\epsilon$ -greedy 사용

2) MDP: S(State), T(Transition Probability), A(Action), R(Reward) 정보 알 수 있음

3) Model을 알 수 있는 경우와 알 수 없는 경우(Model-free) 비교

① Model given

- MDP로 모델링 후 Bellman equation으로 Value function 구함
- Bellman equation은 Bellman expectation equation과 Bellman optimality equation 두 종류
- Prediction: Value function evaluation ( $V^\pi(s), Q^\pi(s,a)$ ), Control: 최적 정책 찾기 by greedy policy

② Model free

- $P_{ss'}^a$ , unknown  $\rightarrow$  시뮬레이터로 대체  $\rightarrow$  return의 평균을 Value function으로 사용
- MC(Monte Carlo), TD(Temporal Difference) 사용
- .MC: t 시점의 return값(t시점 이후 모든 reward들의 discounted 합)을 target으로 사용
- .TD: 다음 step의 state의 Value function( $V(s_{t+1})$ : biased, 초기값 좋지 않음)을 TD Target에 사용
- MC 수렴 조건: Greedy in the Limit with Infinite Exploration(GLIE)

Definition

**Greedy in the Limit with Infinite Exploration (GLIE)**

- All state-action pairs are explored infinitely many times,  
 $(s, a)$   
 $\lim_{k \rightarrow \infty} N_k(s, a) = \infty$  모든 state, action은 충분히 많은 방문횟수를
- The policy converges on a greedy policy, policy가 파지락하는 greedy policy에 수렴함  
 $\lim_{k \rightarrow \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname{argmax}_{a' \in A} Q_k(s, a'))$

- For example,  $\epsilon$ -greedy is GLIE if  $\epsilon$  reduces to zero at  $\epsilon_k = \frac{1}{k}$

- Sarsa(TD계열) 수렴 조건

Theorem

**Sarsa converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$ , under the following conditions:**

- GLIE sequence of policies  $\pi_t(a|s)$
- Robbins-Monro sequence of step-sizes  $\alpha_t$

$\sum_{t=1}^{\infty} \alpha_t = \infty$   $\alpha_t > 0$  : 충분히 많이 탐색해야 함 } 한 term은  $\alpha_t = \frac{1}{k}$  정도면  $\sum \alpha_t > 0$   
 $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$   $\alpha_t \rightarrow 0$  : 너무 많이 돌아다니면 안됨 }  $\sum \frac{1}{k^2} < \infty$   
 $\Rightarrow$  step size는 처음에 커서 나중에 계속 작아져야 함

- Prediction: MC, TD 사용, Control:  $\epsilon$ -greedy 사용
- 4) SARSA: TD +  $\epsilon$ -greedy 사용
  - S, A, R, S', A', R', S'', A'', R'',...
  - MC의 경우 Target으로 Return  $G_t = R + \gamma V_t$  ... 사용(  $\gamma = 1$  )
  - SARSA는 다음과 같이 Sampling: (S, A, R, S', A'), (S', A', R', S'', A'')
  - on-policy: behavior policy와 target policy가 동일하고 모두  $\epsilon$ -greedy 사용
- 5) Q-Learning
  - off-policy: behavior policy와 target policy가 다름
  - behavior policy는  $\epsilon$ -greedy 사용, target policy는 greedy 사용
- 6) TD( $\lambda$ ): n-step TD의 가중평균
- 7) Value Function Approximation: state 또는 action space의 size가 커서 계산 로드와 저장공간 부족 문제를 해결하고자 approximation
  - 단점: convergence 문제 발생

## 2. Value Function Approximation

### 1) Large Scale Reinforcement Learning

#### Large Scale Reinforcement Learning

---

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon:  $10^{20}$  states
  - **Computer Go:  $10^{170}$  states**
  - Helicopter: continuous state space

2

- Computer Go의 경우  $10^{170}$  states가 있음
  - . state 당 1byte 가정하면,  $10^{170}$ byte가 필요함
  - .  $10^{170}$ byte를 저장하는 것은 현실적으로 불가능
    - : RAM 32GB  $\approx 32 \times 10^9$ , Hard Disk 1TB  $\approx 1 \times 10^{12}$  가정
    - :  $10^{170-12} \approx 10^{158}$ 개의 1TB Hard Disk 필요
    - $10^{170}$ 개의 Q value 값을 table 형태로 저장하는 것은 불가능
- Value Function Approximation을 하면 학습모델의 Weight만 저장
  - . 선형회귀 모델의 경우 모든 예측값은 필요 없고 절편과 기울기만 있으면 예측값 계산 가능
  - . 장점: 저장공간이 크게 줄어듦
  - . 단점: 정확도가 떨어지고, 수렴 잘 안됨 → 그래도 못하는 것보다 나음

## 2) Value Function Approximation

### Value Function Approximation

- So far we have represented value function by a *lookup table*
  - Every state  $s$  has an entry  $V(s)$
  - Or every state-action pair  $s, a$  has an entry  $Q(s, a)$
- **Problem** with large MDPs:
  - There are **too many states and/or actions to store in memory**
  - It is **too slow to learn the value of each state individually**
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_{\pi}(s)$$

$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_{\pi}(s, a)$$

- Generalise from seen states to unseen states
- Update parameter  $\mathbf{w}$  using MC or TD learning

4

- State와 Action space의 size가 큰 Large MDP의 문제점과 Approximation 통한 해결방안

① 모든 state과 action을 메모리와 Hard disk에 저장하는 것은 불가능

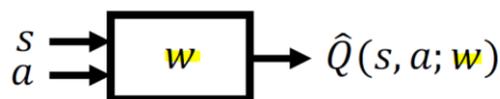
→ 해결 방안: 모든 state과 action의 table을 갖지 않고, approximation 모델의 parameter를 저장

② 모든 state과 action pair를 방문하는 것은 불가능

→ 해결 방안: approximation 통해 모든 state과 action에 대한 정확한 value function을 계산하지 않고 근사값을 찾아서 computational load를 줄임

### Value Function Approximation

- Represent a (state-action/state) **value function** with a **parameterized function** instead of a table



5

- parameter  $w$ 가 바뀌면 다른 Function approximation model이 됨

- gradient descent 알고리즘에 의해  $\text{Loss} = E[(V(s) - \hat{V}(s, W))^2]$ 를 최소화하는  $w$ 값을 구함

## 3) How to approximate?

## How to approximate?

True Value Function

$$V^\pi(s)$$

Approximate VF



- Find  $w$  minimizing mean-squared error

$$J(w) = \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, w))^2]$$

- Gradient Descent Direction

$$\begin{aligned} \Delta w &= -\frac{1}{2} \alpha \nabla_w J(w) \\ &= \alpha \mathbb{E}_\pi [(v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w)] \end{aligned}$$

- Stochastic Gradient Descent samples the gradient

$$\Delta w = \alpha (v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w)$$

10

-  $\{(x_i, y_i), i = 1, 2, 3, \dots, n\}$  데이터가 주어졌을 때, 예측모델  $f(\cdot)$  활용한 예측값은  $\hat{y}_i = f(x_i)$ 로 표현하며, 다양한 종류의 예측모델  $f$ 를 사용할 수 있음

→ 예측모델의 최적 parameter  $w$ 는  $\min \sum (y_i - f(x_i))^2 \rightarrow \min \sum (\hat{V}(w) - V^\pi(w))^2$ 를 최소화하는 parameter 로써 예측값  $\hat{v}$ 가  $V^\pi$ 에 가까워지게 해야함

- Value Function Approximation에서는  $(x_i, y_i)$ 로 state와 그 state에서의 주어진 policy  $\pi$ 에 대한 value function으로 사용:  $(x_i, y_i) \rightarrow (s_i, V^\pi(s_i))$  또는  $(s_i, Q^\pi(s_i, a_i))$

- 그러나, 주어진 데이터는 sample일 뿐이기 때문에, 실제로는  $V^\pi(s_i)$ 과  $Q^\pi(s_i, a_i)$ 를 알 수 없음

:  $s, a, r, s', a', r'$

- MC에서는  $G_t$ 를  $V_\pi$ 로 놓고, TD에서는 TD Target을  $V_\pi$ 로 놓고 문제를 풀어감

- Gradient Descent Direction: gradient descent direction으로 가면  $J(w)$ 가 감소되는게 보장되기 때문에 사용하지만, 반드시 최적값을 도출하지는 않을 수 있음

- Stochastic Gradient Descent: 모든 state에 대해서 gradient descent 알고리즘을 적용하면 computational load가 매우 크지만, 한 개의 state에 대해 업데이트하면 computational load가 작아지게 되지만, state 마다의 variability가 커지는 단점이 있음

- Value Function Approximation에서는 stochastic gradient descent에 의해 approximation 모델의 parameter  $w$ 를 계속 업데이트함.  $V^\pi(s_i)$ 과  $Q^\pi(s_i, a_i)$ 를 업데이트 하지 않음

#### 4) Feature Vectors

## Feature Vectors

---

- Represent **state** by a **feature vector**

$$\mathbf{x}(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

11

- State를 잘 설명할 수 있는 Feature 추출 필요하며 State와 Feature는 같지 않음

### 5) Linear VFA

## Linear VFA

---

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^n x_j(S) \mathbf{w}_j$$

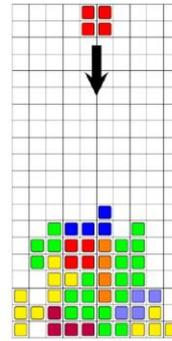
12

- State에서 추출한 Feature들의 선형 조합은 Linear VFA는 제일 간단한 형태

### 6) Linear VFA Example: Tetris

## Linear VFA Example: Tetris

- state: board configuration + shape of the falling piece  $\sim 2^{200}$  states!
- action: rotation and translation applied to the falling piece
- 22 features aka basis functions  $\phi_i$ 
  - Ten basis functions, 0, ..., 9, mapping the state to the height  $h[k]$  of each column.
  - Nine basis functions, 10, ..., 18, each mapping the state to the absolute difference between heights of successive columns:  $|h[k+1] - h[k]|$ ,  $k = 1, \dots, 9$ .
  - One basis function, 19, that maps state to the maximum column height:  $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.



$$\hat{V}_\theta(s) = \sum_{i=0}^{21} \theta_i \phi_i(s) = \theta^\top \phi(s)$$

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

13

-  $2^{200}$  states  $\approx 10^{60}$  Bytes  $\rightarrow 10^{60} - 12 = 10^{48}$ 개의 Hard Disk 필요  $\rightarrow$  저장 불가

-  $2^{200}$ 개의 states에서 22개의 Feature를 뽑고, 선형 결합하면 계수 22개의 값만 저장하면 됨: 약 88bytes

### 7) Incremental Prediction Algorithm

#### Incremental Prediction Algorithm

- Have assumed true value function  $v_\pi(s)$  given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a target for  $v_\pi(s)$ 
  - For MC, the target is the return  $G_t$

$$\Delta \mathbf{w} = \alpha (G_t - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

- For TD(0), the target is the TD target  $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

- For TD( $\lambda$ ), the target is the  $\lambda$ -return  $G_t^\lambda$

$$\Delta \mathbf{w} = \alpha (G_t^\lambda - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

16

- 강화학습에서는 true value function  $v_\pi(s)$ 가 주어지지 않고 오직 reward만 주어지기 때문에,  $v_\pi(s)$ 를 target으로 대체함

.MC:  $G_t$

.TD(0):  $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

.TD( $\lambda$ ):  $G_t^\lambda$

### 8) Monte-Carlo with VFA

## Monte-Carlo with VFA

- Return  $G_t$  is an unbiased, noisy sample of true value  $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \dots, \langle S_T, G_T \rangle$$

- For example, using *linear Monte-Carlo policy evaluation*

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w})) \mathbf{x}(S_t)\end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

17

- Return  $G_t$ 가 high variance지만 unbiased하기 때문에  $v_\pi(s)$  대신 사용하며, 다음과 같이 학습 데이터 구성:  $(S_1, G_1), (S_2, G_2), \dots, (S_T, G_T)$

- MC Evaluation은 non-linear VFA를 사용해도 local optimum에 빠짐

## 9) TD Learning with VFA

### TD Learning with VFA

- The TD-target  $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$  is a biased sample of true value  $v_\pi(S_t)$
- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w}) \rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w}) \rangle, \dots, \langle S_{T-1}, R_T \rangle$$

- For example, using *linear TD(0)*

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \\ &= \alpha \delta \mathbf{x}(S)\end{aligned}$$

- Linear TD(0) converges (close) to global optimum

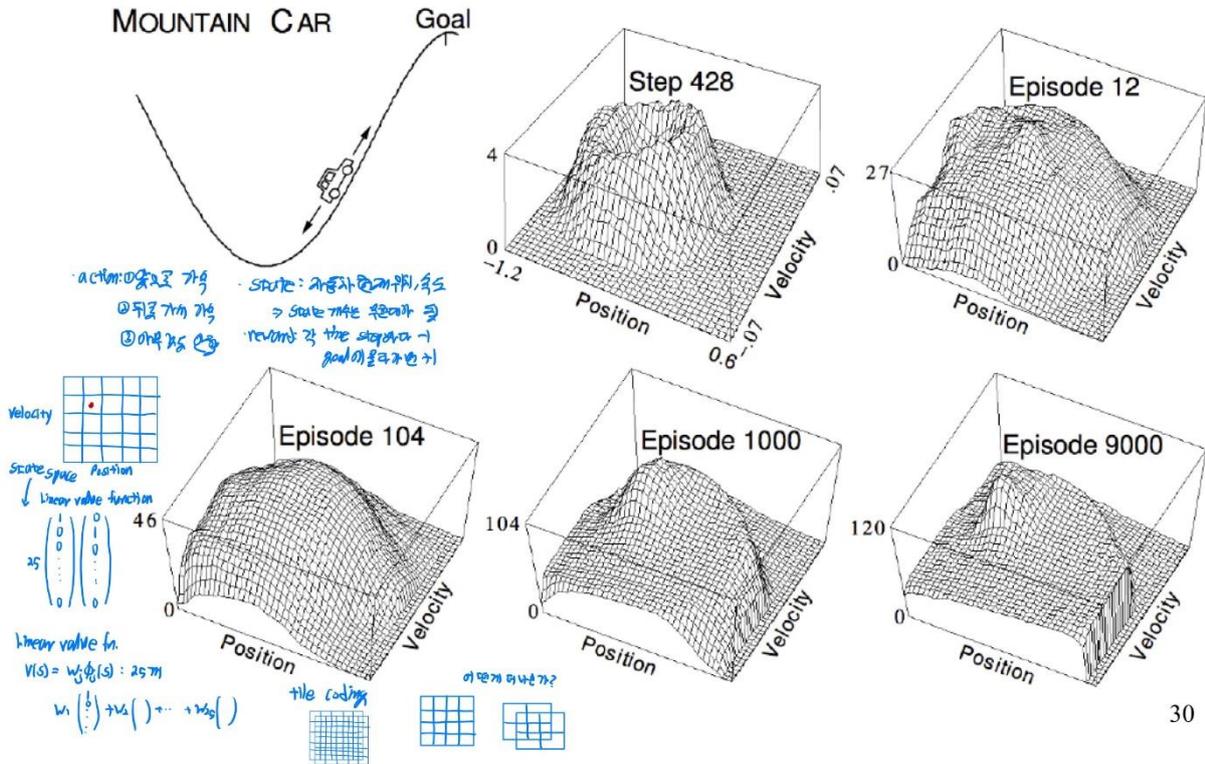
20

- TD target은 biased 하지만, training data의 response 값으로 true value function을 대체하여 사용

- TD(0)는 global optimum에 수렴

## 10) Linear SARSA with Coarse Coding in Mountain Car

# Linear SARSA with Coarse Coding in Mountain Car



- action: 앞으로 가속, 뒤로 이동 후 앞으로 가속, 아무것도 안함
- state: 자동차 현재 위치, 속도 → 위치와 속도는 연속치라서 state space는 무한대가 됨
- state space를 5 x 5 grid로 discretize 하면, 25 x 1 크기의 one-hot vector로 나타낼 수 있음
- Linear Value Function Approximation은  $\hat{v}(s) = \sum w_j \phi_j(s)$ 로 표현