6. 벨류 펑션 근사(Value Function Approximation)

김호철

Contents

1 소개

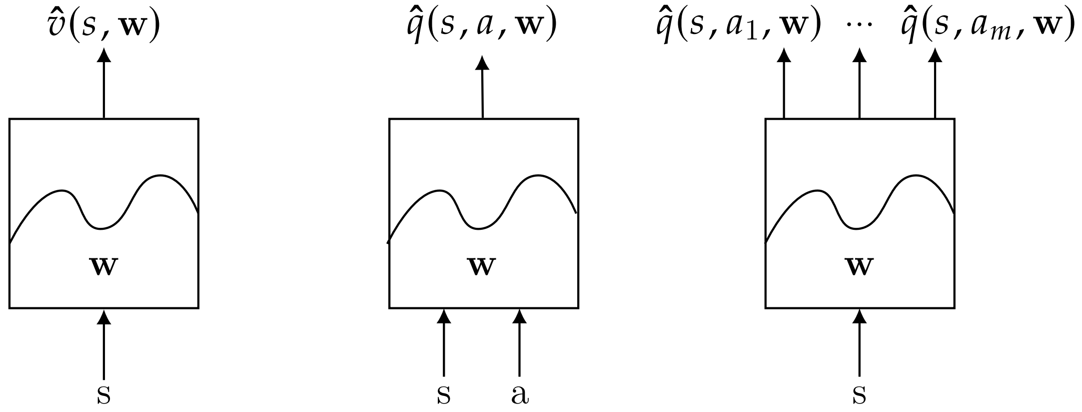
- **대규모 강화학습**
  - 강화 학습은 아주 큰 문제들을 해결하는 데 사용될 수 있다.
  - 백가몬게임(10^{20} 상태), 바둑(10^{170} 상태), 헬리곱터(연속적 상태 공간)
  - 프리딕션과 콘트롤의 모델-프리 방법들을 어떻게 스케일 업 할 수 있을까?
- **벨류 펑션 근사(Value Function Approximation)**
  - 지금까지는 모든 상태나 상태-액션 값을 테이블에 저장하는 방식의 강화 학습
  - 아주 큰 MDP에서는 상태나 액션을 메모리에 모두 저장하기에 너무 많고, 각 상태를 개별적으로 학습하기에 너무 느리다.
  - **솔루션** 대규모 MDP들은 함수 근사(function approximation)로 가치 함수(value function)를 추정한다.

$$\begin{aligned} \hat{v}(s, w) &\approx v_\pi(s) \\ \hat{q}(s, a, w) &\approx q_\pi(s, a) \end{aligned} \tag{1}$$

  - 보이는 상태로 안보이는 상태를 **일반화(Generalise)**

– MC나 TD 러닝으로 **파라메터 w**를 업데이트

- **벨류 펑션 근사의 타입들**



- 다양한 함수 근사 방법들이 있으나, **w**를 구하기 위해 미분 가능(differentiable)한 방법인,
- (1)특성(feature)들의 선형 결합(Linear combinations)과 (2)신경망(Neural network)을 사용

2 증분적 방법들(Incremental Methods)

2.1 기울기 하강법(Gradient Descent)

- 파라메터 벡터 **w**의 미분 가능한 함수를 $J(\mathbf{w})$라고 하자
- $J(\mathbf{w})$를 최소로 하는 입력 **w**를 찾는 문제
- $J(\mathbf{w})$의 기울기 정의(∇:기울기,그레디언트)

$$\nabla_w J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix} \tag{2}$$

- $J(\mathbf{w})$의 지역 최소값을 찾기 위해, 기울기의 음수(-) 방향으로 **w**를 조정 (Δ:델타)

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \tag{3}$$

- α는 스텝 크기 파라메터

2.2 확률적 기울기 하강법(Stochastic Gradient Descent)에 의한 VFA

- 근사 가치 함수 $\hat{v}(S,w)$ 와 실제 가치 함수(True Value Function) $v_\pi(s)$ 사이의 평균 제곱 오차를 최소화 하는 파라메터 벡터 **w**를 찾는 방법

$$J(\mathbf{w}) = \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S,\mathbf{w}))^2] \tag{4}$$

- 기울기 하강법은 지역 최소값을 찾는다.

$$\begin{aligned} \Delta w &= -\frac{1}{2}\alpha\nabla_w J(w) \\ &= -\frac{1}{2}\alpha\mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S,w))^2] \end{aligned} \tag{5}$$

- 여기에 다항 함수 T 거듭 제곱의 미분($(T^n)' = nT^{n-1}T'$)을 적용하면,

$$\begin{aligned} &= -\frac{1}{2}\alpha\mathbb{E}_\pi[2(v_\pi(S) - \hat{v}(S,w))\nabla_w(v_\pi(S) - \hat{v}(S,w))] \\ &= \alpha\mathbb{E}_\pi\ [v_\pi(S) - \hat{v}(S,w)]\nabla_w\hat{v}(S,w) \end{aligned} \tag{6}$$

- 확률적 경사 하강법은 1개의 기울기를 샘플링(Stochastic)한다.

$$\Delta w = \alpha(v_\pi(S) - \hat{v}(S,w))\nabla_w\hat{v}(S,w) \tag{7}$$

- 기대된(샘플링된) 업데이트는 전체 기울기 업데이트와 동일하다.

2.3 선형 함수 근사(Linear Function Approximation)

- **특성 벡터(Feature Vectors)**
  - 상태(state)를 특성(feature) 벡터로 나타내면,

$$x(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix} \tag{8}$$

  - 자동 청소 로봇의 경우 전방 180도에 대한 거리를 알려주는 센서가 있음
  - 특성 벡터는 1도에서 180도까지의 거리들
  - $x_1(s)$에는 1도 방향 목적지까지의 거리, $x_2(s)$에는 2도 방향 목적지까지의 거리, ...

- **선형 VFA(Linear Value Function Approximation)**
  - 특성들의 선형 조합으로 가치함수를 표현하면,

$$\hat{v}(S,w) = x(S)^T w = \sum_{j=1}^{n} x_j(S)w_j \tag{9}$$

  - 목적함수는 파라메터 **w**항의 제곱이고,

$$J(w) = \mathbb{E}_\pi\left[\left(v_\pi(S) - x(S)^T w\right)^2\right] \tag{10}$$

- 확률적 경사 하강법은 글로벌 최적에 수렴한다.
- 업데이트 규칙은 간단한다.

$$\begin{aligned} \nabla_w \hat{v}(S, w) &= x(S) \\ \Delta w &= \alpha(v_\pi(S) - \hat{v}(S, w))x(S) \end{aligned} \tag{11}$$

- 업데이트 = 스텝 싸이즈 × 예측 에러 × 특성 값

- **테이블 룩업 특성들**
  - 테이블 룩업은 선형 VFA의 특별한 경우이다.
  - 테이블 룩업 특성의 사용

$$x^{table}(S) = \begin{pmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{pmatrix} \tag{12}$$

  - 파라메터 벡터 **w**는 각 개별 상태의 값(Value)으로 제공된다.

$$\hat{v}(S, w) = \begin{pmatrix} 1(S = s_1) \\ \vdots \\ 1(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \tag{13}$$

2.4 증분 예측 알고리즘(Incremental Prediction Algorithms)

- **증분 예측 알고리즘**
  - 지금까지 지도자(supervisor)가 제공하는 True Value Function $v_\pi(s)$를 가정 했었다
  - 하지만 강화학습에는 지도자가 없고 오직 보상(reward) 만 있다.
  - 실제에서는 True Value Function $v_\pi(s)$를 해당 목적함수로 대체하면 된다.
  - MC에서 목적함수는 리턴 G_t이다.

$$\Delta w = \alpha(\color{red}{G_t} - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w) \tag{14}$$

  - TD(0)에서 목적함수는 TD 타겟 $R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$이다.

$$\Delta w = \alpha(\color{red}{R_{t+1} + \gamma\hat{v}(S_{t+1}, w)} - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w) \tag{15}$$

  - TD(λ)에서 목적함수는 λ-리턴 G_t^λ이다.

$$\Delta w = \alpha(\color{red}{G_t^\lambda} - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w) \tag{16}$$

- **VFA를 사용하는 몬테 카를로**
  - 리턴 G_t는 True Value $v_\pi(S_t)$에 편향되지 않았지만 노이즈가 있는 샘플이다.
  - 그러므로 "훈련 데이터"를 지도 학습에 적용할 수 있다

$$<S_1, G_1>, <S_2, G_2>, ..., <S_T, G_T> \tag{17}$$

- 예를 들어, 선형 몬테 카를로 정책 평가를 사용하면,

$$\begin{aligned}\Delta w &= \alpha(G_t - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w) \\ &= \alpha(G_t - \hat{v}(S_t, w))x(S_t) \\ &= \alpha(G_t - x(S_t)^T w)x(S_t)\end{aligned} \tag{18}$$

- MC 평가는 로컬 최적에 수렴한다.
- 심지어 비선형 VFA에도 수렴한다.

- **VFA를 사용하는 TD 러닝**
  - TD 타겟 $R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$는 True Value $v_\pi(S_t)$에 편향된 샘플이다.
  - 마찬가지로 "훈련 데이터"를 지도 학습에 적용할 수 있다

$$<S_1, R_2 + \gamma \hat{v}(S_2, w)>, <S_2, R_3 + \gamma \hat{v}(S_3, w)>, ..., <S_{T-1}, R_T> \tag{19}$$

  - 선형 TD(0)를 사용하면,

$$\begin{aligned}\Delta w &= \alpha(R_{t+1} + \gamma \hat{v}(S', w) - \hat{v}(S, w))\nabla_w \hat{v}(S, w) \\ &= \alpha \delta x(S)\end{aligned} \tag{20}$$

  - 선형 TD (0)는 글로벌 최적에 가깝게 수렴한다.

- **VFA를 사용하는 TD(λ)**
  - λ 리턴 G_t^λ는 True Value $v_\pi(S_t)$에 편향된 샘플이다.
  - 이것도 "훈련 데이터"를 지도 학습에 적용할 수 있다

$$<S_1, G_1^\lambda>, <S_2, G_2^\lambda>, ..., <S_{T-1}, G_{T-1}^\lambda> \tag{21}$$

  - 전방 뷰(Forward view) 선형 TD(λ),

$$\begin{aligned}\Delta w &= \alpha\left(G_t^\lambda - \hat{v}(S, w)\right)\nabla_w \hat{v}(S, w) \\ &= \alpha\left(G_t^\lambda - \hat{v}(S, w)\right)x(S_t)\end{aligned} \tag{22}$$

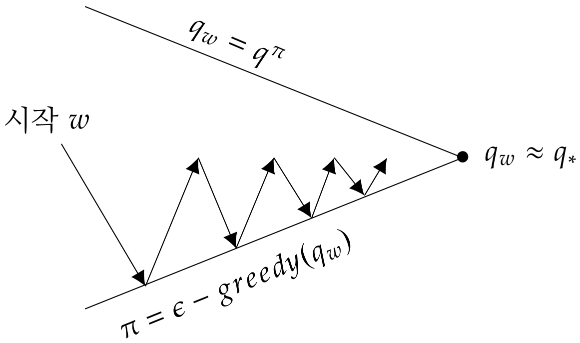
  - 후방 뷰(Backward view) 선형 TD(λ),

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w) \\ E_t &= \gamma \lambda E_{t-1} + x(S_t) \\ \Delta w &= \alpha \delta_t \ E_t\end{aligned} \tag{23}$$

  - 전방 보기 및 후방 보기 선형 TD(λ)는 동일하다.

2.5 증분 콘트롤 알고리즘(Incremental Control Algorithms)

- **VFA에서 콘트롤**



  - **정책 평가** : 정책 평가를 **근사(Approximate)** $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$
  - **정책 개선** : ϵ 탐욕 정책 개선

- **액션-가치 함수의 근사**
  - 액션-가치 함수의 근사

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A) \tag{24}$$

  - 근사 액션-가치 함수 $\hat{q}(S, A, \mathbf{w})$ 와 True 액션-가치 함수 $q_\pi(S, A)$ 사이의 평균 제곱 에러를 최소화

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2\right] \tag{25}$$

  - 로컬 최소값을 찾기위해 확률적 기울기 하강법(stochastic gradient descent)을 사용

$$\begin{aligned} -\frac{1}{2}\alpha\nabla_{\mathbf{w}}J(\mathbf{w}) &= (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w}) \\ \Delta\mathbf{w} &= \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w}) \end{aligned} \tag{26}$$

- **선형 액션-가치 함수의 근사**
  - 특성(feature) 벡터로 상태(state)와 액션(action)을 표현하면,

$$x(S, A) = \begin{pmatrix} x_1(S, A) \\ \vdots \\ x_n(S, A) \end{pmatrix} \tag{27}$$

  - 특성들의 선형 조합으로 액션-가치 함수로 나타내면,

$$\hat{q}(S, A, \mathbf{w}) = x(S, A)^T\mathbf{w} = \sum_{j=1}^{n} x_j(S, A)\mathbf{w}_j \tag{28}$$

  - 확률적 기울기 하강법(Stochastic gradient descent)으로 업데이트

$$\begin{aligned} \nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w}) &= x(S, A) \\ \Delta\mathbf{w} &= \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))x(S, A) \end{aligned} \tag{29}$$

- **증분적 콘트롤 알고리즘**
  - 프리딕션(Prediction)과 마찬가지로 $q_\pi(S, A)$를 타겟으로 대체하면 된다.
  - MC에서 목적함수는 리턴 G_t이다.

$$\Delta \mathbf{w} = \alpha({\color{red}G_t} - \hat{v}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, A_t, \mathbf{w}) \tag{30}$$

  - TD(0)에서 목적함수는 TD 타겟 $R_{t+1} + \gamma\hat{v}(S_{t+1}, A_{t+1}, \mathbf{w})$이다.

$$\Delta \mathbf{w} = \alpha({\color{red}R_{t+1} + \gamma\hat{v}(S_{t+1}, A_{t+1}, w)} - \hat{v}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, A_t, \mathbf{w}) \tag{31}$$

  - 전방 보기 TD(λ)에서 목적함수는 액션-가치 λ-리턴이다.

$$\Delta \mathbf{w} = \alpha({\color{red}q_t^\lambda} - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w}) \tag{32}$$

  - 후방 보기 TD(λ)에서 업데이트는 동일하다.

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma\lambda E_{t-1} + x(S_t) \\ \Delta \mathbf{w} &= \alpha\delta_t \ E_t \end{aligned} \tag{33}$$

2.6 수렴

- **증분적 콘트롤 알고리즘**

	알고리즘	테이블 룩업	선형	비선형
온폴리시	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
오프폴리시	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✓

- **기울기 TD 학습**
  - TD는 어떤 목적함수의 기울기도 따르지 않는다.
  - 이것이 오프폴리시나 비선형 함수 근사에서 TD가 발산할 수 있는 이유이다.
  - **기울기 TD**는 예상된 벨만 에러의 True 기울기를 따른다.

	알고리즘	테이블 룩업	선형	비선형
온폴리시	MC	✓	✓	✓
	TD	✓	✓	✗
	기울기 TD	✓	✓	✓
오프폴리시	MC	✓	✓	✓
	TD	✓	✗	✗
	기울기 TD	✓	✓	✓

- **콘트롤 알고리즘들의 수렴**

알고리즘	테이블 룩업	선형	비선형
몬테카를로 콘트롤	✓	(✓)	✗
살사	✓	(✓)	✗
Q-러닝	✓	✗	✗
기울기 Q-러닝	✓	✓	✗

- (✓) 최적에 가까운 가치 함수

3 배치 방법들

3.1 배치 강화 학습

- 경사 하강법은 단순하고 매력적이지만 샘플링을 하기 어렵다.
- 에이전트의 경험(”훈련 데이터”)을 고려하여, 배치 메쏘드는 가장 적합한 가치 함수를 찾기 위함이다.

3.2 최소 제곱 예측

- 주어진 가치 함수 근사 $v^{(}s, \mathbf{w}) \approx v_\pi(s)$
- 그리고 ⟨state, value⟩ 쌍으로 구성된 경험 $\mathcal{D}$

$$\mathcal{D} = \{<s_1, v_1^\pi>, <s_2, v_2^\pi>, ..., <s_T, v_T^\pi>\} \tag{34}$$

- 가장 적합한 가치 함수 $\hat{v}(s, \mathbf{w})$를 제공하는 파라메터 $\mathbf{w}$는 무엇일까?

- 최소 제곱 알고리즘은 $\hat{v}(s_t, \mathbf{w})$ 와 target value들 v_t^π 간의 제곱 에러들의 합을 최소화하는 파라메터 벡터 $\mathbf{w}$를 찾는다.

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T}(v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_{\mathcal{D}}[(v^\pi - \hat{v}(s, \mathbf{w}))^2] \end{aligned} \tag{35}$$

- **경험 리플라이(Experience Replay)로 통계적 기울기 하강(Stochastic Gradient Descent)**
  - ⟨상태(state), 값(value)⟩ 쌍으로 경험(experience)이 주어지면,

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\} \tag{36}$$

  - 다음을 반복하면,
    1. 경험으로부터 상태와 값을 샘플링

$$\langle s, v^\pi \rangle \sim \mathcal{D} \tag{37}$$

    2. 통계적 기울기 하강 업데이트 적용

$$\Delta w = \alpha(v^\pi - \hat{v}(s, w))\nabla_w \hat{v}(s, w) \tag{38}$$

  - 최소 제곱 해에 수렴한다.

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmin}}\, LS(\mathbf{w}) \tag{39}$$

- **고정된 Q-타겟(Fixed Q-Targets)**
  - Q-러닝 방식에서 계속 타겟이 바뀌어서 문제였지만, 이것을 고정시켜 대처
  - 타겟 업데이트시 사용할 가중치 w^-와, 실제 업데이트 할 가중치 $\mathbf{w}$를 분리해서 사용하는 것
    1. 데이터셋 $<s, v^\pi>$ 나 $<s, a, r, s'> \sim \mathcal{D}$로 부터 경험 튜플을 샘플링
    2. 샘플링된 s($r + \gamma \max_{a'} Q(s', a'; w_i^-)$)로 targte value를 업데이트
    3. 통계적 기울기 하강 업데이트 적용

$$\Delta w = \alpha(r + \gamma max_{a'} Q(s', a'; w_i^-) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s, a; w) \tag{40}$$

  - 타겟에

$$w^-$$

    를, 나머지 부분에는 원래 가중치 w를 적용한다. n번째 마다 (hyper parameter) $w-$값을 w값으로 셋팅
  - Q-러닝에서 계속 w가 바뀌다가 결국 w가 무한(infinity)으로 가는 상황을 방지하여 안정성이 꽤 많이 향상

- **딥 Q-네트워크(DQN)에서 경험 리플라이(Experience Replay)**
  - DQN은 경험 리플라이와 고정된 Q-타겟을 사용
    1. ϵ-탐욕 정책에 따라 액션 a_t를 가져온다.

2. 리플라이 메모리 $\mathcal{D}$에 트랜지션 $(s_t, a_t, r_{t+1}, s_{t+1})$을 저장
3. D에서 트랜지션(s, a, r, s')들을 무작위 미니배치(n개)로 샘플링한다.
4. 이전의 고정된 파라메터 w^-로 Q-러닝 타겟을 계산한다.
5. Q-네트워크와 Q-러닝 타겟들 간의 최소제곱에러(MSE) 최적화

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i}\left[\left(r + \gamma \max_{a'} \ Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right] \tag{41}$$

6. 확률적 경사하강법 사용

- **아타리 게임 점수 비교**

게임	선형	딥고정된 네트워크	고정된 Q-타겟	경험 리플라이	고정된 Q-타겟 경험 리플라이
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Sequest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089