## Chapter 6. Value Function Approximation

## 1. Introduction

**#. Scaled–up Continuous State Space Model-Free인 상황에서, Value Function을 어떻게 표현하고 이를 통해 Prediction, Control 문제를 해결하는 방법을 배운다.**

### 1-1. Tabular Solution Method의 한계

**#. 현실세계에서는 State의 개수가 엄청 많으며, Continuous State 또한 존재하기 때문에 이를 Tabular Method로 풀기는 불가능하다. 지난 단원까지는 Tabular Method를 통해 State Value Function과 Action Value Function을 표현하였는데, 이렇게 Scaled-Up과 Continuous State 문제에서는 Tabular Method를 대체하는 Function Approximation방법을 사용해야 한다.**

### 1-2. Value Function Approximation의 의미

**#. Function Approximation은 우리가 구하고자 하는 State Value Function($v_\pi(s)$), Action Value Function ($q_\pi(s,a)$)를 어떤 Parameter $w$에 의해 근사되는 함수를 의미하며 아래와 같이 표기한다. 결국 이 근사함수로 실제의 Value Functions를 학습하려고 하는 것이다. 즉 Optimal $w$[1]을 찾는 것이 목적이다.**

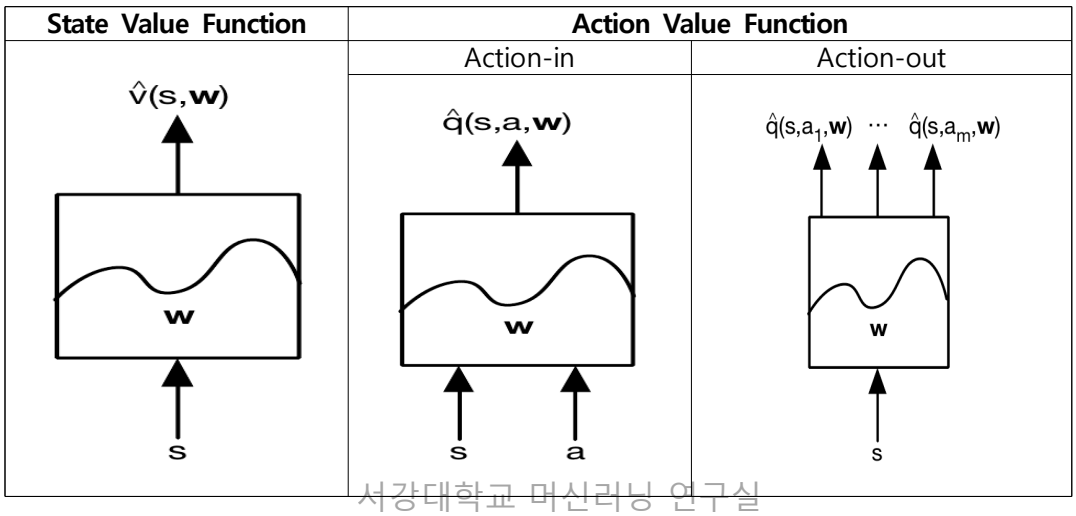
| State Value Function | Action Value Function |
|---|---|
| $\hat{v}(s,w) \approx v_\pi(s)$ | $\hat{q}(s,a,w) \approx q_\pi(s,a)$ |

a. Optimal $w$을 찾게 되면, 기계학습과 같이 경험하지 않은 State에 대해서도 적절한 함수 값을 Return해줄 수 있다. 즉 Generalization할 수 있다는 것이다.

b. Optimal $w$을 찾기 위해 MC 또는 TD Learning을 사용한다.

### 1-3. Value Function Approximation의 종류

**#. Value Function Approximation의 종류는 아래와 같으며, 구체적으로는 미분 가능한 'Linear Combination of Features($\hat{v}(s) = w^T x_s$, $\hat{q}(s,a) = w^T x_{s,a}$)'와 'Neural Network($\hat{v}(s) = f_n(x_s)$, $\hat{q}(s,a) = f_n(x_{s,a})$)'를 사용한다. 또한 Non-Stationary, Non-i.i.d를 가정한다.**






1) 표기상 Input으로 생각할 수 있지만 Internal-Parameter이다.

## 2. Incremental Methods

### 2-1. Premise_1 : Value Function Approximation by Stochastic Gradient Descent

| | |
|---|---|
| Batch or Deterministic Gradient Methods | Full Examples |
| Stochastic or Online Methods | Single Example |
| Mini-batch or Mini-batch Stochastic Methods | Less than Full Examples |
| ∴ Stochastic Methods라 함은 $1 < \#(Examples) < Full$ | |

2-1-1. Gradient Descent Algorithm

**#. 함수의 기울기를 구하여 기울기가 낮은 쪽으로 Input 값을 계속 이동시켜서 극값에 이를 때까지 반복시키는 알고리즘이며, 이를 통해 극값을 갖는 Input[2]을 찾는 것이 목적이다.**
**최적화할 함수(목적함수) $f(x)$에 대하여, 먼저 시작점을 정하고 다음으로 이동할 $X_{i+1}$은 아래와 같이 계산한다.**

$$X_{i+1} = X_i - \frac{1}{2}\alpha \nabla_x f(X_i)$$

a. 이 알고리즘의 수렴 여부는 목적함수$f$의 성질과 Step-Size $\alpha$의 선택에 따라 달라진다. 또한 이 알고리즘은 Local Minimum으로 수렴한다. 따라서 이 값이 Global Minimum이라는 것을 보장하지 않으며 시작점의 선택에 따라서 달라진다. 이에 따라 다양한 시작점에 대해 이 알고리즘을 적용하여 그 중 가장 좋은 결과를 선택할 수도 있다.

2-1-2. Value Function Approximation by Stochastic Gradient Descent

**#. $\hat{v}$함수가 $v$[3]을 모방하는 것이 목적이므로 아래와 같이 목적함수 $J(w)$[4]을 설정해준다. 하지만 Stochastic Gradient Descent는 Sampling을 하면서 방문한 State를 그대로 Input으로 넣어 아래의 Expectation[5]을 없애는 방법이다.**

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

$$\Rightarrow \triangle w = -\frac{1}{2}\alpha \nabla_w J(w) = \alpha E_\pi[(v_\pi(S) - \hat{v}(S,w))\nabla_w \hat{v}(S,w)]$$

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

a. 위의 식에서 $J(w)$은 Cost-Function, $(v_\pi(S) - \hat{v}(S,w))^2$은 Loss-Function이라 하고, $E_\pi(= \frac{1}{m}\sum_{i=1}^{m})$은 $For\,all\, s \in S$을 의미한다. 즉 목적함수의 Input은 $w$이므로 모든 State $s$에 대한 평균값을 내야한다. 그

2) $f(x,y) = 2x - 4y$에서 Input은 $\langle x, y \rangle$와 같이 Vector로 표현된다.
3) Given by Oracle
4) 목적함수에서는 $w$가 Input 이지만 $\hat{v}$에서는 Parameter이다.
5) policy $\pi$을 따랐을 때의 $S$에 대한 차이를 봐야하기 때문에 존재한다.

리고 Cost-Function 뒤에 Over-fitting을 방지하기 위해 Regularization-Term을 더해줄 수 있다.

b. Stochastic Gradient Descent의 방법은 Expectation이 붙은 Full Gradient Update와 동일하게 적용된다. 예를 들어 해당 Policy $\pi$가 1번을 자주 방문하고 7번을 덜 방문하면 자연스럽게 이것이 Expectation의 효과와 동일해지기 때문이다.

## 2-2. Premise_2 : Linear Function Approximation

### 2-2-1. Feature Vectors

**#. 함수와 비슷하게 현재 State $S$을 input으로 넣으면 이에 대한 $n$개의 원소로 이루어진 Feature vector를 아래와 같이 정의할 수 있다.**

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

a. 예를 들어 주식시장의 현재 state를 넣으면 이동평균, 전날 고가, 저가, 상한가 등 여러 개의 feature들이 나오게 된다.

b. Neural Network을 사용하여 Function Approximation을 할 때는 Feature Vector를 따로 사용할 필요 없이 그대로 넣어줄 수도 있다. 하지만 Linear Function을 사용할 경우 Continuous State를 Discrete하게 변환해주기 위해 Polynomials, Fourier Basis, Coarse Coding, Tile Coding, Radial Basis Functions를 사용하여 Feature Vector를 생성해준다.

### 2-2-2. Linear Value Function Approximation

**#. $\hat{v}(S,w)$을 각 Feature들에 Weight($w$)을 가중합한 것으로, 즉 Linear Combination의 형태로 아래와 같이 표현할 수 있다. 이렇게 표현하고 Stochastic Gradient Descent를 통해 $w$을 $v(s)$에 맞도록 아래와 같은 식으로 조정해나가면 된다.**

| Linear Value Function Approximation | Objective Function |
|---|---|
| $\hat{v}(S,\mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$ | $J(\mathbf{w}) = \mathbb{E}_\pi \left[ (v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$ |
| **Update Rule** | |
| $\nabla_w \hat{v}(S,w) = x(S)$<br>$\triangle w = \alpha(v_\pi(S) - \hat{v}(S,w))x(S)$<br>Update = Step-Size × Prediction-Error × Feature Vector | |

a. Linear Function이므로 Stochastic Gradient Descent를 사용하면 Global-Optimal에 수렴한다. 하지만 Linear이므로 Flexibility가 부족하다.

b. Objective Function의 Input은 Weight($w$)이지만 $\hat{v}(S, w)$에서는 Parameter이다.

c. 4장까지 살펴보았던 Tabular Method 또한 Linear Function Approximation의 Special한 예시이다. 왜냐하면 Feature를 State의 개수만큼 생성하고 해당 State에 방문할 때 1이 되는 Feature[6]로 생성하면 해당 State를 Input으로 넣었을 때 나오는 값은 해당 Feature의 Weight이자 State Value이기 때문이다.

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \qquad \hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$



6) State를 One-Hot Encoding으로 표현(Binary Vector)

## 2-3. Incremental Prediction Algorithms

**#. 2-1-2에서 살펴보았지만 $v_\pi(S)$는 Oracle이 주었다고 가정했다. 하지만 강화학습에서는 Reward만 존재하기 때문에 이 True State Value를 어떻게 설정할 것인가가 문제이다. 결론적으로는 이 True State Value Function자리에 MC와 TD를 넣으면 된다. 구체적인 식은 아래와 같다.**

| MC |
| --- |
| $\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$ |
| **TD(0)** |
| $\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$ |
| **TD($\lambda$)** |
| $\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$ |

a. True Value $v(s)$을 구하는 것은 Prediction의 문제이며 일반적인 Model-Free(Unknown MDP)인 상황에선 MC 또는 TD를 사용한다. 또한 MC, TD를 통해 구한 값은 True Value $v(s)$의 Unbiased Sample이므로 Law of Large Number에 의해 수렴한다. 따라서 True Value $v(s)$대신 MC, TD[7]을 사용하는 것이 적절하다.

b. 모방함수가 결국 MC나 TD(0), TD($\lambda$)를 따르게끔 Parameter $w$을 MSE형태의 Loss Function을 만들고, Gradient Descent Algorithm으로 Update하는 것이다.

### 2-3-1. Monte-Carlo with Value Function Approximation

**#. Linear Function을 가정하므로 구체적인 Update식은 아래와 같으며 MC를 사용하므로 한 Episode가 종료된 후에 Update가 가능하다.**

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

a. True Value에 $G_t$을 넣을 수 있는 이유는 Return $G_t$는 True Value $v_\pi(S_t) = E_\pi[G_t|S_t]$의 Unbiased, Noisy한 Sample이기 때문이다. 즉 Value Function의 정의는 Return의 기댓값이므로 매 Episode마다 다른 Return $G_t$을 무수히 많이 Sampling하면 그 평균값은 대수의 법칙에 의해 True value에 수렴하기 때문이다.

b. Linear MC일 때는 Local Optimal로 수렴함이 증명되어 있으며, Neural Net. 같은 Non-Linear를 Function Approximator로 사용해도 수렴함이 증명되어 있다.



7) 항상 수렴하는 것은 아니다.

2-3-2. TD(0), TD($\lambda$) with Value Function Approximation

**#. Linear Function을 가정하므로 구체적인 Update식은 아래와 같으며 MC와 달리 매 Step마다 Update가 가능하다.**

<table>
<tr><th>TD(0)</th><th>TD($\lambda$)</th></tr>
<tr><td rowspan="4">$\Delta \mathbf{w} = \alpha(R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$<br>$= \alpha \delta \mathbf{x}(S)$</td><td>Forward View Linear TD($\lambda$) ($\lambda$-return)</td></tr>
<tr><td>$\Delta \mathbf{w} = \alpha(G_t^{\lambda} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$<br>$= \alpha(G_t^{\lambda} - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$</td></tr>
<tr><td>Backward View Linear TD($\lambda$) (TD($\lambda$))</td></tr>
<tr><td>$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$<br>$E_t = \gamma \lambda E_{t-1} + \mathbf{x}(S_t)$<br>$\Delta \mathbf{w} = \alpha \delta_t E_t$</td></tr>
</table>

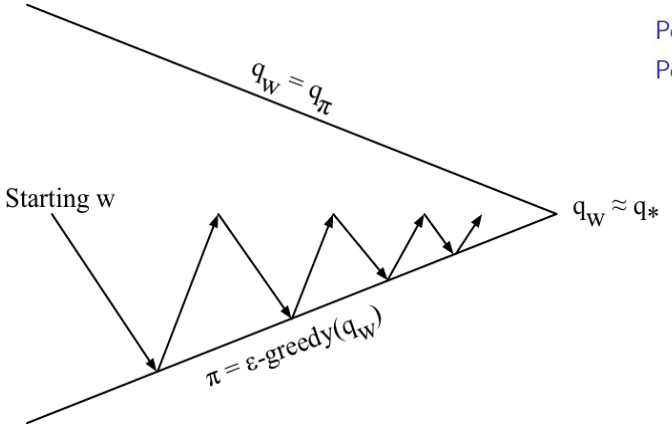
a. TD-Target $R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$는 비록 True State Value의 Biased Sample이지만 위와 같이 Approximation이 가능하다.

b. Linear TD(0)는 Global Optimal에 수렴함이 증명되어 있다.

c. Forward Linear TD($\lambda$)와 Backward Linear TD($\lambda$)는 서로 수학적으로 동일하다.

## 2-4. Incremental Control Algorithms

**#. 이제까지는 State Value Function을 구하는 Prediction에 집중하였다면, Policy를 구하는 Control문제에 대해서도 알아보자. 그런데 5단원에서 Model-Free인 상황에서 Action Value Function을 학습해야 Greedy 방법으로 Policy를 정할 수 있었음을 배웠다. 따라서 $q$도 $\hat{v}$처럼 $\hat{q}$로 표현하고 동일하게 진행한다.**




Policy evaluation Approximate policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$
Policy improvement $\epsilon$-greedy policy improvement

a. Model-Free + Large Scale 문제에서 Policy를 구하기 위해 GPI를 적절히 변형하여 사용하며 Linear Sarsa라고도 한다. 일반적으로 Policy를 구하기 위해서는 Optimal Action Value Function을 구해야하고 이를 구하기 위해선 GPI를 사용한다.

2-4-1. Linear Action-Value Function Approximation

a. $\hat{q}(S, A, w) \approx q_\pi(S, A)$와 같이 Action Value Function을 True Action Value Function으로 Approximation 하는 것이 목적이다.

b. 따라서 목적함수는 아래와 같이 MSE가 되며, Stochastic Gradient Descent Algorithm을 사용하여 Local-minimum을 찾을 수 있다.

| MSE (Objective Func.) | Stochastic Gradient Descent |
|---|---|
| $J(\mathbf{w}) = \mathbb{E}_\pi\left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2\right]$ | $-\frac{1}{2}\nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$<br>$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$ |

c. Feature vector를 사용하여 $\hat{q}(S, A, w)$을 Linear combination으로 표현하고, SGD를 사용하면 아래와 같다.

| Feature Vector | Linear Combination of features |
|---|---|
| $\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$ | $\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$ |
| **Stochastic Gradient Descent** | |
| $\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$<br>$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$<br>Update = Step-Size × Prediction-Error × Feature Vector | |

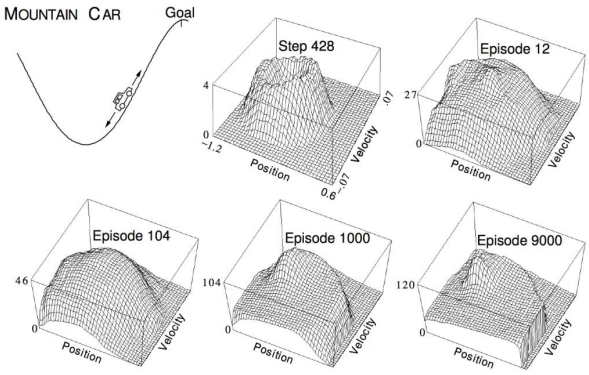
2-4-2. Incremental Control Algorithms(MC, TD(0), TD($\lambda$))

**#. MC, TD(0), TD($\lambda$)을 적용한 Update를 아래와 같이 표현할 수 있다.**

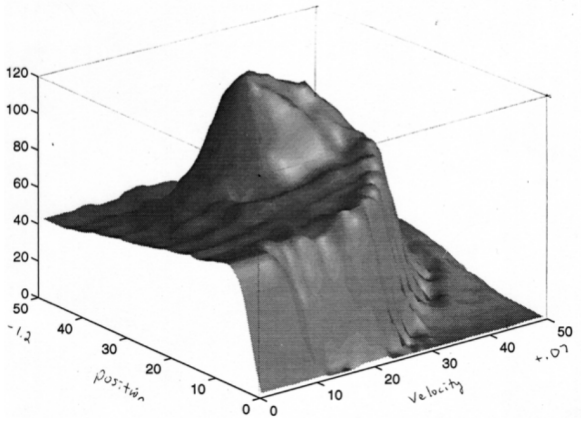
| MC |
|---|
| $\Delta\mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$ |
| **TD(0)** |
| $\Delta\mathbf{w} = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$ |
| **Forward View TD($\lambda$) ($\lambda$-Return)** |
| $\Delta\mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$ |
| **Backward View TD($\lambda$) ($TD(\lambda)$)** |
| $\delta_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})$<br>$E_t = \gamma\lambda E_{t-1} + \nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$<br>$\Delta\mathbf{w} = \alpha\delta_t E_t$ 서강대학교 머신러닝 연구실 서강현 |

## 2-5. Example : Mountain Car with Linear Sarsa

a. State, Action, Reward는 아래와 같이 정의되고 학습한 State Value Function은 아래와 같다.



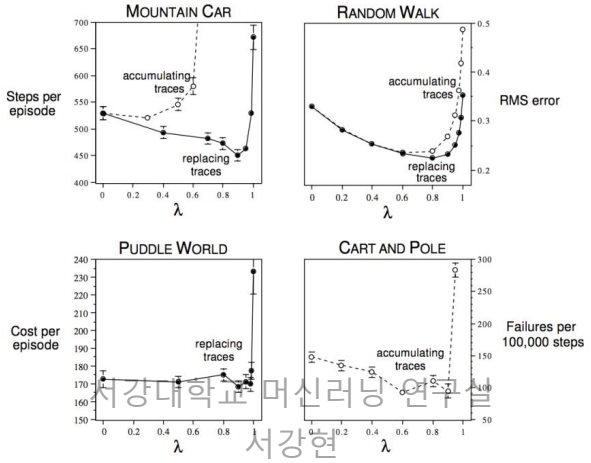

| | |
|---|---|
| **State (Continuous)** | {Position, Velocity} |
| **Action (Discrete)** | {Forward, Hold, Backward} |
| **Reward** | -1 per Time-Step |




b. State Space가 Continuous하므로 TD($\lambda$)을 사용할 때, Eligibility Trace 값을 각 State마다 저장하기 위해선 State를 Discrete하게 변환해야 한다.

### 2-5-1. Should we Bootstrap ?

a. 아래의 그래프는 Control문제를 각기 다른 환경에서 적용한 TD-Learning의 Error를 표현한 예시인데, MC(=TD(1))를 사용하게 되면 Variance가 매우 커서 실제 문제에서는 적용하기 어렵기 때문에 Bootstrapping을 사용하는 것이 좋음을 나타낸다. $\lambda$값에 따른 Sweet Spot이 존재한다.

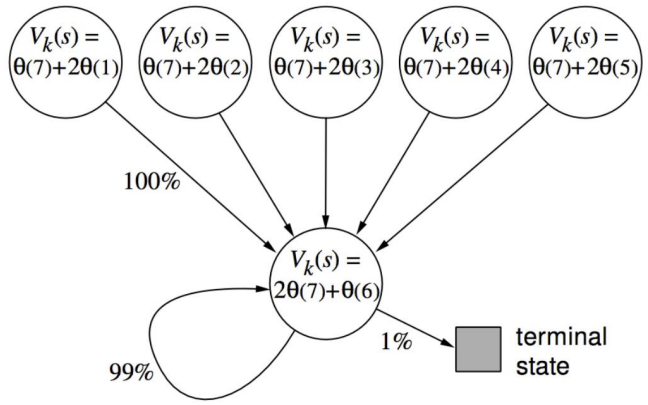


## 2-6. Convergence

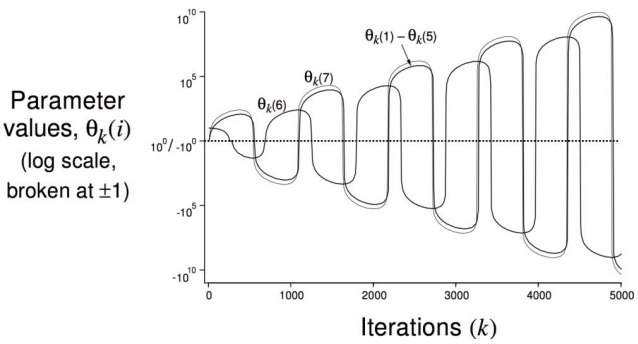
### 2-6-1. Baird's Counterexample for TD(0)'s Convergence

a. TD(0)의 수렴성은 항상 보장되지 않았으며, 아래의 예시는 TD(0)가 발산하는 예시를 보여준다. 그렇지만 현실적으로 사용할 때는 잘 수렴한다.

**Baird's Counterexample**




**Result : parameter divergence**




### 2-6-2. Convergence of Prediction and Control Algorithms

a.

**Convergence of Prediction Algorithms**

| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✓ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✗ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |

**Convergence of Control Algorithms**

| Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|
| Monte-Carlo Control | ✓ | (✓) | ✗ |
| Sarsa | ✓ | (✓) | ✗ |
| Q-learning | ✓ | ✗ | ✗ |
| Gradient Q-learning | ✓ | ✓ | ✗ |

b. GTD(Gradient TD)는 Bellman error의 True Gradient를 따라가기 때문에 Non-Linear에서도 수렴성이 좋고 Off-policy일 때도 수렴성이 좋다.

c. TD는 Objective Function의 Gradient를 따라가지 않기 때문에 Off-policy나 Non-linear Function Approximator를 쓸 때, Divergence한다.

d. 위의 Table에서 '( )' 표시는 Optimal 근처에 수렴한다는 의미이다.

## 3. Batch Methods

### 3-1. SGD with Experience Replay

**#. Incremental Algorithm은 한 번 사용한 Sample은 버려지게 되므로 효율적으로 Sample를 사용하지 못하는 단점이 존재하였다. 따라서 Agent가 쌓아온 경험[8]들을 $D=\{\langle s_1, v_1^\pi\rangle, \langle s_2, v_2^\pi\rangle, ..., \langle s_T, v_T^\pi\rangle\}$에 저장하였다가 Training Data처럼, 아래의 Objective Function에 SGD를 쓸 때, Random Sampling하여 최적의 Parameter $w$을 찾아 $\hat{v}\rightarrow v$하는 것이 목적이다.**

| Objective Function (Least Squares) | SGD update |
|---|---|
| $LS(\mathbf{w}) = \sum_{t=1}^{T}(v_t^\pi - \hat{v}(s_t,\mathbf{w}))^2$<br>$= \mathbb{E}_{\mathcal{D}}\left[(v^\pi - \hat{v}(s,\mathbf{w}))^2\right]$<br>$\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}}\ LS(\mathbf{w})$ | $\Delta\mathbf{w} = \alpha(v^\pi - \hat{v}(s,\mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(s,\mathbf{w})$ |

a. Incremental Methods에서는 Expectation에 $\pi$가 있었다. 이는 $\pi$을 따라가면서 얻는 Data로 Update를 한다는 의미였다. 하지만 Batch Methods에서는 주어진 Data $D$에 대해 Update한다는 의미이다.

b. $D$에서 sampling을 할 때, 같은 Data를 여러 번 Sample할 수 있다. 따라서 이를 Experience Replay라고 한다.

### 3-2. Example : DQN

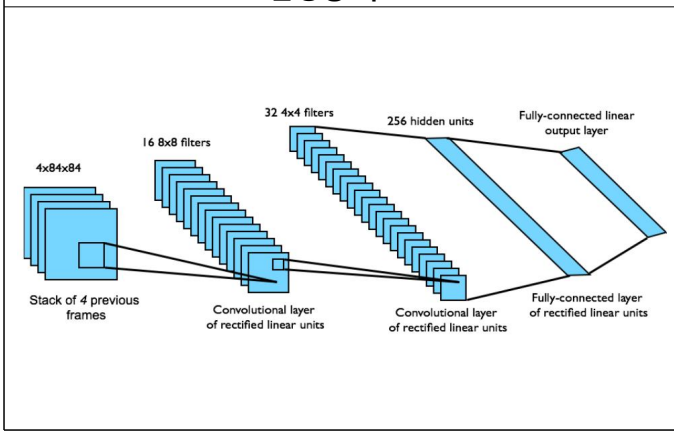
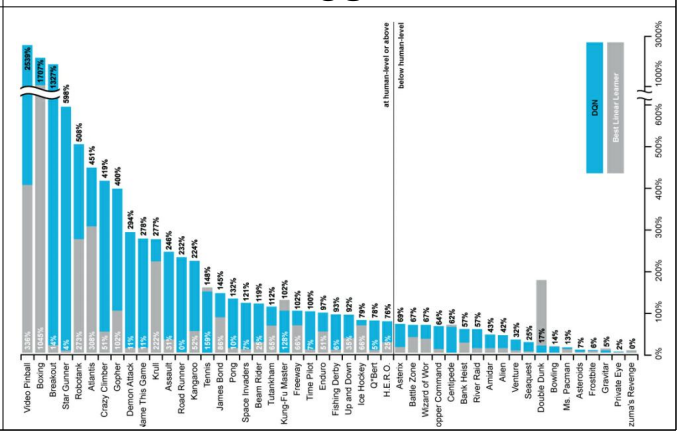
**#. 2-6-2에서 Off-Policy인 상황에서 TD를 이용하여 Action-Value Function을 Non-linear Function Approximation[9]을 하면 수렴하지 않는 문제점이 있었다. DQN에서는 'Experience Replay'와 'Fixed Q-Target'을 이용하여 해결하였는데 각각에 대한 설명은 아래와 같다. 또한 Input에는 게임 Image를 직접 넣고 CNN을 거쳐 Feature Vector를 추출하고, Output으로는 Action-out 형태의 Q-함수가 나온다.**

| | |
|---|---|
| **Experience Replay** | • Replay Memory $D$에 $(s_t, a_t, r_{t+1}, s_{t+1})$의 Data를 저장하고 Random하게 Mini-batch size만큼 Sampling을 하여 SGD할 때 사용한다. |
| **Fixed Q-Target** | $\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$<br>• 위와 같은 MSE형태의 Objective Function을 SGD 알고리즘을 써서 Update할 때, Parameter $w$가 동일하게 있기 때문에 Update의 방향이 계속 바뀌어서 수렴이 불가능하다. 따라서 TD-target에서의 Parameter를 일정 시간동안 고정시킨 상태에서 $\hat{q}$의 Parameter만 몇 회 Update하고 Target의 $w$을 Update하고, 다시 고정시키는 과정을 반복한다. |



8) $\langle s, a, r, s'\rangle$
9) DQN에서는 Neural Network를 사용했으므로 Non-linear이며 Non-linear 중에서는 Neural Net만 알면 된다.

a. DQN 신경망의 구조는 아래와 같으며, 성능 또한 아래의 그래프에 제시되어 있다.

| 신경망 구조 | 성능 |
| --- | --- |
|  |  |



b. 아래의 Table은 'Experience Replay', 'Fixed Q-Target'을 각각 사용하지 않았을 때의 성능을 비교한 결과이다.

| | Replay Fixed-Q | Replay Q-learning | No replay Fixed-Q | No replay Q-learning |
| --- | --- | --- | --- | --- |
| Breakout | 316.81 | 240.73 | 10.16 | 3.17 |
| Enduro | 1006.3 | 831.25 | 141.89 | 29.1 |
| River Raid | 7446.62 | 4102.81 | 2867.66 | 1453.02 |
| Seaquest | 2894.4 | 822.55 | 1003 | 275.81 |
| Space Invaders | 1088.94 | 826.33 | 373.22 | 301.99 |