

## 1. Introduction

### 1-1. Policy Approximation

### 1-2. Policy ?

### 1-3. Advantages of Policy Based RL

### 1-4. Policy Objective Functions and Optimization

## 2. Finite Difference Policy Gradient

### 2-1. Policy Gradient

### 2-2. Finite Difference Policy Gradient

## 3. Policy Gradient Theorem

### 3-1. Likelihood Ratio

### 3-2. Policy Gradient Theorem

## 4. REINFORCE Algorithm : Monte-Carlo Policy Gradient

### 4-1. REINFORCE

### 4-2. REINFORCE with Baseline

## 5. Actor-Critic Methods

### 5-1. One-Step Actor-Critic

### 5-2. Actor-Critic with Eligibility Traces

### 5-3. Action-Value Actor-Critic (QAC)

### 5-4. Advantage Actor-Critic (A2C)

### 5-5. Advantage Actor-Critic with TD-Error

## 6. Summary of Policy Gradient Algorithms

## 1. Introduction

### 1-1. Policy Approximation

#. **Model Free and Scale-up** 상황에서, 6장에서는 Value Function( $v, q$ )을 Neural Net. 또는 Linear Combination을 사용하여  $\hat{v}$ 을 설정한 후 이를 추론하는 Value Based 방법론을 배웠다. 즉  $(v, q)$  각각에 대해 Objective Function( $J(\theta)$ )을 설정하고 SGD를 이용하여 Value Function의 Parameter  $\theta$ 을 Update하는 방법이었다. 이 때 Policy는 수동적으로 정해졌다.

이번 단원에서는 직접 Policy를 6장과 동일한 방식으로 Approximation하는 방법을 알아본다.

a. Policy 또한 하나의 함수이며 아래와 같이 표현한다. ( $\theta$ 는 parameter)

$$\pi_{\theta}(s, a) = P[a | s, \theta]$$

b. 요즘은 Neural Network를 주로 사용하여 Policy를 표현한다.

### 1-2. Policy ?

#. Policy  $\pi(a|s)$ 는 State  $s$ 을 입력으로 받고 해당 State에서 선택 가능한 Action이 선택될 확률 분포<sup>1)</sup>을 Return해주는 함수이다. 함수이므로 다양한 방식(Softmax, Gaussian, Neural Net. etc)으로 표현할 수 있다.

a. Policy는 특정 Time(or State)에서 Agent의 Behavior를 정의해준다.

b. Learning a Policy는 특정 Time(or State)에서 The Best인 Action과의 Mapping을 찾는 것이다.

### 1-3. Advantages of Policy Based RL

#. Policy Based RL의 장단점은 아래와 같다.

Advantages	Disadvantage
<ul style="list-style-type: none"><li>• Well Convergence (Learning Curve가 완만함)</li><li>• Continuous Action Space에서도 사용가능.</li><li>• Stochastic Policy 학습이 가능.</li></ul>	<ul style="list-style-type: none"><li>• Local Optimal에 빠지기 쉽다.</li><li>• 학습 과정이 Stable하지만 효율성이 떨어진다. (High Variance)</li></ul>

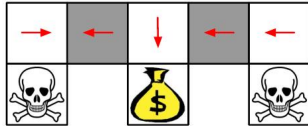
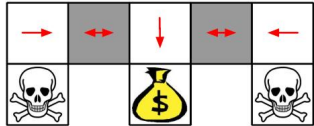
a. Discrete Action Space는 1번 action, 2번 action ... 이처럼 Action이 서로 분리되어 있는 상태이지만 Continuous Action Space의 경우 0과 1사이의 action을 정해주어야 하는 상황이다. 이런 상황에서 Value based 방법론을 쓰게 된다면 Action을 정해주어야 하는 것 또한 또 하나의 최적화 문제가 되어버린다.

b. Value Based는 Value Function( $v, q$ )을 Approximation한 후에 수렴된 이 값을 이용하여 Policy를 찾을 때, Greedy 방법(max 연산)을 사용하기 때문에 Policy가 확확 바뀌는 Aggressive한 방법론이다. 하지만 Policy Based의 경우 확률 값이 조금씩 증가하기 때문에 학습 과정이 stable하지만 높은 Variance가 있다.

c. Value Based 방법론에서는 Stochastic Policy를 고려하지 않고 State마다 취해야할 Action이 정해진 Deterministic Policy만을 고려하였다. 하지만 이는 매우 치명적인 단점이 있으며 그 예시는 아래와 같다.

서강대학교 머신러닝 연구실

1) Action Space가 Discrete이면 Vector, Continuous이면 Probability Distribution으로 표현한다.

Rock-Paper-Scissors	Aliased GridWorld
<ul style="list-style-type: none"> <li>• Deterministic Policy인 경우 예를 들어 가위만 내는 policy라고 한다면, 승률이 좋지 않다.</li> <li>• 따라서 1/3씩 내는 Uniform Random Policy가 Optimal이며 이는 Nash Equilibrium을 이룬다.</li> </ul>	<ul style="list-style-type: none"> <li>• 아래와 같은 상황에서, Model Free POMDP를 가정해보자. 일반적으로 POMDP에선 Feature를 설정하는 것이 불완전하기 때문에 Optimal Solution도출을 못할 가능성이 존재한다.</li> <li>• 아래의 예시에서 Feature를 현 Agent 위치의 위에 벽이 있는지 없는지(0 or 1)로 설정했다고 하자. 그러면 왼쪽의 그림과 같이 Deterministic Policy의 경우에는 다른 State임에도 같은 잘못 설정한 Feature로 인해 동일한 State로 인식하여 Goal에 도달하지 못하는 상황이 발생한다.</li> <li>• 이를 해결하기 위한 방법은 Stochastic Policy를 사용하여 Uniform Random Policy를 설정해주는 것이 최선이다.</li> </ul> $\pi_{\theta}(\text{wall to N and S, move E}) = 0.5$ $\pi_{\theta}(\text{wall to N and S, move W}) = 0.5$ <div style="display: flex; justify-content: space-around; align-items: center;">   </div>

### 1-3. Policy Objective Functions and Optimization

#. Policy Performance를 측정하는 Policy Objective Function  $J(\theta)$ 는 아래와 같이 3가지로 정의할 수 있으며, 이렇게 목적함수를 정의해주고 SGD를 사용하여  $J(\theta)$ 을 Max해주는 Input  $\theta$ (Policy Parameter)를 찾으면 된다. 또한 세 가지 각기 다른 목적이지만, 같은 방법으로 동일하게 작동하는 장점이 있다.

Episodic Environment	Start Value	$J_1(\theta) = V^{\pi_\theta}(s_1) = E_{\pi_\theta}[v_1]$
Continuing Environment (Non-Episodic)	Average Value (State Value관점)	$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$
	Average Reward per Time-Step	$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R_s^a$

Start Value	<ul style="list-style-type: none"> <li>시작하는 State에서 <math>\pi_\theta</math>을 따라서 끝까지 행동하였을 때, 얻을 수 있는 Start State Value의 기댓값.</li> </ul>
Average Value	<ul style="list-style-type: none"> <li>각 State에 있을 확률과 그 State에서의 Value를 곱한 값. 즉 Policy <math>\pi_\theta</math>을 따랐을 때, 모든 State Value들의 평균값을 의미한다.</li> </ul>
Average reward per time-step	<ul style="list-style-type: none"> <li>한 Step만 보는 방법으로, 모든 State에서 각각 <math>\pi_\theta</math>을 따라 한 번의 Action을 하였을 때, 받는 Reward에다 가중치 <math>\pi_\theta</math>을 곱한 값이다. 즉 Policy <math>\pi_\theta</math>을 따른다는 가정 하에 모든 State에서 취할 수 있는 모든 Action에 따른 기대 Reward를 평균 낸 것이다.</li> </ul>

- Objective Function은 Max, Min연산의 대상이다. 그렇다면 여기에서는 어떤 것이 목적함수가 되어야 하는가?에 대한 답은 다음과 같다. Value Function은  $G_t$ 의 기댓값이고 좋은 policy는  $G_t$ 을 크게 받도록 하는 것이기 때문에 Policy Gradient Objective Function에는 Value Function이 들어간다.
- 또한 이 Value Function의 Parameter는 Policy( $\pi$ )가 들어가고 Policy의 Parameter에는  $\theta$ 가 들어간다. 결국에는  $\theta$ 의 Optimal Value를 도출하면 된다. 중요한 점은 Policy Gradient는 Objective Function  $J(\theta)$ 에 대한 Gradient를 Update하는 것이지  $\pi$ 의 Gradient를 Update한다고 생각하면 안 된다.
- State Distribution  $d^{\pi_\theta}(s)(\mu(s) \geq 0, \sum_s \mu(s) = 1)$ 는 Markov Chain에서  $\pi_\theta$ 을 따라 수렴할 때까지 움직이면, 각 State별로 머무는 확률들이 나오는데 이를 분포로 표현한 것이다.
- 정리하면, Policy Gradient는 직접 Policy를 Approximation하는 과정이며 이를 위해서는 3가지 방법으로 Policy Objective Function을 설정해주고 이를 최적화하는 방법론이다. 즉 Policy의 Parameter인  $\theta$ 의 Optimal Policy를 찾는 것이 목적이다.

## 2. Finite Difference Policy Gradient

### 2-1. Policy Gradient

#. Policy Gradient의 핵심은 앞서 1-4에서 정의한 Objective Function  $J(\theta)$  (Performance Measure)를 Max하는 Input  $\theta$ 을 찾는 것이므로, Gradient Ascent를 사용하면 된다. 아래와 같이  $\theta$ 을  $n$ 개의 Parameter로 Vector로 표현할 수 있으며 각  $\theta_n$ 에 대해 편미분한 값을 이용하여 Update할 수 있다.

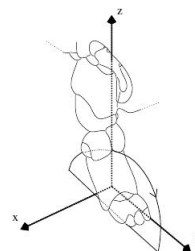
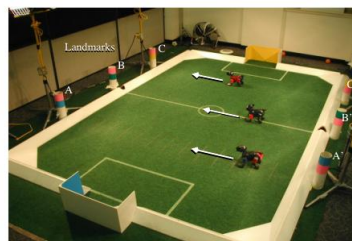
Parameter Vector	Gradient Ascent Update Rule
$\theta = \langle \theta_1, \theta_2, \dots, \theta_n \rangle$	$\alpha \nabla_{\theta} J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$

### 2-2. Finite Difference Policy Gradient

#.  $\theta$ 가 아래와 같이  $n$ -Dimension에서  $n$ 개가 있다고 하면 각각에 대해 아래와 같이  $\theta_n$  값을 조금씩 바꾸면서 Gradient Ascent Update Rule을 구하는 방법이다.

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- 이 방법은 한번의 Update가 매우 느리다. 왜냐하면  $n$ 차원의 Gradient를 구하려면 매번 Parameter 하나씩 계산을 해야 하기 때문이다. Neural Net.의 경우 Weight의 개수가 엄청 많은 점을 고려하면 이 방법은 비효율적임을 알 수 있다.
- 그렇지만 임의의 Policy Functions에 대해서도 사용할 수 있다는 장점이 있다.
- 아래의 예시는 위의 방법을 사용하여 ALBO라는 축구 로봇을 개발한 것이다. 여기에서 사용된 Parameter는 12개<sup>2)</sup>이다.



서강대학교 머신러닝 연구실

서강현

2) # of parameters = # of dimension

### 3. Policy Gradient Theorem

#### 3-1. Likelihood Ratios

#. Policy Objective Function( $J(\theta)$ )에 Gradient Ascent를 적용하기 위해서는 Policy의 Gradient( $\nabla_{\theta}\pi_{\theta}(s, a)$ )를 구해야한다. 이 때, 계산상의 불편함 때문에 아래와 같은 Likelihood Ratio Tricks를 사용하여 계산상의 편리함을 이룰 수 있다.

Policy Gradient	Eligibility Vector
$\begin{aligned}\nabla_{\theta}\pi_{\theta}(s, a) &= \pi_{\theta}(s, a) \frac{\nabla_{\theta}\pi_{\theta}(s, a)}{\pi_{\theta}(s, a)} \\ &= \pi_{\theta}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a)\end{aligned}$	$\nabla_{\theta} \log \pi_{\theta}(s, a)$

- Policy Gradient를 계산하기 위한 가정 2가지는 'Policy  $\pi$ 가 항상 미분가능하고', ' $\pi$ 에 대한 Gradient를 알고 있다.'이다.
- Likelihood Ratios는 결국 log 함수의 미분을 이용한 것이다.
- Policy  $\pi_{\theta}$ 는 우리가 자율적으로 정해주어야 하는 것(=Policy Parameterization)이다. 대표적인 예시로 아래와 같이 'Softmax'와 'Gaussian'을 이용할 수 있으며 각 State마다 동일하게 정의된다. (주로 Neural Network를 사용한다.)

Softmax Policy
$\pi_{\theta}(s, a) \propto e^{\phi(s, a)^T \theta}$ $\nabla_{\theta} \log \pi_{\theta}(s, a) = \phi(s, a) - E_{\pi}[\phi(s, \cdot)] = \phi(s, a) - \sum_b \pi(b s) \phi(s, b)$
<ul style="list-style-type: none"> <li>Discrete Action Space에서 사용하는 Policy Parameterization의 한 방법이다.</li> <li>Feature의 선형결합(<math>\phi(s, a)^T \theta</math>)을 이용하여 각 Action에 Exponential Weight를 주어 각 Action을 확률형 태로 표현하는 Policy.</li> </ul>
Gaussian Policy
$a \sim N(\mu(s), \sigma^2)$ $\nabla_{\theta} \log \pi_{\theta}(s, a) = \frac{(a - \mu(s)) \phi(s)}{\sigma^2}$
<ul style="list-style-type: none"> <li>Continuous Action Space에서 주로 사용하는 Policy Parameterization의 한 방법으로 Mean은 <math>\mu(s) = \phi(s)^T \theta</math>, Variance는 <math>\sigma^2</math>로 고정되거나 다른 함수로 표현할 수 있다.</li> <li><math>\mu(s)</math>가 있어서 어떤 State에서 대체로 이 Action을 할 것인데, Variance를 설정하여 다른 Action들도 할 수 있도록 한 Policy.</li> </ul>

### 3-2. Policy Gradient Theorem

#. Policy의 Performance를 나타내는  $J(\theta)$ 에 Gradient Ascent를 적용해도 과연 성능이 향상되는지에 대한 의문이 발생한다.(a에 자세한 설명) 즉 Function Approximation의 방법론으로 Performance를  $J(\theta) = v_{\pi_\theta}(s_0)$ 로 설정하였을 때, Policy Parameter  $\theta$ 의 변화가 Performance를 Improvement해준다고 보장하기 어렵다는 것이다. 이를 해결하기 위해 Policy Gradient Theorem은 아래와 같이  $\theta$ 의 변화에 따른 Performance의 변화를 나타내주는 식을 변형하여 제시한다.

Policy Gradient Theorem
$\nabla_{\theta} J(\theta) = \nabla_{\theta} v_{\pi_{\theta}}(s_0) \rightarrow \nabla_{\theta} J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a s) q_{\pi}(s, a)$
<ul style="list-style-type: none"> <li>논의의 간략화를 위해 Episodic Environment로 설정하고 Policy Objective(Performance Measure) <math>J(\theta)</math>을 State Value(<math>v_{\pi_{\theta}}(s)</math>)로 가정한다.</li> </ul>

- a. 위에 제기한 의문을 구체화 하면 Performance  $J(\theta)$ 에 영향을 미치는 요인은 Action Selection과 Selection이 이루어지는 State Distribution( $\mu(s)$ ) 2가지가 있다. 또한  $\mu(s)$ 는 Policy Parameter  $\theta$ 에 영향을 받는 구조인데,  $\theta$ 가  $\mu(s)$ 에 영향을 끼칠 때, 환경에 따라 미치는 정도가 다르기 때문에 선불리  $\theta$ 의 변화가  $J(\theta)$ 에 어떤 영향을 끼치는지 파악하기 어렵다는 것이다.
- b.  $\nabla_{\theta} J(\theta)$ 을 위와 같이 변형하여도 Gradient Ascent를 직접적으로 사용하기에는 계산상의 번거로움이 있다. 따라서 아래의 과정과 같이  $\nabla_{\theta} J(\theta)$ 을 변형하여 Update할 때 사용한다. 아래의 과정에서 Action  $a$ 을 Sample  $A_t \sim \pi$ 로 대체한다.

$$\begin{aligned}
 \nabla_{\theta} J(\theta) &\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\
 &= E_{\pi} \left[ \sum_a q_{\pi}(S_t, a) \nabla \pi(a|S_t, \theta) \right] \\
 &= E_{\pi} \left[ \sum_a \pi(a|S_t, \theta) q_{\pi}(S_t, a) \frac{\nabla \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right] \\
 &= E_{\pi} \left[ q_{\pi}(S_t, A_t) \frac{\nabla \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right] = E_{\pi} [q_{\pi}(S_t, A_t) \nabla \ln \pi(A_t|S_t, \theta)]
 \end{aligned}$$

- c. 위의 결과를 이용하여 Update를 할 때는 SGD를 사용하므로  $\nabla_{\theta} J(\theta) = q_{\pi}(S_t, A_t) \nabla \ln \pi(A_t|S_t, \theta)$ 을 사용하면 된다. 즉 아래와 같이 Update를 진행한다.

$$\theta_{t+1} = \theta_t + \alpha q_{\pi}(S_t, A_t) \nabla \ln \pi(A_t|S_t, \theta)$$

## 4. Monte-Carlo Policy Gradient (REINFORCE Algorithm)

### 4-1. REINFORCE

#. Policy Gradient Theorem을 통해 Policy Gradient Ascent의 구체적인 Update식을 도출하였다. 하지만  $q_\pi(S_t, A_t)$ 은 True-Value이므로 구체적인 수치를 알 수 없다. 따라서 이 값의 Unbiased Sample인 Return  $G_t$ 로 대신사용하여 Update하는 것이 REINFORCE Algorithm이다. ( $E_\pi[G_t|S_t, A_t] = q_\pi(S_t, A_t)$ ) 구체적으로는 매 Episode를 직접 실행하고 끝난 후에 그 Episode에서의 매 Step마다  $\theta$ 을 Update해주는 방법이다.

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic), for estimating  $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$

Algorithm parameter: step size  $\alpha > 0$

Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$  (e.g., to  $\mathbf{0}$ )

Loop forever (for each episode):

    Generate an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi(\cdot|\cdot, \theta)$

    Loop for each step of the episode  $t = 0, \dots, T - 1$ :

$G \leftarrow$  return from step  $t$  ( $G_t$ )

$\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \ln \pi(A_t|S_t, \theta)$

- Policy Parameter  $\theta$ 가 Update되면서 Policy  $\pi$ 또한 점점 Update된다.
- MC방식을 사용했기 때문에 Episodic Env.에서만 사용가능하고, High Variance, Slow Learning의 특징이 있다.
- Update 과정은 Return에 비례하여 Parameter Vector( $\theta$ )를 Update해주는 방향으로 진행되지만, Action Selection Probability와 반비례하여 Update해준다. 왜냐하면  $\theta$ 의 Update는 Highest Return을 주었던 Action을 선택하는 방향으로 이루어지는 것이지, 가장 많이 선택하는 Action으로 Update가 되는 것이 아니기 때문이다.
- 위에 적힌  $\gamma$ 는 Discount Factor를 사용한다는 의미이다.
- Alpha-Go에서도 쓰인 알고리즘이며, 'Initialise  $\theta$  Arbitrarily' 대신 SL로 학습한  $\theta$ 을 넣어 강화학습을 한 것이다.



## 4-2. REINFORCE with Baseline

#. 기존 REINFORCE Algorithm을 일반화하기 위해<sup>3)</sup> 임의의 Baseline Function  $b(s)$ 을 아래와 같이 도입한다. 이렇게 되면  $q_\pi(s, a) - b(s)$ 은 Generalization이 가능해진다.  $b(s)$  Function은 주로 Estimated State Value Function  $\hat{v}(S_t, w)$ 을 사용하며,  $\hat{v}(S_t, w)$ 의 Parameter  $w$  또한 MC방법으로 Update해주는 것이 핵심이다.

Policy Gradient Theorem with Baseline	Update
$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a) - b(s)) \nabla \pi(a s, \theta)$	$\theta_{t+1} = \theta_t + \alpha (G_t - b(s)) \frac{\nabla \pi(A_t S_t, \theta_t)}{\pi(A_t S_t, \theta_t)}$

### REINFORCE with Baseline (episodic), for estimating $\pi_\theta \approx \pi_*$

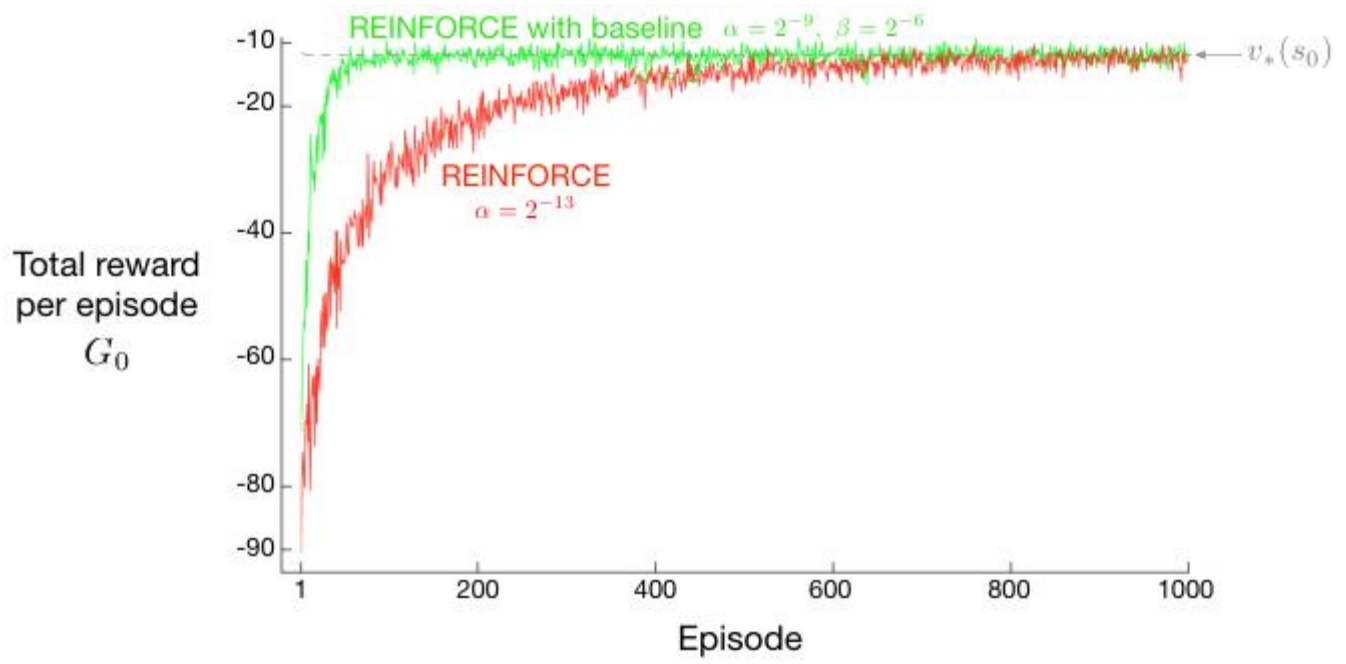
Input: a differentiable policy parameterization  $\pi(a|s, \theta)$   
 Input: a differentiable state-value function parameterization  $\hat{v}(s, \mathbf{w})$   
 Algorithm parameters: step sizes  $\alpha^\theta > 0$ ,  $\alpha^{\mathbf{w}} > 0$   
 Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$  and state-value weights  $\mathbf{w} \in \mathbb{R}^d$  (e.g., to  $\mathbf{0}$ )  
 Loop forever (for each episode):  
   Generate an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi(\cdot|\cdot, \theta)$   
   Loop for each step of the episode  $t = 0, \dots, T-1$ :  
      $G \leftarrow$  return from step  $t$  ( $G_t$ )  
      $\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$   
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$   
      $\theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla_\theta \ln \pi(A_t|S_t, \theta)$

a. Baseline을 추가하여도 Policy Gradient Theorem이 성립하는지에 대한 증명은 아래와 같다.

$$\sum_a b(s) \nabla_\theta \pi(a|s, \theta) = b(s) \nabla_\theta \sum_a \pi(a|s, \theta) = b(s) \nabla_\theta 1 = 0.$$

b. 위의 Pseudo Code에서 표시된 부분은 Estimated State Value Function의 Parameter  $w$ 을 Update하는 과정이다. MC방법을 사용하였기 때문에 동일하게 MC방식으로 한 Episode가 끝난 후  $\theta$ 와 함께 Update를 진행한다.

c. 아래의 그래프는 Baseline과 Non-Baseline의 성능을 비교한 것이다.



## 5-1. One-Step Actor-Critic

#. REINFORCE with Baseline Algorithm에서 단순히 Full Return( $G_t$ )을 One-Step Return으로 바꾼 형태이다. 하지만 이러한 변화 때문에 기존과는 다르게 Bootstrapping<sup>4)</sup>이 가능해졌으며 이로 인해 Low Variance와 학습속도가 빨라진 장점이 생겼다. One-Step Return( $TD(0)$ )으로 바뀌면서 Prediction 역할을 하는 Critic의 개념이 생성되었고 Online Learning, Continuous Learning이 가능해졌다.

## Update

$$\begin{aligned}
 \theta_{t+1} &\doteq \theta_t + \alpha \left( G_{t:t+1} - \hat{v}(S_t, \mathbf{w}) \right) \frac{\nabla_{\theta} \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)} \\
 &= \theta_t + \alpha \left( R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \right) \frac{\nabla_{\theta} \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)} \\
 &= \theta_t + \alpha \delta_t \frac{\nabla_{\theta} \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)}.
 \end{aligned}$$

One-Step Actor-Critic (episodic), for estimating  $\pi_0 \approx \pi_*$ 

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization  $\hat{v}(s, \mathbf{w})$

Algorithm parameters: step sizes  $\alpha^{\theta} > 0$ ,  $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$  and state-value weights  $\mathbf{w} \in \mathbb{R}^d$  (e.g., to  $\mathbf{0}$ )

Loop forever (for each episode):

    Initialize  $S$  (first state of episode)

$I \leftarrow 1$

    Loop while  $S$  is not terminal (for each time step):

$A \sim \pi(\cdot | S, \theta)$

        Take action  $A$ , observe  $S', R$

$\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

(if  $S'$  is terminal, then  $\hat{v}(S', \mathbf{w}) \doteq 0$ )

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$

$\theta \leftarrow \theta + \alpha^{\theta} I \delta \nabla_{\theta} \ln \pi(A | S, \theta)$

$I \leftarrow \gamma I$

$S \leftarrow S'$

a. 위의 Pseudo Code에서 빨간색 부분은 Policy Evaluation을 하는 Critic, 검은색 부분은 Actor이다.

b. REINFORCE with Baseline에서 Estimated State Value Function  $\hat{v}(s)$ 은 단지 Baseline 역할만 하였지만 One-Step Return으로 바뀌어 지면서 즉 Bootstrapping이 적용되면서  $\hat{v}(s)$ 은 Critic이 되었다.

## 5-2. Actor-Critic with Eligibility Traces

#. 5-1.에서  $TD(0)$ 을 사용하였기 때문에  $\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$ 에서  $R + \gamma \hat{v}(S', w)$ 을 확장시켜 Forward View로 손쉽게  $n$ -Step( $G_{t:t+n}$ ),  $\lambda$ -Return( $G_t^\lambda$ )로 대체할 수 있으며 Backward View의  $TD(\lambda)$ 로도 쉽게 바꿀 수 있다.

### Actor-Critic with Eligibility Traces (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$   
 Input: a differentiable state-value function parameterization  $\hat{v}(s, \mathbf{w})$   
 Algorithm parameters: trace-decay rates  $\lambda^\theta \in [0, 1]$ ,  $\lambda^{\mathbf{w}} \in [0, 1]$ ; step sizes  $\alpha^\theta > 0$ ,  $\alpha^{\mathbf{w}} > 0$   
 Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$  and state-value weights  $\mathbf{w} \in \mathbb{R}^d$  (e.g., to  $\mathbf{0}$ )

Loop forever (for each episode):  
 Initialize  $S$  (first state of episode)  
 $\mathbf{z}^\theta \leftarrow \mathbf{0}$  ( $d'$ -component eligibility trace vector)  
 $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$  ( $d$ -component eligibility trace vector)  
 $I \leftarrow 1$   
 Loop while  $S$  is not terminal (for each time step):  
 $A \sim \pi(\cdot|S, \theta)$   
 Take action  $A$ , observe  $S', R$   
 $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$  (if  $S'$  is terminal, then  $\hat{v}(S', \mathbf{w}) \doteq 0$ )  
 $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + I \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$   
 $\mathbf{z}^\theta \leftarrow \gamma \lambda^\theta \mathbf{z}^\theta + I \nabla_\theta \ln \pi(A|S, \theta)$   
 $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$   
 $\theta \leftarrow \theta + \alpha^\theta \delta \mathbf{z}^\theta$   
 $I \leftarrow \gamma I$   
 $S \leftarrow S'$

a. One-Step Actor-Critic과 다른 점은 Critic, Actor 각각에 Eligibility Traces를 생성하여 Update해주는 것 뿐이다.

### 5-3. Action-Value Actor-Critic (QAC)

#. 5-1. ~ 5-2.에서의 Critic은 Estimation of State-Value Function만 Update하였다. 하지만 Value Function에는 Action-Value Function도 있으므로 Critic이 Action-Value Function을 동일한 방법으로 Estimation할 수 있다.

#### QAC Algorithm

```
function QAC
  Initialise  $s, \theta$ 
  Sample  $a \sim \pi_\theta$ 
  for each step do
    Sample reward  $r = \mathcal{R}_s^a$ ; sample transition  $s' \sim \mathcal{P}_{s,\cdot}^a$ .
    Sample action  $a' \sim \pi_\theta(s', a')$ 
     $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$ 
     $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$ 
     $w \leftarrow w + \beta \delta \phi(s, a)$ 
     $a \leftarrow a', s \leftarrow s'$ 
  end for
end function
```

- 위의 Pseudo Code에서는 TD(0)를 사용하였으며  $Q_w(s, a) = \phi(s, a)^T w$ 으로 표현한다. 즉 Linear Combination of Feature Function Approximation을 사용한 것이다.
- 결국 TD-Error인  $\delta$ 는  $w$ 을 update하는데 쓰이고,  $w$ 는  $\theta$ 을 Update하는데 쓰인다. 즉  $\theta$ 는 Action을 바꾸는 방향인데, 바꿀 때  $Q_w$ 을 참고하여 좋았으면 좋게 바꾸는 것이고, 안 좋았으면 그 Action을 하지 않도록 확률을 조정하는 방식이다.
- 정리하면 Critic은 Estimated (Action / State) Value Function의 Parameter인  $w$ 을 Update하는 역할이고, ( $\hat{V}_w(s) \approx V^{\pi_\theta}(s)$ ,  $\hat{Q}_w(s, a) \approx Q^{\pi_\theta}(s, a)$ ) Actor는 Critic이 Update된 방향으로 Policy의 Parameter인  $\theta$ 을 아래와 같이 Update하는 역할이다.

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

$$\nabla \theta = \alpha \nabla_\theta \log_{\pi_\theta}(s, a) Q_w(s, a)$$

#### 5-4. Advantage Actor-Critic (A2C)

#. 5-3을 보완한 방법으로, 여전히 Variance가 존재하기 때문에 이를 더 줄이고자 하는 방법이다. 예를 들어  $Q$ 값대로  $\theta$ 를 Update하고 있는데,  $Q$ 값들의 절대적 수치가 너무 크면 계산량이 많아진다. 하지만 Policy는 Action들 간의 상대적인 우열을 정하는 것이기 때문에  $Q$ 값을 줄이기 위해, REINFORCE with Baseline Algorithm과 동일한 방법으로 State  $s$ 에 대한 Baseline Function  $B(s)$ 을 도입하여 각  $Q$ 값에서 빼준다. Baseline Function 중 Value Function이 가장 좋으므로  $B(s) = V^{\pi_\theta}(s)$ 으로 설정해주면 Advantage Function은 아래와 같이 정의되며 이는 policy gradient의 variance를 획기적으로 줄일 수 있다.

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

a. Baseline Function을 추가해도 수학적으로 동일하다는 증명은 아래와 같다.

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

b. Advantage Function은 State  $s$ 에 있는 것보다 Action  $a$ 을 하면 얼마나 더 좋은지, 안 좋은지의 상대적인 평가 기준을 만들어 준다.

c. 하지만 결론적으로 학습해야 하는 Parameter의 수는 하나 더 증가하였다. Policy의 Parameter  $\theta$ ,  $Q$ 의 Parameter  $w$ ,  $V$ 의 Parameter  $v$ 가 그것들이다.

$$\begin{aligned} V_v(s) &\approx V^{\pi_\theta}(s) \\ Q_w(s, a) &\approx Q^{\pi_\theta}(s, a) \\ A(s, a) &= Q_w(s, a) - V_v(s) \end{aligned}$$

## 5-5. Advantage Actor-Critic with TD-Error

#. 5-4에서 나온 문제점은 학습해야 할 Parameter가 많아졌다는 것이다. 하지만 TD-error는 Advantage Function의 Unbiased Estimation<sup>5)</sup>이기 때문에 TD-error로 Advantage Function을 대신할 수 있다. 그렇게 되면  $Q(s, a)$ 가 사라지기 때문에 Parameter의 개수가 줄어든다.

$$A(s, a) = \boxed{Q(s, a)} - V(s)$$

$$A(s, a) = \underbrace{r + \gamma V(s') - V(s)}_{\text{TD Error}}$$

a. TD-error가 Advantage Function의 Unbiased Sample인 것의 증명은 아래와 같다.

■ For the true value function  $V^{\pi_\theta}(s)$ , the TD error  $\delta^{\pi_\theta}$

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

■ is an unbiased estimate of the advantage function

$$\begin{aligned} \mathbb{E}_{\pi_\theta} [\delta^{\pi_\theta} | s, a] &= \mathbb{E}_{\pi_\theta} [r + \gamma V^{\pi_\theta}(s') | s, a] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s, a) \end{aligned}$$

■ So we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) \delta^{\pi_\theta}]$$

■ In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

■ This approach only requires one set of critic parameters  $v$

## 5. Summary of Policy Gradient Algorithms

---

#. Policy Gradient는 다양하지만 그 의미가 동일한 형태들이 있으며 각 형태마다 Algorithm이름이 붙는다.

$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{v}_t]$	REINFORCE
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{Q}^w(s, a)]$	Q Actor-Critic
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{A}^w(s, a)]$	Advantage Actor-Critic
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta]$	TD Actor-Critic
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta \mathbf{e}]$	TD( $\lambda$ ) Actor-Critic
$G_{\theta}^{-1} \nabla_{\theta} J(\theta) = w$	Natural Actor-Critic