

1. Introduction

2. On-Policy Monte-Carlo Control

- 2-1. Generalized Policy Iteration (Policy Evaluation 측면)
- 2-2. Exploration (Policy Improvement 측면)
- 2-3. GLIE and Monte-Carlo Control
- 2-4. Blackjack Example

3. On-Policy Temporal-Difference Learning Control

3-1. Introduction

3-2. Sarsa

- 3-2-1. Sarsa
- 3-2-2. Convergence
- 3-2-3. Windy World Example

3-3. Sarsa(λ)

- 3-3-1. n -Step Sarsa
- 3-3-2. Forward View Sarsa(λ)
- 3-3-3. Backward View Sarsa(λ)
- 3-3-4. Grid-World Example

4. Off-Policy Learning

4-1. Introduction

- 4-1-1. Backgrounds
- 4-1-2. Off-Policy Learning의 의미와 특징

4-2. Importance Sampling in MC and TD

- 4-2-1. 의미
- 4-2-2. Importance Sampling for Off-Policy Monte-Carlo
- 4-2-3. Importance Sampling for TD

4-3. Q-Learning

- 4-3-1. General Q-Learning
- 4-3-2. Q-Learning

5. Summary

5-1. Relationship Between DP and TD

서강대학교 머신러닝 연구실
서강현

1. Introduction

#. Model-Free(unknown MDP)인 상황에서 Control 문제를 풀 것이다. 즉 State Transition Probability와 Reward를 모르는 상태에서 최적의 Policy를 찾는 문제인데, Policy에는 Agent가 직접 Update하는 Target Policy(π)와 Agent의 Behavior를 Generate하는 Behavior Policy(μ)가 있다. 이 두 종류의 Policy가 같으면 On-policy, 다르면 Off-policy이다. 전자의 경우 Agent가 하나의 Policy를 이용하여 Action을 Selection하고, 이를 바탕으로 Policy를 최적화하는 방식이라면, 후자의 경우 별도로 존재하는 Behavior Policy에 따라 Action을 정하고 이에 따른 결과를 이용하여 Target Policy를 Update하는 방식이다. 이 때, 학습이 진행될수록 Behavior Policy에 따른 Action Selection의 빈도는 감소해야 한다.

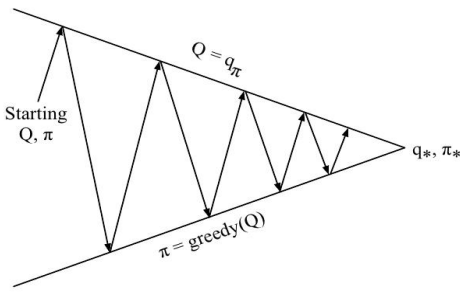
a. Model-Free Control은 MDP가 Unknown이거나, Known일 때도 Sampling(직접 Env.를 경험하는 방법)으로만 접근할 수 있을 때 사용한다.

b. 5장까지 유지할 방식은 'Tabular Method¹⁾' 방식이다. 즉 Value Function을 표현할 때 Table을 사용하여 이를 Update하는 방법이다. 하지만 이 방식은 State의 수가 늘어나면 더 이상 쓸모가 없어지기 때문에 이후에는 Function Approximator방법으로 Value Function을 표현한다. 즉 Env.의 특징에 따라 (Action / State) Value Function을 표현하는 방법이 Tabular or Function Approximation으로 나뉜다.

2. On-Policy Monte-Carlo Control

2-1. Generalized Policy Iteration (Policy Evaluation 측면)

#. 일반적인 상황인 Model-Free(Unknown MDP)에서는 Policy Iteration을 사용할 때, Policy Evaluation에서 MC를 사용하여 State Value대신 Action Value를 구하고 Policy Improvement 단계에서 Action-Value를 바탕으로 Greedy하게 Action을 선택한다. 하지만 이렇게 되면 모든 State를 방문하지 못하여 Exploration이 충분하지 않게 되는 문제점이 발생한다.

기존 Policy Iteration	Generalized Policy Iteration
$\pi'(s) = \operatorname{argmax}_{a \in A} (R_s^a + P_{ss'}^a V(s'))$	<div style="text-align: center;"> $\pi'(s) = \operatorname{argmax}_{a \in A} Q(s, a)$  </div> <p>Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$ Policy improvement Greedy policy improvement?</p>

- 기존에 살펴본 Policy Iteration은 MDP를 안다는 가정이므로 다음 State가 무엇인지 확률적으로 알고 있었다. 이 때문에 Policy Evaluation을 통해 State Value를 구하고, 이를 이용하여 Improvement 단계에서 Greedy하게 State Value에 따라 Action을 선택할 수 있었다.
- 하지만 Model-Free인 상황에서는 Agent가 직접 Action을 취하기 전까진 다음 State에 대한 정보가 가려져 있기 때문에 MC를 사용해서 State Value를 구하더라도 이를 이용하여 Improvement가 불가능하다. 따라서 이 대신 Action Value를 MC방법으로 구한 후 Greedy하게 Improvement한다.

2-2. Exploration (Policy Improvement 측면)

#. Policy Improvement에서 Action-Value를 바탕으로 Greedy로만 Action을 선택하게 되면, Exploration이 부족해지는 문제가 발생한다. 이를 해결하기 위해 아래와 같은 식으로 ϵ 의 확률로 Random하게, $1 - \epsilon$ 의 확률로 Greedy하게 Action을 선택하도록 한다. 더 나아가 MC는 하나의 Episode만 있어도 Update가 가능하기에 좀 더 효율적인 Iteration을 위해 한 Episode가 끝나는 대로 Policy Evaluation을 진행한다. 즉 Asynchronous Backup으로 모든 State-Action Pair를 경험하지 않고 Update가 가능하다는 것이다. 그러면 위의 방법이 Monte-Carlo Control²⁾이 된다.

ϵ - Greedy Exploration	
$\pi(a s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \operatorname{argmax}_{a \in A} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$	
<p>Monte-Carlo Policy Iteration</p> <p>Starting Q, π</p> <p>$Q = q_\pi$</p> <p>$\pi = \epsilon\text{-greedy}(Q)$</p> <p>$Q^*, \pi^*$</p> <p>Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$</p> <p>Policy improvement ϵ-greedy policy improvement</p>	<p>Monte-Carlo Control</p> <p>Starting Q</p> <p>$Q = q_\pi$</p> <p>$\pi = \epsilon\text{-greedy}(Q)$</p> <p>$Q^*, \pi^*$</p> <p>Every episode:</p> <p>Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$</p> <p>Policy improvement ϵ-greedy policy improvement</p>

a. ϵ - Greedy Exploration에서 Action의 개수는 m 개다.

b. Policy Improvement에서 ϵ - Greedy를 해도 Improve가 되는지에 대한 증명은 아래와 같다. 이 증명은 3장에서 Greedy로 하여도 Improve가 되는지에 대한 증명과 동일하게 이루어진다. 그리고 'Policy Improvement Theorem'에 의해 한 Step에서 Improvement를 보이면 그 이후 여러 단계에서도 Improvement가 가능하다.

Theorem
<ul style="list-style-type: none"> For any ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$ $ \begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in A} \pi'(a s) q_\pi(s, a) \\ &= \epsilon/m \sum_{a \in A} q_\pi(s, a) + (1 - \epsilon) \max_{a \in A} q_\pi(s, a) \\ &\geq \epsilon/m \sum_{a \in A} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in A} \frac{\pi(a s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\ &= \sum_{a \in A} \pi(a s) q_\pi(s, a) = v_\pi(s) \end{aligned} $ <p>\therefore Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$</p>

c. 위의 증명에서, $q_\pi(s, \pi'(s))$ 는 현재의 선택은 π' 로 하고, 그 다음 부터는 π 을 사용한다는 의미이다. 그리고 부등호가 생기는 이유는 가중치의 합보다 \max 값이 항상 크기 때문이다.

2) 아직 완전한 것은 아니지만, 곧 다음에 나올 GLIE의 2가지 속성을 만족해야 한다.

2-3. GLIE and Monte-Carlo Control

#. 2-2에서 언급한 MC Control이 잘 수렴하기 위해서는 Greedy in the Limit with Infinite Exploration³⁾의 2가지 속성이 만족되어야 한다. 그 속성은 아래와 같으며, 이를 만족시키는 GLIE Monte-Carlo Control을 식으로 표현하면 아래와 같다.

조건 1	$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$	학습이 진행될수록 모든 State-Action Pair가 무한히 반복되어야 한다. -> Exploration
조건 2	$\lim_{k \rightarrow \infty} \pi_k(a s) = 1 (a = \operatorname{argmax}_{a' \in A} Q_k(s, a'))$	학습이 충분히 진행되었다면, ϵ 값은 감소 ⁴⁾ 하고, 최종적으로는 Greedy Policy가 되어야 한다. -> Exploitation

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

k -th Episode는 $\pi : S_1, A_1, R_2, \dots, S_T \sim \pi$ 에서 Sampling할 수 있다.

한 Episode 단위로, 아래와 같은 Evaluation, Improvement과정을 수행한다.

Policy Evaluation

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

Policy Improvement

$$\epsilon \leftarrow 1/k$$

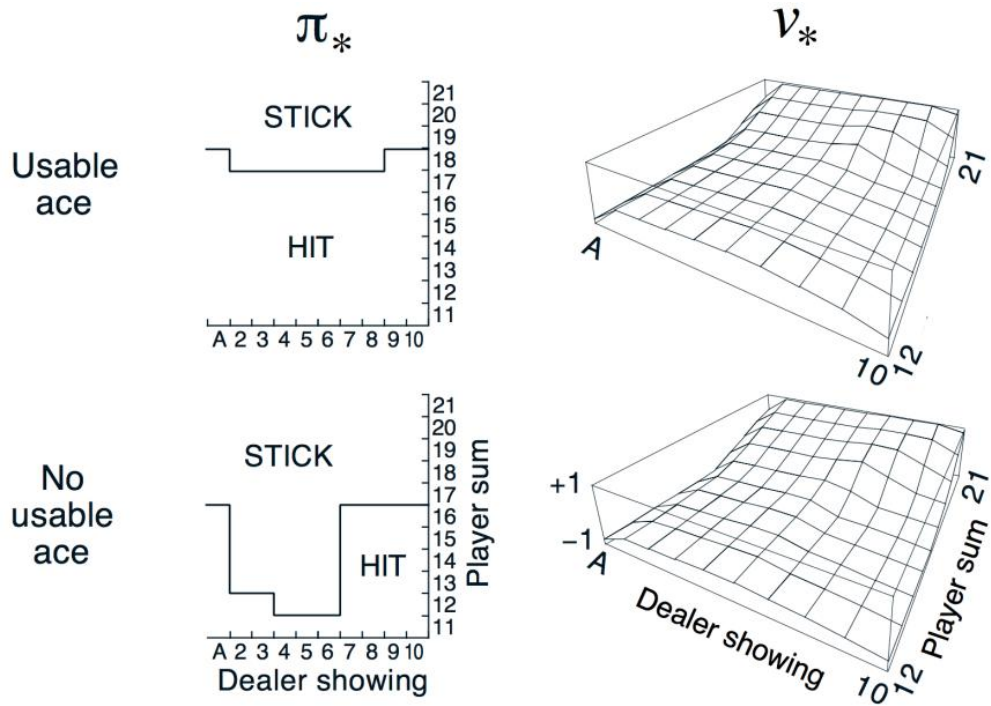
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

3) 무한한 Exploration에 제한을 걸고 Greedy를 해야 한다는 의미이다.

4) $\epsilon_k = \frac{1}{k}$ 로 설정한다.

2-4. Blackjack Example

- a. 4장에서 보았던 Blackjack Example은 Prediction이었으므로 Optimal Value Function을 구했지만, 여기에서는 Optimal Policy를 도출할 것이다. 그 결과는 아래와 같다.



3. On-Policy Temporal-Difference Learning Control

3-1. Introduction

#. MC와 TD를 비교했을 때처럼, TD는 MC보다 Variance가 낮으며, Online-Learning이 가능하므로, Terminal State가 없는 Episode에서도 학습이 가능하다는 장점이 있었다. 따라서 Model-Free Policy Iteration 과정에서 Policy Evaluation(=Prediction)할 때, MC 대신 TD⁵⁾로 대체할 수 있는데, 이것이 Sarsa Model이다.

3-2. Sarsa

3-2-1. Sarsa

#. Model-Free Policy Iteration 중 Policy Evaluation(=Prediction)에서 아래와 같이 TD-Learning을 사용하는 방법을 Sarsa라고 한다. 따라서 Monte-Carlo Control에서 한 Episode마다 Policy Evaluation을 했다면, 이제는 한 Step마다 Policy Evaluation을 진행하며 Q-table⁶⁾ 또는 Function Approximator의 Weight 값을 Update하는 것이다.

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

$R + \gamma Q(S', A') : TD-Target$

$(R + \gamma Q(S', A') - Q(S, A)) : TD-Error$

$\alpha : \text{얼만큼 update 할 것인지}$

Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ -greedy policy improvement

Sarsa Algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize S

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Repeat (for each step of episode):

Take action A , observe R, S'

Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

until S is terminal

5) MC와 TD는 Model-Free Prediction(=Evaluation)의 대표적인 기법이다.

6) Look up Table이며 (# of state \times # of actions per state) size이다.

3-2-2. Convergence of Sarsa

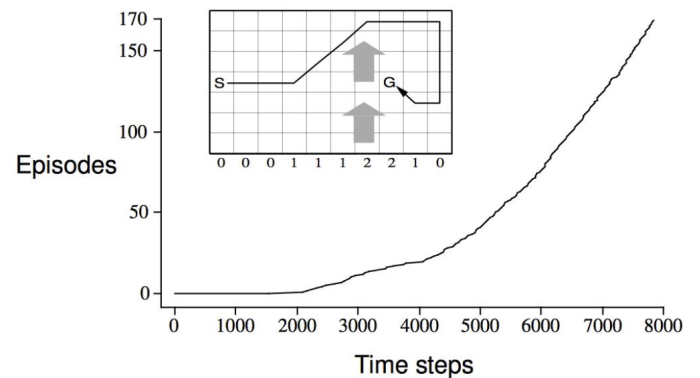
#. Action Value인 Q-Value가 Sarsa를 통해 Optimal Action Value Function($q_*(s, a)$)로 수렴하기 위해서, 이론적으로는 아래의 2가지 조건이 필요하지만, 실제적으로는 두 조건들을 신경 쓰지 않아도 수렴함이 보장된다.

조건 1	GLIE	<ul style="list-style-type: none"> 결국엔 모든 State-Action Pair를 방문해야 한다. 결국엔 ϵ 값이 줄어들어서 Policy가 Greedy Policy로 되어야 한다.
조건 2	Robbins-Monro Sequence	<ul style="list-style-type: none"> $\sum_{t=1}^{\infty} \alpha_t = \infty$: Step-Size가 Q값을 먼 곳으로 이끌 수 있도록 설정. $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$: Q값을 수정하는 것이 점점 작아진다. 즉 수렴한다.

3-2-3. Windy world Example

a. 아래의 MDP에서 Sarsa를 사용한 예시와 그 결과이다. State에서는 바람이 불어서 숫자가 적힌 해당 열에 위치하면, 그 값만큼 위로 올라간다.

State(S)	
Action(A)	<div>standard moves king's moves</div>
Reward(R)	한 time-step당 -1
Transition Probability(P)	1 (deterministic)
Discount Factor(γ)	None



- b. 위의 결과는 King's Moves를 사용한 결과이며, 처음 성공하기 까지 2000번의 Time steps가 필요하다. 하지만 성공 이후에는 기울기가 급격하게 증가하여 성공하는 Episode가 기하급수적으로 늘어난다.
- c. 성공하게 되면 Goal인 지점에서부터 주변 State로 역으로 정보가 전파되기 시작하여 이전보다 적은 Time-step으로도 Goal에 도착할 수 있기 때문에 이렇듯 기하급수적으로 증가하는 것이다.
- d. 구체적으로 한번 성공하였을 때, 70×8 크기의 Q-table에서 Goal State에 도달 직전의 State의 각 Column 값(Q-value)이 Update되고, 다음 Time-Step에서는 그 State의 직전 State의 Q-Value 값이 Update되는 방식이다.

3-3. Sarsa(λ)

3-3-1. n-step Sarsa

#. n -Step TD와 동일한 방식으로 Bootstrapping을 진행하는데, n -step까지는 직접 가서 Reward를 얻고 그 이후부터는 Bootstrapping을 진행하는 Sarsa이다.

n -step Q-Return
$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$
n -step Sarsa Update
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)$

3-3-2. Forward View Sarsa(λ)⁸⁾

#. MC와 동일하게 한 Episode내에서, Terminal State까지 가면서 모든 Step에 대한 Q-Return에 Weighted-Sum을 하여 이를 TD-Target으로 설정한 TD(λ)와 동일한 Sarsa이다. 하지만 Terminal State가 존재해야하고, 이에 도달해야 Update가 가능한 단점이 있었다.

TD-Target	Forward-view Sarsa(λ)
$q_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$	$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (q_t^\lambda - Q(S_t, A_t))$
<p style="text-align: center;">Sarsa(λ)</p> <p>The diagram shows a sequence of states and actions over time steps $t, t+1, \dots, T-1$. At each step, a state s and action a are input to a function block. The output is a weighted sum of future rewards, with weights $(1-\lambda), (1-\lambda)\lambda, (1-\lambda)\lambda^2, \dots$. The total sum is labeled $\Sigma = 1$. The final state is labeled s_T.</p>	

3-3-3. Backward View Sarsa(λ)

#. Forward View의 TD-Error 부분에 각 State-Action Pair마다 책임을 묻는 Eligibility trace값⁹⁾을 곱하여 책임 사유와 비례하여 Update되도록 하는 Backward View TD(λ)와 비슷하지만 State뿐만 아니라 State-Action Pair를 고려한다는 점에서 차이가 있다. Forward View와는 다르게 Terminal State가 없어도 Update가 가능하며, 한 Step마다 모든 칸을 Update하여 계산량은 많지만, 정보의 전파는 빠르다.

Eligibility Trace	Backward View Sarsa(λ)
$E_0(s, a) = 0$	$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$
$E_t(s, a) = \gamma\lambda E_{t-1}(s, a) + \mathbf{1}(S_t = s, A_t = a)$	$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_t E_t(s, a)$

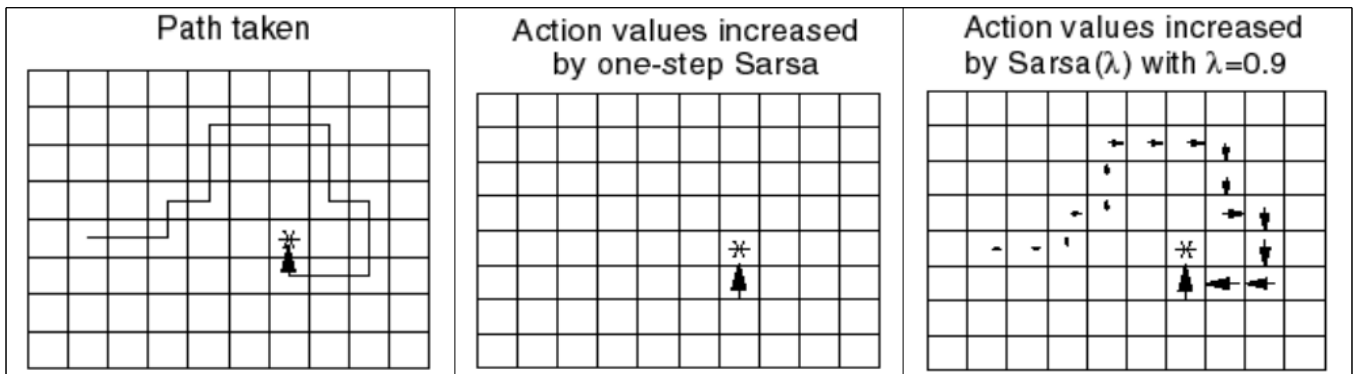
Sarsa(λ) Algorithm
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ Repeat (for each episode): $E(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ Initialize S, A Repeat (for each step of episode): Take action A , observe R, S' Choose A' from S' using policy derived from Q (e.g., ϵ -greedy) $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$ $E(S, A) \leftarrow E(S, A) + 1$ For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$: $Q(s, a) \leftarrow Q(s, a) + \alpha\delta E(s, a)$ $E(s, a) \leftarrow \gamma\lambda E(s, a)$ $S \leftarrow S'; A \leftarrow A'$ until S is terminal

a. Sarsa Algorithm과는 다르게 한 Step 이후에 지나온 모든 State, Action에 대해 Eligibility Trace값을 구해 준 후 Update해준다. 따라서 위와 같이 Loop가 존재하며, 계산량은 많지만 정보 전파는 기존 Sarsa보다 빠르다.

b. 빨간 박스 부분의 구현 방법은 Tabular Method를 사용하여 한 번에 해당 연산을 적용해주면 된다.

3-3-4. Grid-World Example

- a. Episode가 끝났을 때, One-Step Sarsa와 Sarsa(λ)의 Action-Value가 Update되는 방식의 차이를 살펴보면, 아래와 같다. 즉 One-Step Sarsa는 Goal직전의 State만 Update되는 반면, Sarsa(λ)는 Agent가 지나왔던 모든 경로에 대해 Eligibility Trace의 값에 따라 Update가 진행된다.



- b. Sarsa(λ)의 Action Value가 Update되는 것을 보면, 지나온 경로에서 Eligibility Trace에 따라 Frequency는 모두 동일하지만, Recency가 다르기 때문에 화살표의 굵기에 차이가 있다.

4. Off-Policy Learning

4-1. Introduction

4-1-1. Backgrounds

- a. 확률분포의 의미 : x 축은 발생 가능한 사건들, y 축은 각 사건이 발생할 확률로 설정하고 이를 함수로 표현한 것이다. 따라서 어떤 확률분포에서 어떤 사건을 Sampling을 한다는 것은 그 사건의 발생 확률과 비례하여 그 사건을 택한다는 것이다.
- b. Policy의 의미 : Policy는 State가 주어졌을 때의 Action에 대한 확률분포($\pi(a|s)$)이므로, 각 State마다 따로 존재하며, x 축에는 Actions, y 축에는 각 Action을 취할 확률로 표기된다.
- c. Behaviour Policy(μ)와 Target Policy(π) :

Target Policy(π)	Optimal Policy를 찾기 위해 Evaluation and Improvement가 되는 Policy이다.
Behaviour Policy(μ)	Exploration을 위한 Policy이며 Behavior를 Generation하는 Policy이다.

4-1-2. Off-Policy Learning의 의미와 특징

- #. Off-Policy Learning은 On-policy와 다르게 $\pi \neq \mu$ 인 상황이며 μ^{10} 는 누군가 이미 했던 것(Data-Set)인데 이를 따라 Action Selection을 하고 Data를 생성한 후 State Value($v(s)$)나 Action Value($q(s, a)$)를 구하는 것이 목적이다. 하지만 π 가 학습될수록 μ 에 의한 Action Selection의 빈도가 줄어들도록 해야 한다.
- a. 하지만 Supervised Learning과는 다르게 타산지석을 통해 더 나은 것을 학습하는 것이다.
- b. Simulation이 불가능하지만 기존의 수집된 Data-Set이 있으며 이를 기반으로 학습해야 할 때 사용할 수 있다.
- c. On-policy는 한 번 경험 후에 그 경험을 버렸다. 왜냐하면 같은 Policy를 Update하기 때문이다. 하지만 Off-Policy는 π, μ 을 재사용할 수 있다.
- d. 하나의 Policy(μ)를 따르면서 Multiple policy를 학습할 수 있으므로, Optimal Policy를 학습하면서 Exploration이 가능하다. 이는 Exploration과 Trade-off 관계에 있는 Exploitation을 고려했을 때 중요한 특징이기도 하다.
- e. Off-Policy는 두 Policy가 서로 관련이 없으므로 Target Policy는 Deterministic(=Greedy)하지만, Behavior Policy는 Uniform Random Policy와 같이 모든 가능한 Action을 Selection하는 Policy가 될 수 있다.
- f. Off-Policy Methods는 Variance가 높고 수렴속도가 늦지만 일반적으로 강력한 성능을 발휘하므로 다양한 활용 가능성이 있다.
- g. 이러한 Off-Policy Learning이 가능한 방법론은 크게 Importance Sampling과 Q-learning이 있으며 각각 MC와 TD의 방법론을 구체적으로 배운다.

¹⁰⁾ $\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$

4.2 Importance Sampling in MC and TD

4-2-1. Importance Sampling

#. 어떤 확률분포($P(X)$)의 기댓값을 구하고 싶지만 직접적으로 구할 수 없을 때, 하지만 해당 분포의 각 사건의 발생 확률 값을 알 수 있을 때, 이와 비슷한 다른 확률 분포($Q(X)$)를 사용하여 Sampling하고 Importance Sampling Ratio를 이용하여 아래와 같은 방법으로 $P(X)$ 의 기댓값으로 변환하는 방법이다.

$$\begin{aligned}\mathbb{E}_{X \sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X) \frac{P(X)}{Q(X)} f(X) \\ &= \mathbb{E}_{X \sim Q} \left[\frac{P(X)}{Q(X)} f(X) \right]\end{aligned}$$

a. 이해의 편의를 위해 아래와 같이 특정 State s 가 주어진 상황에서의 Table이 있다고 하자. 즉 s 에서 얻을 수 있는 Return의 가짓수는 5가지이다.

$G_t(= X)^{11)}$	3	21	10	14	11
Probability of the Subsequent State-Action(= $P(X)$) by Target Policy($\pi(a s)$)	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
Probability of the Subsequent State-Action(= $Q(X)$) by Behavior Policy($b(a s)$)	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$
Importance Sampling Ratio(= $\frac{P(X)}{Q(X)}$)	$\frac{3}{5}$	$\frac{6}{5}$	$\frac{6}{5}$	$\frac{6}{5}$	$\frac{6}{5}$
Arbitrary Function	$f(x) = x$				

Probability of the Subsequent State-Action ¹²⁾
$\begin{aligned}\Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ &= \pi(A_t S_t)p(S_{t+1} S_t, A_t) \underbrace{\pi(A_{t+1} S_{t+1})}_{\text{by Policy}} \cdots \underbrace{p(S_T S_{T-1}, A_{T-1})}_{\text{by Env.}} \\ &= \prod_{k=t}^{T-1} \pi(A_k S_k)p(S_{k+1} S_k, A_k),\end{aligned}$
Importance Sampling Ratio
$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k S_k)p(S_{k+1} S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k S_k)p(S_{k+1} S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k S_k)}{b(A_k S_k)}.$

11) Monte-Carlo의 경우엔 G_t 이므로 \prod 연산이 필요하지만, TD의 경우엔 한 Step만 보기 때문에 해당 연산은 필요 없다.

12) 위의 Table에서 확률 값 하나가 Policy와 State Transition Probability 값을 연속적으로 곱하여 나온 결과이다.

- b. 위의 Table에서 Sampling하여 State Value Function($v(s) = E[G_t | S_t = s]$)을 구해보자. 즉 Behavior Policy($b(a|s)$)를 사용하여 Sampling을 하고 State Value Function을 구한 후 이를 Target Policy($\pi(a|s)$)를 이용해 구한 것으로 변환해보자.

3번의 Episode후 얻은 Returns = (21, 3, 10)	
Target Policy $v_\pi(s)$	$21 \times \frac{1}{5} + 3 \times \frac{1}{5} + 10 \times \frac{1}{5} = \frac{34}{5}$
Prediction via Importance Sampling $E[\rho_{t:T-1} G_t S_t = s]$	$\frac{1}{3} \times \frac{3}{5} \times 21 + \frac{1}{6} \times \frac{6}{5} \times 3 + \frac{1}{6} \times \frac{6}{5} \times 10 = \frac{34}{5}$
$\therefore E[\rho_{t:T-1} G_t S_t = s] = v_\pi(s)$	

- c. 위의 결과에서 알 수 있듯이 Importance Sampling Ratio가 Behavior Policy를 통해 구한 Return G_t 을 Target Policy를 통해 구한 Return으로 변환해준다.

4-2-2. Importance Sampling for Off-Policy MC

- #. MC에 따라서, Behavior Policy μ 을 따라 한 Episode를 마쳤을 때, 얻은 Return G_t 에 $\frac{\pi(A_T | S_T)}{\mu(A_T | S_T)}$, $P(a|s) = \pi(a|s)$, $Q(a|s) = \mu(a|s)$ 을 G_t 을 얻을 때 까지 Selection한 Action의 개수만큼 곱해주고(=Importance Sampling Ratio) 아래와 같이 Update해주면 Target Policy $\pi(a|s)$ 을 따랐을 때의 Return을 도출할 수 있다.

$$G_t^{\pi/\mu} = \frac{\pi(A_t | S_t)}{\mu(A_t | S_t)} \frac{\pi(A_{t+1} | S_{t+1})}{\mu(A_{t+1} | S_{t+1})} \cdots \frac{\pi(A_T | S_T)}{\mu(A_T | S_T)} G_t$$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{\pi/\mu} - V(S_t) \right)$$

- a. Off-Policy Method에선 거의 대부분 Importance Sampling을 사용한다.
- b. 하지만 Importance Sampling Correction Term의 Variance가 매우 크고 수렴이 느린 문제점이 있다. 왜냐하면 $0 \leq \frac{\pi(a|s)}{\mu(a|s)} \leq 1$ 인 상황에서 한 Episode의 Action이 많아지면 0으로 $G_t^{\pi/\mu}$ 의 값이 0으로 수렴하기 때문이다. 또한 $\pi(a|s) = 0$ 이면 $G_t^{\pi/\mu} = 0$ 인 문제 또한 발생하기 때문이다.

4-2-3. Importance Sampling for Off-Policy TD

- #. 위의 방법에서 MC대신 TD를 사용하는 방법이다. 즉 한 Step인 하나의 Action에 대해서만 Behavior Policy μ 을 따랐을 때의 TD-target에 Importance Sampling Ratio을 곱해주면 된다. MC와 다른 점은 Probability of the Subsequent State-Action의 길이가 1이므로 $\frac{\pi(a|s)}{\mu(a|s)}$ 을 TD-Target에 한 번만 곱해주면 된다.

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right)$$

4-3. Q-Learning

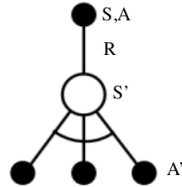
4-3-1. General Q-Learning

- #. TD-Learning¹³⁾에서 State Value 대신 Action Value를 Bootstrapping하는데, 먼저 현 시점(t)에서는 Behavior Policy(μ)를 따라 실제 Action을 취하고, 다음 시점($t+1$)에서의 Action Value는 우리가 학습하고자 하는 Target Policy(π)로 추측하는 방식이다. 즉 S_t 에서 μ 에 따라 A_t 을 했을 때의 Action Value는 A_t 을 따라 S_{t+1} 로 갔을 때의 π 을 따른 A' 의 Action Value를 이용하여 구한다는 것이다.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

- 예를 들어 현 시점에서 오른쪽으로 이동하는 Action Value를 구하려면, 실제 오른쪽으로 가고 난 후 다음 State에서 μ 가 위로 가는 것, π 가 아래로 가는 것이라 한다면, 아래로 가는 방향으로 오른쪽으로 가는 Action Value를 Update하는 것이다.
- 아직까진 Tabular Method를 사용하고 있으므로 $Q(s, a)$ 는 (# of States \times # of Actions Per State) 크기의 Q-Table로 표현된다. 하지만 State Space가 Continuous이거나 많아지면 Function Approximation을 사용하여 $Q(s, a)$ 을 나타낸다. 예를 들어 Linear Function Approximation은 $\hat{Q}(s, a) = \theta^T \phi_{s, a}$, Neural Network은 $\hat{Q}(s, a) = f_n(\phi_{s, a})$ 이 되고 $\phi_{s, a}$ 은 Action-State Feature Vector이다.

#. Behavior, Target Policy(μ, π) 모두 Improve 시키기 위한 방법으로 General Q-Learning에서 Target Policy π 는 Greedy하게 정하여 Update하는 방식이고, Behavior Policy μ 는 ϵ -greedy하게 정하여 학습초반에는 Action을 Selection을 주로 하여 Exploration을 도모하고 Target Policy가 어느정도 학습이 되었으면 Behavior Policy에 의한 Action Selection을 줄여나가도록 하는 방법이다. (아래 그림에서 호는 max 연산을 의미)



$$\begin{aligned}
 Q(S, A) &\leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right) \\
 A' &= \pi(S_{t+1}) = \operatorname{argmax}_{a'} Q(S_{t+1}, a') \\
 &\Rightarrow R_{t+1} + \gamma Q(S_{t+1}, A') \\
 &= R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax}_{a'} Q(S_{t+1}, a')) \\
 &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')
 \end{aligned}$$

Theorem :

Q-Learning control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
 Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ -greedy)

Take action A , observe R, S'

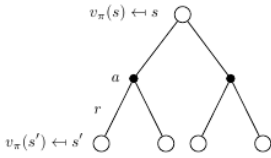
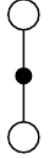
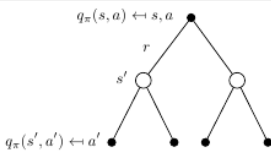
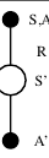
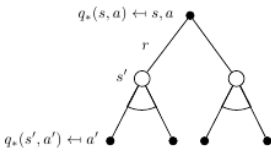
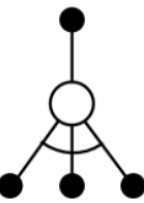
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$;

until S is terminal

5. Summary

5-1. Relationship Between DP and TD

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $v_{\pi}(s)$	 <p>Iterative Policy Evaluation</p>	 <p>TD Learning</p>
Bellman Expectation Equation for $q_{\pi}(s, a)$	 <p>Q-Policy Iteration</p>	 <p>Sarsa</p>
Bellman Optimality Equation for $q_{*}(s, a)$	 <p>Q-Value Iteration</p>	 <p>Q-Learning</p>
Full Backup (DP)		Sample Backup (TD)
Iterative Policy Evaluation		TD Learning
$V(s) \leftarrow \mathbb{E} [R + \gamma V(S') \mid s]$		$V(S) \stackrel{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration		Sarsa
$Q(s, a) \leftarrow \mathbb{E} [R + \gamma Q(S', A') \mid s, a]$		$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma Q(S', A')$
Q-Value Iteration		Q-Learning
$Q(s, a) \leftarrow \mathbb{E} \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a \right]$		$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$