Chapter 9. Exploration and Exploitation

1. Introduction

1-1. Exploration vs. Exploitation Dilemma

#. Agent가 action을 취할 때 발생하는 근본적인 문제는 현재 알고 있는 정보를 바탕으로 가장 좋은 선택을 내리는 Exploitation과 새로운 정보를 모으는 Exploration과의 trade-off 적인 관계를 어떻게 꾸준하게 유지할 것인가이다.

따라서 이 단원에서는 둘 사이의 관계를 적절히 조절할 수 있는 action-value $\hat{Q}_t(a)$을 estimation하는 5가지의 algorithms를 소개한다.

1-2. The Multi-Armed Bandit

#. 분석을 쉽게 하기 위해 MDP에서 $\langle A, R \rangle$만 남은 multi-armed bandit(one step MDP)을 고려해보자. 이 환경은 m개의 slot machine이 있으며 각 machine의 reward분포($R^a(r) = P(r|a)$)는 각각 다르며 알려져 있지 않다. 이 환경에서의 action은 m개가 있으며 한 step(t)마다 하나의 machine을 당기는 것($a_t \in A$)이고 이 때 마다 환경은 reward($r_t \sim R^{a_t}$)를 준다. 이때의 목적은 cumulative reward($\sum_{\tau=1}^{t} r_\tau$)를 최대화하는 것이다.

a. machine 마다 reward 분포가 다르다는 것은 각 분포마다 variance, expectation value가 다르다는 것을 의미한다. 즉 꾸준히 일정 수준의 reward를 주는 기계가 있는가하면, 다양하게 reward를 주는 기계가 있다는 것이다.

2. Naive Exploration

2-1. (Total) Regret and Gap

#. Total Regret[1]은 Gap과 count의 함수로 아래와 같이 정의된다. count($N_t(a)$)는 action별로 몇 번을 시행했는지를 나타내고, Gap($\triangle_a = V^* - Q(a)$)은 각 machine별로 정의되어 있는데 가장 높은 reward 기댓값을 주는 machine의 그 값과 각 machine에서의 reward 기댓값을 뺀 것이다.

$$
\begin{aligned}
L_t &= \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right] \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right](V^* - Q(a)) \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right] \Delta_a
\end{aligned}
$$

a. 위의 식에서 $E[N_t(a)]$은 만약 총 N회의 다양한 action을 시행하였고, 특정 action a을 한 횟수가 m이면 m/N을 의미한다.

b. $Q(a) = E[r \mid a]$는 action-value이다. action-value는 원래 return의 기댓값인데, 여기에서는 one-step MDP이므로 action-value가 reward의 기댓값이다.

c. $V^* = Q(a^*) = \max_{a \in A} Q(a)$는 제일 좋은 machine을 당겼을 때의 reward 기댓값이다.

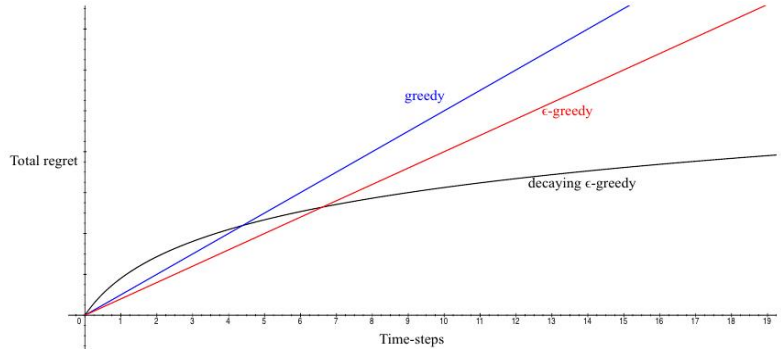
d. 한 step에 대한 regret은 $l_t = E[V^* - Q(a_t)]$으로 정의된다.

e. 하지만 V^*을 실질적으로 알 수 없기 때문에 이는 이론적인 접근방법에 불과하다.

f. 결국 우리의 목표인 cumulative reward의 최대화는 total regret을 최소화하는 것과 동일하다고 볼 수 있다. 하지만 왜 regret이라는 개념을 도입하는지에 대한 의문이 생긴다. regret은 어떤 algorithm이 가능한 가장 잘할 수 있는 한계에 대한 정보를 제공해주기 때문에 regret의 개념을 도입하는 것이다.

g. 아래의 그래프는 time-steps에 따른 regret의 변화 추이를 보여주는 것이다. 따라서 우리의 목적은 decaying ϵ-greedy처럼 sub-linear를 그리도록 하는 exploration과 exploitation을 조절하는 적절한 $\hat{Q}_t(a)$을 estimation하는 algorithm을 만드는 것이다.

1) opportunity loss라고도 한다.



2-2. Greedy and ϵ-greedy

2-2-1. Greedy Algorithm

#. action을 선택할 때, agent가 exploration한 정보를 바탕으로 $\hat{Q}_t(a) \approx Q(a)$을 추론한 후 이를 바탕으로 항상 높은 값을 갖는 action을 선택하는 algorithm이다. 하지만 항상 선택한 값이 좋을 것이라는 보장이 없기 때문에(sub-optimal) Optimal action-value와 gap이 항상 존재하여 linear하게 그려진다.

Estimation	Estimation by MC evaluation	Action selection
$\hat{Q}_t(a) \approx Q(a)$	$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{t=1}^{T} r_t 1(a_t = a)$	$a_t^* = \arg max_{a \in A} \hat{Q}_t(a)$

2-2-2. ϵ-greedy Algorithm

#. ϵ의 확률로 random한 action을 선택(exploration)하고 $1-\epsilon$의 확률로 exploitation한 action을 선택하는 algorithm이다. ($a_t^* = \arg max_{a \in A} \hat{Q}_t(a)$) 이 방법은 일정 확률 ϵ으로 random한 행동을 하기 때문에 그 만큼의 gap이 지속적으로 더해진다. 따라서 linear한 그래프가 그려지게 된다.

a. 일정 확률 ϵ으로 random한 행동을 하므로 $l_t \geq \frac{\epsilon}{A} \sum_{a \in A} \triangle_a$이 성립한다.

2-3. Decaying ϵ_t-greedy

#. ϵ-greedy Algorithm 바탕에서 어떤 schedule에 따라 ϵ을 계속 decay하는 방법이다. 즉 처음에는 높은 ϵ로 exploration을 하다가 정보가 쌓이면 이를 줄여나가는 방법이다. 결국 schedule을 정하는 것이 중요하며, reward에 대한 정보 없이 gap에 대한 정보만으로 이를 heuristic으로 정하여 sub-linear regret 함수를 갖도록 하는 것이 목적이다.

a. 아래의 scheduling 예시는 total regret이 logarithmic으로 수렴한다. 아래의 식에서 d는 1등 machine과 2등 machine과의 차이를 나타낸다.

$$c > 0,\ d = \min_{a | \triangle_a > 0} \triangle_i,\ \epsilon_t = \min\left\{1,\ \frac{c|A|}{d^2 t}\right\}$$

3. Optimistic Initialisation

#. 초기 action-value를 모두 높은 값으로 설정한 후에 greedy or ϵ-greedy로 action을 selection하고 incremental Monte-Carlo Evaluation을 사용하여 Q-value를 update하고 greedy or ϵ-greedy로 다시 action selection을 반복하는 algorithm이다. 때문에 초기에는 모든 높은 값이기 때문에 어느 정도 값까지는 exploration이 활발하게 이루어진다.
하지만 sub-optimal에 빠질 수 있으므로 greedy, ϵ-greedy 모두 linear total regret이 도출된다.

a. Incremental Monte-Carlo Evaluation은 아래와 같다.

$$\hat{Q}_t(a_t) = \hat{Q}_{t-1} + \frac{1}{N_t(a_t)}(r_t - \hat{Q}_{t-1})$$

b. Total regret 그래프가 linear하지만 실무에서도 충분히 좋은 방법론이다.

4-1. Lower-Bound

#. Exploration과 Exploitation을 조절하는 algorithm을 아무리 잘 만들어도 performance는 log함수가 lower bound이다. 이는 아래와 같이 Lai and Robbins의 정리로 증명되어 있다. 즉 어떤 algorithm의 performance인 total regret은 제일 좋은 machine의 reward 분포(R^{a^*})와 다른 machine의 reward 분포(R^a)의 차이 또는 gap에 의존한다는 것이다.

$$\lim_{t\to\infty} L_t \geq \log t \sum_{a|\triangle_a > 0} \frac{\triangle_a}{KL\left(R^a \| R^{a^*}\right)}$$

a. 즉 reward의 분포가 비슷할수록 난이도가 결정되는데, 그 이유는 예를 들어 실제 두 분포의 평균은 다른데, 비슷하기 때문에 이 둘에 대해 계속 해볼수록 틀리는 양이 쌓이기 때문이다.

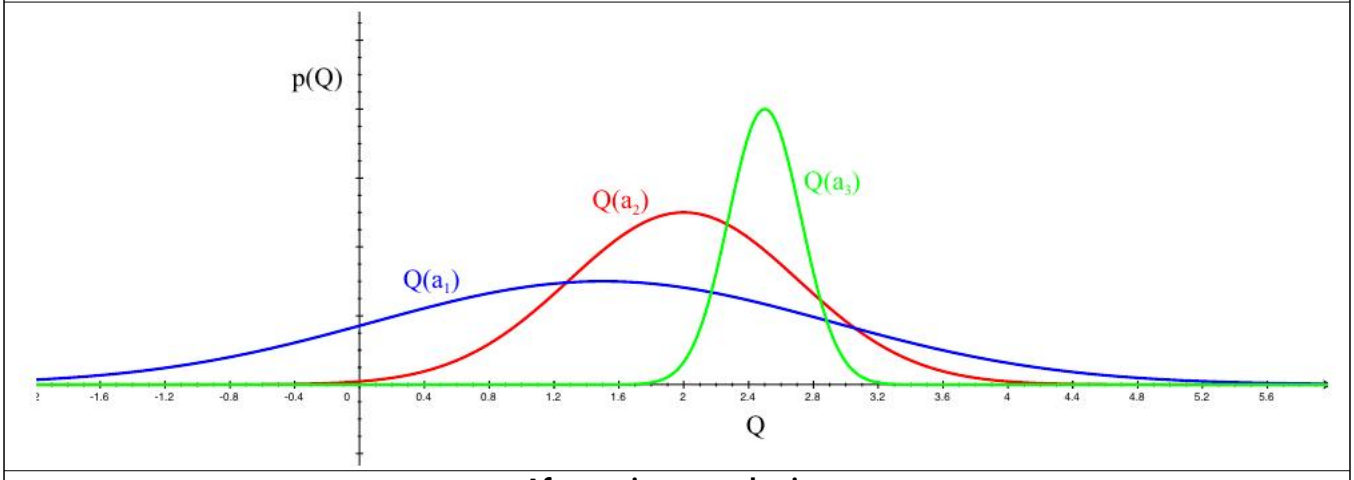
b. 위의 정리에서 t는 number of steps이고 $a|\triangle_a > 0$은 가장 좋은 machine을 제외한 다른 machine에 대한 것을 의미한다.

c. KL-Divergence($KL(R^a \| R^{a^*})$)는 두 분포간의 차이를 나타내는 지표로서 차이가 클수록 큰 값을 가지게 된다.

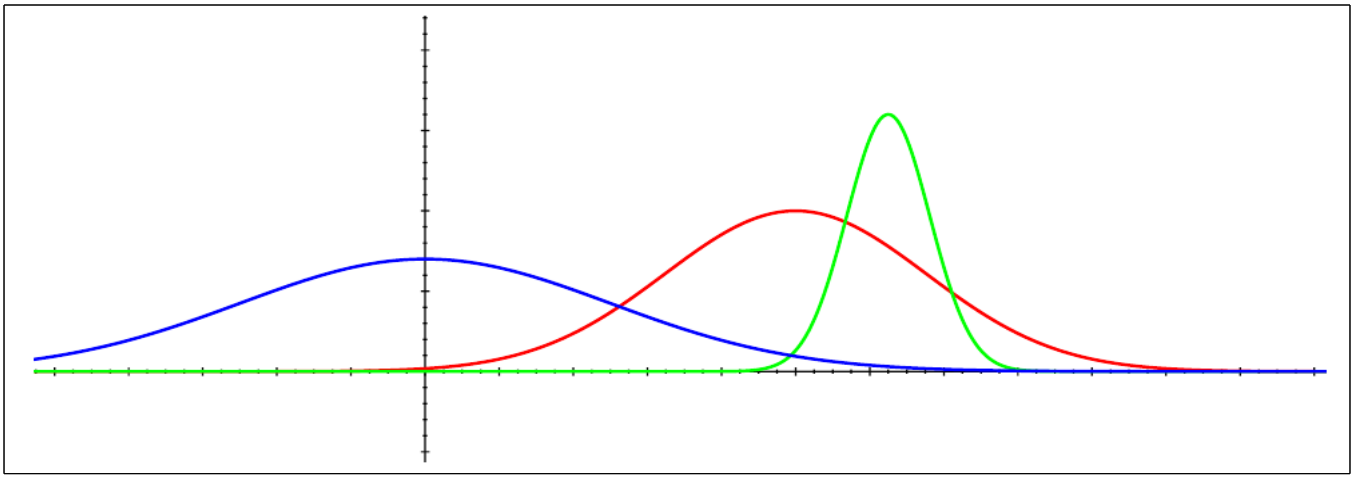
4-2. Optimism in the Face of Uncertainty

#. 어떤 action-value가 uncertain(high variance)일수록 exploration 대상의 가치가 있다는 관점이다. 즉 아래의 3개의 action-value 분포가 있다. 여기에서 기댓값이 가장 높은 $Q(a_3)$을 선택하는 것이 합리적이겠지만 신뢰구간(confidence interval)이 큰 $Q(a_1)$을 선택함으로써 신뢰구간이 줄어들어 a_1 action에 대한 불확실성이 감소함과 동시에 더 좋지 않은 action(Q-value의 기댓값이 왼쪽으로 이동=분포가 왼쪽으로 이동)이라는 것을 알게 되었다. 따라서 다음에는 다른 action을 선택할 가능성이 높아졌다.

Before action a_1 selection



After action a_1 selection



a. 어떤 분포의 신뢰구간은 분포가 유효한 값을 갖는 범위를 의미하며 이 구간이 클수록 uncertainty(=variance)가 크다.

b. 신뢰구간(confidence-interval)은 length의 의미를 내포하고, 신뢰(Upper/Lower confidence)는 수평선(x축)에서의 한 지점을 의미한다.

4-3. Upper Confidence-Bound

#. 고등학교 수학 통계시간에서 다룬 신뢰구간과 동일한 내용이다. 즉 모평균을 $Q(a)$, 표본평균을 $\hat{Q}_t(a)$이라 하였을 때, 95%의 신뢰구간을 사용하여 모평균 $Q(a)$을 추론할 때, $\hat{Q}(a) \pm 1.96\frac{s}{\sqrt{n}}$으로 정의하였다. 여기에서 오른쪽 부분을 고려하면 $Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a)$이 된다. 이 때 아래와 같이 UCB가 최대화되는 action을 선택해 주면 4-2와 같이 exploration이 이루어지게 된다.

$$a_t = \arg max_{a \in A} \hat{Q}_t(a) + \hat{U}_t(a)$$

a. 위의 식에서 action a을 선택한 횟수 $N(a)$의 크기에 따라 달라지는 특징이 있다.

Small $N_t(a)$ ⇒ Large $\hat{U}_t(a)$	Large $N_t(a)$ ⇒ Small $\hat{U}_t(a)$
• Estimated Value is Uncertain	• Estimated Value is Accurate

b. 신뢰구간이 길수록 자신감이 떨어지는 것을 의미한다.

c. UCB는 idea단계였고 이를 구체적으로 구현하기 위한 방법론은 Frequentist View와 Bayesian Approach가 있다.

4-3-1. Frequentist View : UCB1

#. 모분포를 아예 몰라도 사용할 수 있는 방법으로 4-3에서 $\hat{U}_t(a)$의 값을 Hoeffding's Inequality를 사용하여 구하고 $a_t = \mathrm{argmax}_{a \in A} \hat{Q}_t(a) + \hat{U}_t(a)$으로 action을 selection하는 algorithm이다.

Hoeffding's Inequality
Let $X_1, ..., X_t$ be $i.i.d.$ random variables in $[0,1]$, and let $\overline{X}_t = \frac{1}{\tau}\sum_{\tau=1}^{t} X_\tau$ be the sample mean. Then $$P[E[X] > \overline{X}_t + u] \leq e^{-2tu^2}$$
$$P[Q(a) > \hat{Q}_t(a) + U_t(a)] \leq e^{-2N_t(a)U_t(a)^2}$$

위의 식에서 아래와 같이 $\hat{U}_t(a)$값을 구할 수 있다.

$$e^{-2N_t(a)U_t(a)^2} = p$$

$$U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

reward를 얻으면 얻을수록 p값은 감소하므로 대략 $p = t^{-4}$로 설정하고 이를 대입하면, 아래와 같으며 UCB1 algorithm은 완성된다.

$U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}}$	$a_t = \underset{a \in \mathcal{A}}{\mathrm{argmax}}\ Q(a) + \sqrt{\frac{2\log t}{N_t(a)}}$

a. UCB와 함께 사용할 수 있는 부등식[2]은 많지만 이 중 Hoeffding's Inequality를 사용한 것이며, 이 부등식은 분포의 종류와 관계없이 항상 성립하는 부등식이다.

b. 현재 가정한 환경에서 action-value는 reward의 기댓값 이므로 우리의 분포는 reward분포이다. 따라서 Hoeffding's Inequality의 내용과 같이 $Q(a)$는 reward 기댓값의 true-value이고 $\hat{Q}_t(a)$는 sample reward mean이다. 그리고 $\hat{U}_t(a)$은 틀리는 정도를 의미한다. 즉 true-value값이 sample reward mean보다 $\hat{U}_t(a)$ 만큼 틀릴 확률이 어떤 값 이하라는 의미이다.

c. 위의 algorithm을 사용하면 결국 exploration과 exploitation을 둘 다 잘하게 되며 아래와 같이 total-regret은 logarithmic이다.

$$\lim_{t \to \infty} L_t \leq 8\log t \sum_{a \mid \Delta_a > 0} \Delta_a$$

2) Berstein 부등식, Azuma 부등식 등이 있음.

d. 아래의 그래프는 ϵ-greedy와 UCB를 비교하였는데 하나는 'best machine playes' 관점에서, 하나는 'regret'의 관점에서 측정한 결과를 보여주고 있다. 이 그래프에서 알 수 있는 것은 UCB는 체계적으로 성능이 좋게 나오고 있지만 ϵ-greedy는 간단하고 UCB보다 비슷하지만 좀 더 높은 성능을 보이지만 tuning을 잘못하면 큰 재앙이 될 수도 있다.

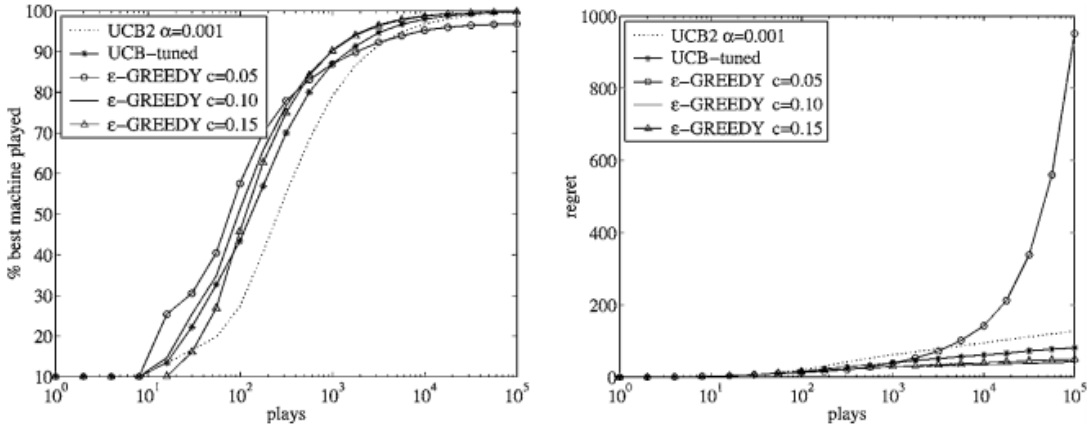


Figure 9. Comparison on distribution 11 (10 machines with parameters 0.9, 0.6, . . . , 0.6).

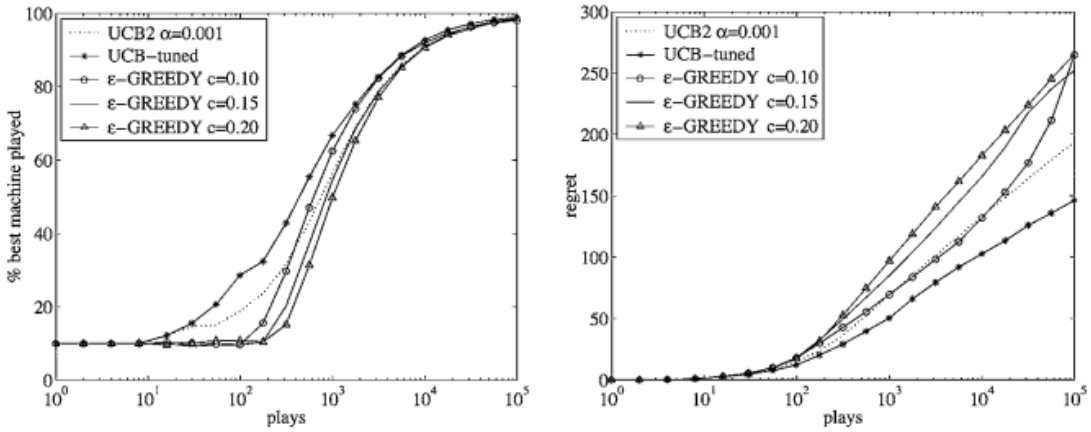


Figure 10. Comparison on distribution 12 (10 machines with parameters 0.9, 0.8, 0.8, 0.8, 0.7, 0.7, 0.7, 0.6, 0.6, 0.6).

e. UCB가 ϵ-greedy와 비슷한 성능이 나오는 이유는 Hoeffding's Inequality의 가정이 하나 밖에[3] 없기 때문에 약하기 때문이다. 이는 반대로 말하면 Random Variable이 $[0,1]$로 clipping 되어 있다면 Hoeffding's Inequality를 항상 사용할 수 있다는 의미이기도 하다. 따라서 성능을 높이기 위해선 좀 더 빡빡한 가정들이 필요할 것이다.

3) Random Variable은 $[0,1]$이어야 한다.

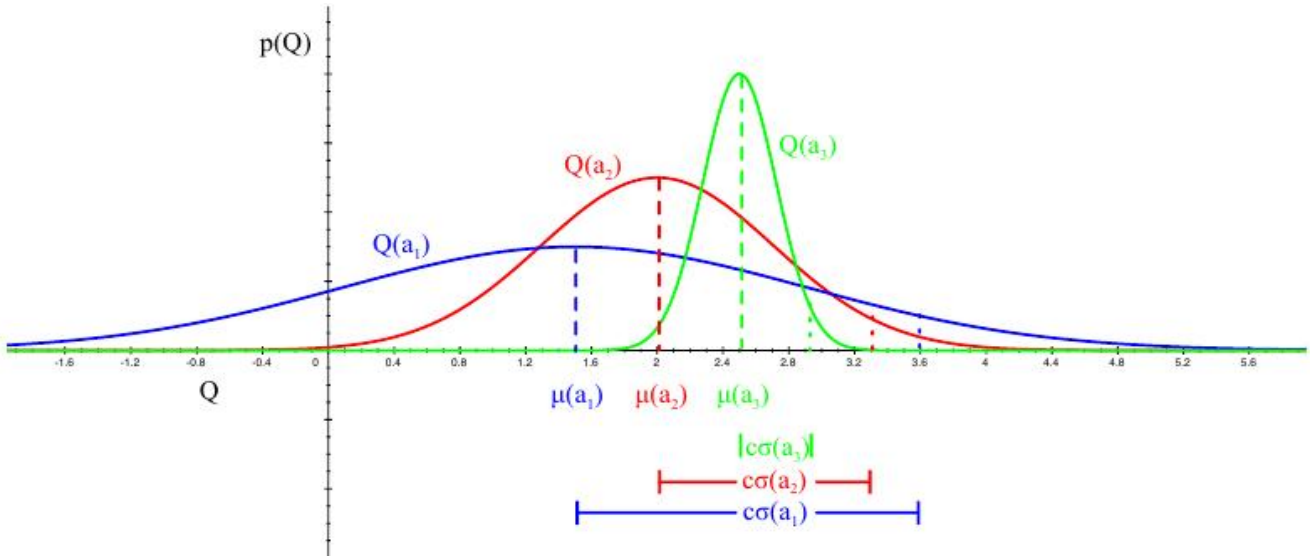
4-3-2. Bayesian View : Bayesian Bandit

#. reward distribution의 prior $p[R]$을 주어져있다 가정하고 실제 당겨서 나온 rewards(history)를 이용하여 posterior($p[R|h_t]$)을 계산한다. 계산된 posterior를 가지고 UCB(Bayesian UCB) 또는 Thompson Sampling algorithm을 사용하여 exploration을 guide하고 다시 posterior를 계산하는 방법(loop)이다.

a. prior 분포의 정확도에 따라 performance가 영향을 많이 받는다.

b. Bayesian UCB의 Example은 아래와 같다. 각 slot machine의 reward 분포는 독립적이며 Gaussian 분포이다.

Reward Distributions for each Machines



$$\mathcal{R}_a(r) = \mathcal{N}(r; \mu_a, \sigma_a^2)$$

Compute Gaussian Posterior by Bayes Law

$$p\left[\mu_a, \sigma_a^2 \mid h_t\right] \propto p\left[\mu_a, \sigma_a^2\right] \prod_{t \mid a_t = a} \mathcal{N}(r_t; \mu_a, \sigma_a^2)$$

Select action that maximises standard deviation of $Q(a)$ (Exploration)

$$a_t = \text{argmax}_{a \in A}\, \mu_a + \frac{c\sigma_a}{\sqrt{N(a)}}$$

5. Probability Matching

#. 확률적으로 optimal action인 것 같은 a을 선택하는 방법론이며 아래와 같이 표현할 수 있다. 즉 history[4] 가 주어진 상태에서 이를 바탕으로 확률적으로 optimal action을 선택하는 방법인데 Thompson Sampling 과 같은 방법으로 구체화 될 수 있다.

$$\pi(a|h_t) = P[Q(a) > Q(a'), \forall a' \neq a|h_t]$$

5-1. Thompson Sampling

#. Probability Matching을 구현하기 위한 방법 중 하나이며, Simulation 방법론과 같이 Reward Distribution 의 model로 posterior distribution $p[R|h_t]$을 설정하고 여기에서 sampling을 한 후 action-value를 계산하고 ($Q(a) = E[R_a]$) 이를 바탕으로 action을 selection($a_t = \arg max_{a \in A} Q(a)$) 한다. 그러면 실제 data를 얻었기 때문에 이를 바탕으로 posterior를 update한다. 즉 bayesian 방법론이다.

$$\pi(a|h_t) = P[Q(a) > Q(a'), \forall a' \neq a|h_t]$$

$$= E_{R|h_t}\left[1(a = \arg max_{a \in A} Q(a))\right]$$

a. Reward Distribution은 x축은 각 slot machine, y축은 각 slot machine의 expected reward $Q(a) = E[R_a]$ 로 구성된다.

b. posterior에서 sampling을 하는 것은 simulation을 하는 것이기 때문에 실제 data를 얻는 것이 아니며, 몇 번을 sampling을 하는지는 잘 모른다.

c. Bayesian 방법론을 사용하기 때문에 prior는 Oracle에 의해 주어져있으며 성능은 prior에 의해 좌우된다.

d. Bernoulli Bandit일 때는 이 방법이 Lai and Robbins lower bound로 수렴하는 것이 증명되어있으며 Multi Bandit인 경우에는, 최근 연구에 따르면, prior가 좋아야 성능이 좋다고 한다.

4) 직접 machine을 당겨보았던 기록을 의미한다.

6-1. Information State Space

#. One-step MDP State space에 history h_t을 요약해주는 information state $\tilde{s}=f(h_t)$을 더해준다. 그러면 state가 새롭게 정의되며 같은 state일지라도 어떤 action을 통해 information을 gain 했느냐에 따라 다르다.[5] 또한 action에 따라 information을 얻고 state가 변화하게 되기 때문에 state transition prob.을 $\tilde{P}^{a}_{\tilde{s},\tilde{s}'}$ 정의할 수 있다. 결국 아래와 같이 One step MDP에서 information을 추가함으로써 MDP가 만들어진다.

$$M=\langle A,R\rangle \Rightarrow \widetilde{M}=\langle \widetilde{S},A,\widetilde{P},R,\gamma\rangle$$

a. uncertain한 상황에서 explore를 통해 information을 얻게 되는데, 얻게 되는 information의 value를 수치화 한다면 exploration과 exploitation을 optimal하게 조절할 수 있다.

b. 주로 uncertain인 상황에서 얻는 information의 value는 높다.

c. 결국 MDP가 정의되었기 때문에 배운 방법대로 아래의 algorithm으로 MDP를 풀 수 있다.

Model-Free RL	Bayesian Model-Based RL
• Q-Learning	• Gittins indices : Bayes-adaptive RL

d. Reward가 Bernoulli Dist.를 통해 정해지는($R^a=B(\mu_a)$) Bernoulli Bandits의 예시를 이용하여 state를 정의해보자. μ_a은 게임에서 지거나 이길 확률을 의미하며 목적은 어떤 slot machine의 arm이 가장 높은 μ_a을 가지고 있는지 찾는 것이다. 그러면 state은 아래와 같이 정의할 수 있다.

$\tilde{S}=\langle \alpha,\beta\rangle$
α_a는 machine a가 reward 0을 준 횟수, β_a는 machine a가 reward 1을 준 횟수를 의미한다.

6-2. Bayesian Model-Based RL : Bayes-Adaptive Bernoulli Bandits

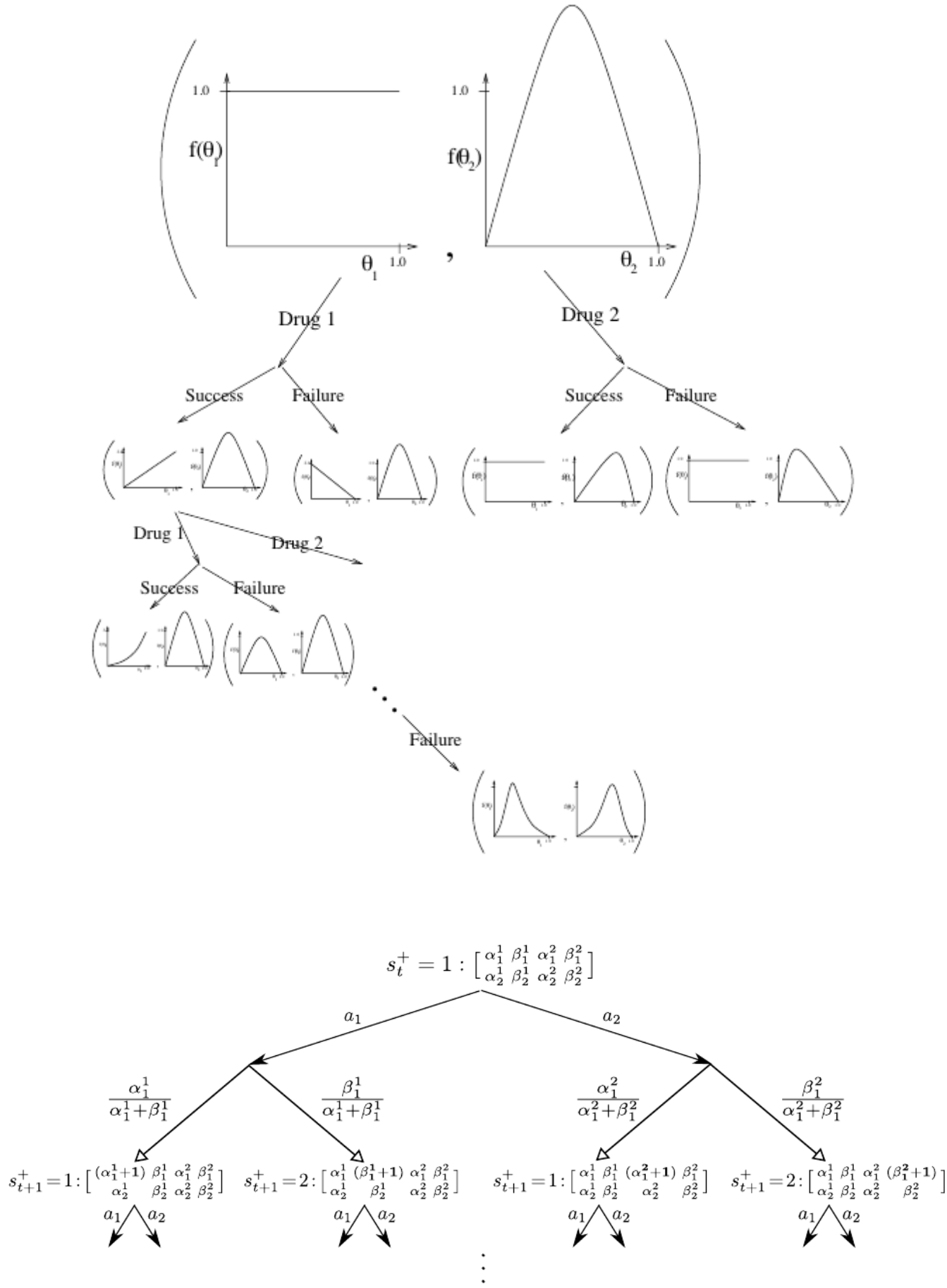
#. Reward Dist. R^a의 Prior를 Beta Dist. $B(\alpha_a,\beta_a)$로 설정한 후에 매 step 마다 action을 통해 bayes-rule로 아래와 같이 posterior를 update 하게 되면 transition function $\tilde{P}$을 정의할 수 있으며, 이 때 posterior(reward model, $B(\alpha,\beta)$)가 information state($\langle\alpha,\beta\rangle$)이다.

$$B(\alpha_a+1,\beta_a)\ \text{if}\ r=0$$

$$B(\alpha_a,\beta_a+1)\ \text{if}\ r=1$$

a. 예를 들어 2개의 신약 개발을 한다고 하자. prior는 각각 약별로 다르게 설정한 후 실험의 결과를 통해 update하는 과정은 아래와 같다.

5) state를 vector로 표현하였을 때, information을 concat해준 것이다.



b. 결국 매번의 state transition마다 Bayesian Model update가 이루어진다.