

### 1. Introduction

#### 1-1. Summary of RL Flow

#### 1-2. Model-Based vs. Model-Free

### 2. Model-Based Reinforcement Learning

#### 2-1. Introduction

##### 2-1-1. Procedure of Model-Based RL

##### 2-1-2. Advantages of Model-Based RL

##### 2-1-3. Representation of Model

#### 2-2. Learning a Model

#### 2-3. Planning with a Model

### 3. Integrated Architectures

#### 3-1. Dyna-Q

### 4. Simulation-Based Search

#### 4-1. Simulation-Based Search

##### 4-1-1. Forward Search

##### 4-1-2. Simulation-Based Search

#### 4-2. Monte-Carlo Search

##### 4-2-1. Simple Monte-Carlo Search

##### 4-2-2. Monte-Carlo Tree Search

#### 4-3. Temporal-Difference Search

# 1. Introduction

## 1-1. Summary of RL Flow

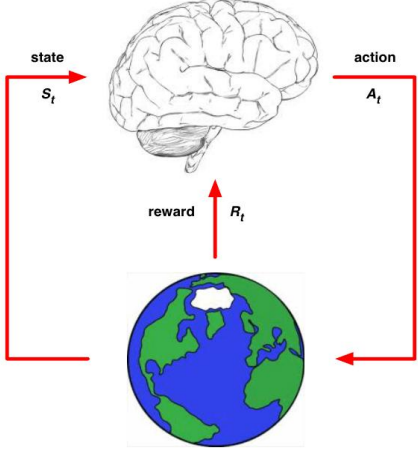
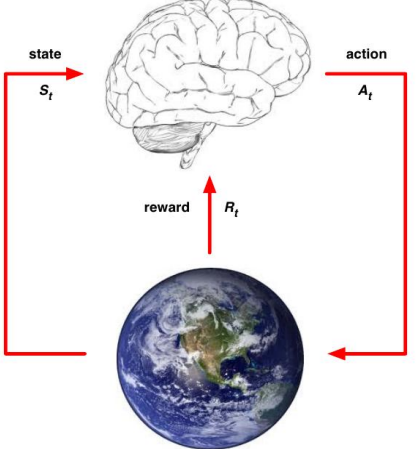
#. 강화학습의 핵심은 Prediction, Control을 구하는 것이 핵심이다. 7단원 이전까지는 각 단원에서 제시한 상황에서 Prediction<sup>1)</sup>, Control<sup>2)</sup> 문제를 푸는 방법을 배워왔다. 강화학습의 큰 흐름은 Model-Based, Model-Free로 크게 분류할 수 있는데 전자는 Model을 agent가 경험을 통해 학습하고 Planning을 하여 Prediction과 Control을 푸는 것<sup>3)</sup>이고, 후자는 Model을 배우지 않고 직접 경험을 통해 Learning을 하여 Prediction과 Control을 푸는 것이다.

각 단원의 흐름을 정리하면 아래의 표와 같다.

1단원			• Introduction of RL
Model-Based	Small Scaled	2단원	• MDP가 무엇인가?
		3단원	• Dynamic Programming을 통해 MDP를 푸는 법
Model-Free	Small Scaled	4단원	• MDP푸는 법 = Prediction (Policy Evaluation)
		5단원	• MDP푸는 법 = Control
	Large Scaled	6단원	• MDP푸는 법 = Prediction (Policy Evaluation)
		7단원	• MDP푸는 법 = Control

## 1-2. Model-Based vs. Model-Free

#. Model-Based와 Model-Free를 아래와 같이 비교할 수 있다.

Model-Based	Model-Free
<ul style="list-style-type: none"> <li>Agent가 Env.와의 상호작용을 통해 Experiences를 쌓고, 이를 이용하여 Model을 배우고 이 Model을 이용하여 Prediction과 Control을 한다. <b>(Planning)</b></li> </ul>	<ul style="list-style-type: none"> <li>Agent가 Env.와의 상호작용을 통해 Experiences를 쌓고, 이를 이용하여 바로 Prediction과 Control을 한다. <b>(Learning, Direct RL)</b></li> </ul>
	

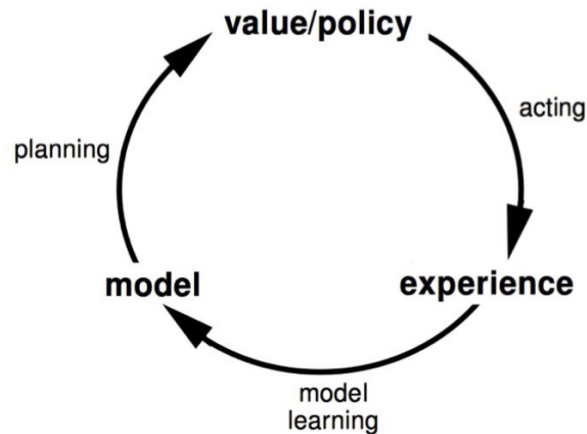
1) Optimal State value fuction, Action value function을 구하는 것을 의미.  
2) Optimal Policy를 구하는 것을 의미  
3) MDP를 푼다고 표현한다.

## 2. Model-Based Reinforcement Learning

### 2-1. Introduction

#### 2-1-1. Procedure of Model-Based RL

#. 아래의 그림과 같이, 먼저 value/policy에 따라 action을 하게 되면 experiences가 생기게 되는데, 이를 이용하여 Model을 학습하는 것이 첫 번째 단계이고, 학습한 Model을 바탕으로 Planning을 하여 value/policy를 구하는 것이 두 번째 단계이다.



#### 2-1-2. Advantages of Model-Based RL

#. 장단점을 아래와 같이 분류할 수 있다.

<b>Advantages</b>	<ul style="list-style-type: none"><li>체스와 같이 규칙이 곧 Model이어서 간단한 경우 Supervised Learning으로 효율적으로 학습할 수 있다.</li><li>model의 불확실성에 대해 알맞게 학습할 수 있다. 즉 model을 학습하면 Reward와 State Transition Prob.을 학습하는 것이다. 하지만 모든 것을 완벽하게 파악하기 어렵기 때문에 학습이 잘 된 부분을 통해 효율적으로 학습이 가능하다.</li></ul>
<b>Disadvantage</b>	<ul style="list-style-type: none"><li>Model을 먼저 추론하고 이를 바탕으로 prediction, control을 하기 때문에 error source가 두 군데나 있다.</li></ul>

#### 2-1-3. Representation of Model

#. MDP( $\langle S, A, P, R, \gamma \rangle$ )를 parameter  $\eta$ 을 이용하여 표현한 Model( $M$ )은 state space  $S$ 와 action space  $A$ 을 알고 있다고 가정한다. 따라서  $M = \langle P_\eta, R_\eta \rangle$  ( $P_\eta \approx P, R_\eta \approx R$ )로 표현할 수 있다.

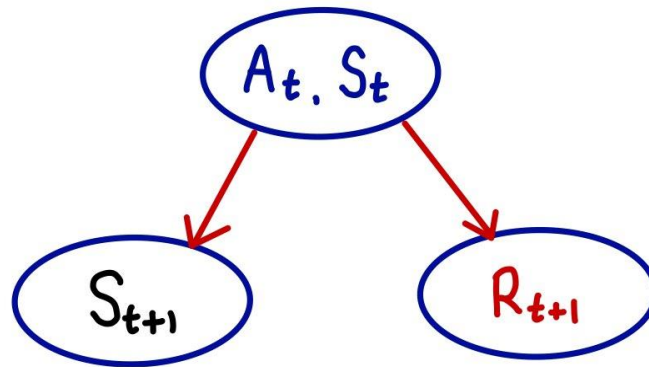
a.  $P_\eta$ 는 확률분포,  $R_\eta$ 는 함수를 의미하므로  $S_{t+1} \sim P_\eta(S_{t+1}|S_t, A_t)$ ,  $R_{t+1} = R_\eta(R_{t+1}|S_t, A_t)$ 로 표현할 수 있다.

b. 전형적으로 state transition prob.과 reward는 conditional independence를 가정하므로 아래와 같은 식이 성립하며, 이는 Bayesian Network에서 Common Parent의 구조로 설명할 수 있다.

서강대학교 머신러닝 연구실

서강현

$$\mathbb{P}[S_{t+1}, R_{t+1} \mid S_t, A_t] = \mathbb{P}[S_{t+1} \mid S_t, A_t] \mathbb{P}[R_{t+1} \mid S_t, A_t]$$



## 2-2. Learning a Model

#. Agent가 Env.와 상호작용하면서 아래와 같이 Experiences를 쌓으면 이것이 곧 data이다. 이를 이용하여 Supervised Learning으로  $(s, a) \rightarrow r$ 은 Regression으로  $(s, a) \rightarrow s'$ 은 Density estimation방법으로 학습하면 된다. 전자의 경우의 loss function은 mean-squared error을 이용하고, 후자는 확률 분포이므로 확률 분포의 차이를 나타내는 KL-Divergence를 loss function으로 사용하면 된다. 그리고  $\eta$ 을 empirical loss가 감소하도록 update하면 된다.

Data-set for Learning a Model
$S_1, A_1 \rightarrow R_2, S_2$
$S_2, A_2 \rightarrow R_3, S_3$
$\vdots$
$S_{T-1}, A_{T-1} \rightarrow R_T, S_T$

- 결국 MDP를 푸는 것은 reward function과 state transition prob.을 구하는 것과 마찬가지이다. 왜냐하면 MDP의 구성요소 중  $S, A$ 는 주어졌다고 가정하기 때문이다.
- $M = \langle P_\eta, R_\eta \rangle$ 을 Neural Network 또는 Gaussian Model 등 다양한 model을 사용할 수도 있다. 이 때 parameter는 당연히  $\eta$ 이다.
- 예를 들어 Table Lookup Model을 사용해보자. 이 model은 state-action pair를 만들어두고 이를 방문할 때 마다 counting을 하는 방법이다. 구체적으로  $P, R$ 은 아래와 같이 계산한다.

$$\hat{P}_{s,s'}^a = \frac{1}{N(s,a)} \sum_{t=1}^T \mathbf{1}(S_t, A_t, S_{t+1} = s, a, s')$$

$$\hat{R}_s^a = \frac{1}{N(s,a)} \sum_{t=1}^T \mathbf{1}(S_t, A_t = s, a) R_t$$

d. Table Lookup Model의 단점은 agent가 경험하지 못한 state transition prob., reward는 계산하지 못한다는 것이다.<sup>4)</sup> 이 방법과 유사한 것은 매 time-step마다  $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$ 을 저장하였다가 나중에  $S_t, A_t$ 을 하려고 할 때, 이와 일치하는 tuple을 uniform sampling하는 방법이 있다.

e. 아래의 그림은 AB-Example<sup>5)</sup>에 Table Lookup Model을 적용하여 학습한 Model이다.

Two states  $A, B$ ; no discounting; 8 episodes of experience

$A, 0, B, 0$

$B, 1$

$B, 1$

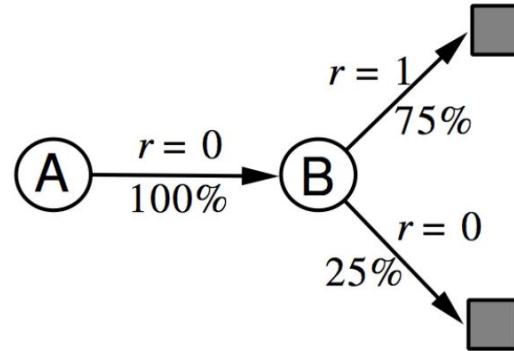
$B, 1$

$B, 1$

$B, 1$

$B, 1$

$B, 0$



### 2-3. Planning with a Model

#. Model-Based RL의 두 번째 단계로, 학습한 Model을 사용하여 Planning을 해야 한다. 현 상태는 MDP를 학습한 상태이므로 3단원에서 배운 Dynamic Programming을 사용할 수도 있고, 반대로 Model-Free에서 사용하는 방법론을 사용할 수 있는데 이를 Sample-Based Planning이라 한다. 이 방법의 경우 학습한 model은 sampling을 하기 위해서만 사용된다.

a. MDP를 학습했음에도, Sample-Based Planning을 사용하는 이유는 차원의 저주를 해결할 수 있기 때문이다. 즉 'sample은 population에서의 data간 상대적인 속성을 반영한다.<sup>6)</sup>'는 전제로 인해 Naive한 Dynamic Programming을 사용하는 것 보다는 sampling을 통해 이를 이용하여 차원의 저주로 인한 계산 복잡도<sup>7)</sup>를 줄이고 학습을 좀 더 효율적으로 할 수 있다는 것이다.

b. 결국 Model 학습 후에 사용할 수 있는 Planning Algorithm은 아래와 같다.

<b>Model-Based</b>	<ul style="list-style-type: none"> <li>Value iteration</li> <li>Policy iteration</li> <li>Tree Search</li> </ul>
<b>Model-Free</b>	<ul style="list-style-type: none"> <li>Monte-Carlo Control</li> <li>Sarsa</li> <li>Q-Learning</li> </ul>

4) 이 한계는 Neural Network으로 극복할 수 있다.

5) action이 없기 때문에 MRP라고 보아도 된다.

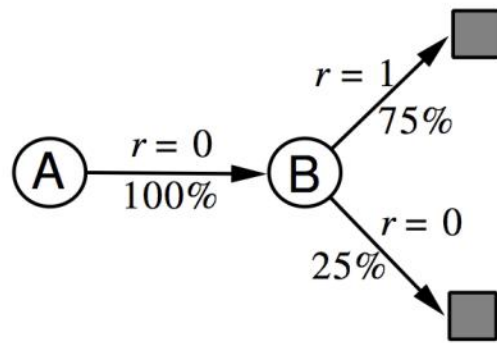
6) population에서 A class가 많이 나왔으면, sample에서도 A class가 많이 나올 것이다.

7) 차원 당 계산을 해야 하는데, 차원이 늘어날수록 계산복잡도가 늘어난다.

c. AB-Example에 Sample-Based Planning을 적용하면 아래와 같다.

#### Real experience

A, 0, B, 0  
 B, 1  
 B, 1  
 B, 1  
 B, 1  
 B, 1  
 B, 1  
 B, 1  
 B, 0



#### Sampled experience

B, 1  
 B, 0  
 B, 1  
 A, 0, B, 1  
 B, 1  
 A, 0, B, 1  
 B, 1  
 B, 0

e.g. Monte-Carlo learning:  $V(A) = 1, V(B) = 0.75$

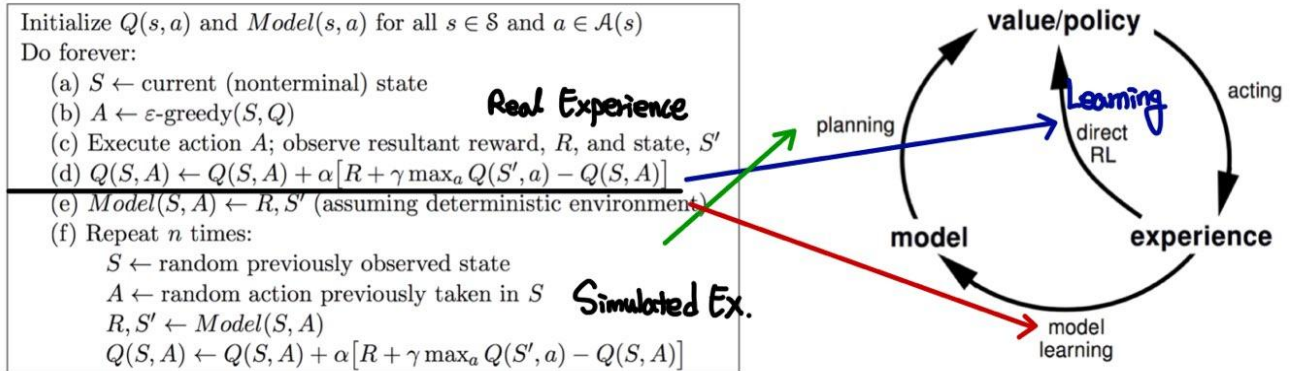
즉 이미 Model은 Real experience로 가운데 graph처럼 생성하였으므로 이는 치워버리고, 학습한 model을 이용하여 무한한 data를 생성할 수 있다.

- d. 하지만 우리가 추론한 model은 항상 정확하지 않다.  $\langle P_\eta, R_\eta \rangle \neq \langle P, R \rangle$  이 경우엔 planning이 sub-optimal policy로 계산될 것이다. 이를 해결하기 위해선, 첫 번째로 단순히 Model-Free방법론을 사용하거나, 두 번째로 model의 uncertainty를 정확하게 표현하여 model을 학습하는 것이다. 이러한 방법에는 Bayesian Model-Based RL이 있다.
- e. Bayesian Model-Based RL은 Bayes inference와 동일한 방식으로 어떤 prior분포를 설정하고 agent가 model을 경험할 때 마다 이를 update하면서 점점 정교해지도록 (Variance=uncertainty가 줄어들도록) model을 학습하는 방법론이다.

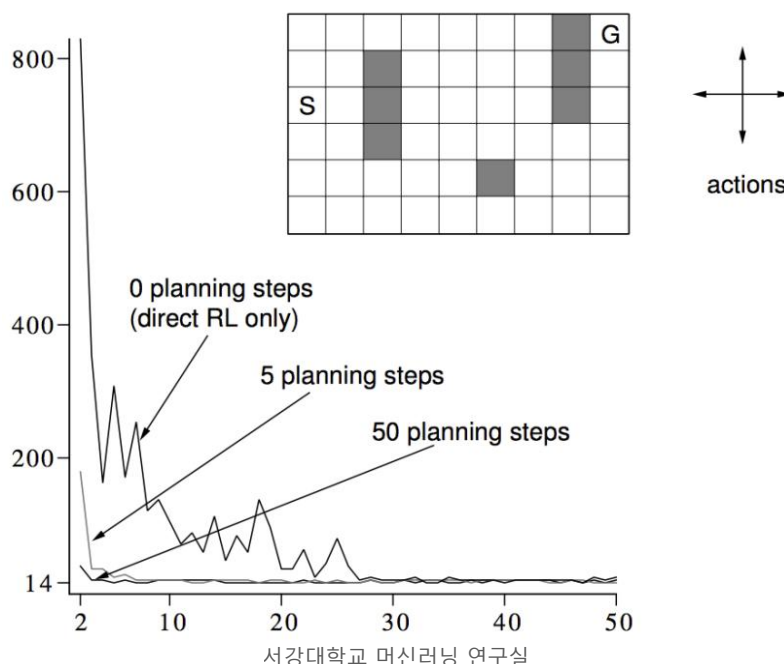
### 3. Integrated Architectures

#### 3-1. Dyna-Q

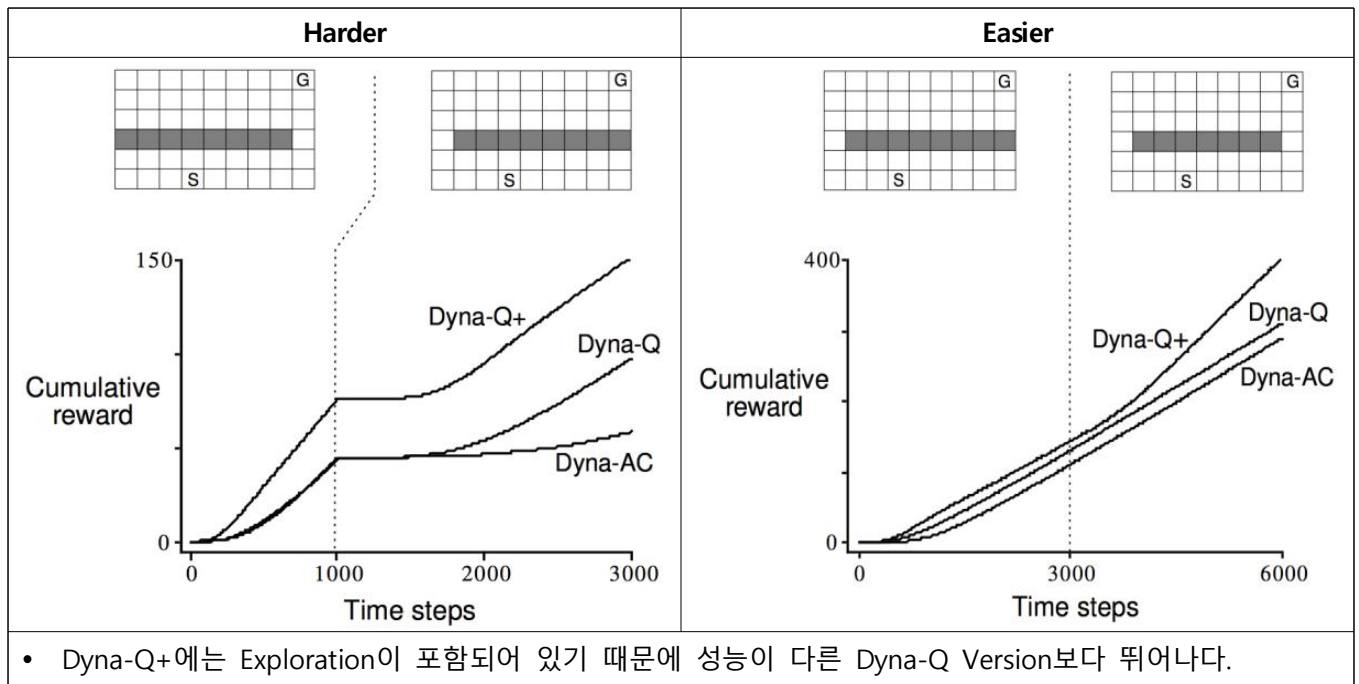
#. Agent가 생성하는 Experience 즉 data의 source는 실제 Env.과 상호작용하여 만든 Real Experience와 model을 학습하여 이와 상호작용하여 만든 Simulated Experience가 있다. 실제 Env.와 상호작용하는 것은 비용이 크기 때문에 한번 Env.과 상호작용 후에 Q-update(Learning)를 하고 Model을 학습하고 이 model을 사용하여  $n$ 번 Q-update(Planning)하는 방법이 Dyna-Q이다.



- 즉 learning과 planning을 합해서 사용하는 방식이다.
- (a)-(d)에선 실제 Env.와 상호작용한 결과를 사용하여 Q update를 진행하고, (e)에서는 Real Experiences(이용하여  $S$ 에서  $A$ 을 하면  $R$ 을 받는)를 이용하여 model을 학습한다. 그리고 (f)에서는 실제 방문했던 states중 하나를 뽑고 거기에서  $A$ 을 했던 것을 뽑는다. 그리고  $Model()$ 을 이용하여  $(S, A)$ 의 결과인  $(R, S')$ 을 sampling하고 tuple  $(S, A, R, S')$ 을 이용하여  $Q$ 를  $n$ 회 update한다.
- Dyna-Q의 성능은 아래와 같다.  $x$ 축은 Episode를,  $y$ 축은 Episode당 걸린 steps를 의미하며 이 수치가 낮을 수록 성능이 뛰어난 것이다. 결국 적은 수의 Experiences(data)를 쥐어짜서 상상 속에서 여러 번 해보는 방법의 성능이 뛰어나다.



d. 아래의 그래프는 model을 부정확하게 학습한 경우에 대한 성능을 나타낸 그래프이며 오른쪽의 model이 실제 model이다.





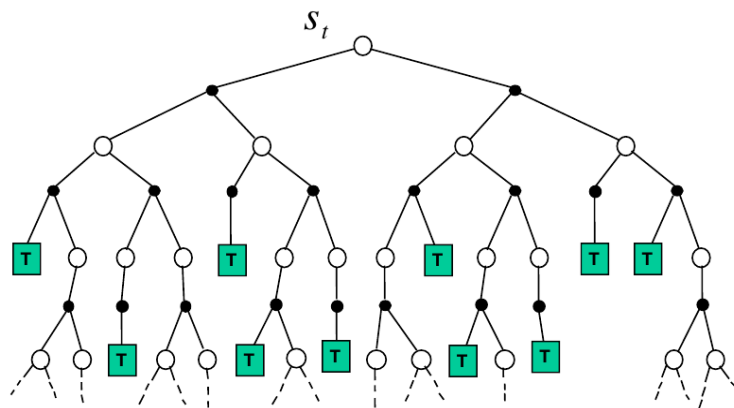
## 4. Simulation-Based Search

#. Planning을 좀 더 효율적으로 하는 방법에 대해서 배운다. Planning은 model을 이미 학습한 상태에서 이루어지기 때문에 MDP가 이미 주어졌다고 생각한다.

### 4-1. Simulation-Based Search

#### 4-1-1. Forward Search

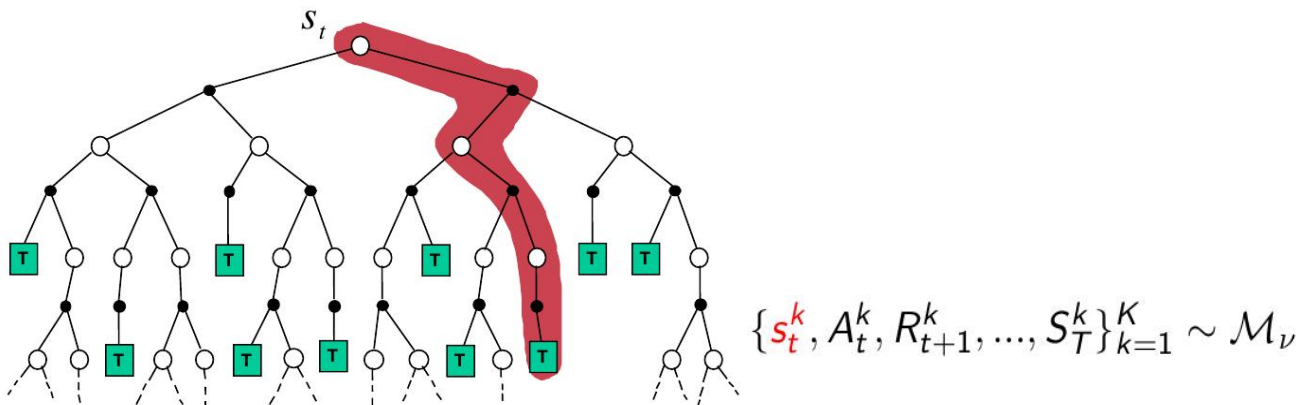
#. 현재의 state  $s_t$ 을 root-node로 설정하고, 학습한 Model을 사용하여 lookahead를 통해 search tree를 만들고 best action을 search하는 algorithm이다.



- 현재 state를 root-node로 설정했기 때문에 전체 MDP를 고려할 필요 없이 sub-MDP만 고려하면 되며, 현재 상황으로부터 미래만 보겠다는 것이다.
- 위의 tree에서 ○은 state, ●은 action을 의미한다.

#### 4-1-2. Simulation-Based Search

#. 학습한 model을 이용하여, Simulated Experiences를 통해, 현재로부터 시작되는 아래의 빨간 색과 같이, 여러 Episodes를 생성한 다음 Episode에 Model-Free RL 방법론(Monte-Carlo, Sarsa, TD 등)을 사용하는 방법론이다. 즉 Forward Search에 sample-based planning을 사용한 것이다.



- Model-Free RL 방법론 중 여기에 MC를 사용한 것이 Monte-Carlo Search이다.

## 4-2. Monte-Carlo Search

#. Monte-Carlo Search는 Planning이므로 model이 주어져 있다. 따라서 이를 이용하여 Episode를 생성하는 Simulation 단계와 action을 평가하는 Evaluation 단계로 이루어져 있다.

### 4-2-1. Simple Monte-Carlo Search

#. Model과 Policy  $\pi$ 가 주어진 상태에서 어떤 state  $s_t$ 에 있을 때, 가능한 모든 action에 대해 각각  $K$ 만큼 Episodes를 생성(Simulation)하고 해당 action을 따랐을 때의 return 값을 평균을 낸다.(Evaluation) 그리고 이 중에서 가장 높은  $Q$ 을 선택하면 된다.

Do Search  
 For each action  $a \in A$  in  $s_t$  :  
   do **Simulate**  $K$  episodes by fixed  $\pi$   
   do **Evaluate** action  $a$   
**Select Action**

Simulation	Evaluation	Select Action
$\{s_t^k, A_t^k, R_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu$	$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^K G_t \xrightarrow{P} q_\pi(s_t, a)$	$a_t = \operatorname{argmax}_{a \in A} Q(s_t, a)$

### 4-2-2. Monte-Carlo Tree Search in GO

#. 현재 주어진 policy  $\pi$ <sup>8)</sup>을 이용하여 통으로  $K$ 개의 Episodes를 생성하고(Simulation<sup>9)</sup>), root-node만 고려하는 것이 아니라 sub-nodes까지 고려<sup>10)</sup>한다. Model을 이용하여 한 번의 Episode를 실행할 때 마다 in-tree, out-of-tree여부<sup>11)</sup>에 따라 Tree Policy( $\pi$ ), Default Policy를 각각 사용한다. 그리고 Episode가 끝나면 MC Evaluation 방법으로 각 state의  $Q(s, A)$ 을 구하고  $\epsilon$ -greedy를 사용하여 Tree Policy를 improvement한다.

Simulation
$\{s_t^k, A_t^k, R_{t+1}^k, S_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$
Evaluation
$Q(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^K \sum_{u=t}^T \mathbf{1}(S_u, A_u = s, a) G_u \xrightarrow{P} q_\pi(s, a)$
Select Action
$a_t = \operatorname{argmax}_{a \in A} Q(s_t, a)$

8) Tree-Policy

9) Tree Policy와 Default Policy의 2가지로 나뉜다.

10) 해당 node의 방문 횟수와 취했던 action들을 저장한다.

11) Model내에 있으면 in-tree, 아니면 out-of-tree.

## Pseudo Code in Black View

### Do Evaluation :

For all child-nodes of  $c_0$  :

Store (Total  $r$ , Total visit cnt) info. // MC Evaluation

### Do Search :

Create Tree-Policy empty tree 'In-Tree'

Let the current state be  $c_0 \leftarrow s_t$  // Selection

Set the root-node  $c_0$  to In-Tree

Let  $K$  be a number of Simulations

For each **Simulation** in  $K$  :

$s_t \leftarrow c_0$

Repeat Until Terminal-State :

if  $s_t$  is node of In-Tree :

$a = \operatorname{argmax}_a Q(S, A)$  by Tree Policy( $\pi$ ) // **Improve** tree policy( $\pi$ ) by  $\epsilon$ -greedy

else (out-of-tree) :

$a$  = Default Policy (Rollout Policy)

if  $s_{t+1}$  is first-visited state by Default Policy : Let it be child-node of  $s_t$

$s_t \leftarrow s_{t+1}$  // Expansion

if Black win :  $r = 1$

else :  $r = 0$

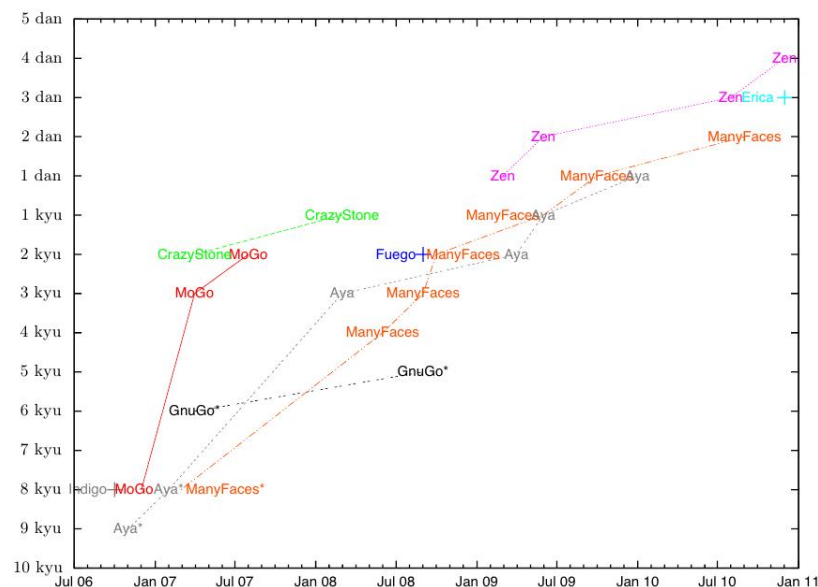
<b>Reward Function (undiscounted)</b>	$R_t = 0$ for all non-terminal steps $t < T$ $R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$
<b>Policy</b>	$\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
<b>Value Function</b>	$v_\pi(s) = \mathbb{E}_\pi [R_T \mid S = s] = \mathbb{P} [\text{Black wins} \mid S = s]$ $v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$

<p>Monte-Carlo Search : 1회 Simulation + Evaluation</p>	<p>Monte-Carlo Search : 2회 Simulation + Evaluation</p>
<p>Monte-Carlo Search : 3회 Simulation + Evaluation</p>	<p>Monte-Carlo Search : 4회 Simulation + Evaluation</p>
<p>Monte-Carlo Search : 5회 Simulation + Evaluation</p>	<p>Evaluation State</p> <p><math>V(s) = 2/4 = 0.5</math></p> <p>Current position <math>s</math></p>

- a. Improvement 연산이 등장하였기 때문에 MC control 방법론이 simulated experience에 적용되었다고 할 수 있다. 즉 simple mc와는 다르게 simulation policy  $\pi$ 가 improve된다.
- b. 이러한 방식으로 진행하면 Optimal Search Tree에 수렴한다.  $Q(S, A) \rightarrow q_*(S, A)$
- c. AlphaGo에서는 Default Policy를 Rollout-Policy로 사용하였다.
- d. MCTS는 RL에서 Planning에 속하고, Planning중에서도 Forward Search에 속하고, 이 중에서도 Simulation-Based Search에 속한다. Simulation-Based Search 중에서도 Monte-Carlo를 사용하기 때문에 Monte-Carlo Search에 속하는 것으로 정리할 수 있다.
- e. MCTS의 장점은 아래와 같다.

- Tree의 node는 여러 번 방문한 것이기 때문에 'Highly selective best-first search'이다.
- State Evaluation의 값이 상황에 맞게 Dynamic하게 변한다.
- Sampling을 사용하기 때문에 차원의 저주를 극복할 수 있다.
- Model이 Black-Box이다. 즉 model에 '현재 state에서 이 action을 하면 다음 state이 무엇이고 reward가 뭐야' 라는 query만 날리면 sample을 얻을 수 있다.

- f. 아래의 그래프는 기존 GO-Program이며  $x$  축은 시간이고  $y$ 는 실력이다. 위에 있는 Program들은 MCTS를 사용한 것들이다.



### 4-3. Temporal-Difference Search

#. Monte-Carlo Search와 비슷한 방법으로 현재 state로부터 시작되는 episode를 생성하는데, 매 simulation의 step마다 Sarsa식을 이용하여 action-value function(Q)을 update해준다. 그리고 action 선택은  $\epsilon$ -greedy를 사용한다. 당연히 Q에 function approximation을 사용할 수 있다.

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

- a. MCTS가 sub-MDP에 대해 MC control을 적용한 것처럼, TD-Search 또한 sub-MDP에 Sarsa를 적용한 것이다.
- b. 중요한 점은 현재 상황에서 Forward Search에 더 특화된 방법론을 사용하는 것이지 MC, TD의 여부는 중요하지 않다.
- c. MC Search와 TD Search의 장단점은 원래의 MC, TD와 장단점과 동일하다. 왜냐하면 학습한 Model에 planning을 사용하는 것이기 때문이다.
- d. TD-Search의 예시는 Dyna-2이다. 이 model의 agent는 두 가지의 Feature Weights를 저장한다. 하나는 Real-Experience를 학습하는 'Long-term Memory'이고 다른 하나는 Simulated-Experience를 학습하는 'Short-term Memory'이다. 이렇게 나누어서 학습을 한 후에 전체 value function은 이 둘을 합해서 계산하게 된다.
- e. 아래의 그래프는 Dyna-2의 성능을 보여주는 그래프이다. GO의 상황에서는 Search 또한 중요하기 때문에 Planning을 사용하지 않는 단순 TD-Learning은 성능이 떨어진다.

