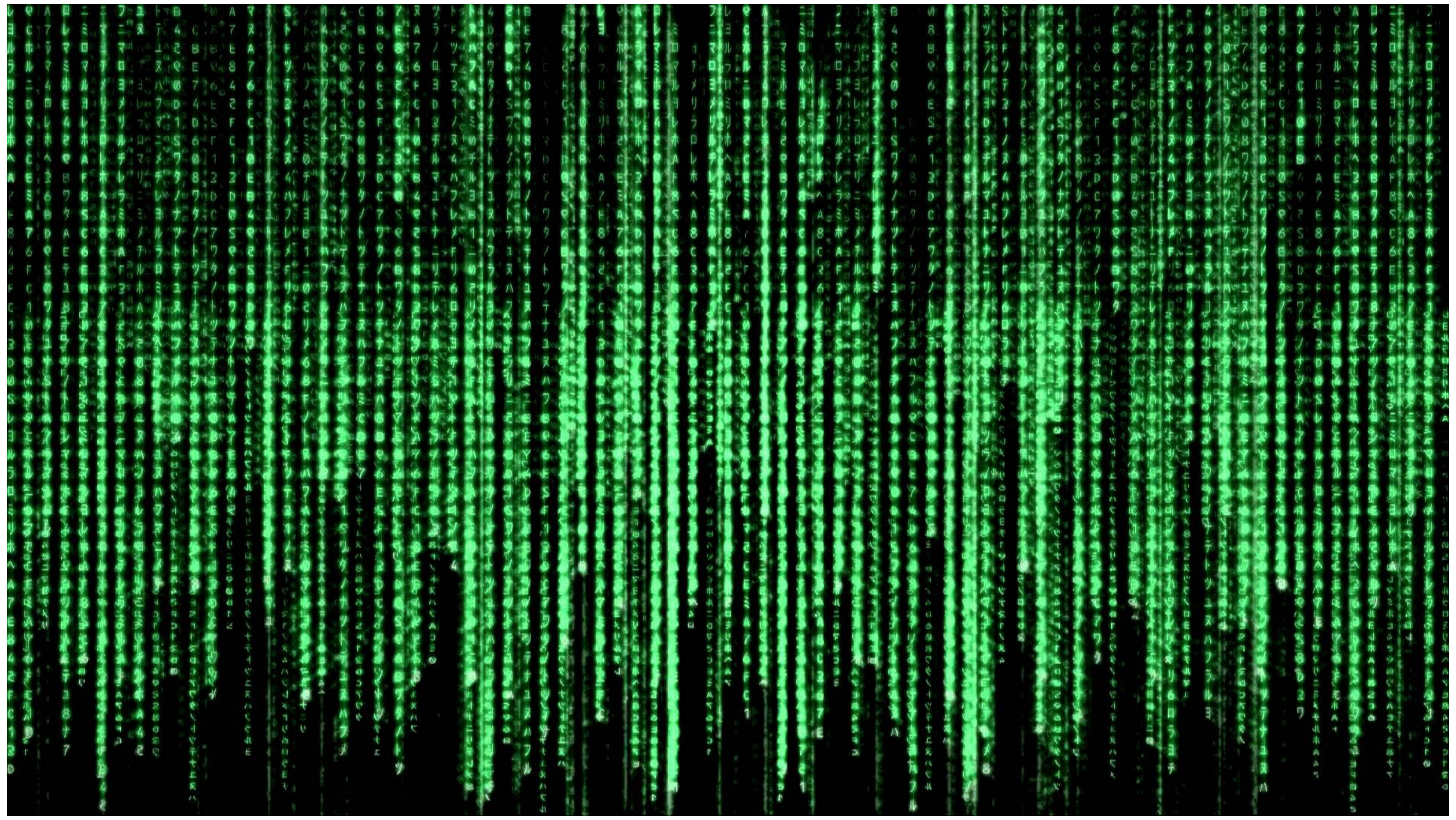
# Introduction of Reinforcement Learning

# Artificial Intelligence

- **지능이란?**

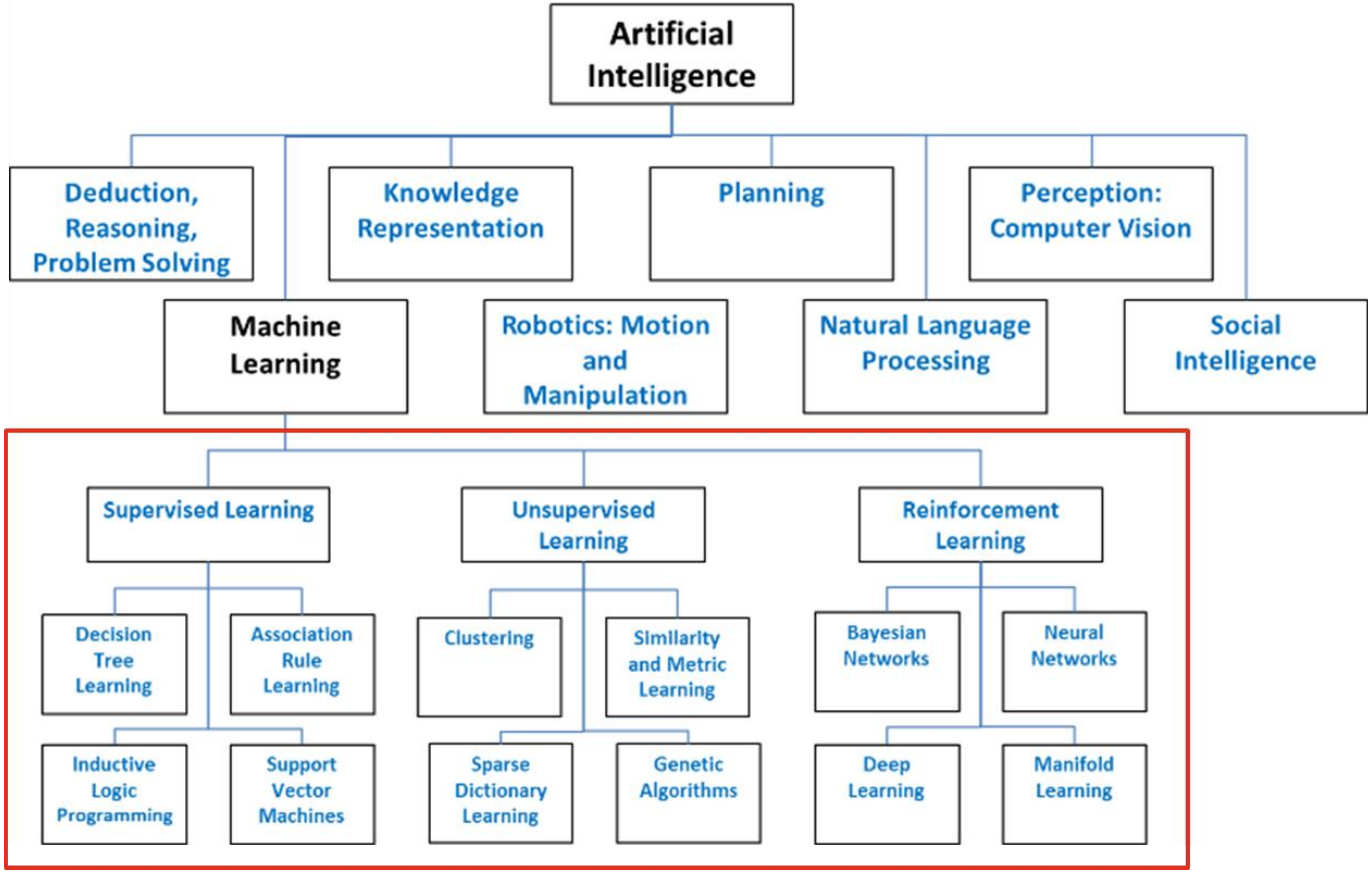
  → 보다 추상적인 정보를 이해하는 능력



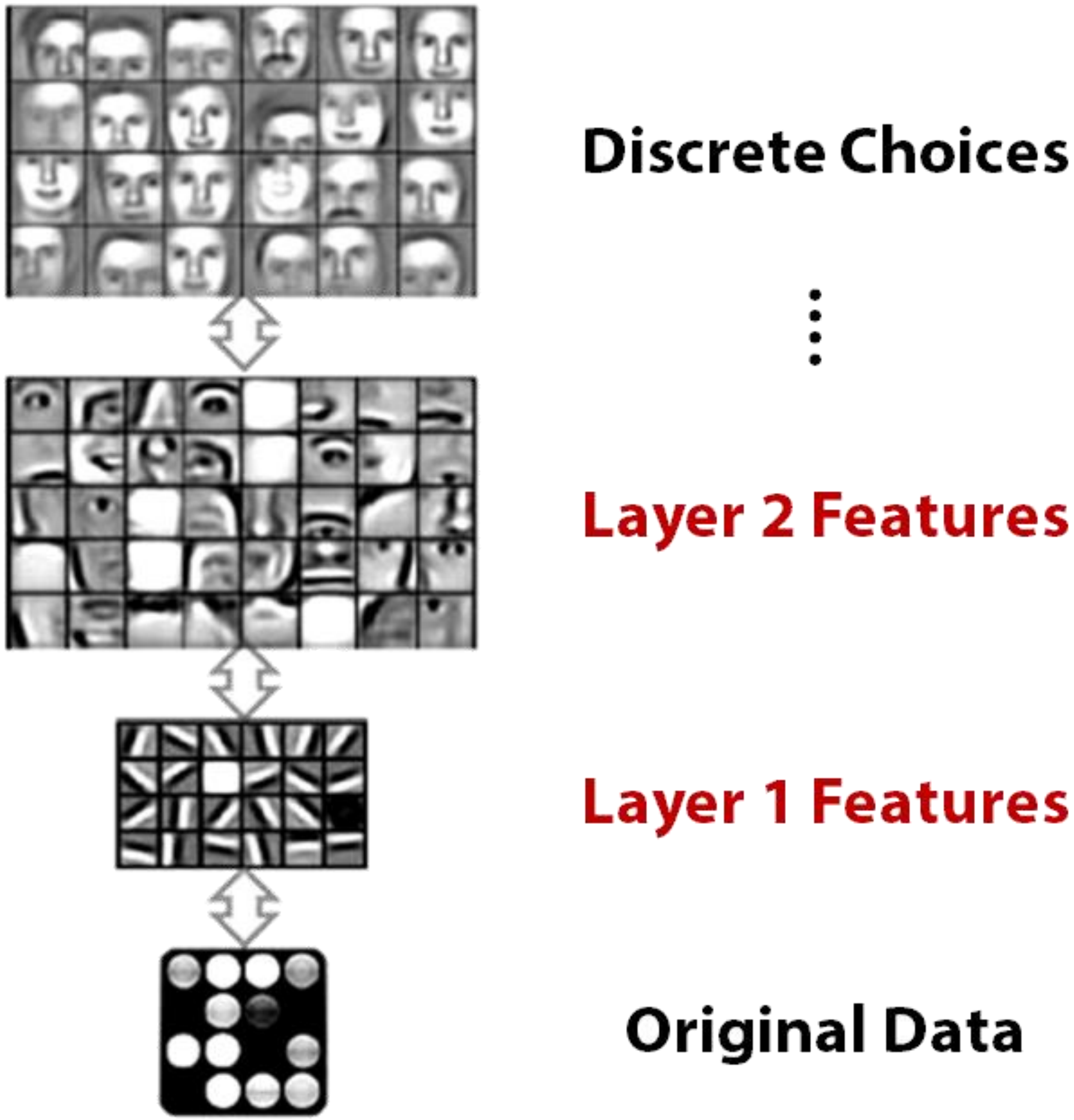
- **인공 지능이란?**

  → 이러한 지능 현상을 인공적으로 구현하려는 연구

# Artificial Intelligence & Machine Learning



# Deep Learning in RL




- DeepRL에서 딥러닝은 그저 하나의 module로써만 사용된다.

- **Deep Learning의 강점 :**

1. 층을 쌓을 수록 더욱 Abstract Feature Learning이 가능한 유일한 알고리즘.

2. Universal Function Approximator.

# Universal Function Approximator


Figure 5. A selection of evaluation results, grouped by human rating.

# Reinforcement Learning이란?

# Machine Learning

- Supervised Learning :

  $$y = f(x)$$

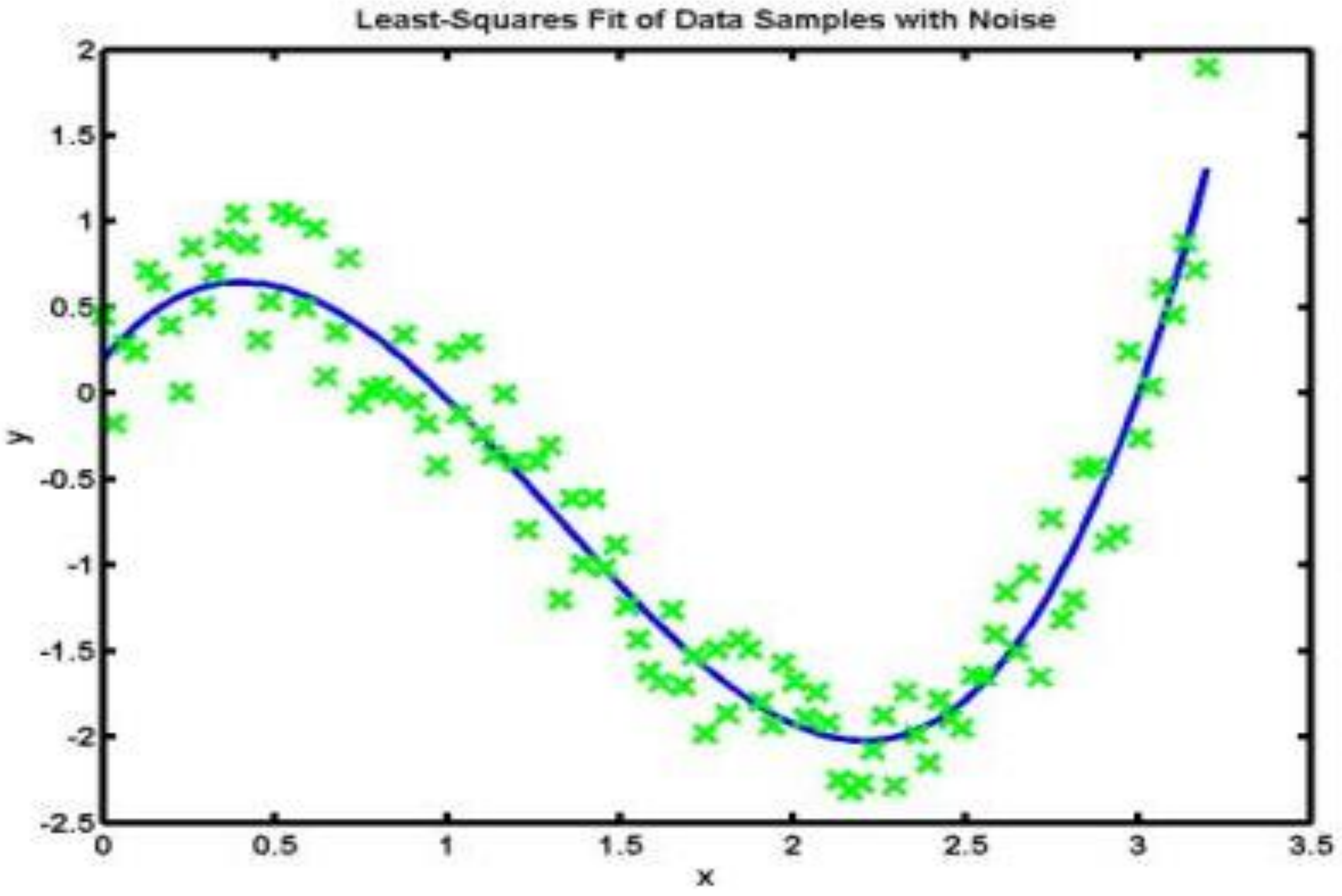
- Unsupervised Learning :

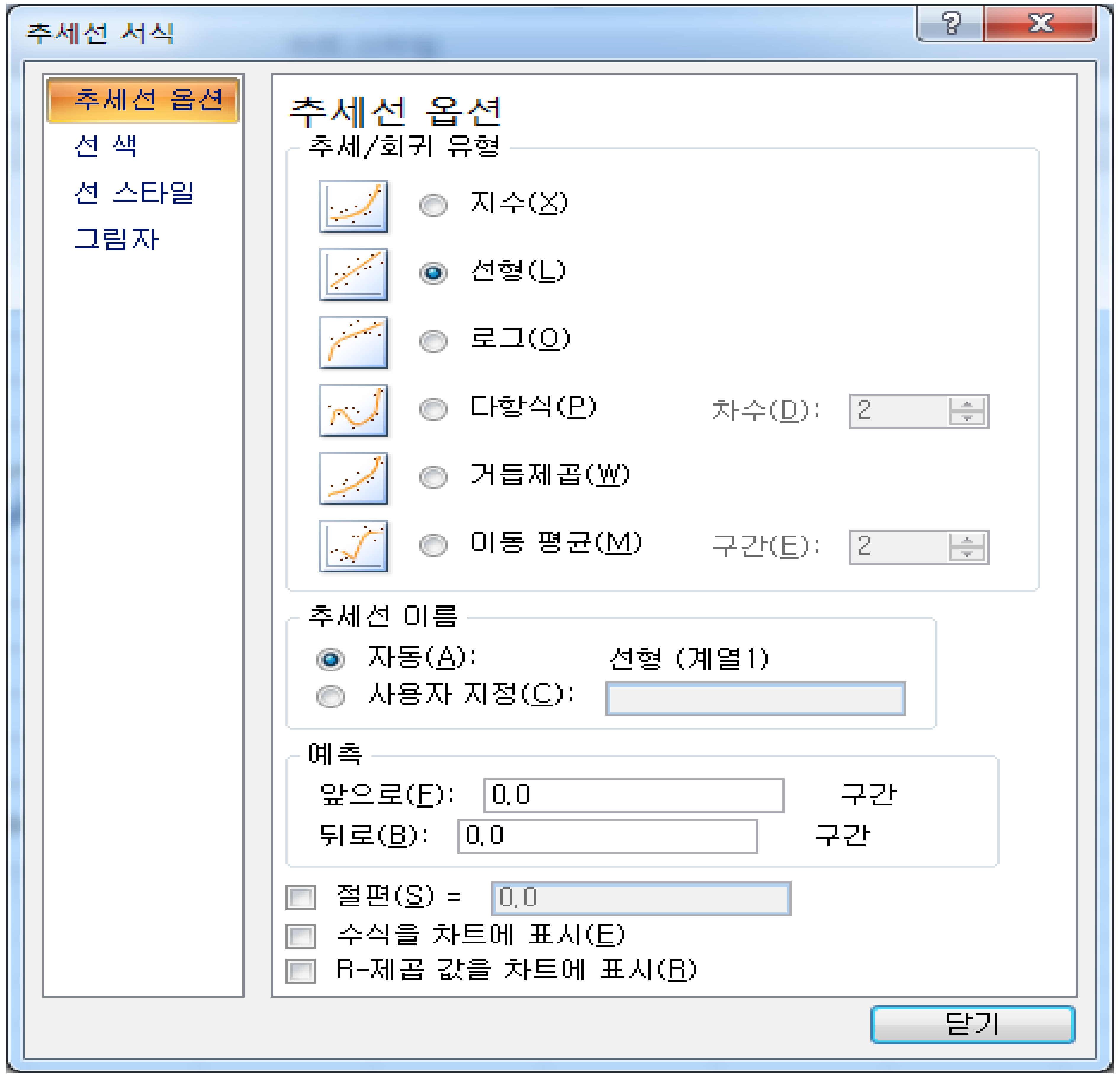
  $$x \sim p(x) \quad \text{or} \quad x = f(x)$$

- Reinforcement Learning :

  **Find a policy, $p(a|s)$ which maximizes the sum of reward**

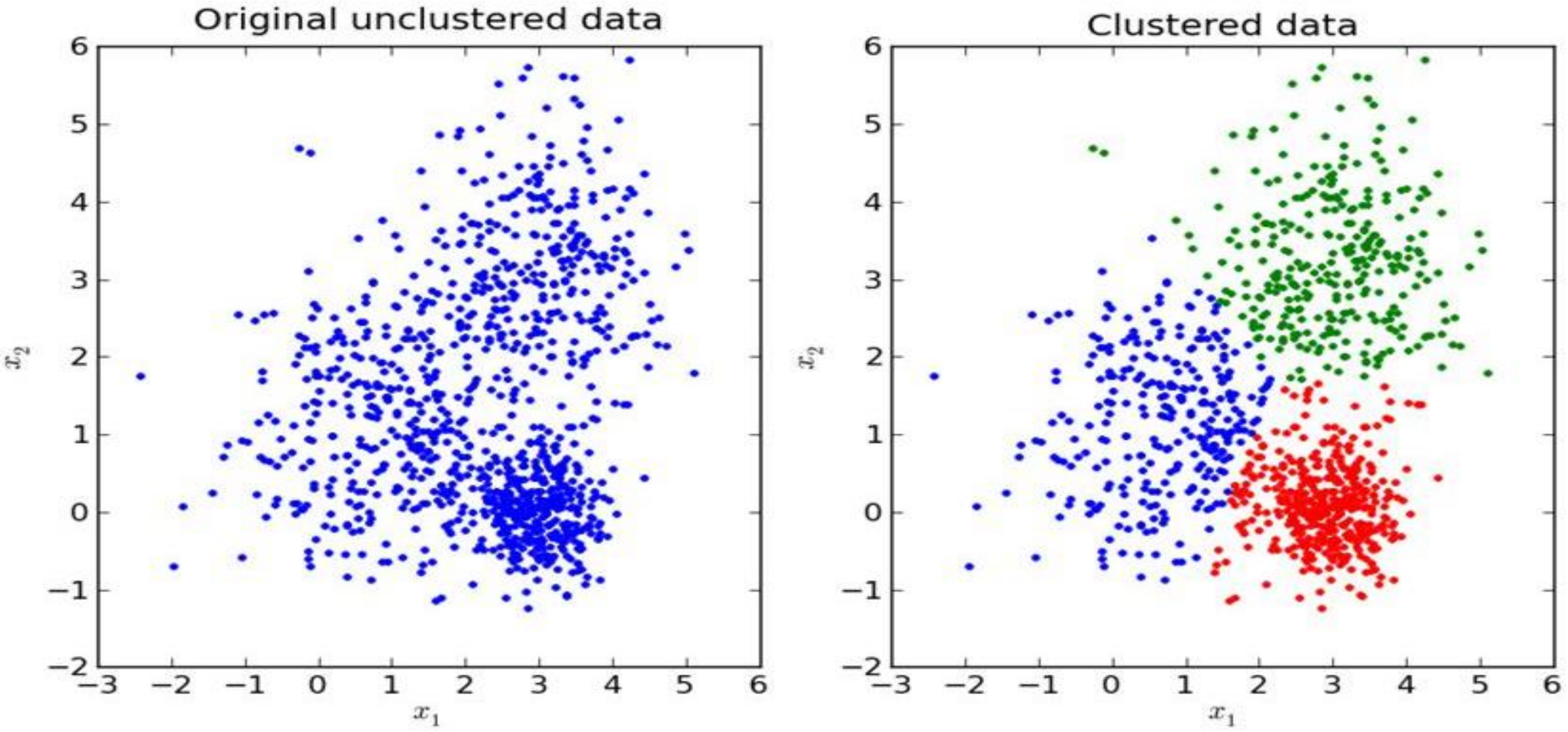
# Example of Supervised Learning : **Polynomial Curve Fitting**



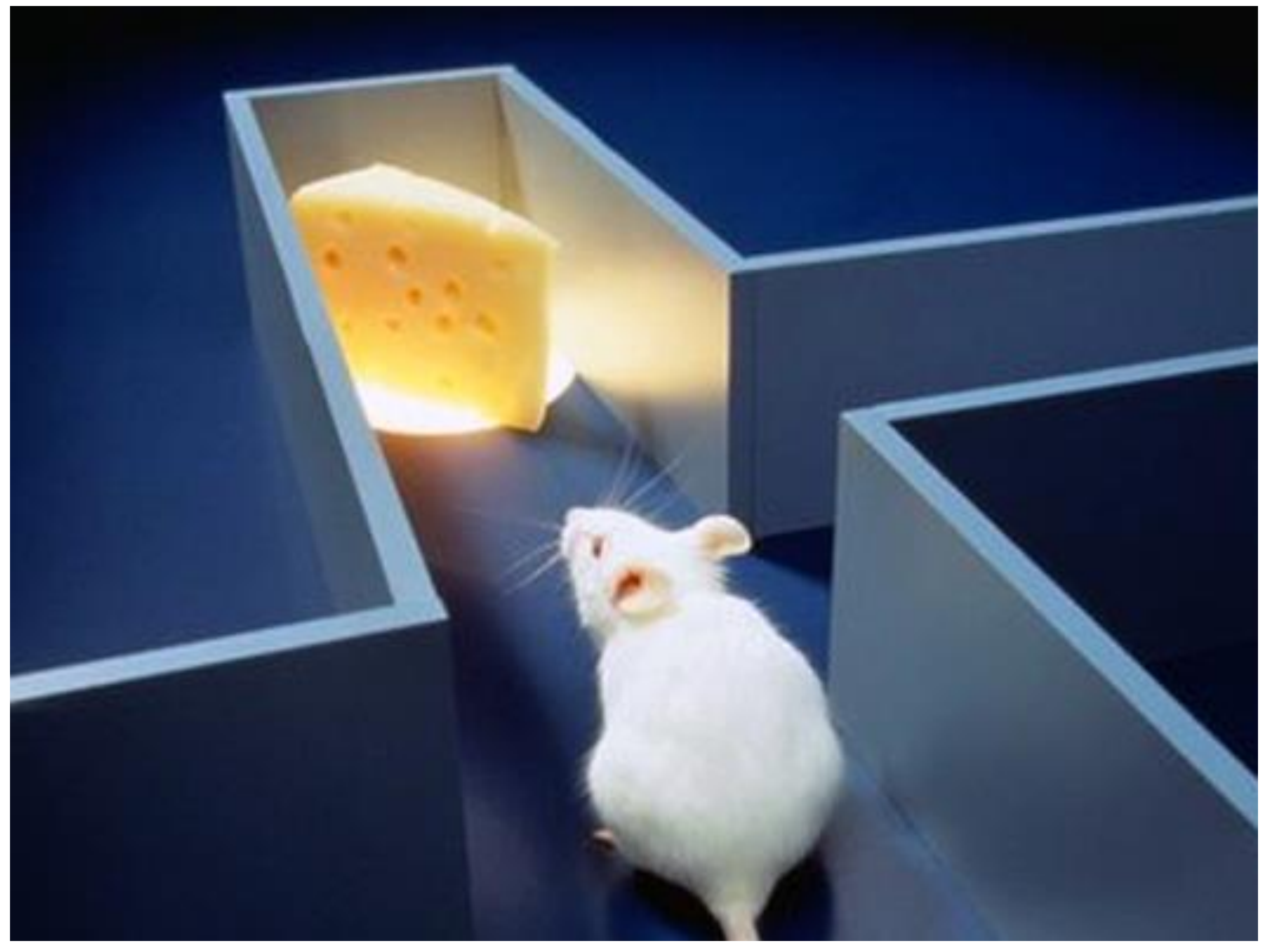

Microsoft Excel 2007의 추세선

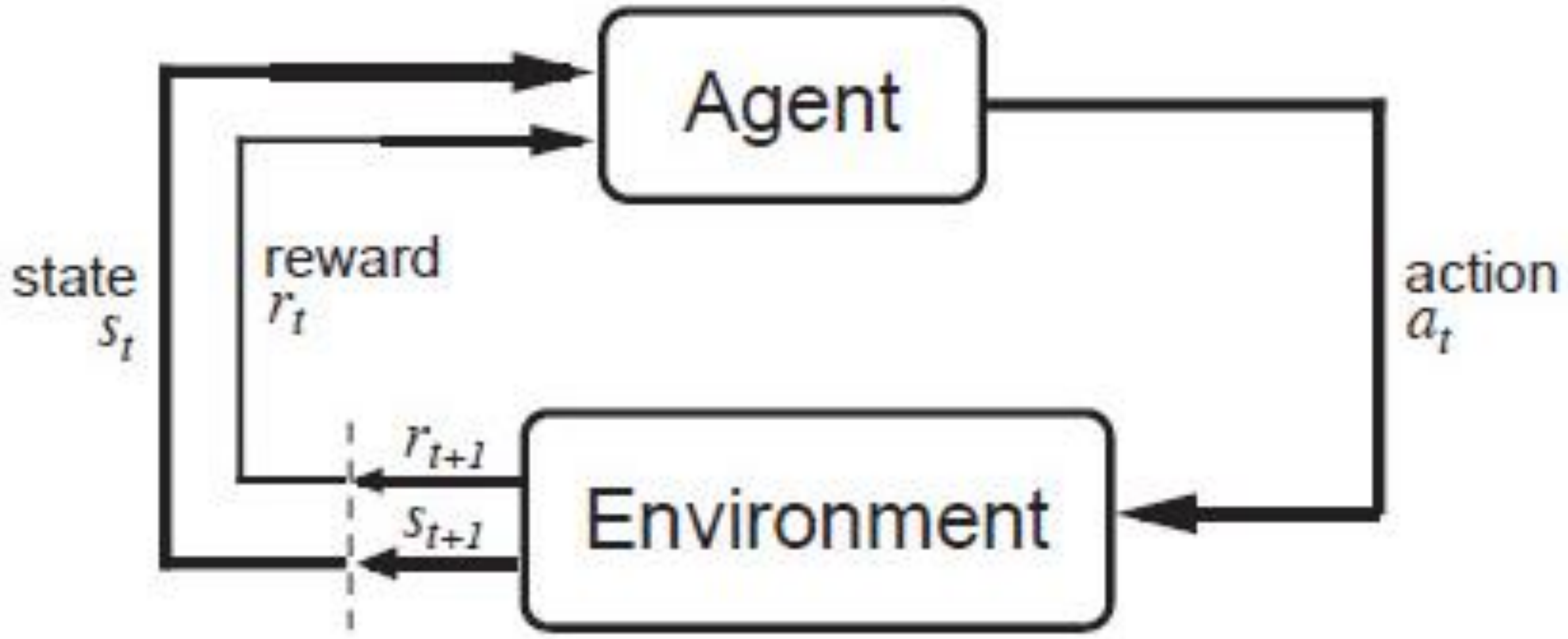
# Example of Unsupervised Learning : **Clustering**

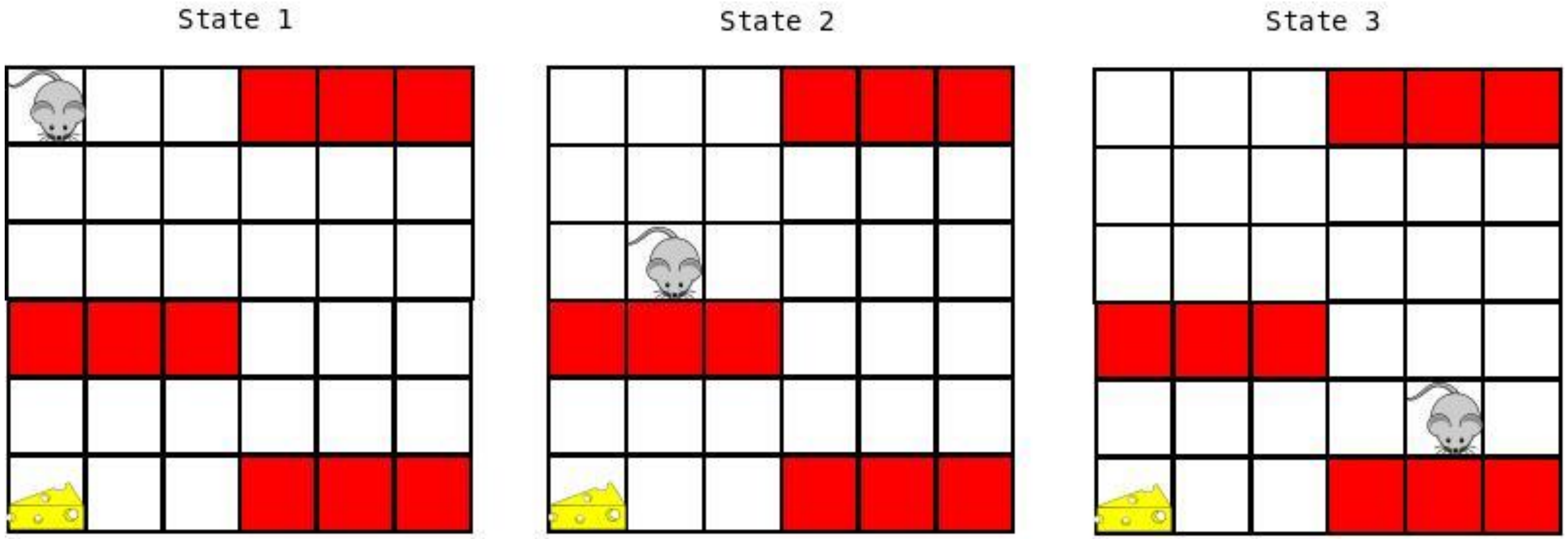



http://www.frankichamaki.com/data-driven-market-segmentation-more-effective-marketing-to-segments-using-ai/

# Example of Reinforcement Learning : **Optimal Control Problem**





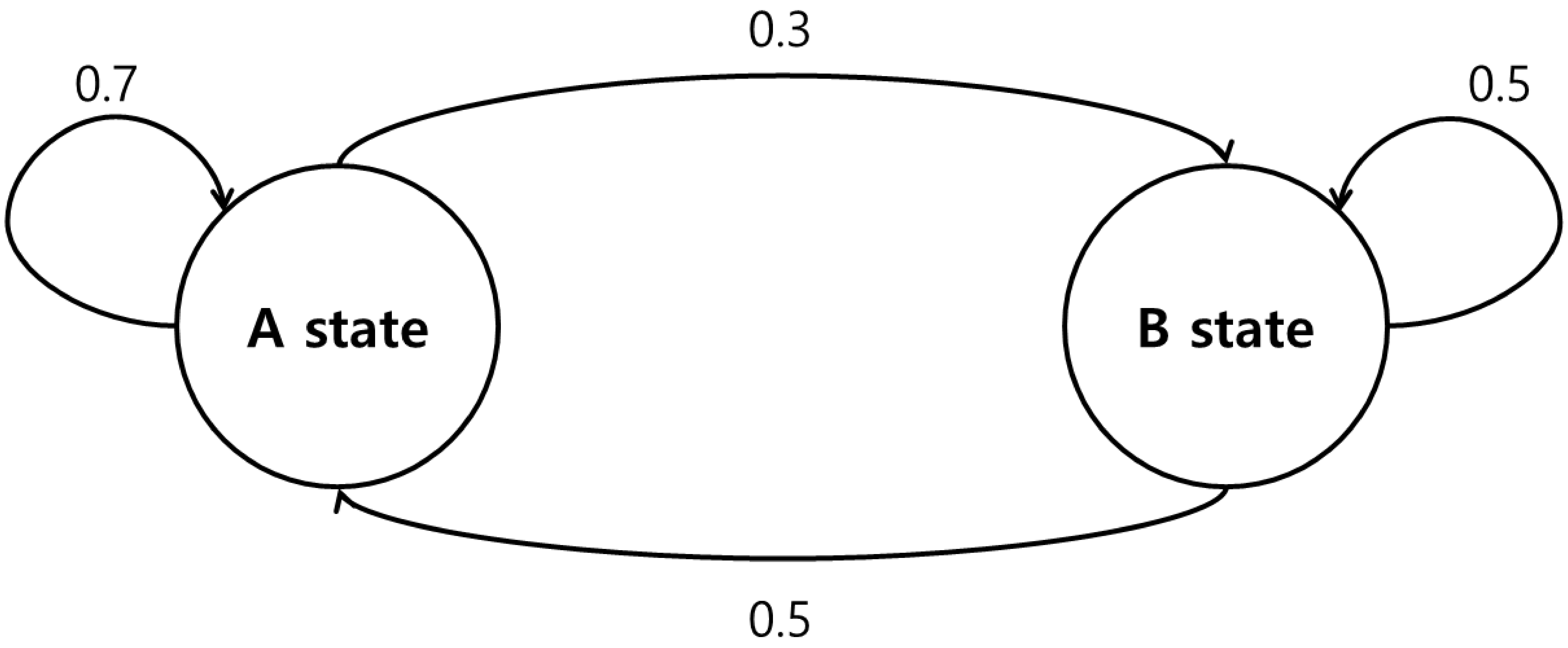




http://graveleylab.cam.uchc.edu/WebData/mduff/older_papers.html
https://studywolf.wordpress.com/2015/03/29/reinforcement-learning-part-3-egocentric-learning/

# Markov Decision Processes(MDP)

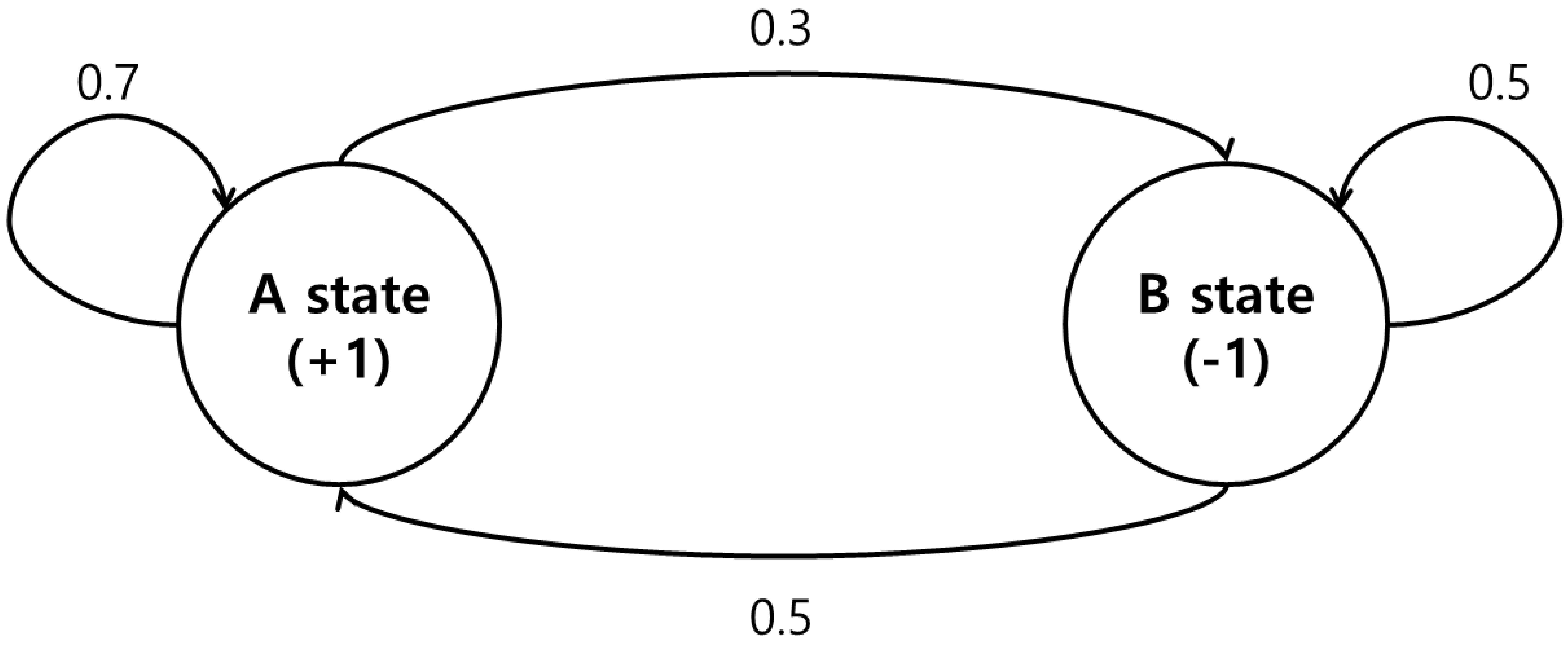
# Markov Processes

- Discrete state space : $\mathbf{S = \{ A , B \}}$
- State transition probability : $\mathbf{P(S'|S) = \{P_{AA} = 0.7 , P_{AB} = 0.3 , P_{BA} = 0.5 , P_{BB} = 0.5 \}}$
- Purpose : Finding a steady state distribution



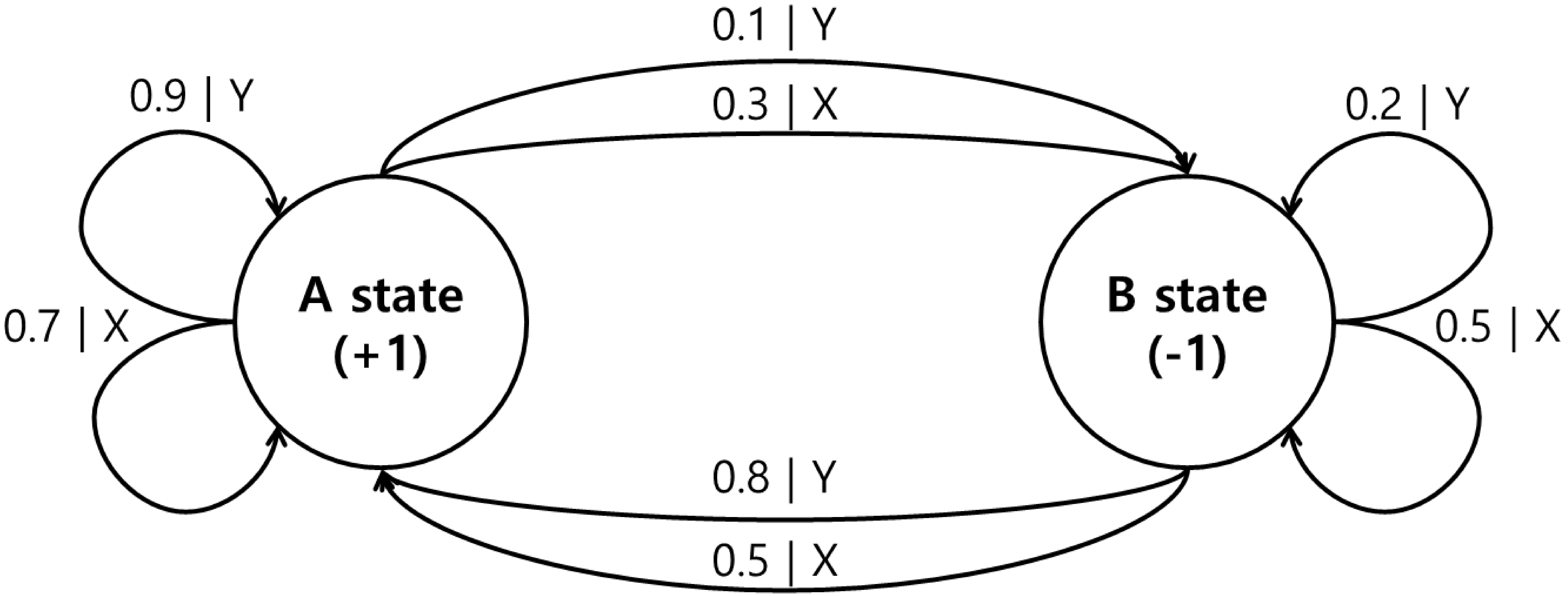
# Markov Reward Processes

- Discrete state space : **S = { A , B }**
- State transition probability : **P(S′|S) = {$P_{AA}$ = 0.7 , $P_{AB}$ = 0.3 , $P_{BA}$ = 0.5 , $P_{BB}$ = 0.5 }**
- Reward function : **R(S′=A) = +1 , R(S′=B) = -1**
- Purpose : Finding an expected reward distribution



# Markov Decision Processes

- Discrete state space : **S = { A , B }**
- Discrete action space : **A = { X, Y }**
- (Action conditional) State transition probability : **P(S'|S , A) = { … }**
- Reward function : **R(S'=A) = +1 , R(S'=B) = -1**
- Purpose : Finding an optimal policy (maximizes the expected sum of future reward)



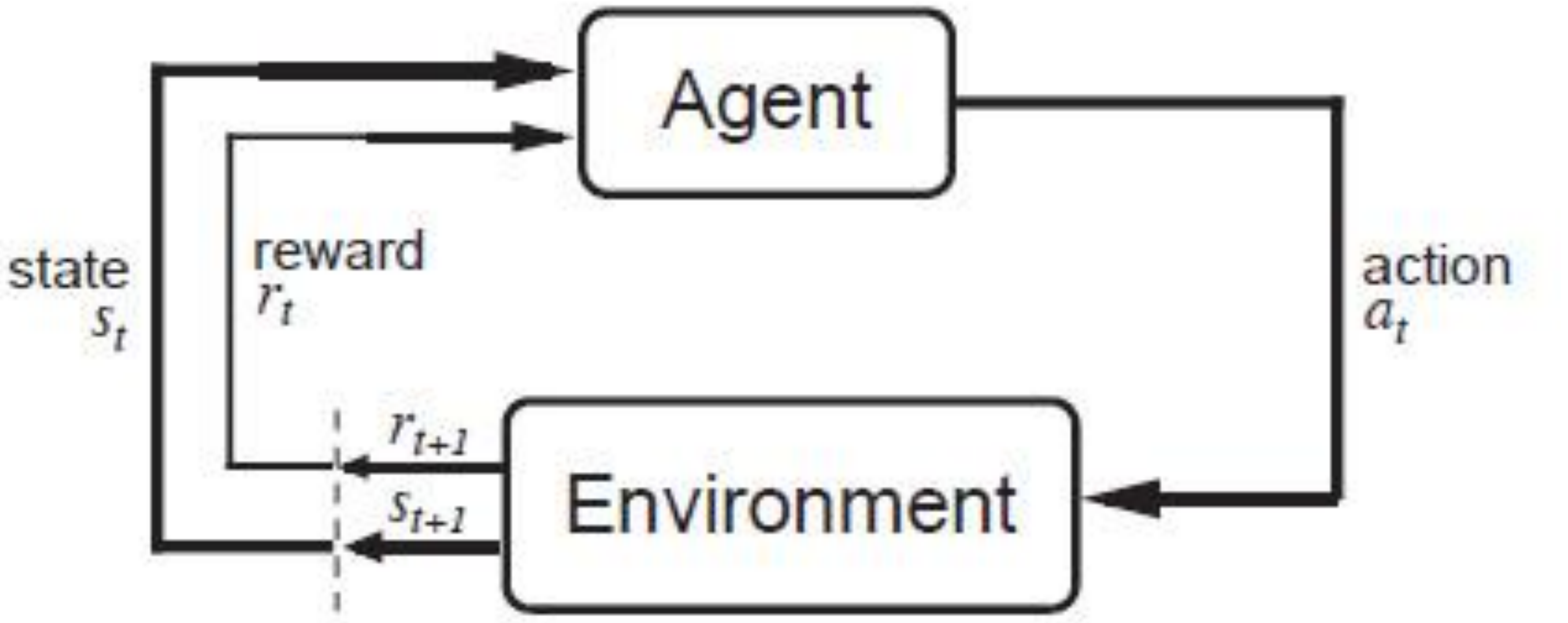
# Markov Decision Processes

- **Markov decision processes** :
  a mathematical framework for modeling decision making.

- MDP are solved via dynamic programming and reinforcement learning.
- Applications : robotics, automated control, economics and manufacturing.

- Examples of MDP :
  1) AlphaGo에서는 바둑을 MDP로 정의
  2) 자동차 운전을 MDP로 정의
  3) 주식시장을 MDP로 정의

# Agent-Environment Interaction




- Objective : Finding an optimal policy which maximizes the expected sum of future rewards
- Algorithms
  1) **Planning** : Exhaustive Search / Dynamic Programming
  2) **Reinforcement Learning** : MC method / TD Learning(Q-learning , ...)

# Discount Factor

- Sum of future rewards **in episodic tasks**

➔ $G_t := R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$

- Sum of future rewards **in continuous tasks**

➔ $G_t := R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T + \ldots$

$G_t \rightarrow \infty$ (diverge)

- Sum of discounted future rewards **in both case**

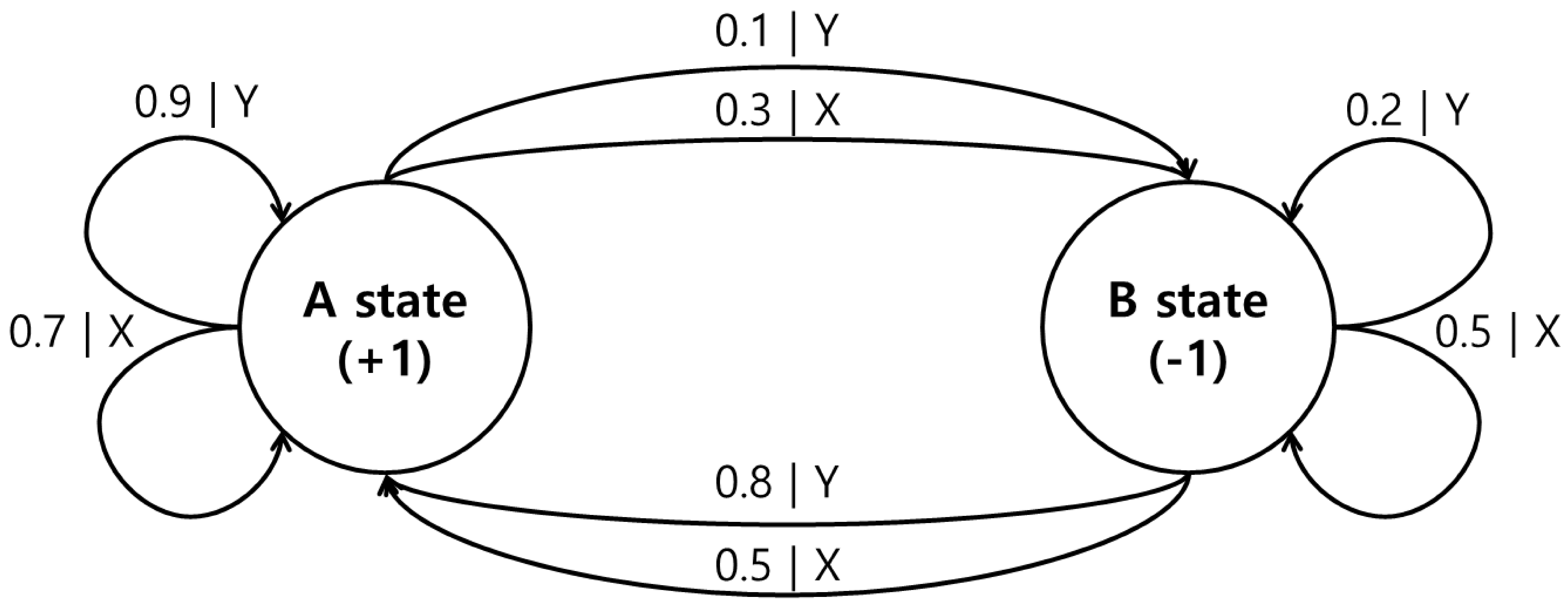
➔ $G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T + \ldots$

$$= \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \text{ (converge)}$$

($R_t$ is bounded / $\gamma$ : discount factor, $0 <= \gamma < 1$)

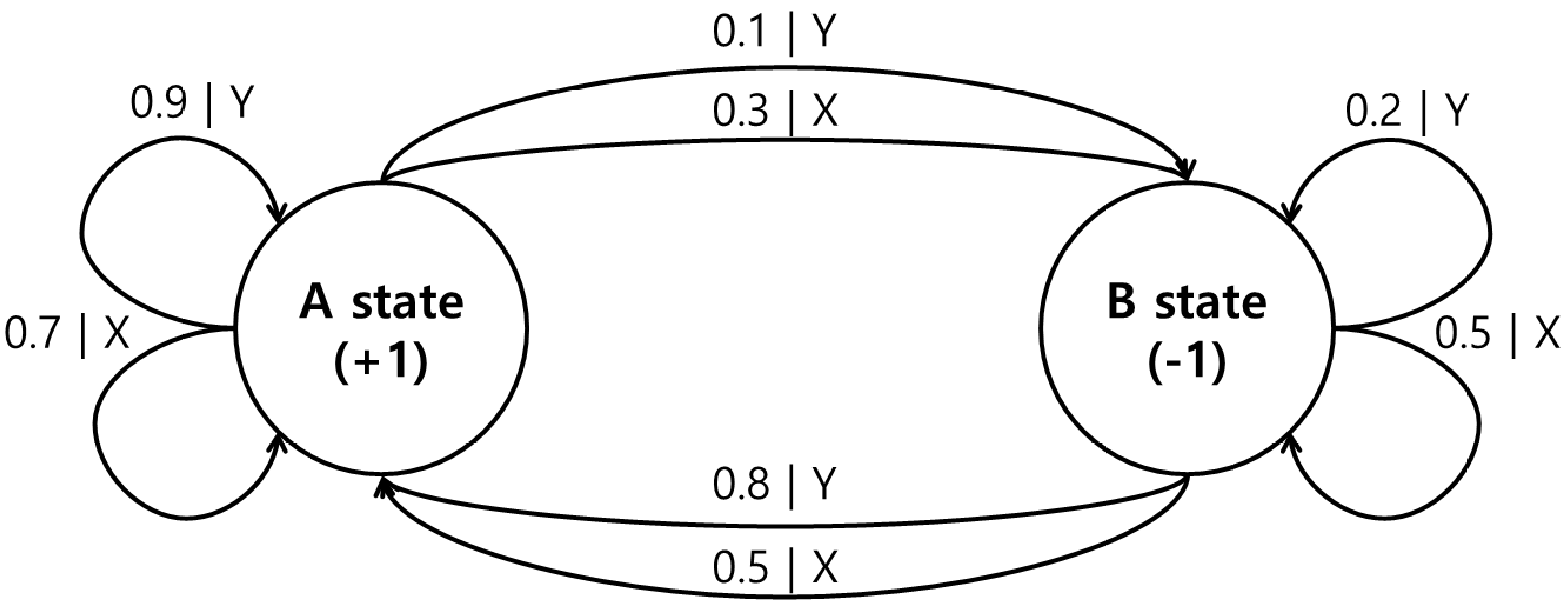
# Deterministic Policy, a=f(s)

| State S | f(S) |
|---|---|
| A | Y |
| B | Y |

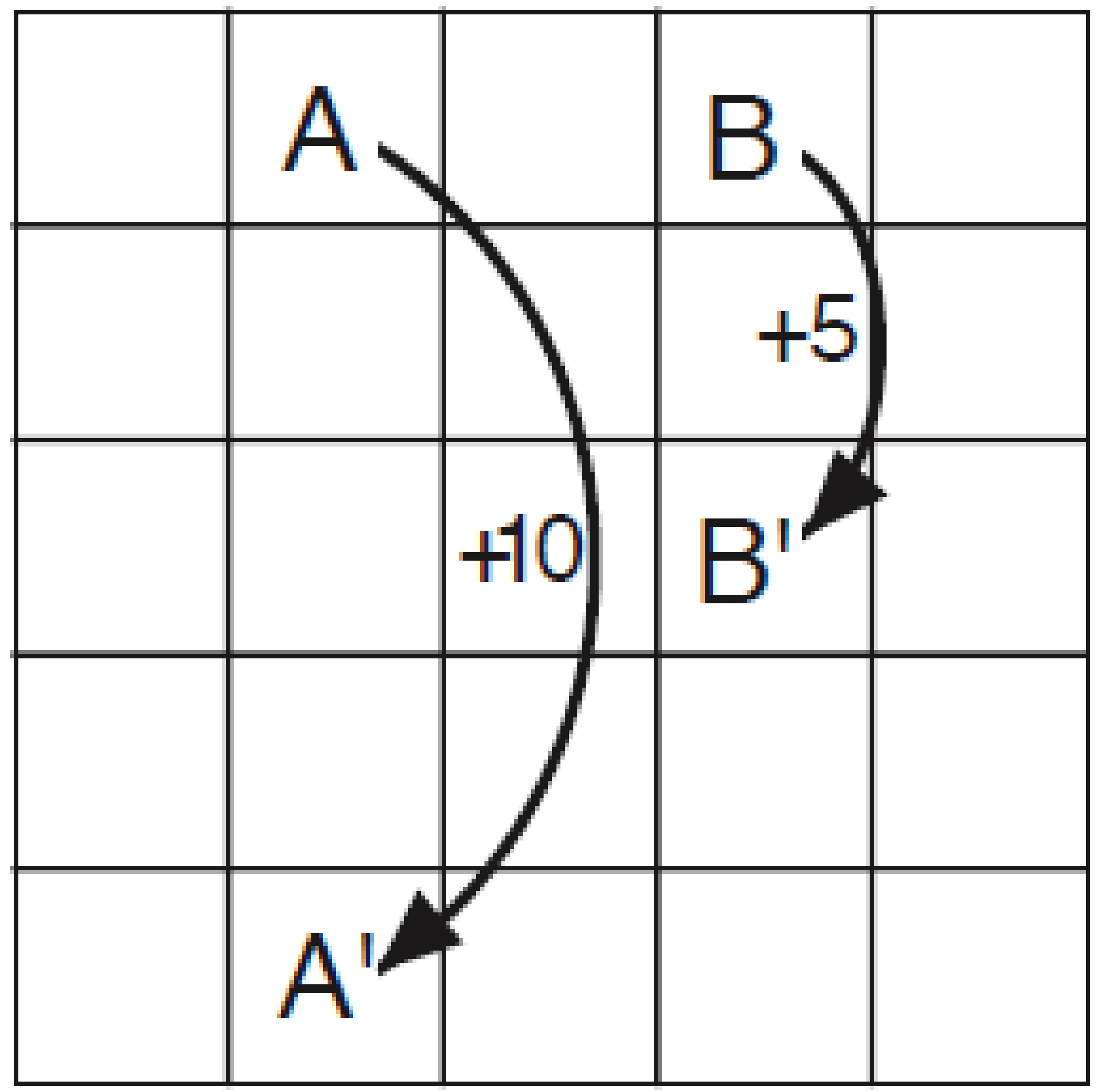


# Stochastic Policy, p(a|s)

| State S | P(X\|S) | P(Y\|S) |
|---|---|---|
| A | 0.3 | 0.7 |
| B | 0.4 | 0.6 |



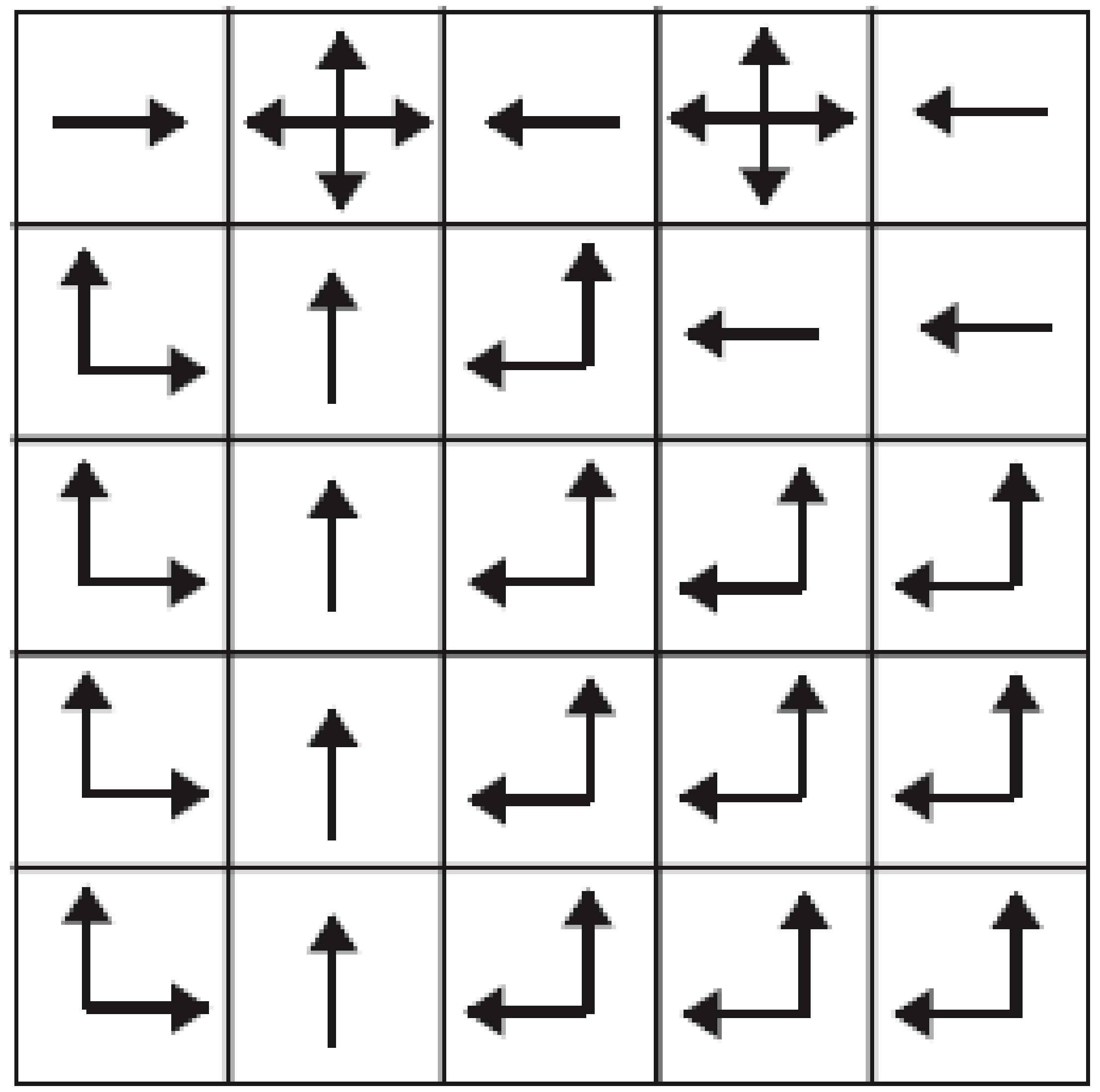
# Gridworld




a) gridworld

| | | | | |
|---|---|---|---|---|
| 3.3 | 8.8 | 4.4 | 5.3 | 1.5 |
| 1.5 | 3.0 | 2.3 | 1.9 | 0.5 |
| 0.1 | 0.7 | 0.7 | 0.4 | -0.4 |
| -1.0 | -0.4 | -0.4 | -0.6 | -1.2 |
| -1.9 | -1.3 | -1.2 | -1.4 | -2.0 |

b) $v$



c) $\pi_*$

- State : 5x5
- Action : 4방향이동
- Reward : A에 도착하면 +10, B에 도착하면 +5, 벽에 부딪히면 -1, 그이외 0
- Discounted Factor : 0.9

# Value-based Approach

# Value Function

- We will introduce a **value function** which let us know the expected sum of future rewards at given state s, following policy π.

## 1) State-value function

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

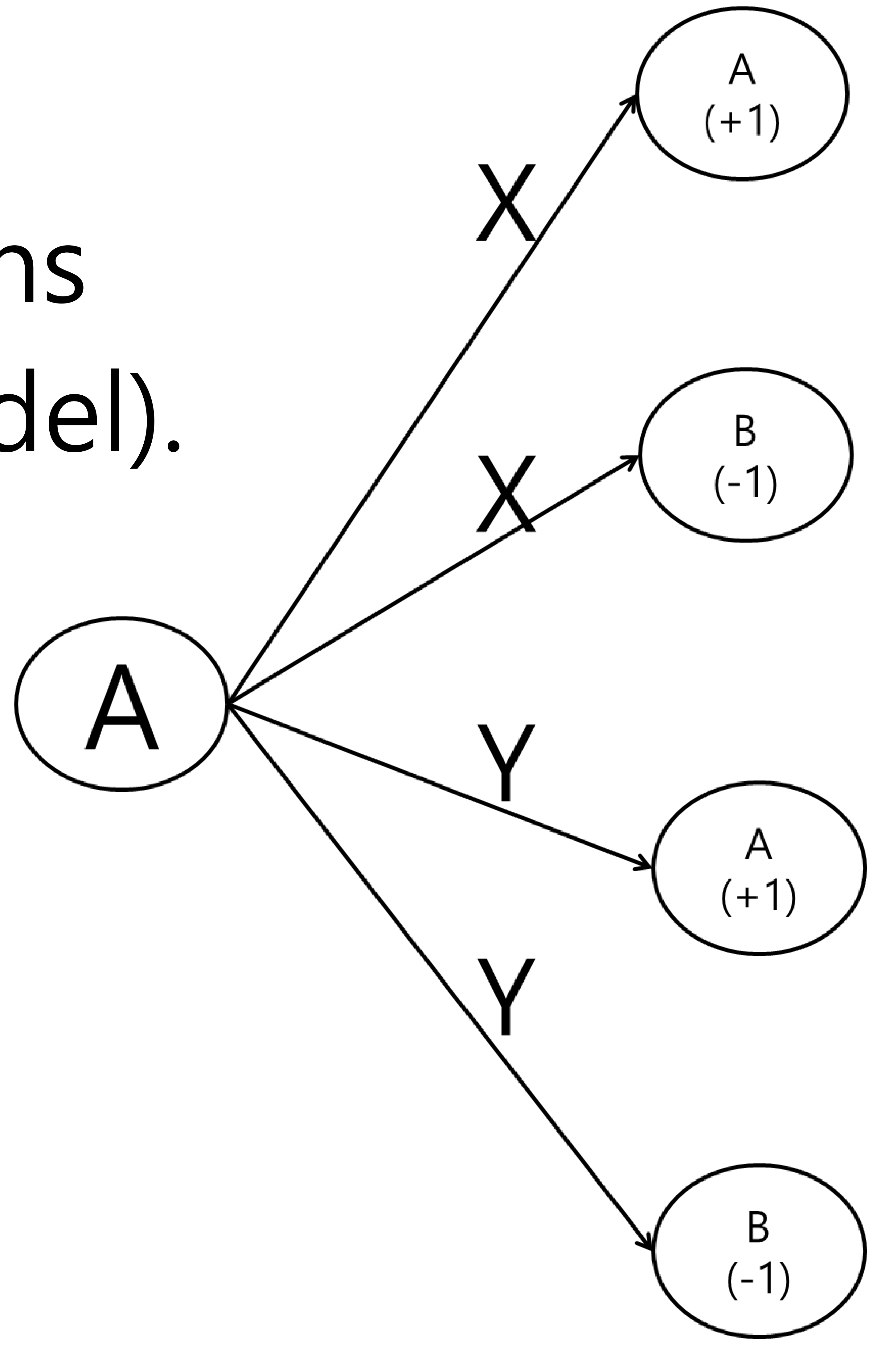
## 2) Action-value function

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

# Optimal Control(Policy) with Value Function

## 1) Policy from state-value function

→ One-step ahead search for all actions
with state transition probability(model).




## 2) Policy from action-value function

→ a = f(s) = $argmax_a\{q_\pi(s,a)\}$

# Planning vs Learning

- Objective : Finding an optimal policy which maximizes the expected sum of future rewards

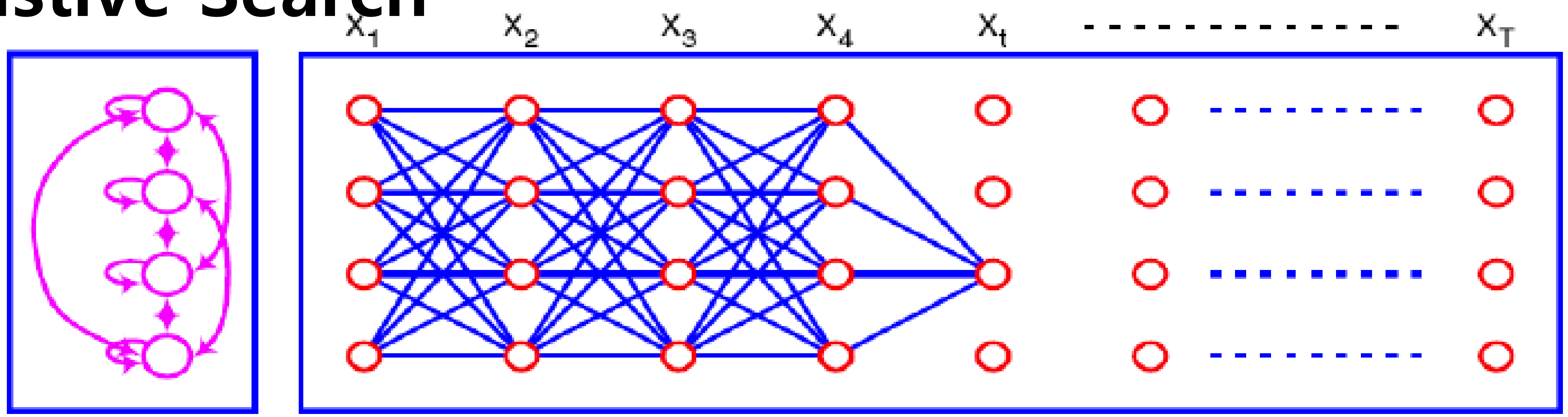
- Algorithms
  1) **Planning** : Exhaustive Search / Dynamic Programming
  ➔ 주사위 눈의 평균 = 1*1/6 + 2*1/6 + … + 6*1/6 = 3.5

  2) **Reinforcement Learning** : MC method / TD Learning
  ➔ 주사위 눈의 평균 = 100번을 던져서 나온 눈을 평균 냄 = 3.5

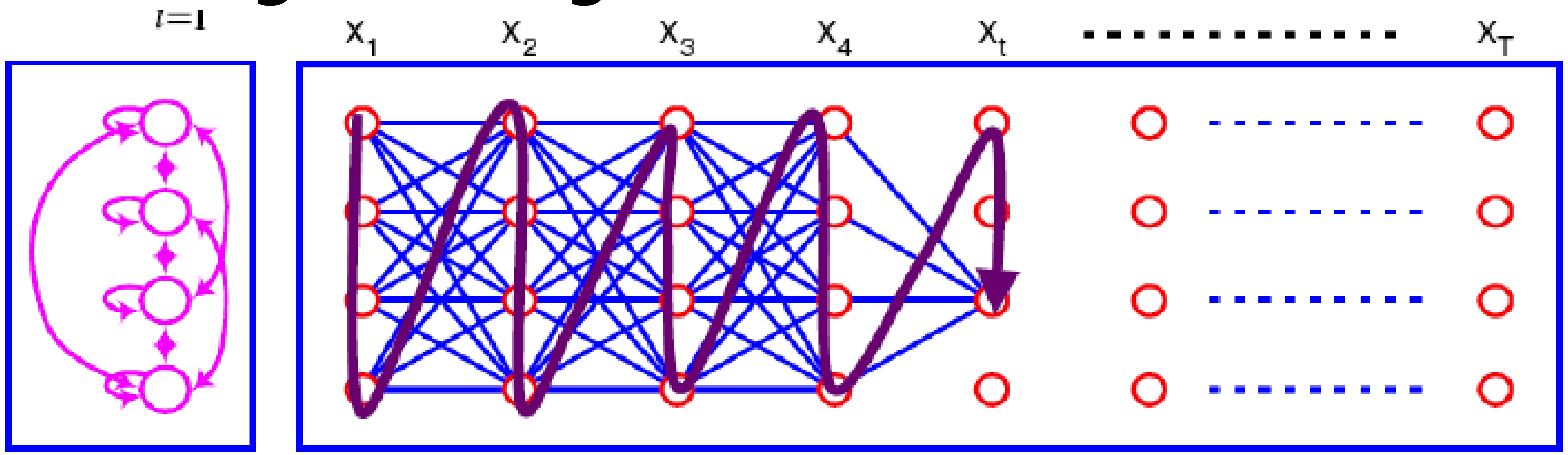
# Solution of the MDP : Planning

- So our last job is find optimal policy and there are two approaches.

  1) **Exhaustive Search**



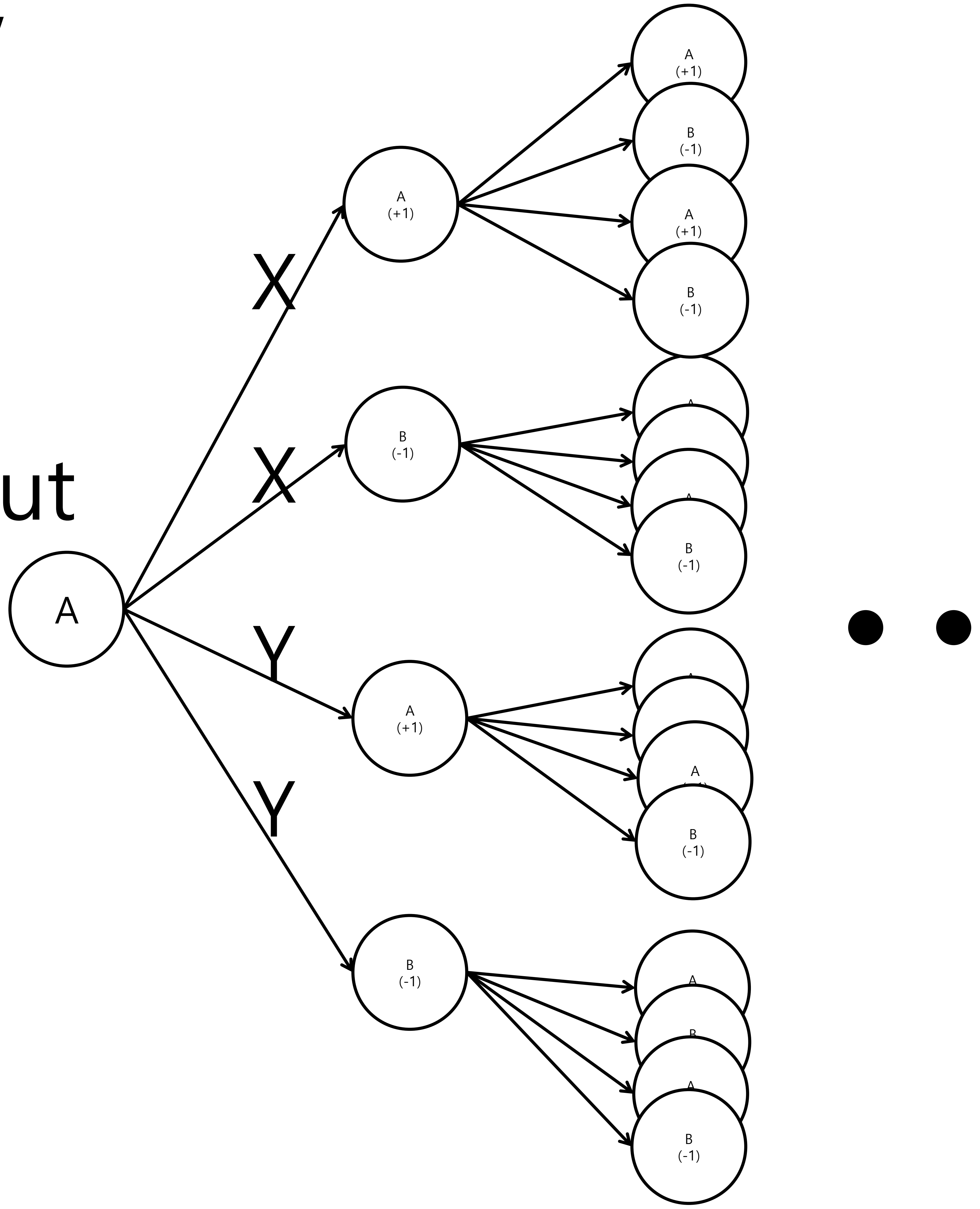

  2) **Dynamic Programming**

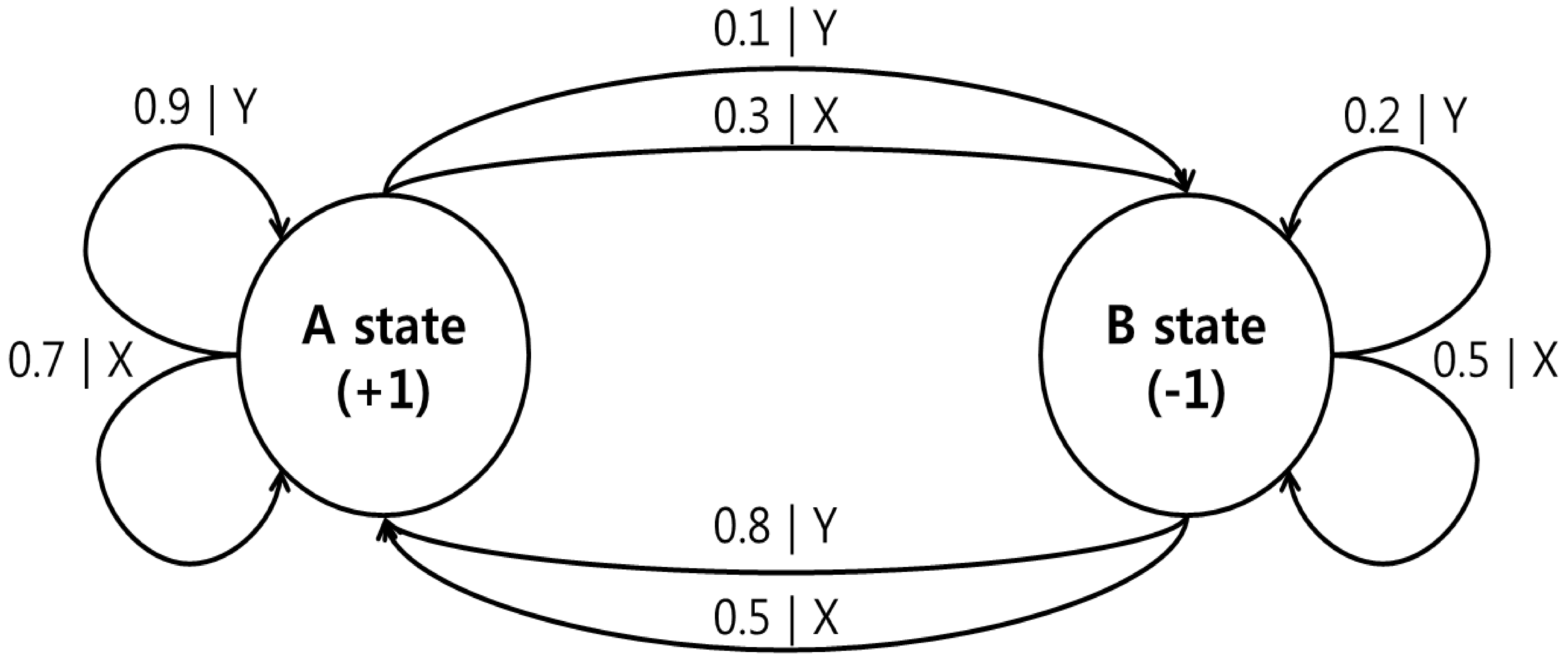


# Find Optimal Policy with Exhaustive Search

If we know the one step dynamics of the MDP, **P(s',r|s,a)** , we can do exhaustive search iteratively until the end step T.

And we can choose the optimal action path, but this needs **O(N$^T$)!**

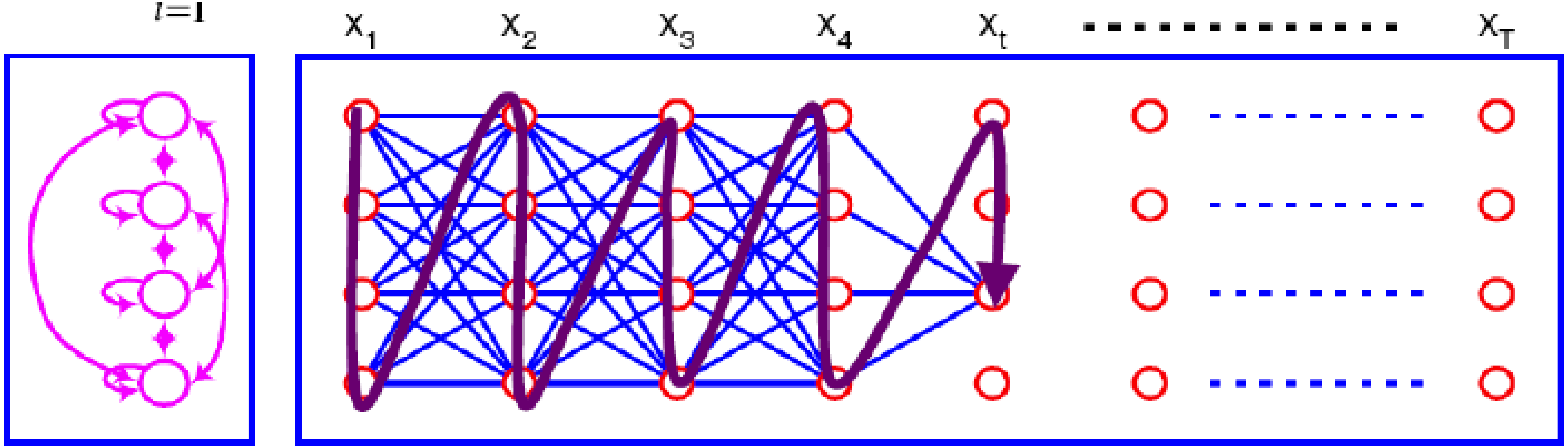


# Dynamic Programming

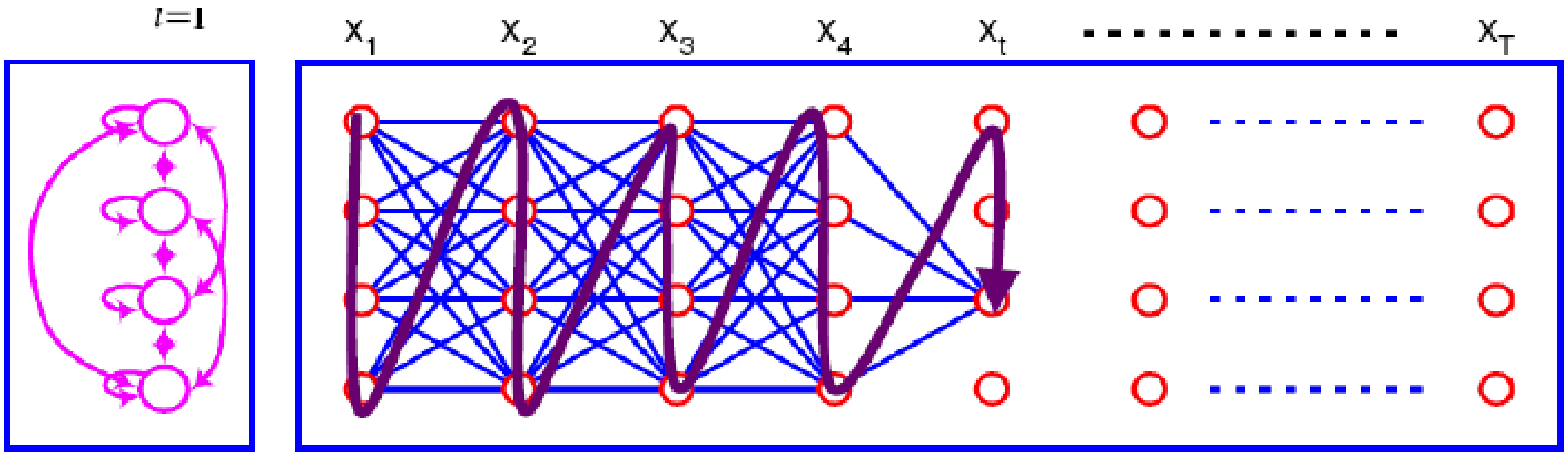
- 전체 문제를 풀 때 중복되는 계산이 발생 →
  이를 subproblem으로 나누어 풀어 중복계산을 없앰

- 전체 path단위로 계산을 하면 앞부분 계산이 항상 중첩 →
  두 step 단위로 subproblem 계산을 끝내고 다음 step으로 넘어감.



# Dynamic Programming

- We can apply the DP in this problem, and the computational cost reduces to **O($N^2$T).** (<u>But still we need to know the environment dynamics.</u>)
- DP is a computer science technique which calculates the final goal value with compositions of cumulative partial values.



# Policy Iteration

- Policy iteration consists of two simultaneous, interacting processes.

- **Policy evaluation (Iterative formulation):** First, make the value function more precise with the current policy.

- **Policy improvement (Greedy selection):** Second, make greedy policy greedy with respect to the current value function.

# Policy Iteration

- How can we get the state-value function with DP?

  (The state-value is subproblem. And action-value function is also similarly computed.)

◆**Policy Iteration = Policy Evaluation + Policy Improvement**

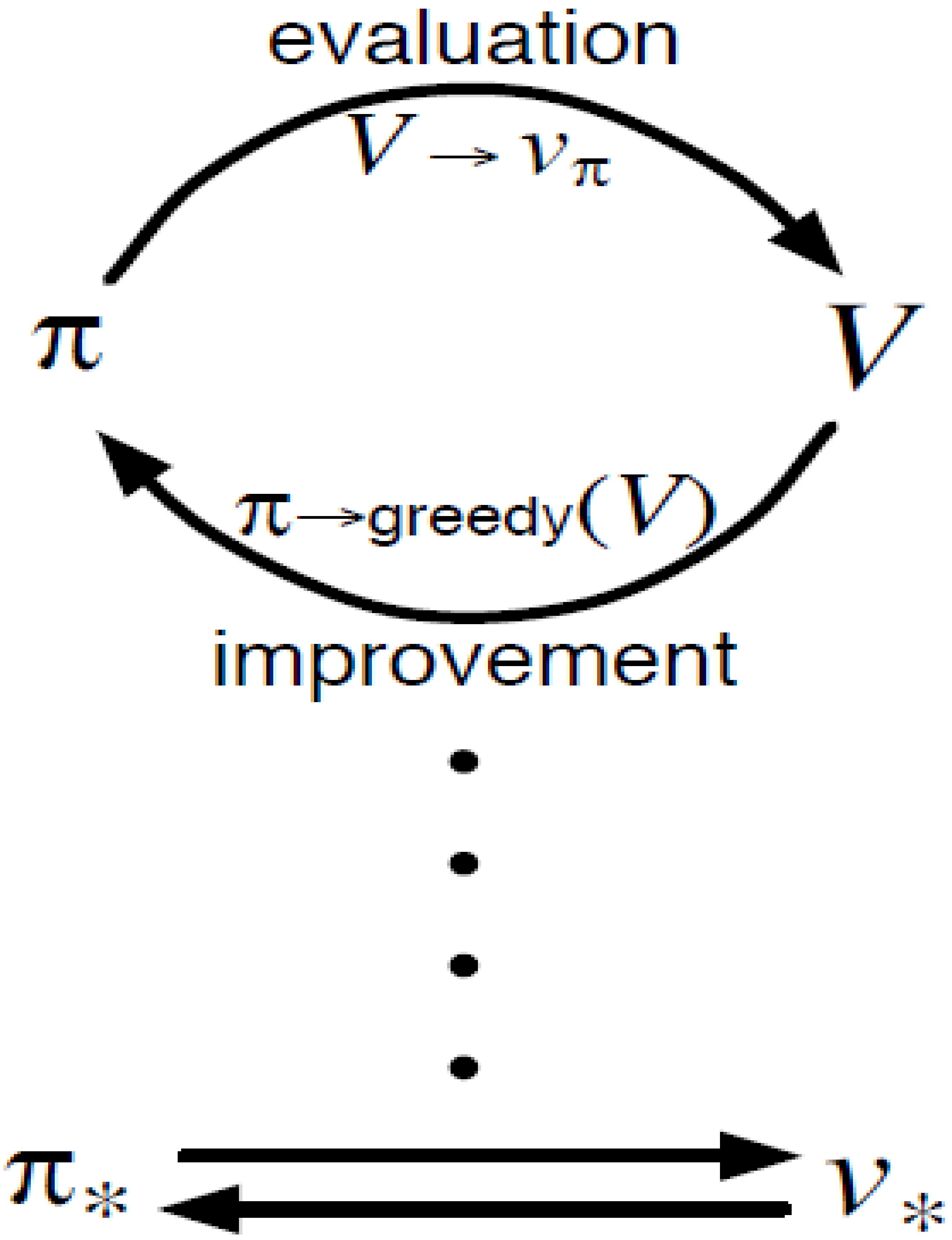
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\
&= \boxed{\mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]} \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}
$$

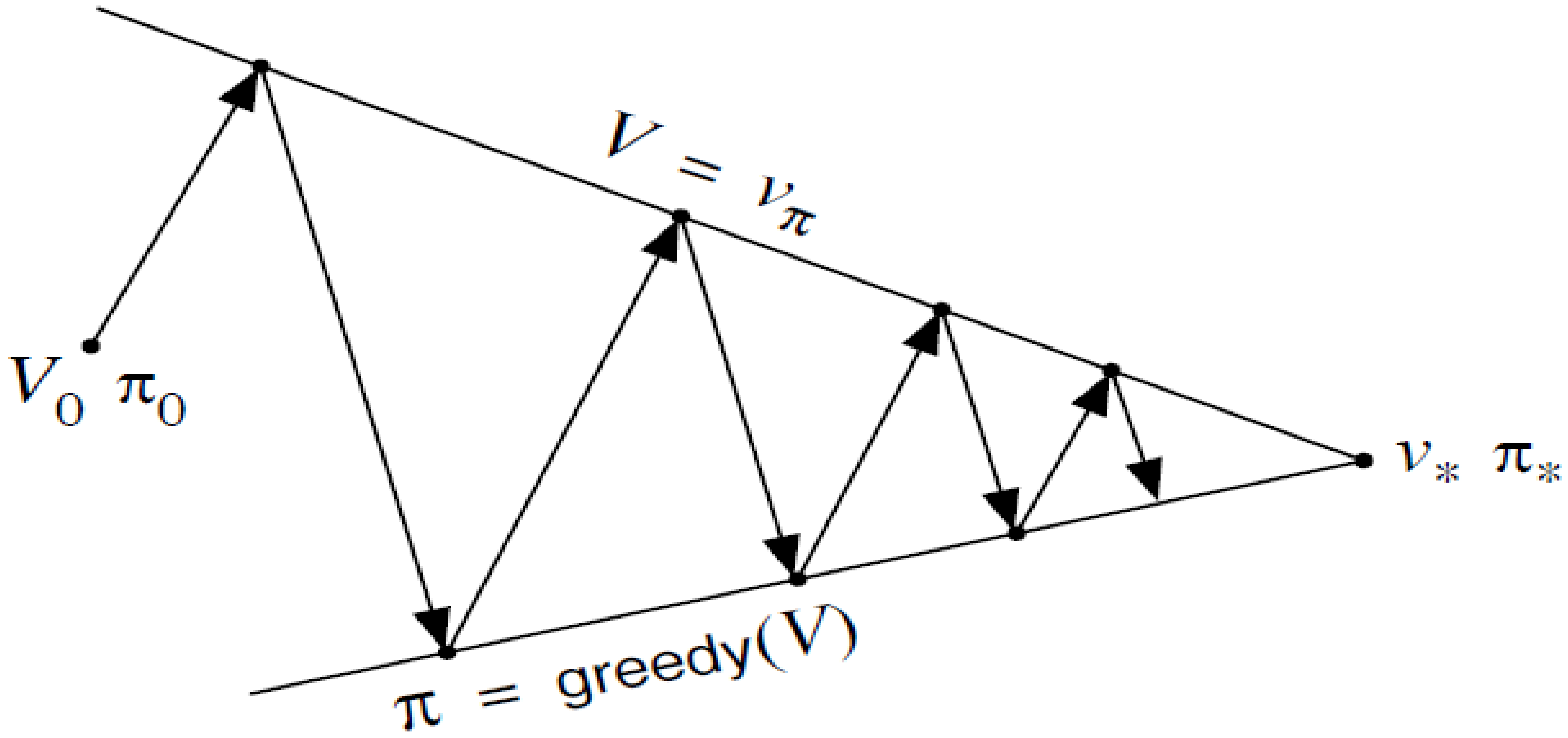
---

$$
\begin{aligned}
\pi'(s) &\doteq \boxed{\arg\max_a q_\pi(s, a)} \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}
$$

# Policy Iteration

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$



# Solution of the MDP : Learning

- The planning methods must know the perfect dynamics of environment, **P(s',r|s,a)**
- But typically this is really hard to know and empirically impossible. Therefore we will ignore this and just calculate the mean from samples.
- This is the starting point of the machine learning is embedded.

1) **Monte Carlo Methods**
2) **Temporal-Difference Learning (a.k.a reinforcement learning)**

# Monte Carlo Methods

Initialize:
  $\pi \leftarrow$ policy to be evaluated
  $V \leftarrow$ an arbitrary state-value function
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

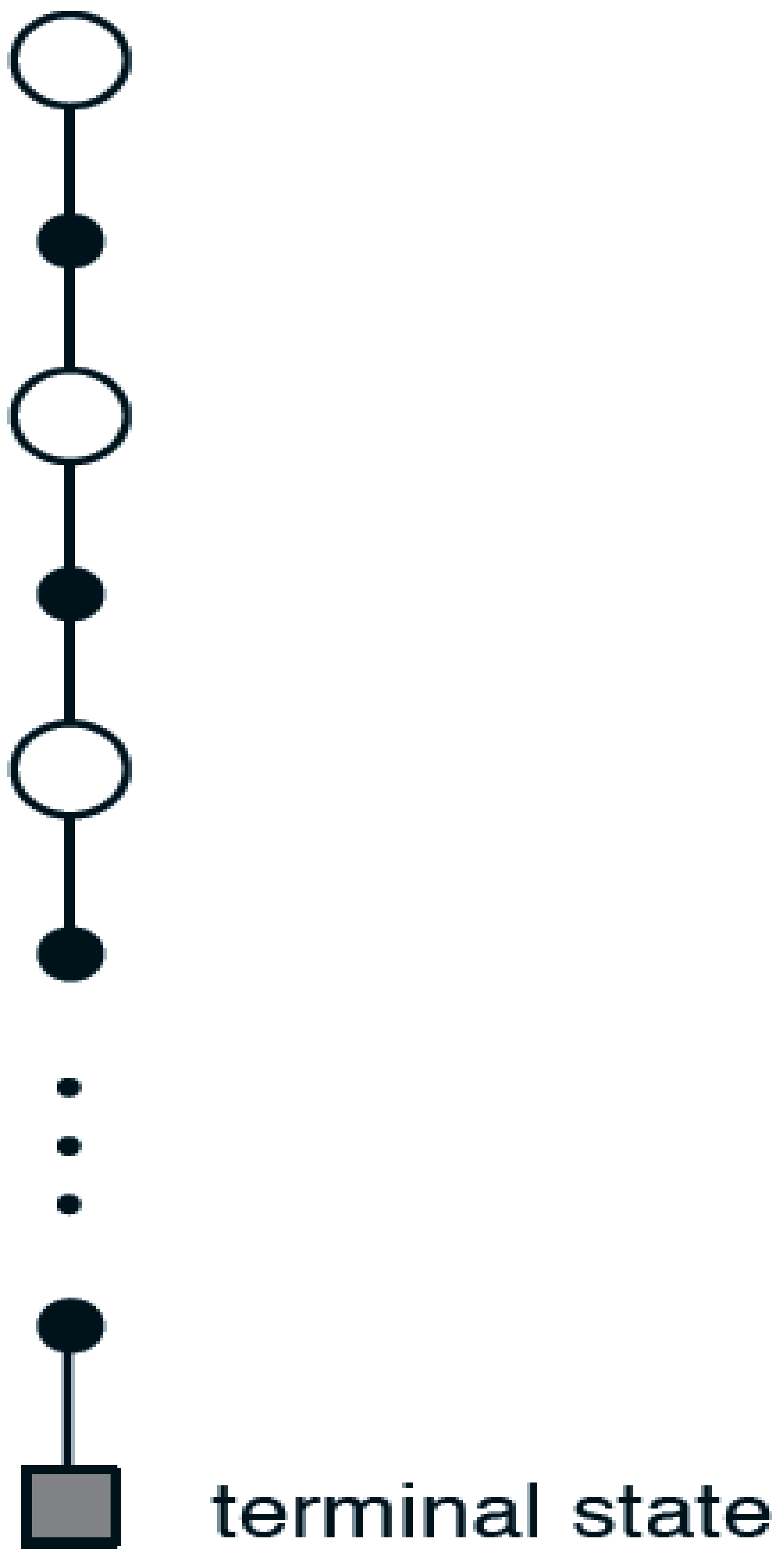
Repeat forever:
  Generate an episode using $\pi$
  For each state $s$ appearing in the episode:
    $G \leftarrow$ return following the first occurrence of $s$
    Append $G$ to $Returns(s)$
    $V(s) \leftarrow \text{average}(Returns(s))$

Tabular state-value function

| Starting State | Value |
|---|---|
| S1 | Average of G(S1) |
| S2 | Average of G(S2) |
| S3 | Average of G(S2) |

# Monte Carlo Methods

- We need a full length of experience for each starting state.
- This is really time consuming to update one state while waiting the terminal of episode.
- Continuous task에는 적용 불가



# Q-learning (Temporal Difference Learning)

# Bellman Equation

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right]
\end{aligned}
$$

$$\boxed{= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]}$$ **(iterative formulation)**

---

$$q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

# Temporal-Difference Learning

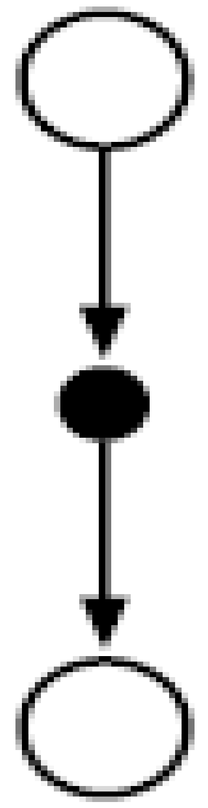
- TD learning is a combination of Monte Carlo idea and dynamic programming (DP) idea.

- Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.

- Like DP, TD methods update estimates based in part without waiting for a final outcome (they bootstrap).

# Temporal-Difference Learning

Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
        $S \leftarrow S'$
    until $S$ is terminal



TD(0)

# Q-learning Algorithm

$$\underbrace{Q(s_t, a_t) \leftarrow Q(s_t, a_t)}_{\text{The New Action Value = The Old Value}} + \underbrace{\alpha}_{\text{The Learning Rate}} \left[\underbrace{r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a)}_{\text{The New Information}} - \underbrace{Q(s_t, a_t)}_{\text{The Old Information}}\right]$$

The New Action Value = The Old Value + The Learning Rate × ( The New Information – The Old Information )

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until $s$ is terminal

# Temporal-Difference Learning


Figure 6.2: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

# On policy / Off policy

- **On policy** : Target policy = Behavioral policy
  there can be only one policy.

  → This can learn a stochastic policy. Ex) SARSA

- **Off policy** : Target policy != Behavioral policy
  there can be several policies.

  → Sample efficient. Ex) Q-learning

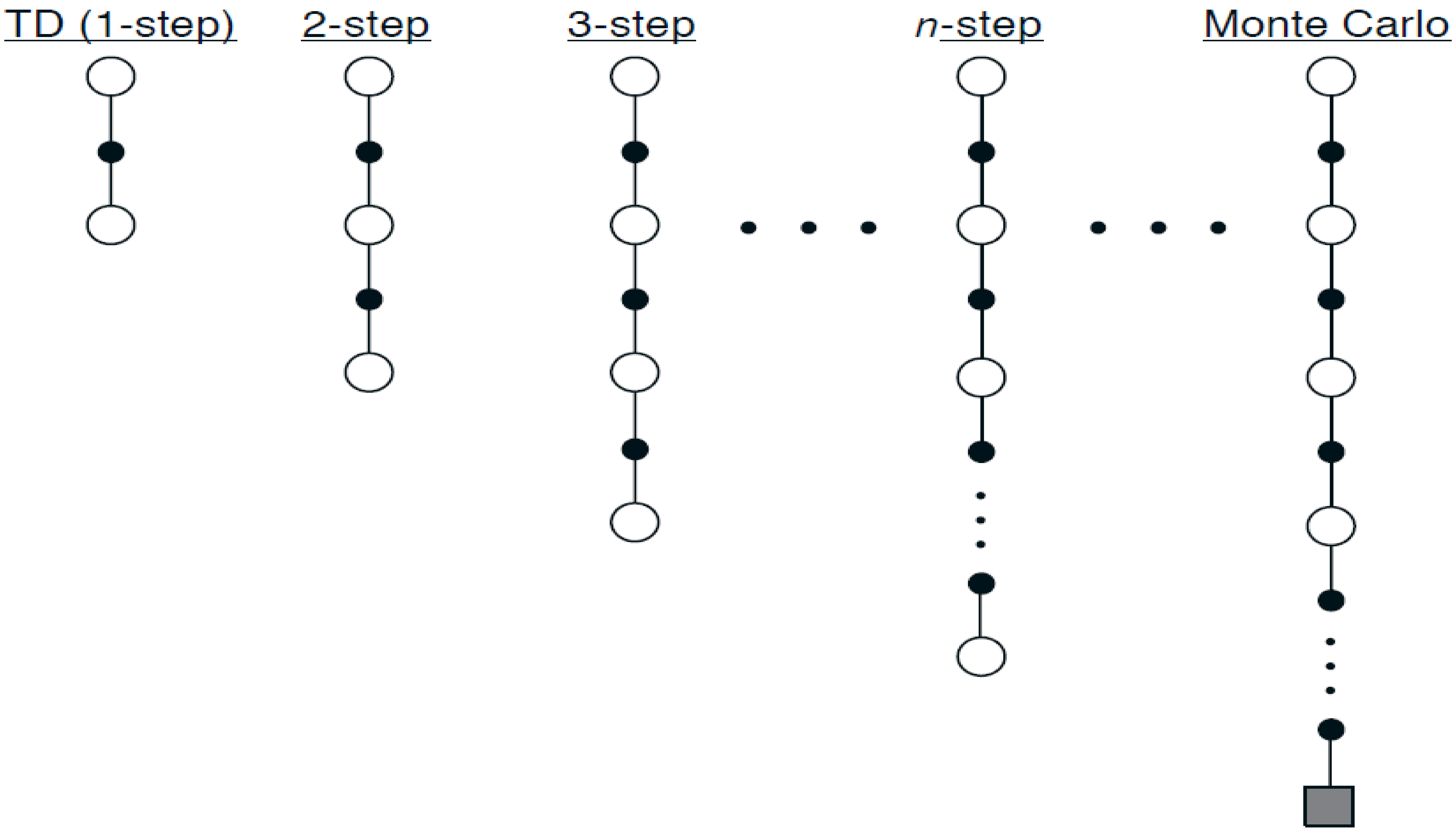
# Sarsa: On-Policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[ R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

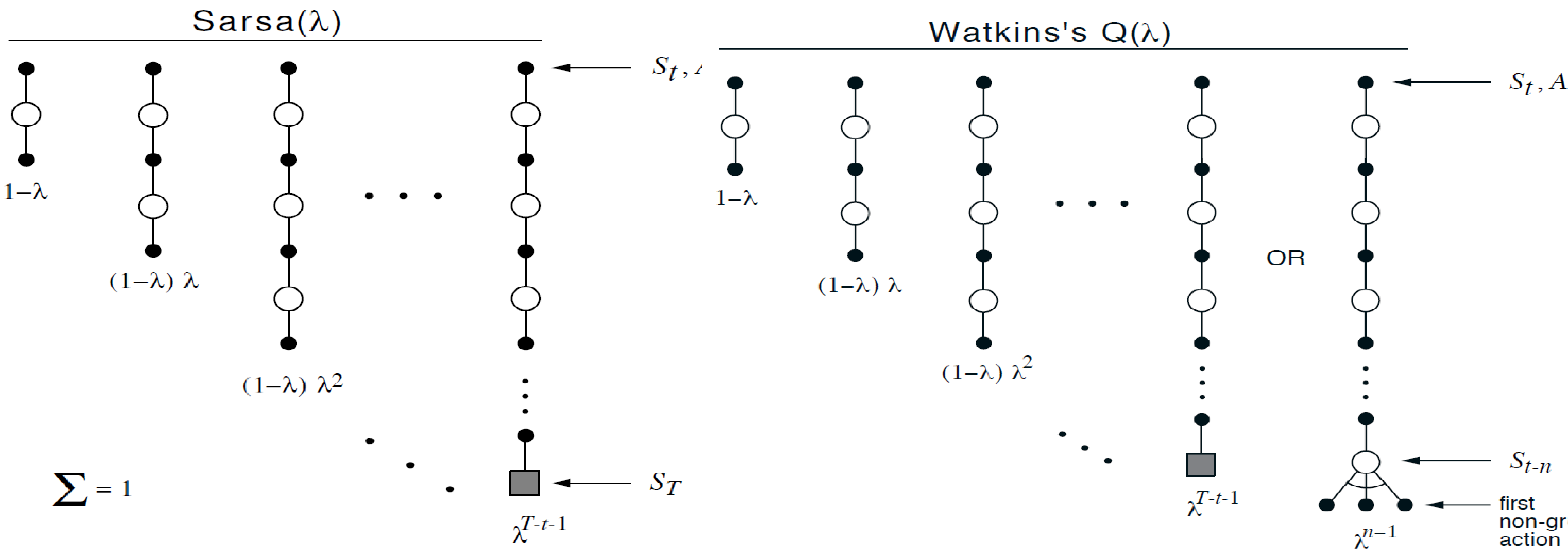
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-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

# Eligibility Trace

- **Smoothly combining the TD and MC.**

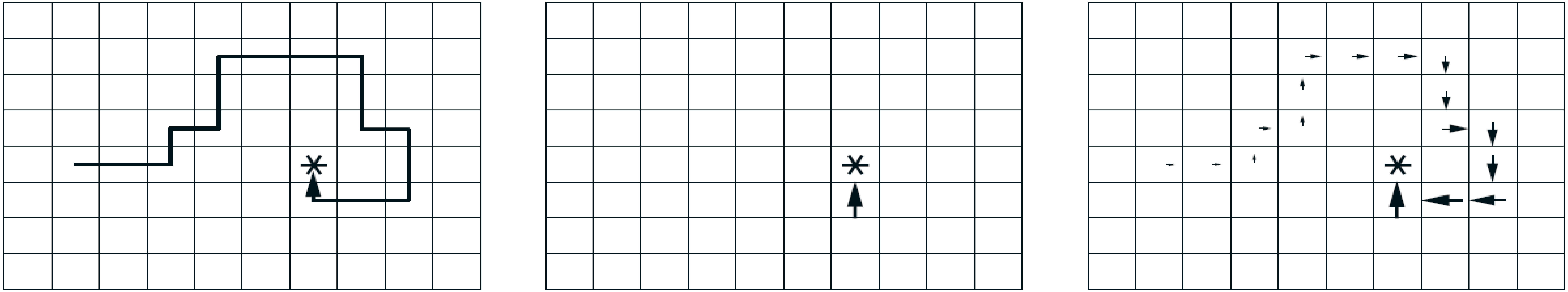


# Eligibility Trace



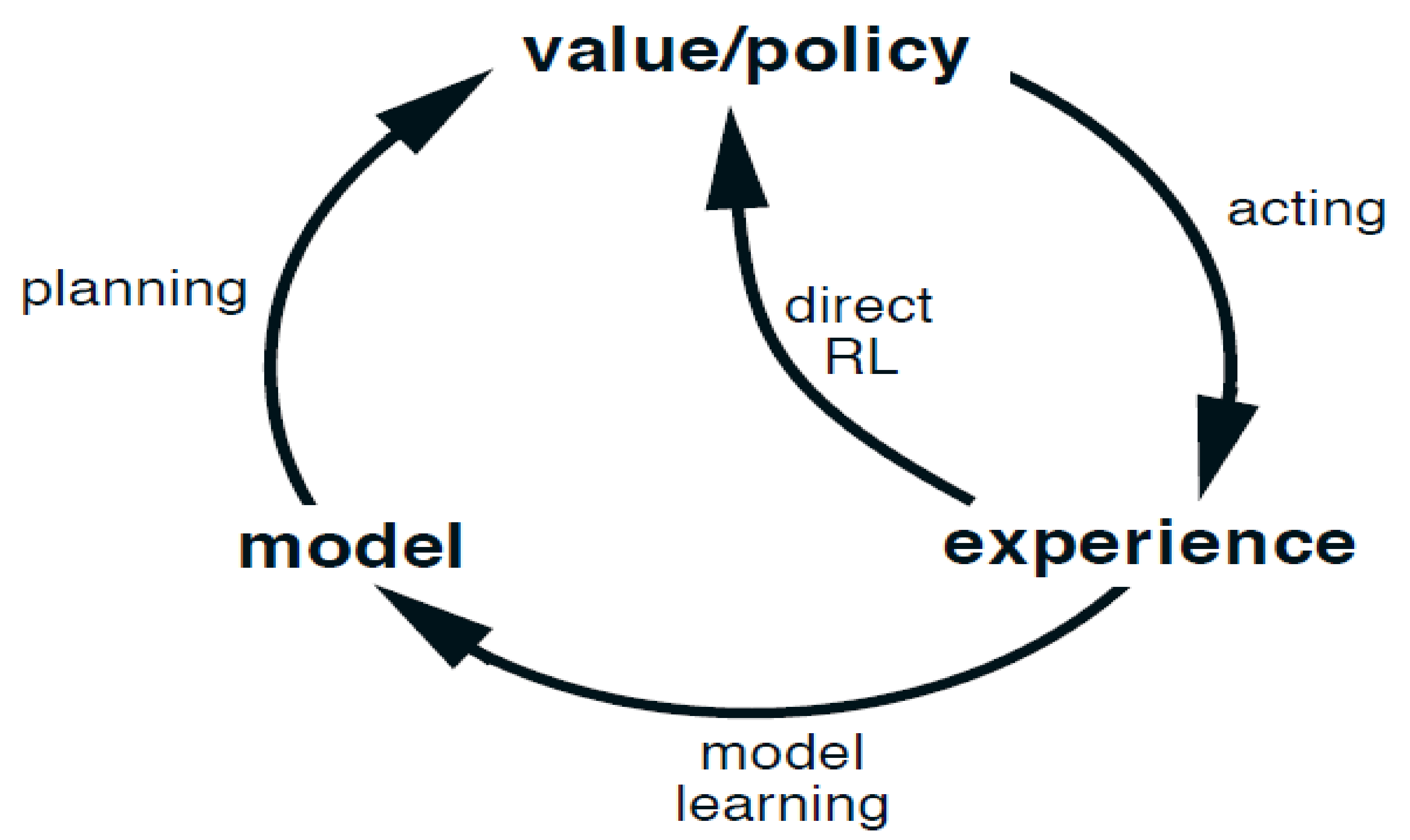
# Comparisons

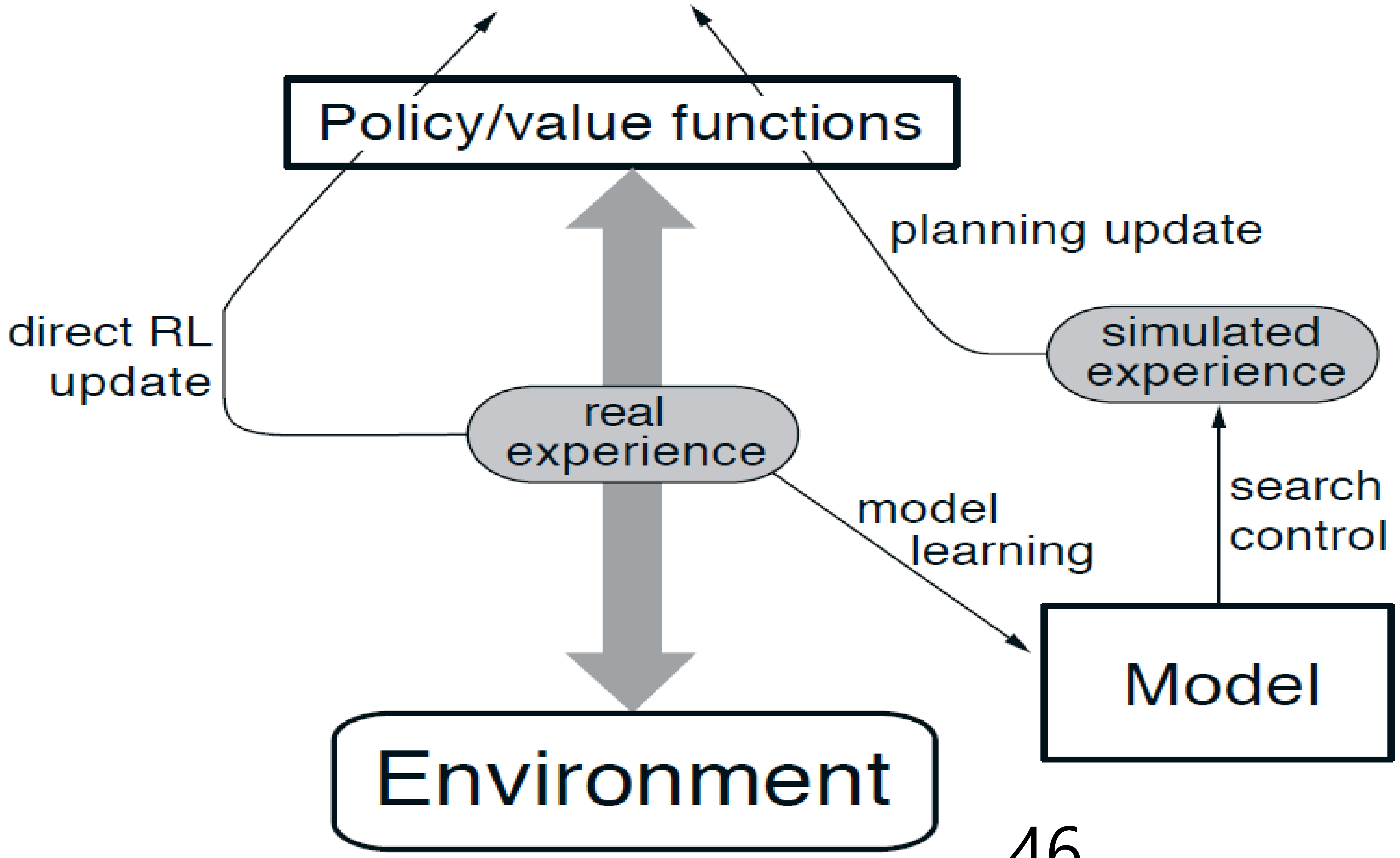
Path taken | Action values increased by one-step Sarsa | Action values increased by Sarsa($\lambda$) with $\lambda$=0.9



# Planning + Learning

- There is only a difference between planning and learning. That is the existence of model.
- So we call planning is **model-based** method, and learning is **model-free** method.



# Planning + Learning

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow \epsilon$-greedy$(S, Q)$
  (c) Execute action $A$; observe resultant reward, $R$, and state, $S'$
  (d) $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
  (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
  (f) Repeat $n$ times:
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in $S$
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

Figure 8.4: Dyna-Q Algorithm. $Model(s, a)$ denotes the contents of the model (predicted next state and reward) for state–action pair $s, a$. Direct reinforcement learning, model-learning, and planning are implemented by steps (d), (e), and (f), respectively. If (e) and (f) were omitted, the remaining algorithm would be one-step tabular Q-learning.

# Deep Reinforcement Learning

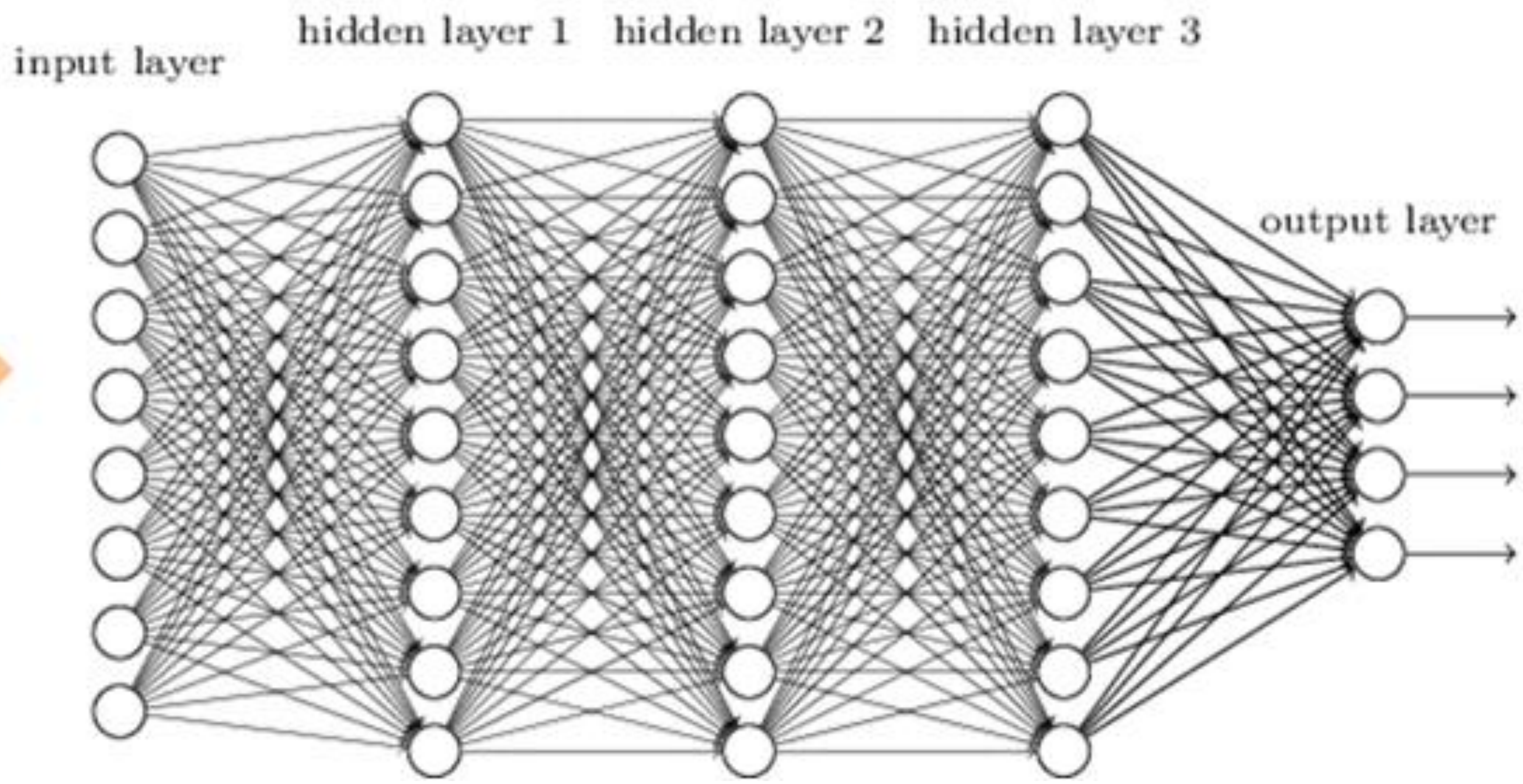
# Generalization with deep learning

- Value function approximation with deep learning
  → **Large scale or infinite dimensional state can be solvable**

Tabular Q value function

$$\hat{\mathbf{Q}} = \begin{array}{c} \textit{state} \backslash \textit{action} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \left[\begin{array}{cccccc} - & - & - & - & 80 & - \\ - & - & - & 64 & - & 100 \\ - & - & - & 64 & - & - \\ - & 80 & 51 & - & 80 & - \\ 64 & - & - & 64 & - & 100 \\ - & 80 & - & - & 80 & 100 \end{array}\right] \end{array}$$

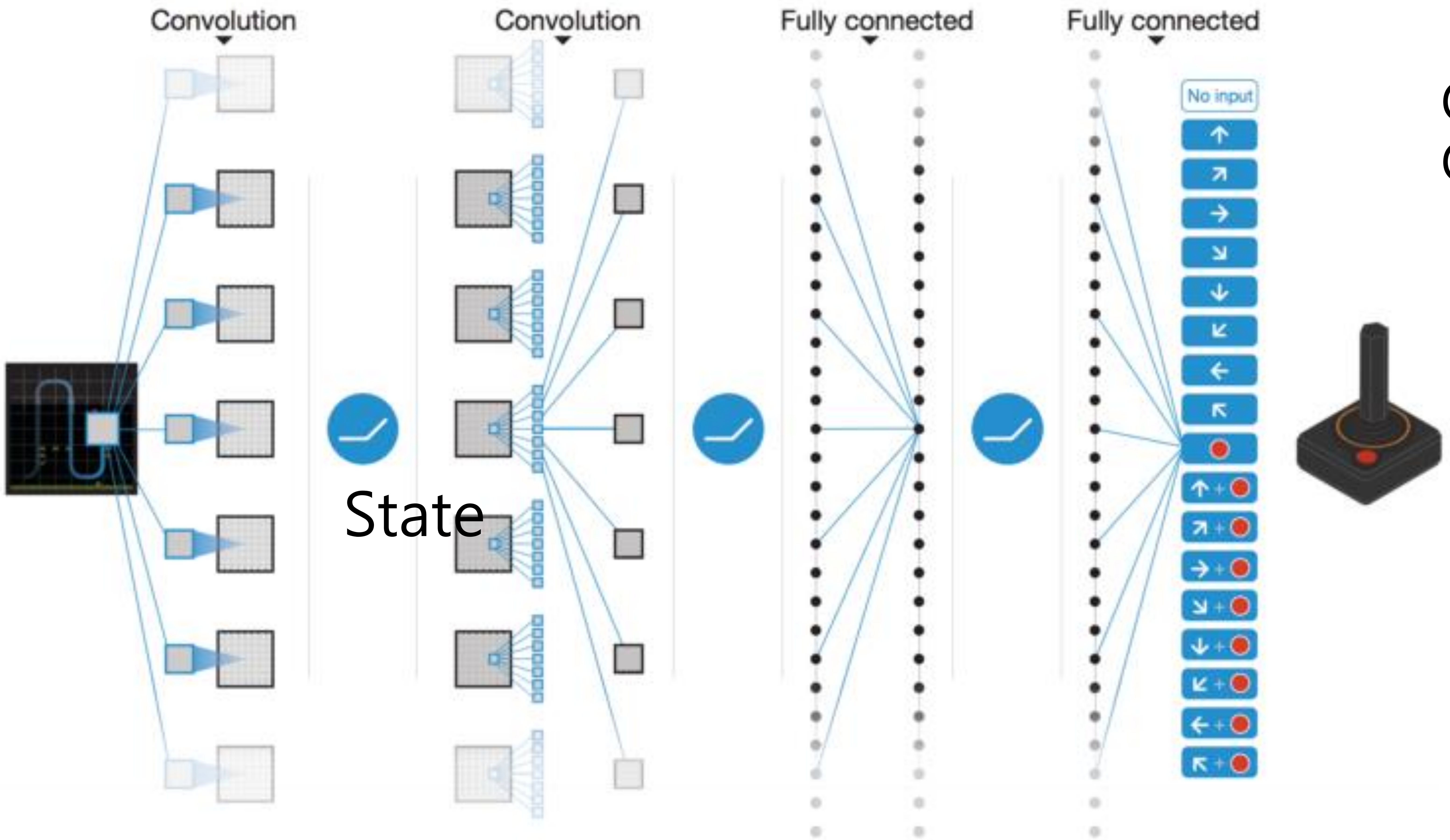
Deep neural network




This needs supervised learning techniques and online moving target regression.

# Deep Q Networks (DQN)

- Q learning에서 특정 state에 대한 action-value function으로 CNN을 사용




Q(State,Action=0)
Q(State,Action=1)

.
.
.

Q(State,Action=K)

# Loss function

- Q learning의 업데이트 식

TD error

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \boxed{\left[ r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]}.$$

- TD error의 크기를 Loss로 사용, regression 방식으로 학습

$$L = \frac{1}{2}[\underbrace{r + max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

# Target Network

$$L = \frac{1}{2}[\underbrace{r + max_{a'}\boxed{Q(s',a')}}_{\text{target}} - \underbrace{Q(s,a)}_{\text{prediction}}]^2$$

- **Moving Target Regression** :
  Q를 학습하는 순간, target 값도 같이 변화 → 학습 성능 저하

➔ **Target Network** : DQN과 동일한 구조를 가지고 있으며 학습 도중 weight값이 변하지 않는 별도의 네트워크로 Q(s',a')로 **고정된 target value를 사용하여 학습 안정성 개선**

(Target Network의 weight값들은 주기적으로 DQN의 것을 복사)

# Replay Memory

- **Data inefficiency** : 그 동안 경험한 experience들을 off-policy learning으로 재활용(Mini-batch learning)

- **Data correlation** : Mini-batch를 On-line으로 구성할 경우 Mini batch 내의 데이터들이 서로 비슷함(Correlated)
  → 한쪽으로 치우친 불균형한 학습이 발생함

➔ **Replay Memory** : (State, Action, Reward, Next State) 데이터를 Buffer에 저장해놓고 그 안에서 Random Sampling하여 Mini-batch를 구성하는 방법으로 해결

# Reward Clipping

- **Reward Scale problem** : 도메인에 따라 Reward의 scale이 다르기 때문에 학습되는 Q-value의 크기도 제각각.

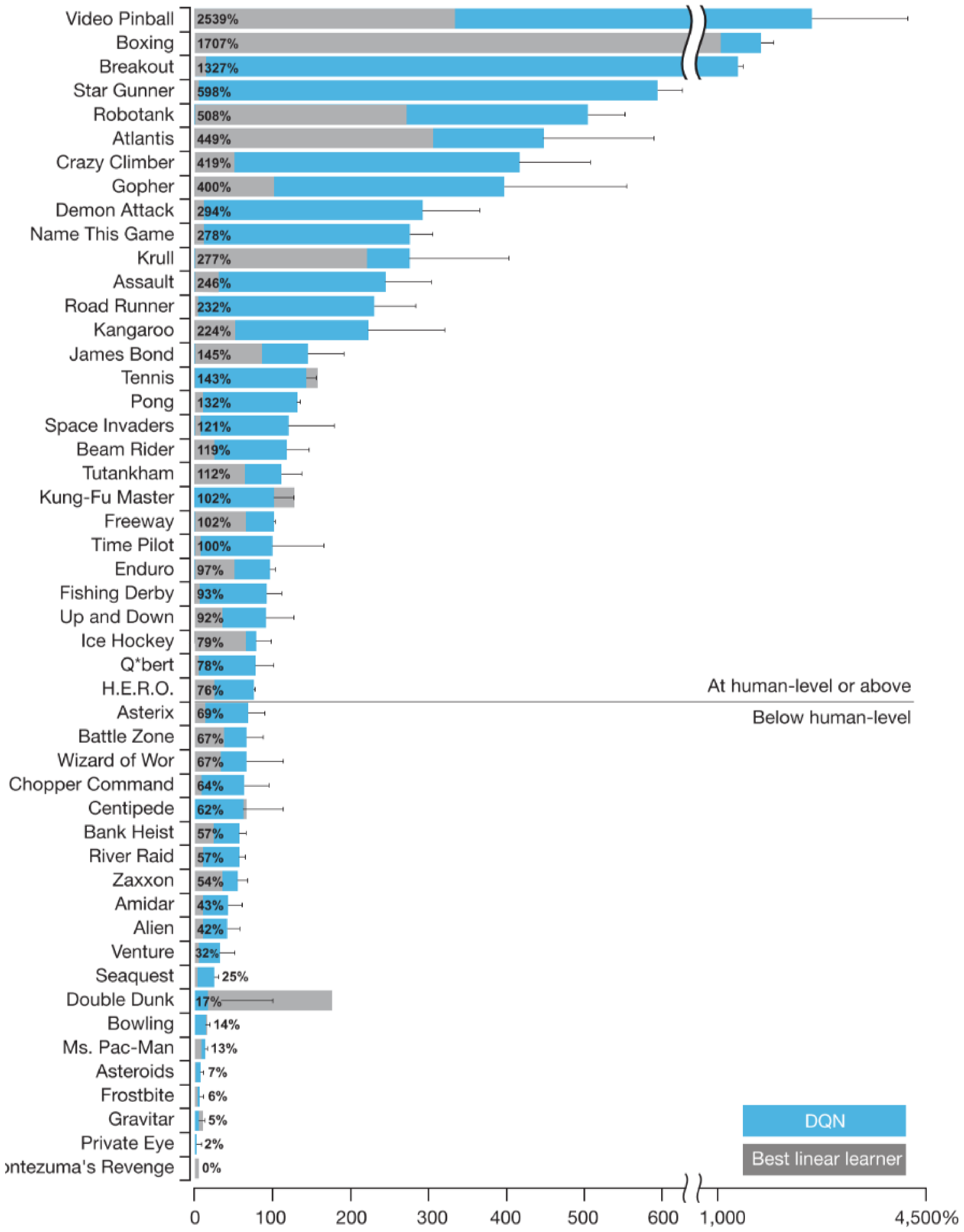
  이렇게 학습해야하는 Q-value의 variance가 매우 큰 경우 (ex : -1000~1000) 신경망 학습이 어려움.

➔ **Reward Clipping**: Reward의 크기를 -1~1 사이로 잘라내어 안정적으로 학습

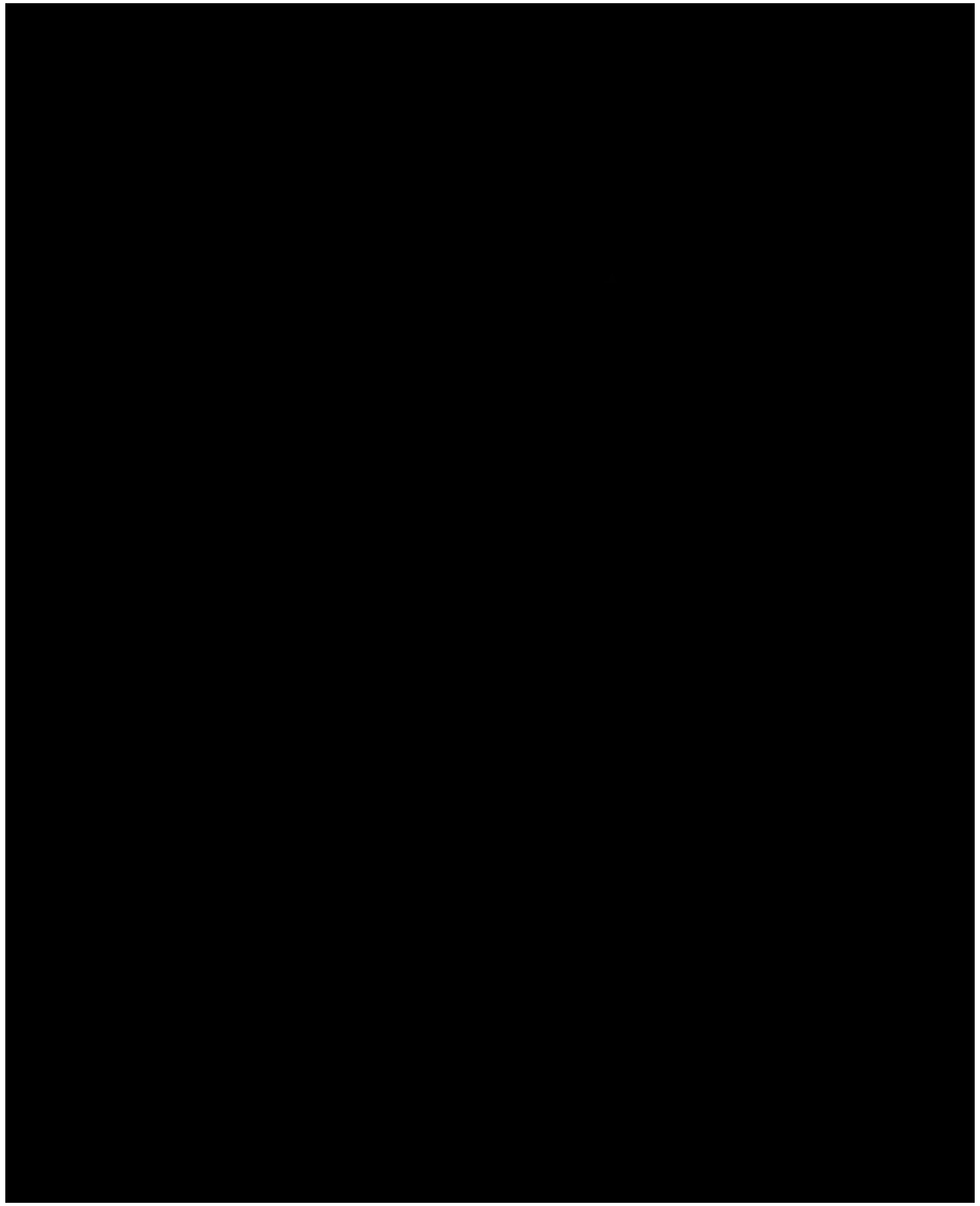
# DQN의 성능

- ATARI 2600 고전게임에서 실험
- 절반 이상의 게임에서 사람보다 우수
- 기존방식 (linear)에 비해 월등한 향상
- 일부 게임은 학습에 실패함.

(sparse reward, 복잡한 단계를 가진 게임)



# DQN 데모



# DQN의 발전된 모델들 - Double DQN

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) .$$

$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t') .$$

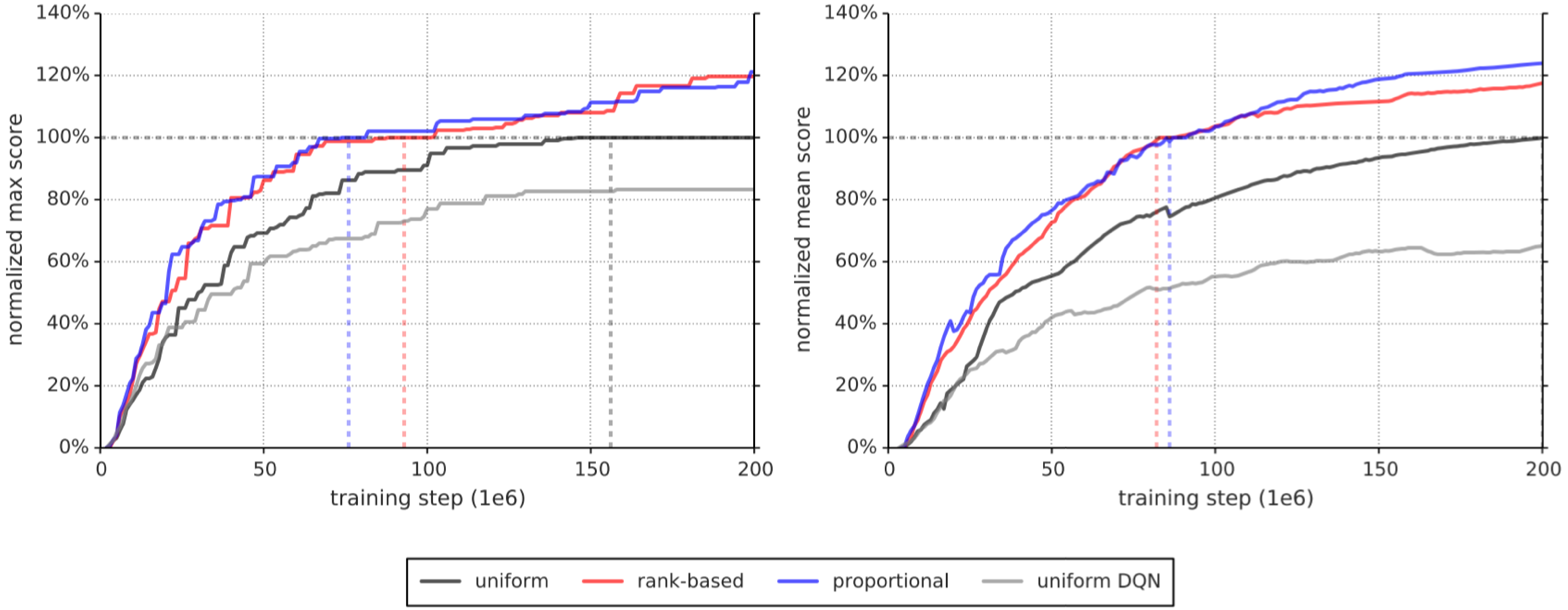
- **Positive bias**: max 연산자에 의해 Q-value를 실제보다 높게 평가하는 경향이 있음

➔ Double Q learning을 사용하여 이 현상을 방, DQN에 보다 빠른 학습이 가능

# DQN의 발전된 모델들 - Prioritized Replay Memory

- 일반적인 DQN은 Replay Memory에서 Uniform Sampling

➔ TD error가 높은 데이터들이 선택될 확률을 높여,
  더 빠르고 효율적인 학습이 가능



# References

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 1998.

[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[3] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." AAAI. 2016.

[4] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581 (2015).

[5] Schaul, Tom, et al. "Prioritized experience replay." arXiv preprint arXiv:1511.05952 (2015).

# Appendix

- Atari 2600 - https://www.youtube.com/watch?v=iqXKQf2BOSE
- Super MARIO - https://www.youtube.com/watch?v=qv6UVOQ0F44
- Robot Learns to Flip Pancakes - https://www.youtube.com/watch?v=W_gxLKSsSIE
- Stanford Autonomous Helicopter - Airshow #2 - https://www.youtube.com/watch?v=VCdxqn0fcnE

- OpenAI Gym - https://gym.openai.com/envs
- Awesome RL - https://github.com/aikorea/awesome-rl
- Udacity RL course
- TensorFlow DRL - https://github.com/nivwusquorum/tensorflow-deepq
- Karpathy rldemo - http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

감사합니다.