Introduction of Reinforcement Learning

곽동현

서울대학교 바이오지능 연구실

Table of Contents

Background

- 기존의 reinforcement learning에서 Q function을 DNN 혹은 CNN으로 근사하여 문제를 해결하는 시도가 최근 Google DeepMind를 필두로 활발히 연구가 되고 있다.
- 최근 연구에서는 Atari 2600, 바둑을 인간보다 더 잘 플레이하는 수준의 경이적인 성과를 보이고 있으며, 나아가 3D 게임이나 로봇 컨트롤 문제에도 적용되고 있다.

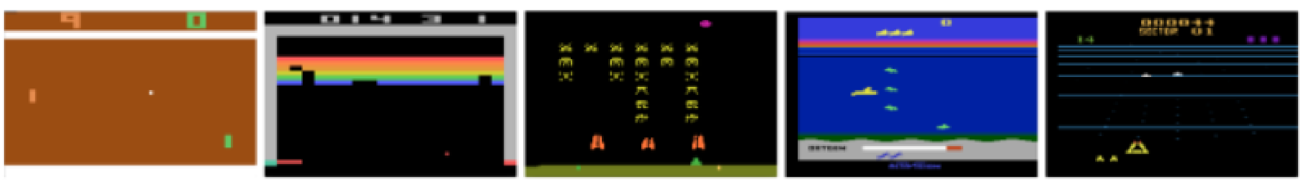
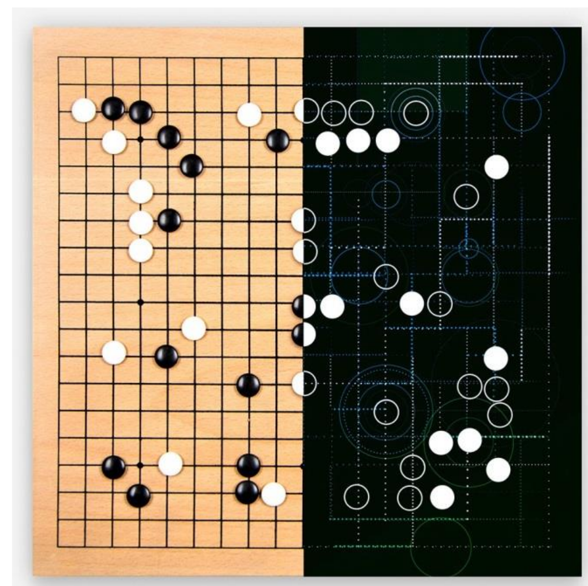
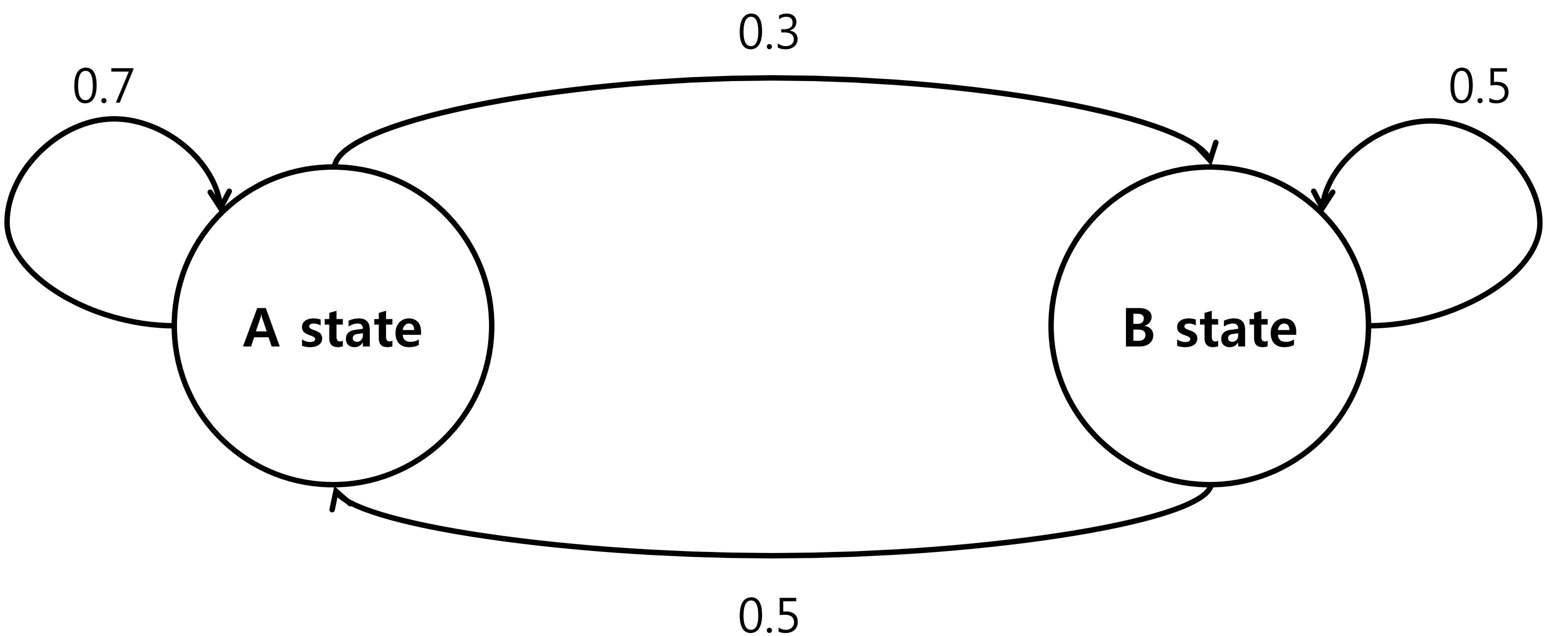


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider



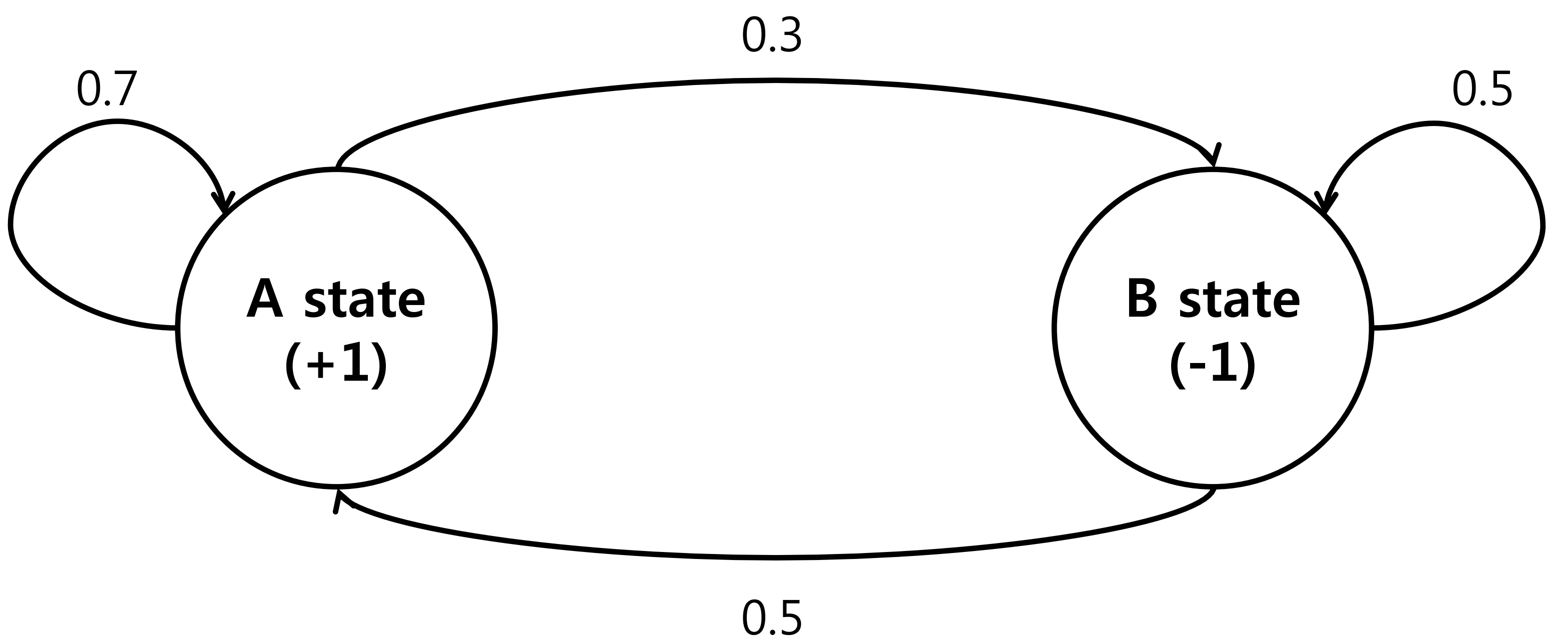
Markov Process

- State space : S = { A , B }
- State transition probability : P(S`|S) = {Paa= 0.7 , Pab = 0.3 … }
- Purpose : find steady state distribution



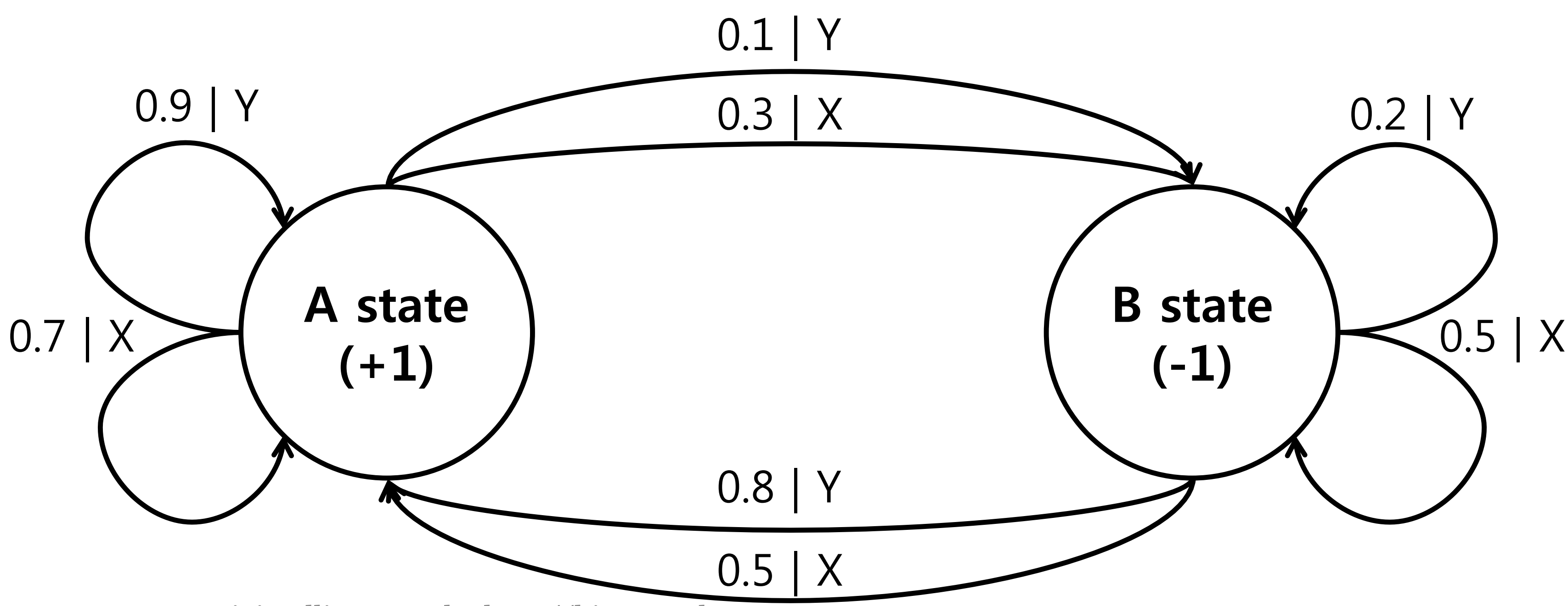
Markov Process with rewards

- State space : S = { A , B }
- State transition probability : P(S`|S) = {Paa= 0.7 , Pab = 0.3 … }
- Reward function : R(S) = +1 if (S=A), -1 if (S=B)
- Purpose : find expected reward distribution



Markov Decision Process

- State space : S = { A , B }
- Action conditional state transition probability : P(S`|S , A)
- Reward function : R(S) = +1 if (S=A), -1 if (S=B)
- Action space : A = { X, Y }
- Purpose : find optimal control policy



Markov Decision Process

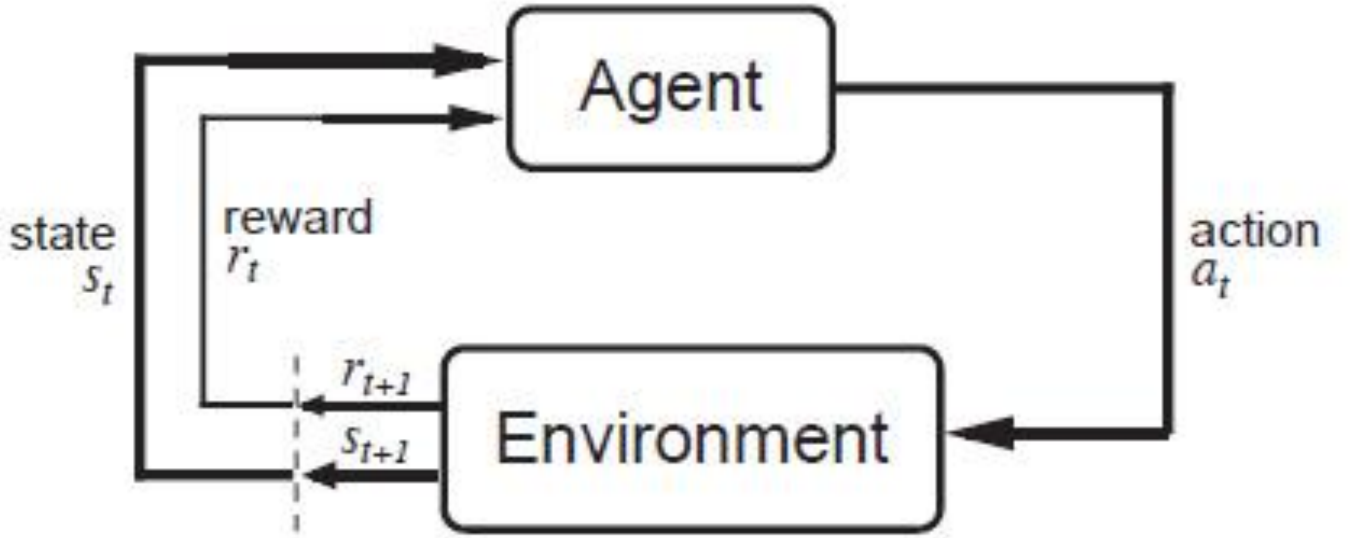
- **Markov decision processes** (**MDPs**) provide a mathematical framework for modeling decision making

  - in situations where outcomes are partly random and partly under the control of a decision maker.

- MDPs are useful for studying a wide range of optimization problems solved via dynamic programming and reinforcement learning.

- They are used in a wide area of disciplines, including robotics, automated control, economics, and manufacturing.

Agent-Environment Interaction



- Objective : maximize the sum of future rewards

- Algorithms
  1) Planning : Exhaustive Search / Dynamic Programming
  2) Reinforcement Learning : MC method / TD Learning

Discounted Reward

- Sum of future rewards (in episodic task)

➔ Gt := Rt+1 + Rt+2 + Rt+3 + … + RT

- Sum of future rewards (in continuous task)

➔ Gt := Rt+1 + Rt+2 + Rt+3 + … + RT + …

Gt → ∞

- Sum of discounted future rewards (in both case)

➔ $$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

(γ : discount rate , 0 <= γ < 1)

Policy

- Deterministic policy : a = f(s)

State	Optimal Action
1	X
2	Y
3	X

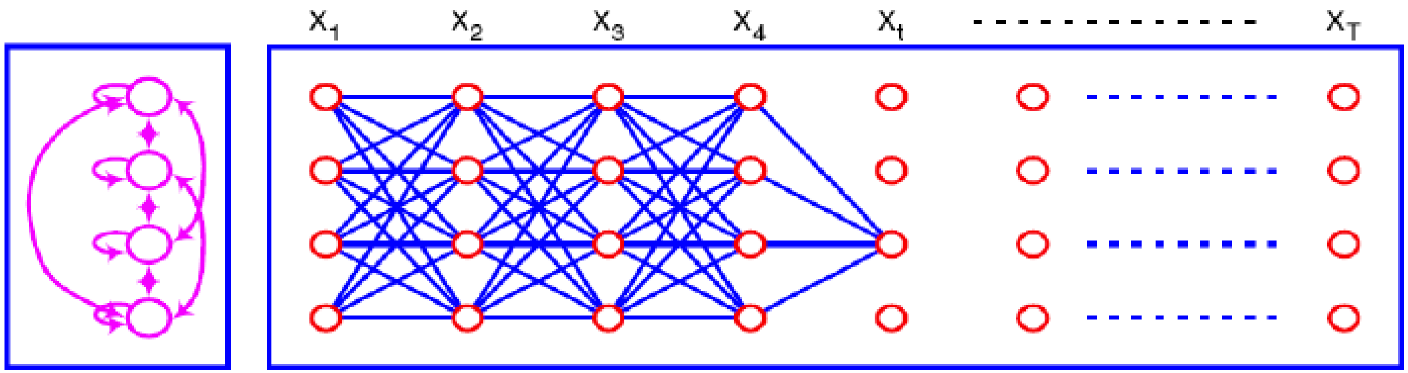
- Stochastic policy : p(a|s)

State	P(Action X)	P(Action Y)
1	0.8	0.2
2	0.4	0.6
3	0.99	0.01

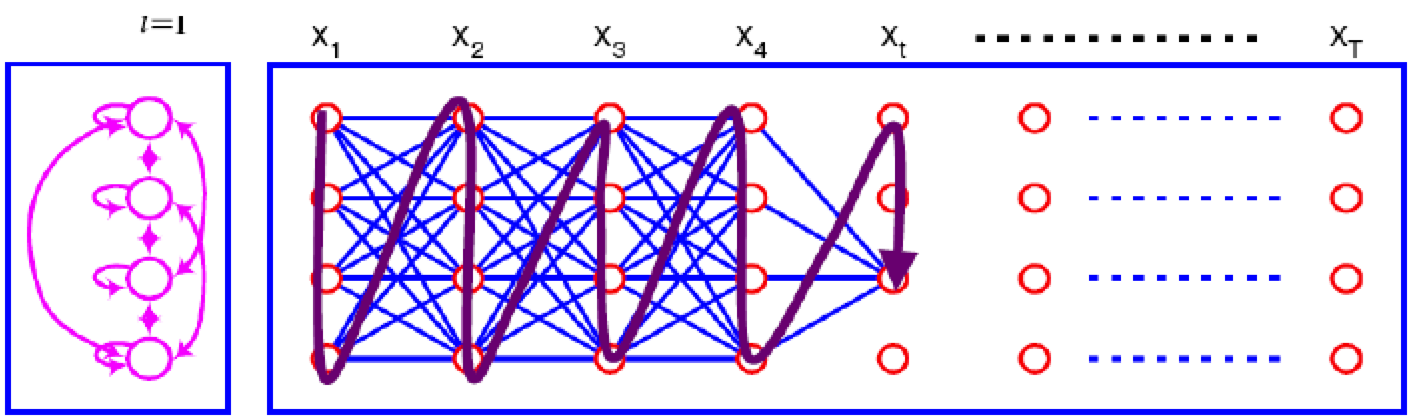
Solution of the MDP : Planning

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

1) **Exhaustive Search**



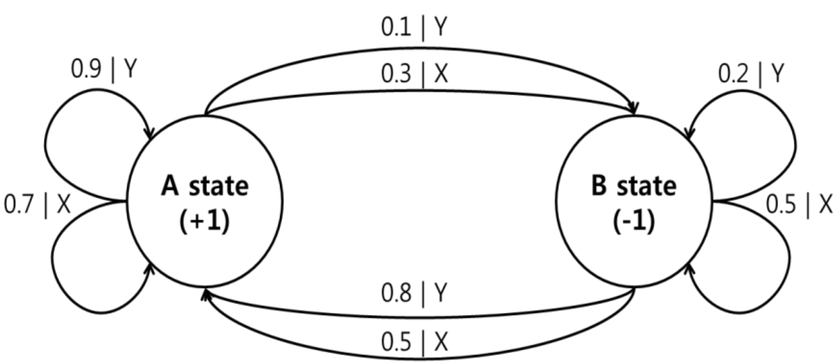
2) **Dynamic Programming**

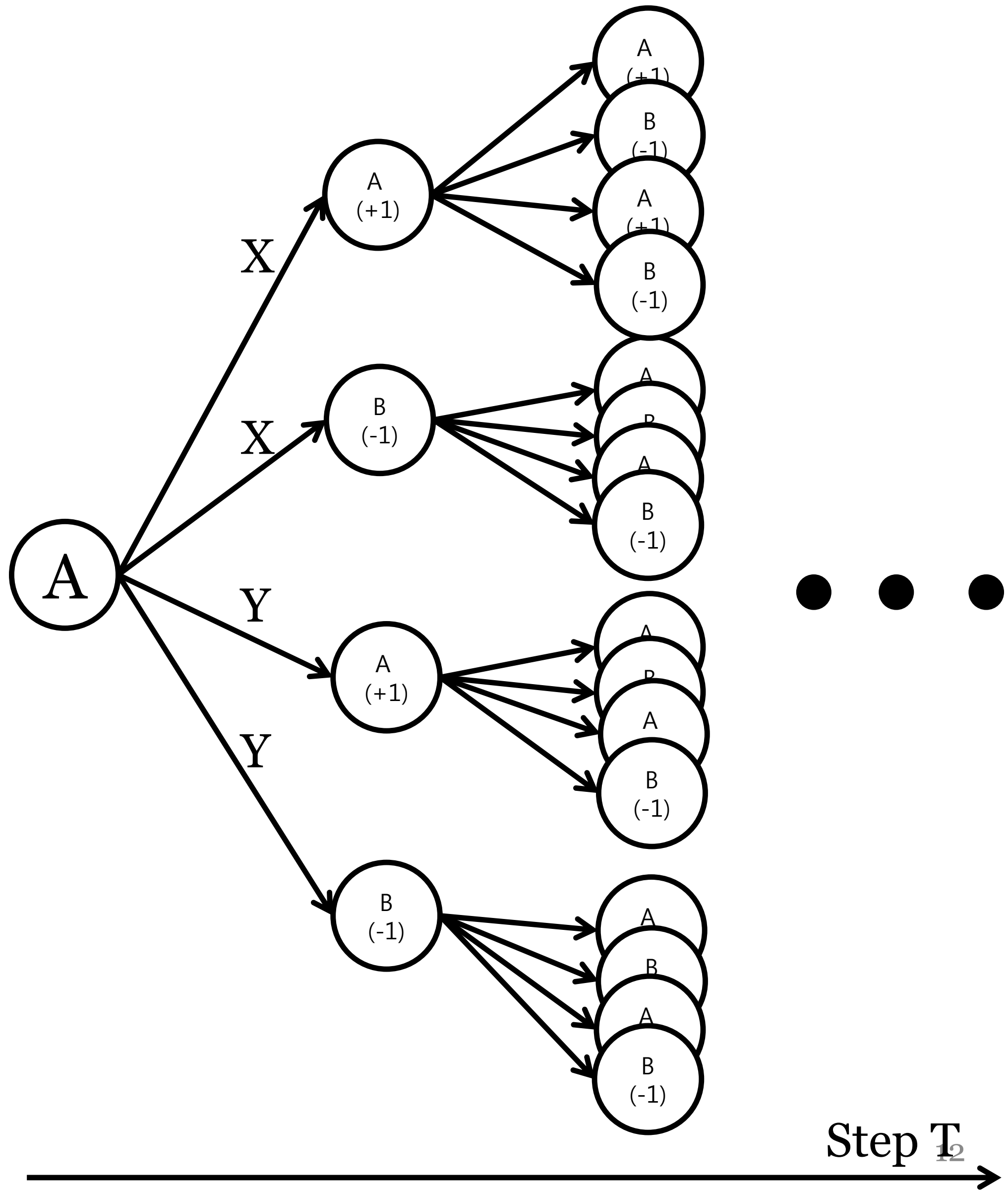


Find Optimal Policy with Exhaustive Search

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.

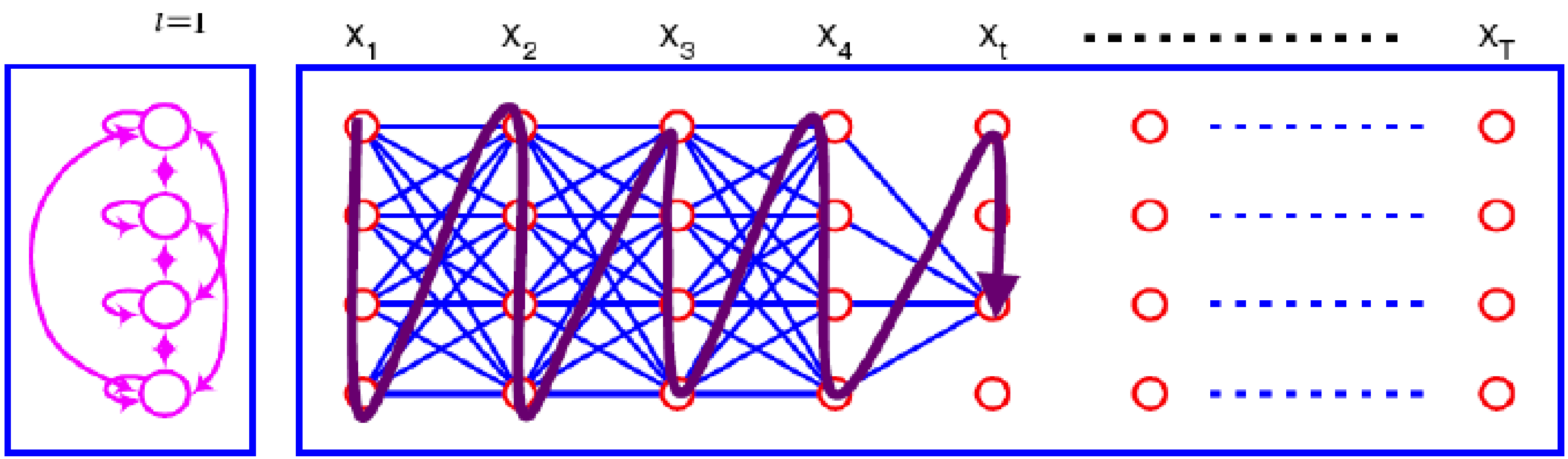
And we can choose the optimal action path, but this needs **O(N^T)!**





Dynamic Programming

- We can apply the DP in this problem, and the computational cost reduces to **$O(N^2T)$.** (<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.



Value Function

- In DP approach, we will introduce a **value function** which let us know the expected future sum of rewards at given state.

1) State-value function

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

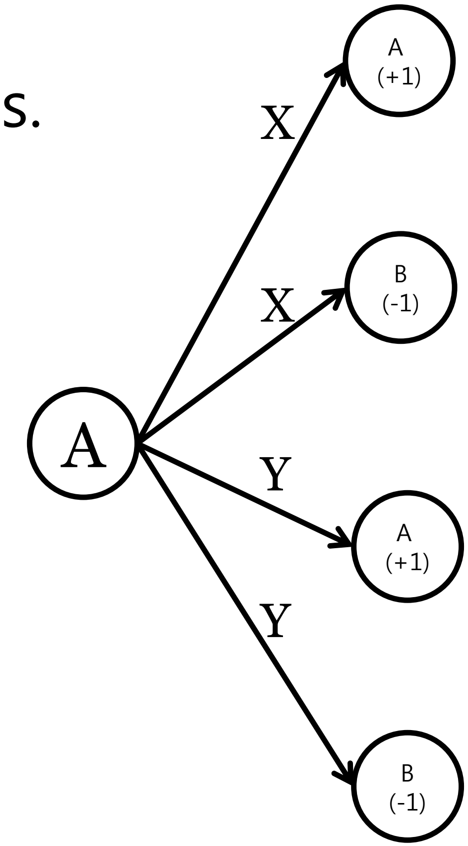
2) Action-value function

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

Find Optimal Policy from Value Function

1) State value function

→ One step search for all actions.



2) Action value function

→ Policy = argmax_a { $q_\pi(s,a)$ }

Policy Iteration

- How can we get the state-value function with DP?
  (action-value function is similarly computed.)

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

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

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

Policy Iteration

- Policy iteration consists of two simultaneous, interacting processes.

- **(policy evaluation)** One making the value function consistent with the current policy

- **(policy improvement)** And the other making the policy greedy with respect to the current value function

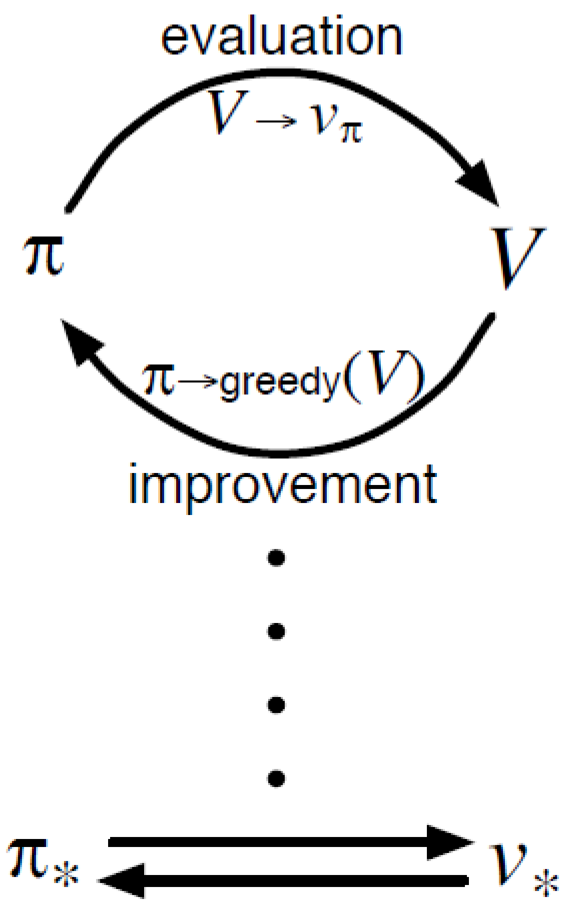
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

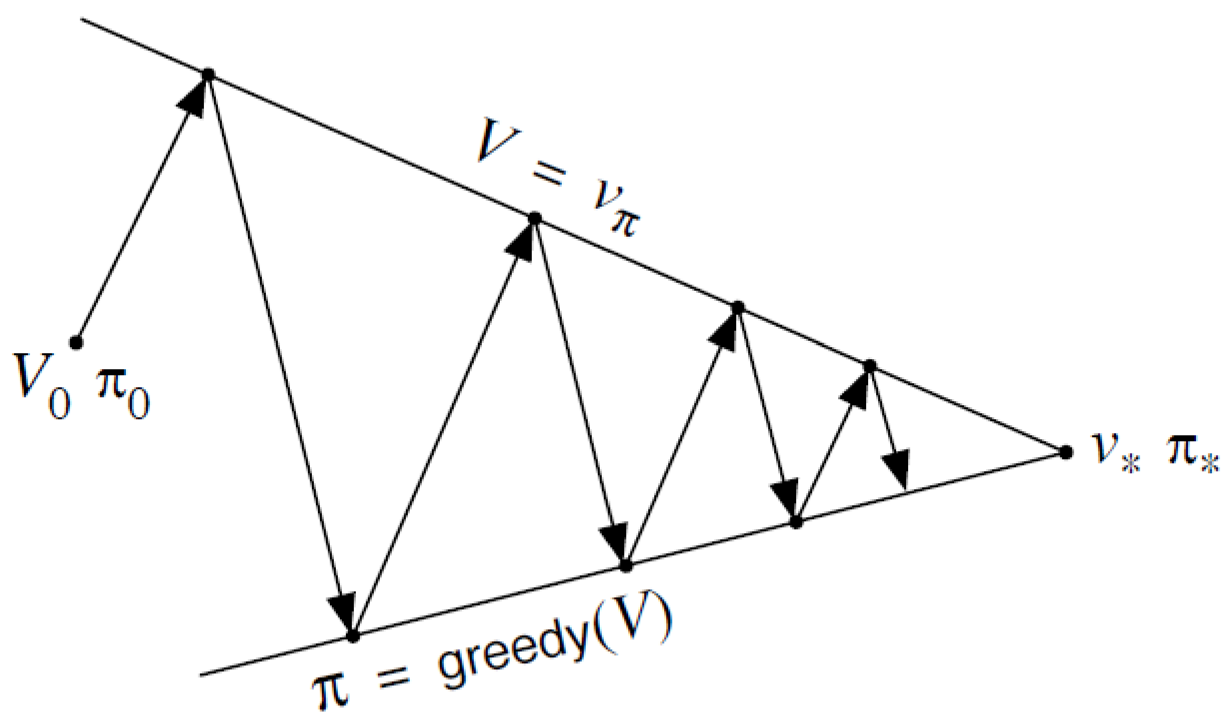
2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad a \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Policy Iteration

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





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 term and just calculate the mean of reward with sampling method. This is the starting point of the machine learning is embedded.

1) Monte Carlo Methods

2) Temporal-Difference Learning (some kind of 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 π

For each state s appearing in the episode:

$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

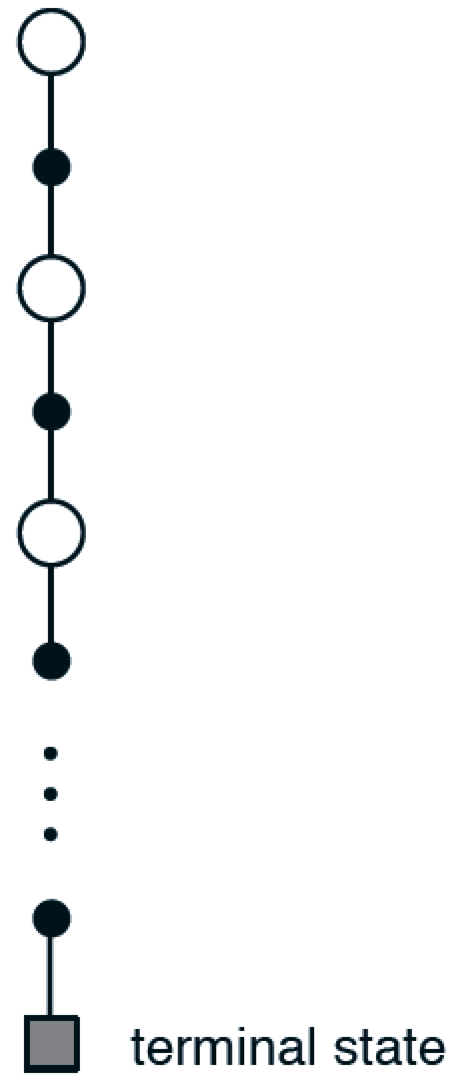
$V(s) \leftarrow \text{average}(Returns(s))$

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 started state. This is really time consuming to update one state while waiting the terminal of episode.



BI BIOINTELLIGENCE

Temporal-Difference Learning

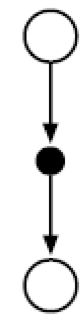
- TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.

- 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 π 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 π 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)

Temporal-Difference Learning

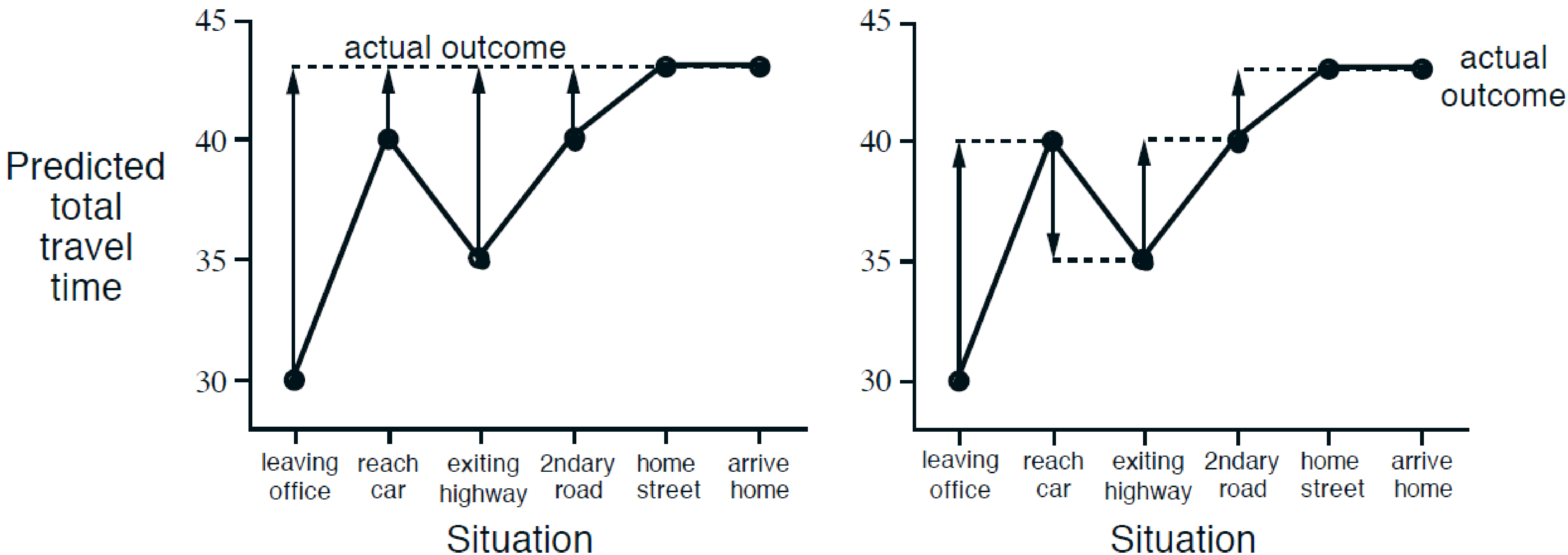


Figure 6.2: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

On policy / Off policy

- **On policy** : Target policy = Behavioral policy
  there can be only one policy.
  → This can learn a stochastic policy. Ex) Q-learning

- **Off policy** : Target policy != Behavioral policy
  there can be several policies.
  → Broad applications. Ex) SARSA

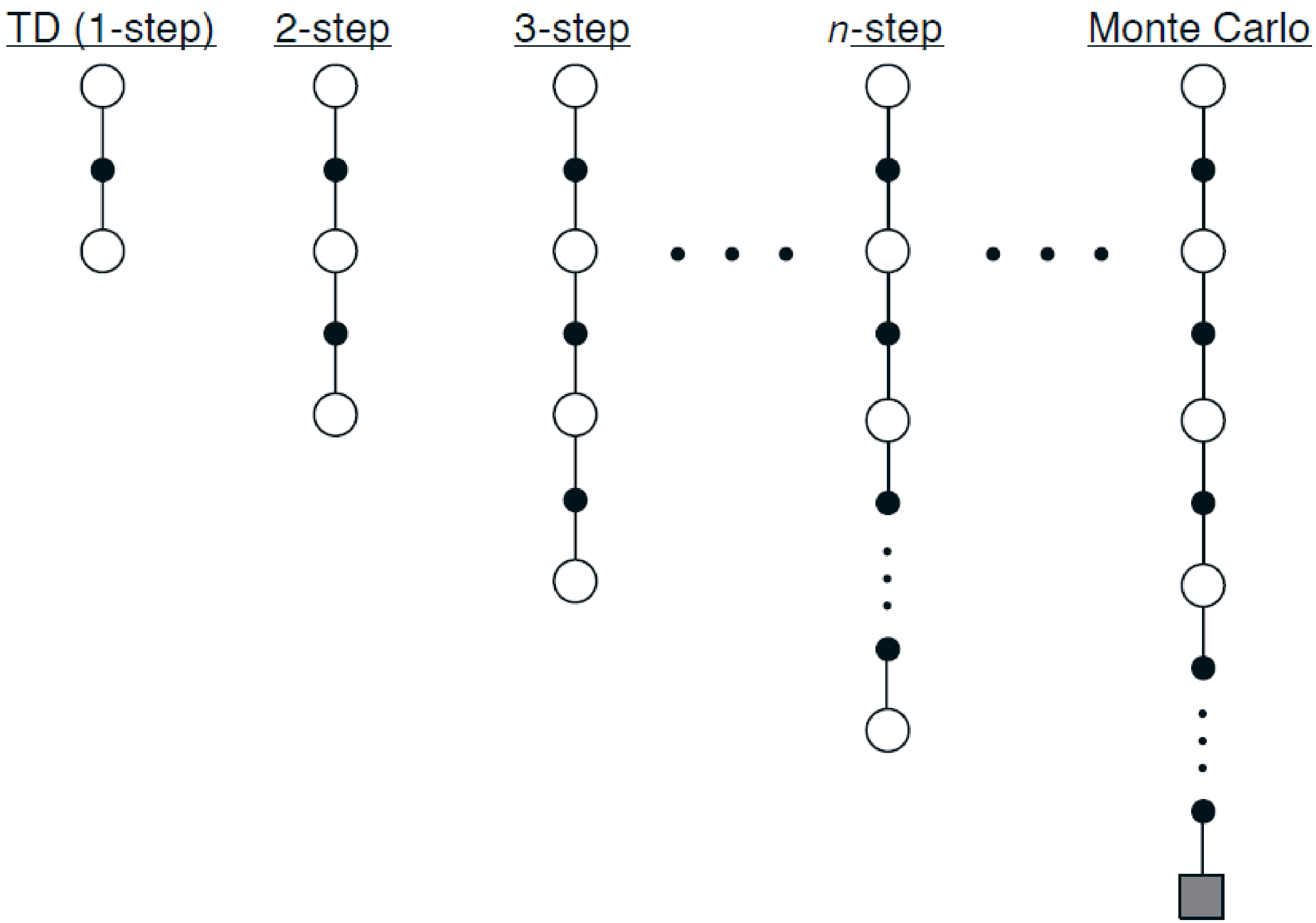
Sarsa: On-Policy TD Control

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

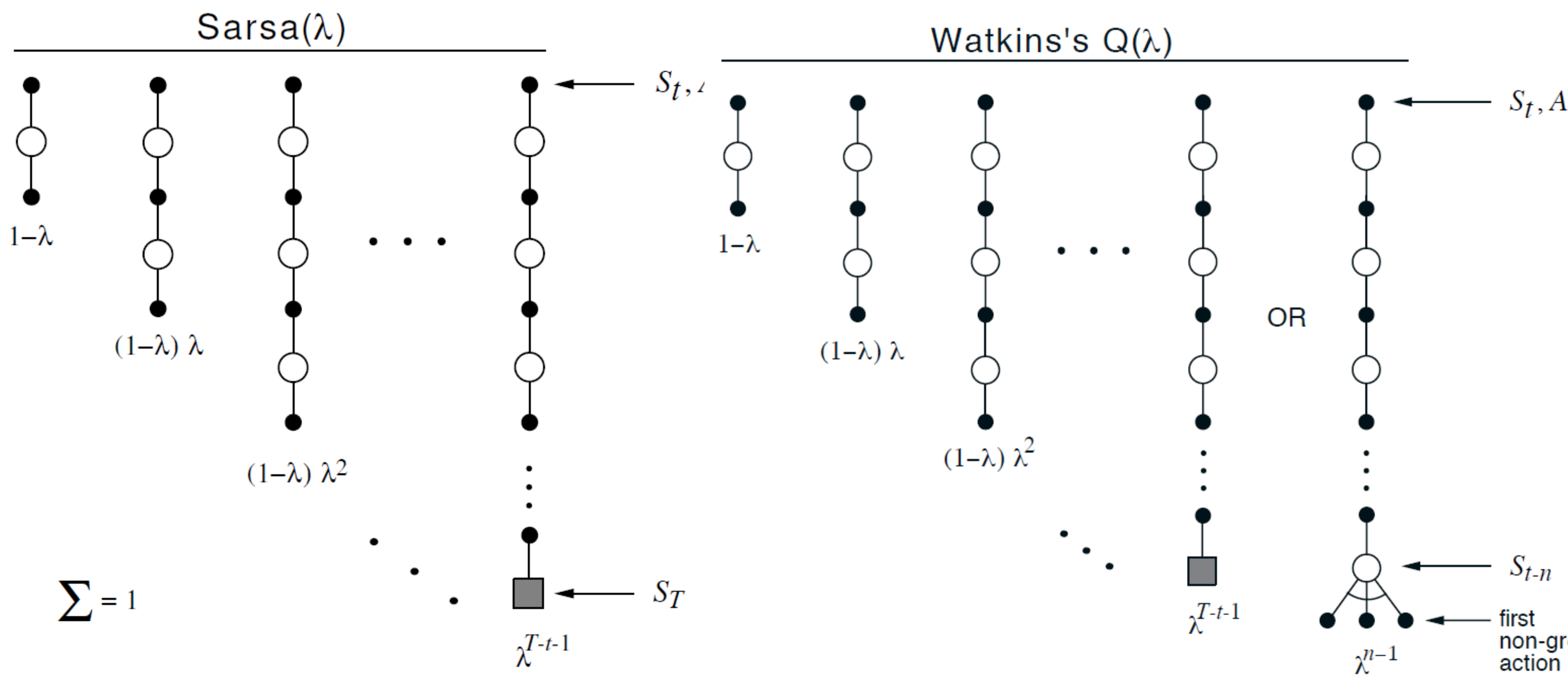
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each step of episode):
        Take action A, observe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

Eligibility Trace

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

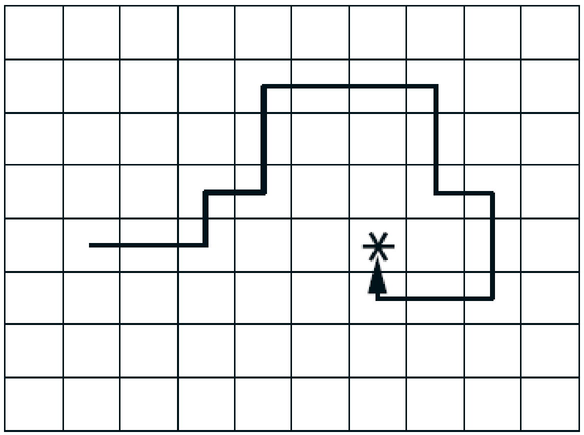


Eligibility Trace

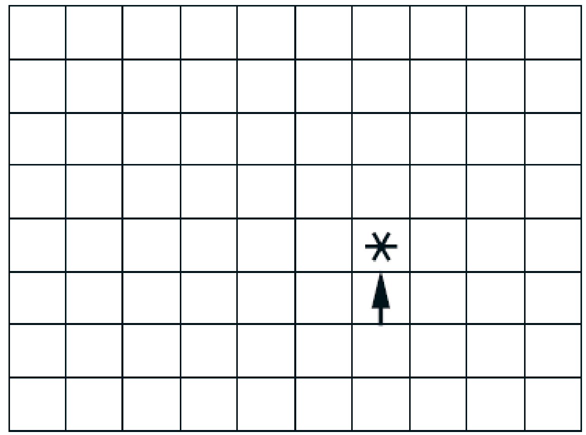


Comparisons

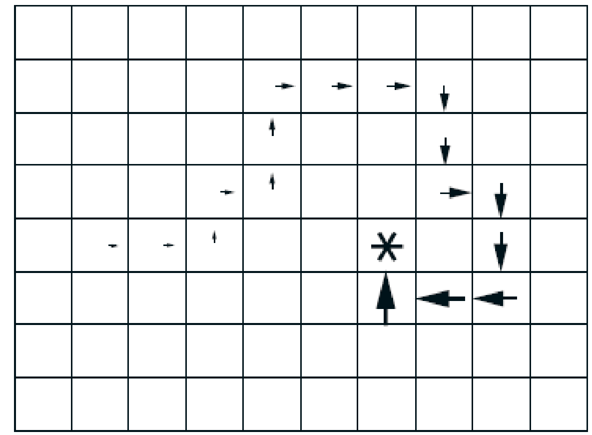
Path taken



Action values increased by one-step Sarsa

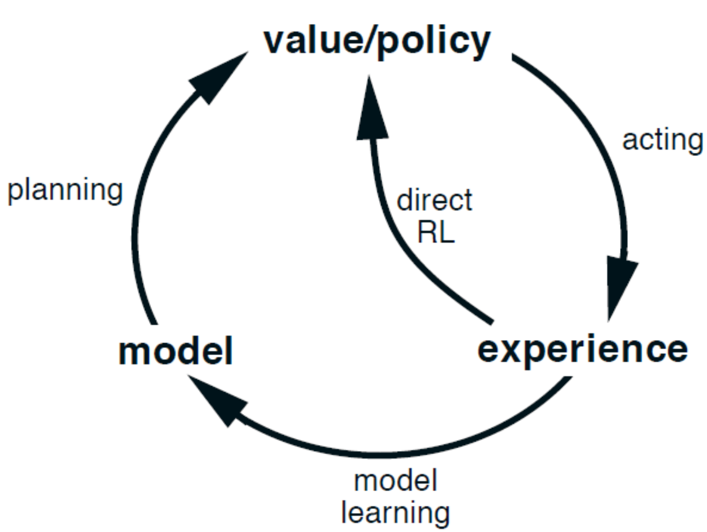


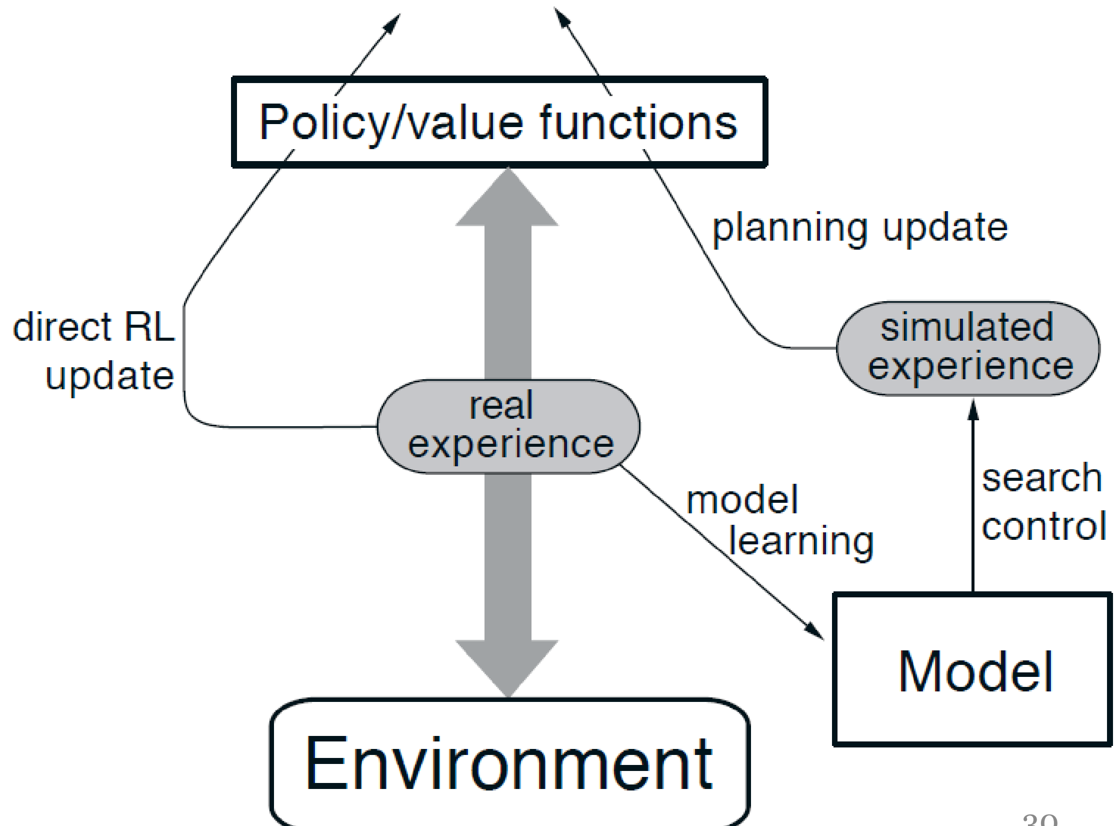
Action values increased by Sarsa(λ) with λ=0.9



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.





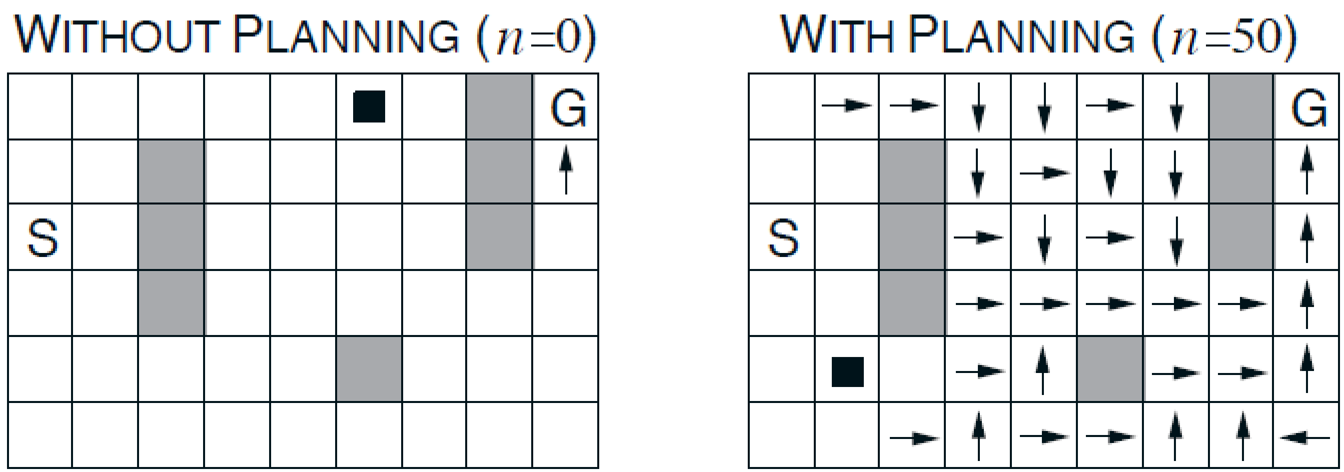
Planning + Learning

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

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.

Planning vs Learning

- After one-step learning.



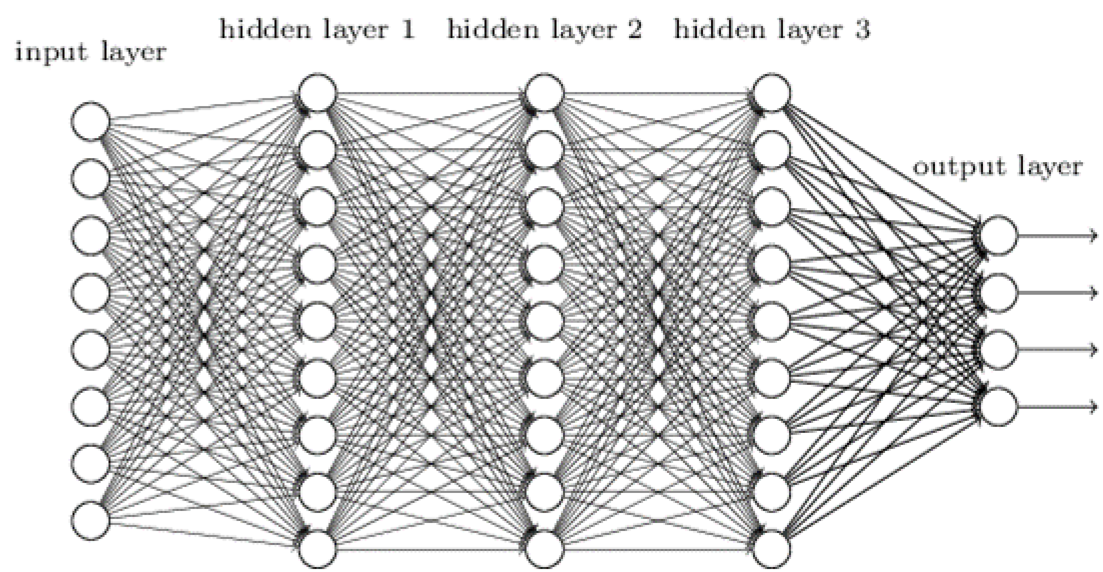
What is Deep RL?

- Approximate the value function with deep learning to generalized the huge state space.
- This needs supervised learning techniques and online moving target regression.

Tabular State-value function

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

Deep neural network



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

References

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

THANK YOU