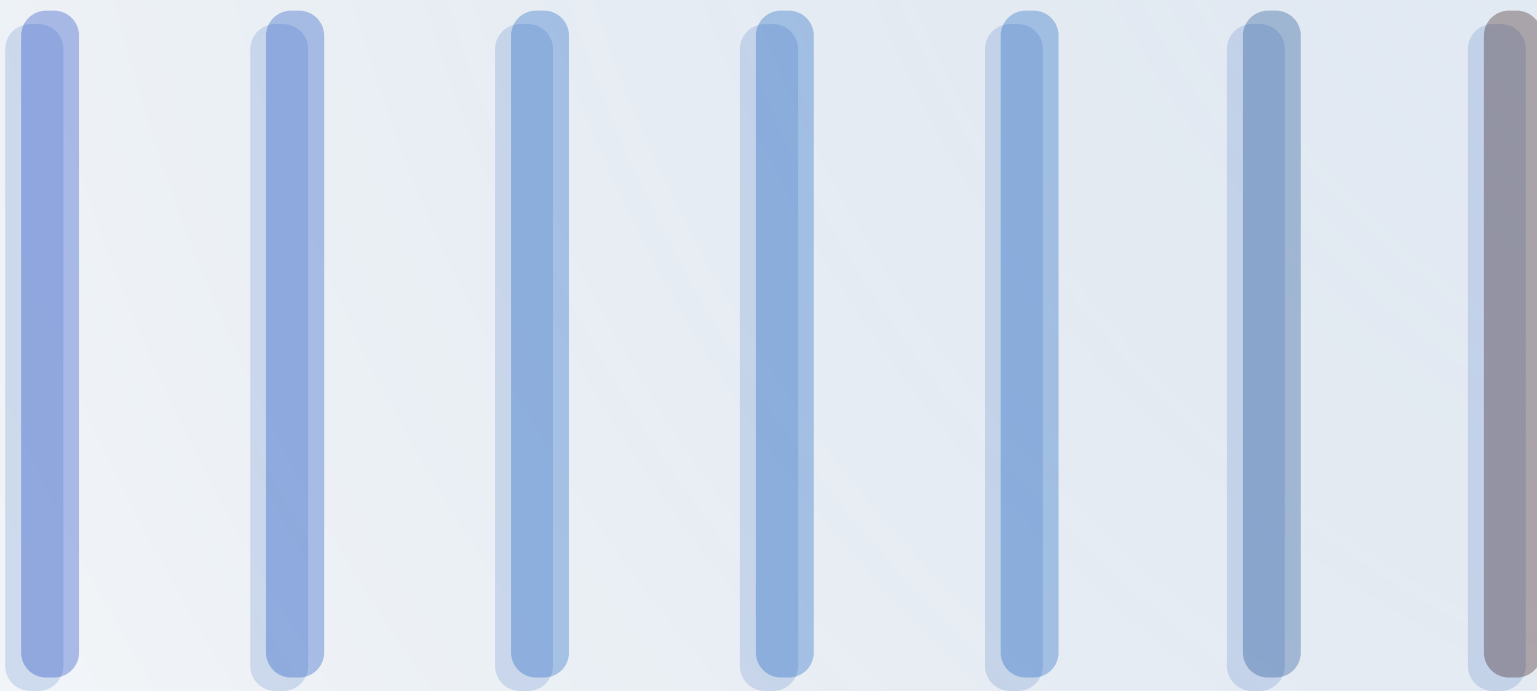


don



Reinforcement Learning

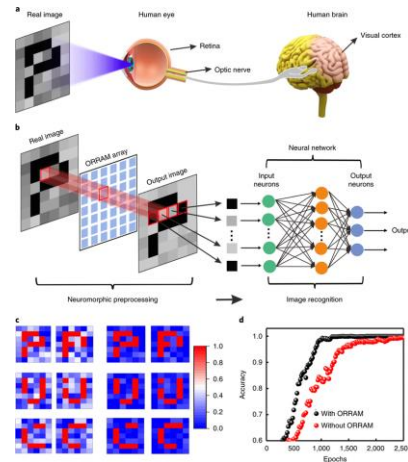


Reinforcement Learning

Neuro Morphic

Low Abstract Level

About How Human
Brain Works
By Neuron



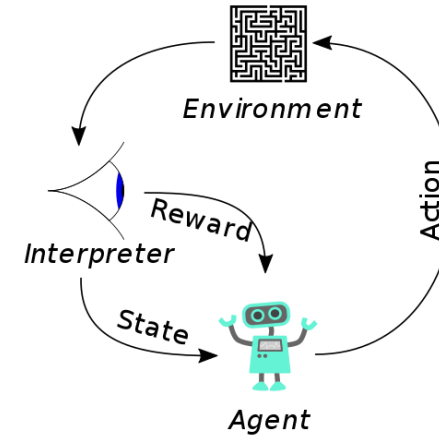
Designed Based on Neuron
So it update Itself by updating
Neurons Weight Value

Almost Every MPL Based
Neural Net Models

Reinforcement

High Abstract Level

About How Human
Learn from
Environment

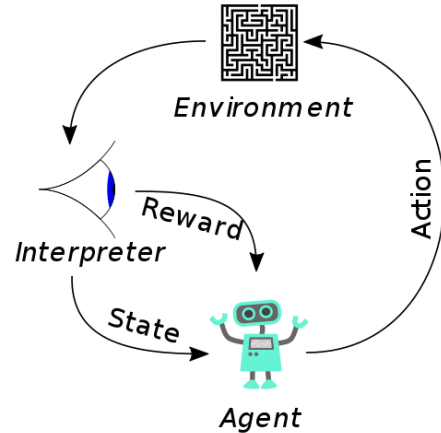


Designed Based on Action
So it update itself by updating
Rule about Environment and Action

Almost Every Environment Based
Agent Models

Reinforcement Learning

Reinforcement Learning



Reinforcement Learning : Interaction with environment

Learning through interaction is an basis idea for basic learning and intelligence theory

A computational approach to interaction and learning

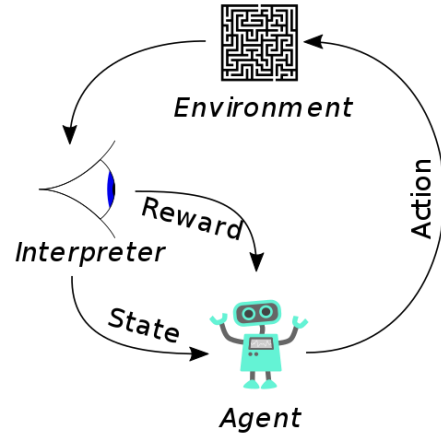
Goal-directed learning through interaction



Reinforcement Learning

Reinforcement Learning

Reinforcement Learning



Reinforcement Learning

Learning about **actions** that can maximize **rewards** in given **situation**

A number of trials and errors are required before the learning **agent** can find the **action** that **best suits** the situation

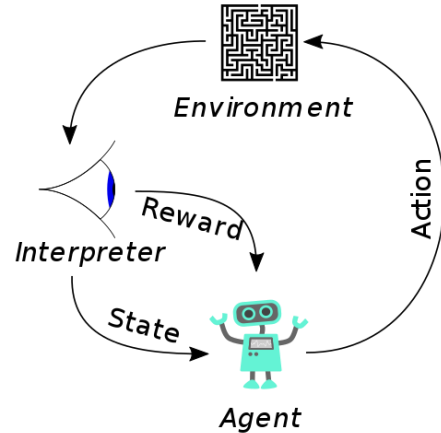
} Trial and Error Search

In complex situations, the **actions** **current** selected will affect **future sequential** rewards

} Delayed Reward

Reinforcement Learning

Reinforcement Learning



Exploration / Exploitation

Exploration (탐험)

To learn about the value of each action, you need to explore in advance.

In order to explore, model should be able to give up what believe as best current policy.

Exploitation (이용)

In order to receive a higher reward, a more appropriate action in a given situation must be exploited.

Exploration-Exploitation Dilemma

Reinforcement Learning

Reinforcement Learning

- Policy
- Reward Signal
- Value function
- Model



Policy plays a role in **determining** what action is taken on the current state
It can be a lookup table simply or a **search process** that requires a lot of computational costs
Usually policy is **stochastic**

Reinforcement Learning

Reinforcement Learning

- Policy
- **Reward Signal**
- Value function
- Model



The number that the agent give to the agent whenever it does an action
It's ok to say that the **definition of a Reward signal is the definition of goal**
Normally, the forward signal is a **probability function for the state and the action**

Reinforcement Learning

Reinforcement Learning

- Policy
- Reward Signal
- Value function
- Model



Value function means reward from a more **long-term perspective** than the reward signal
The value of specific state means the **sum of reward** that agent can get during specific state
The **highest value is given priority** in making an evaluation or selection prior to the judgment

Reinforcement Learning

Reinforcement Learning

- Policy
- Reward Signal
- Value function
- Model

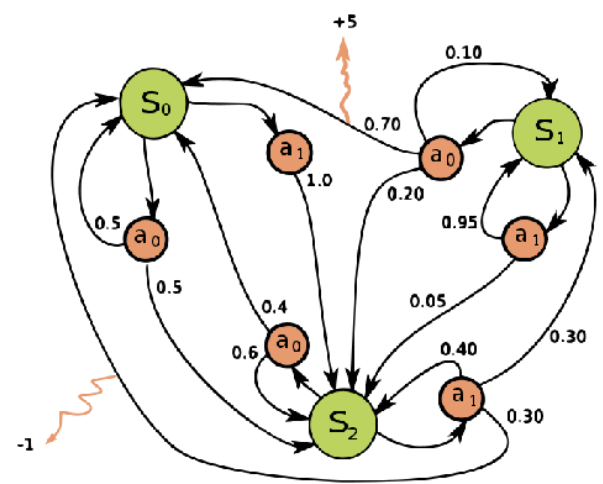


Model is something that **imitates** the behavior of the environment

When a state and action are given, the **model** returns following state and reward as a result (as in the environment).

Reinforcement Learning

Markov Decision Process [MDP]



Markov Decision Process

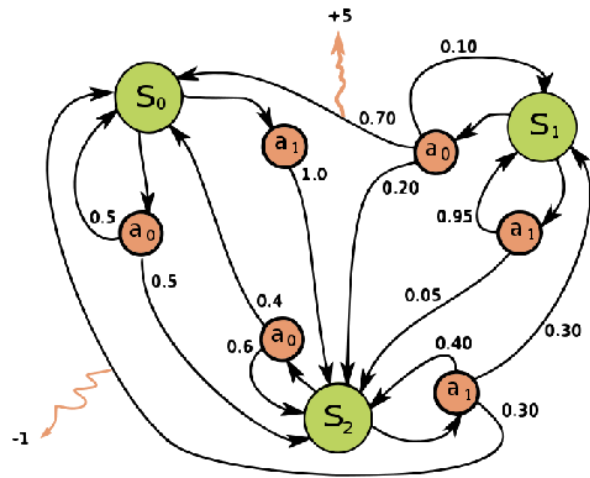
Markov Decision Process (MDP) for represent reinforcement learning problems

Three Key Concept for MDP

Sensation	Action	Goal
The agent should be able to recognize what state the environment is in.	The agent determines the action according to the given state.	Enhancement learning problem must have a specific goal

Reinforcement Learning

Markov Decision Process [MDP]



State

$$S_t = (1,3) \quad S = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$$

Action

$$A_t = a \quad A_t = A = \{up, down, left, right\}$$

Reward

$$R_S^a = E [R_{t+1} | S_t = s, A_t = a] \quad \begin{array}{l} E: \text{expectation} \\ \text{Get reward at } t+1 \text{ step} \end{array}$$

State Transition Probability

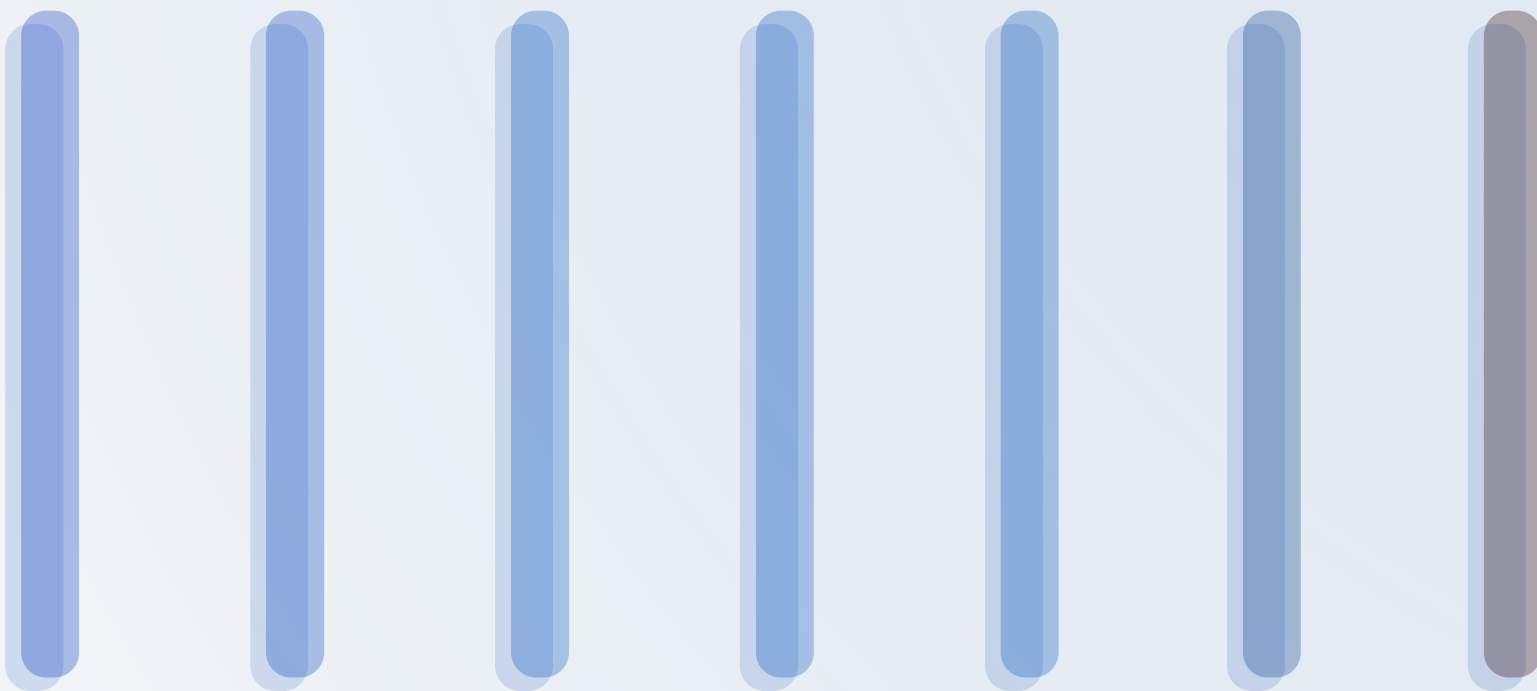
$$p_{ss}^a = P [S_{t+1} | S_t = s, A_t = a]$$

Discount Factor

$$\gamma \in [0,1] \quad \gamma^{k-1} R_{t+k}$$

Policy

$$\pi(a|s) = P[A_t = a | S_t = s]$$



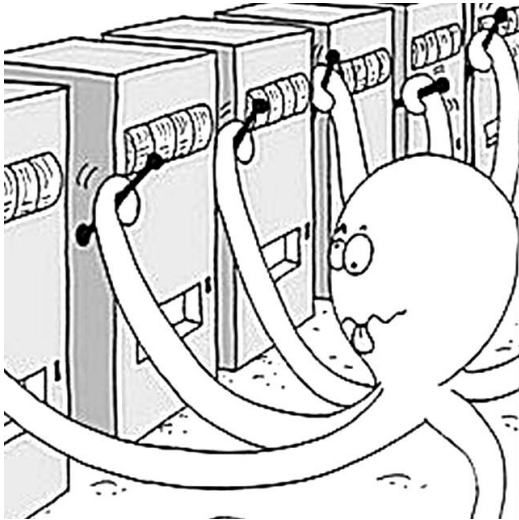
Finite Markov Decision Process



Finite Markov Decision Process

Multi-armed bandit problem [k-armed bandit]

One-armed bandit : 도박용 슬롯머신



Problem Definition

Select one of the k slot machines for each attempt. This is the same as having a choice of k action each time.

The selected slot machine returns numerical reward through a fixed probability distribution (stationary probability distribution)

GOAL

Repeat the above procedure any number of times to **find the slot machine with the highest expected value for the total sum of rewards**

Finite Markov Decision Process

Value Function

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}$$

$$v(s) = E [G_t | S_t = s]$$

$$v(s) = E [R_{t+1} + \gamma G_{t+1} | S_t = s]$$

$$v(s) = E [R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

$$v_{\pi}(s) = E_{\pi} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$$

Value Function

Focus on actions that you expect to maximize reward

The goal is to select the action that can make highest action value.

Expected value considering how much reward will be received later when in some condition

Finite Markov Decision Process

Value Function Estimation

$$Q_t(a) \quad \text{or } q_\pi(s, a)$$

$$v(s) = \sum_{a \in A} \pi(a|s) q_\pi(s, a)$$

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

Value Function Estimation

Unfortunately, in most cases, we do not have information about optimal action value in the dictionary

Some estimates can be made based on the information obtained by repeating the experiment

The estimated function is expressed as follows and call as **Q function**

Finite Markov Decision Process

Terminologies

Greedy actions

A set of actions with the highest value of the Q function in each attempt.

Exploiting

Choosing one of the Greedy actions

Exploring

Choosing one of the non-greedy actions for exploration

Exploitation and extension are trade-off relationships

It is important to strike a proper balance between the **Exploiting and the Exploring**

Finite Markov Decision Process

Action-value Method

$$Q_t(a) = \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_t=a}}{\sum_{i=1}^{t-1} 1_{A_t=a}}$$

$1_{\text{predicate}}$ = Returns 1 if the predicate is true, or 0 if false.

If the denominator is 0, define $Q_t(a)$ as any constant value

Sample-average method

Finite Markov Decision Process

Action Selection

$$A_t = \operatorname{argmax}_q Q_t(a)$$

$\operatorname{argmax}_q Q_t(a)$ function return action a that maximizing $Q_t(a)$

Non-greedy action selection can be performed with a probability of ϵ at every attempt

Greedy Action Selection

ϵ - Greedy Action Selection

Finite Markov Decision Process

Example

When $Q_t(a) = Q^*(a)$ is satisfied and ϵ -greedy action selection is performed for 10 actions by $\epsilon = 0.1$

At this time, what is the probability that the automatic action is selected?

$Q_t(a) = Q^*(a)$ 를 만족하고 10개의 action에 대해 $\epsilon = 0.1$ 로 ϵ -greedy action selection을 하고 있다
이때 optimal action이 선택될 확률은?

Finite Markov Decision Process

Example

When $Q_t(a) = Q^*(a)$ is satisfied and ϵ -greedy action selection is performed for 10 actions by $\epsilon = 0.1$

At this time, what is the probability that the automatic action is selected?

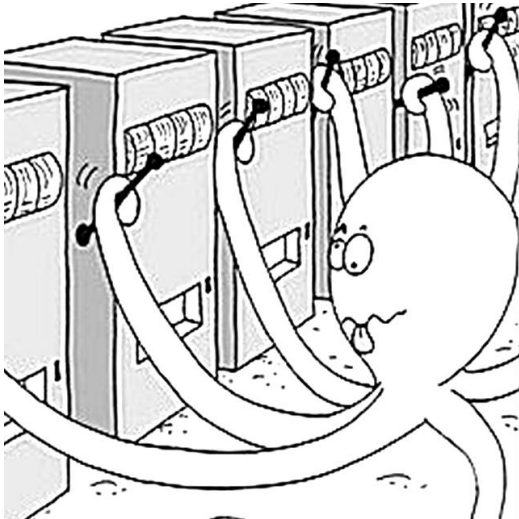
Optimal action is selected by the Greedy action, probability is
 $1 - \epsilon = 1 - 0.1 = 0.9$

Non-greedy action selection selects randomly 10 action as a probability of ϵ .
So the Optimal action can be chosen as below equation
 $0.1 * 1/10 = 0.01$

So, the probability that the optimal action is selected is $0.9 + 0.01 = 0.91$

Finite Markov Decision Process

Example [10-armed bandit]



Problem Definition

Generate 2,000 randomly 10-armed bandit problems and visually output the ratio of average response to each time step.

A Simple Experiment for Comparing the Performance of Greedy Method to ϵ -Greedy Method

Finite Markov Decision Process

k-armed bandit
Incremental Implementation

Is there a more efficient way to calculate sample average method?

nth action value evaluation for some action selected:

$$Q_n = \frac{R_1 + R_2 + R_3 + R_4 \dots R_{n-1}}{n - 1}$$

Should know every rewards for each steps
Must calculate every some of rewards for each action

Finite Markov Decision Process

k-armed bandit
Incremental Implementation

Solution

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i &= \frac{1}{n} (R_n + (n-1)Q_n) \\ &= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) &= \frac{1}{n} (R_n + nQ_n - Q_n) \\ &= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) &= Q_n + \frac{1}{n} [R_n - Q_n], \end{aligned}$$

Finite Markov Decision Process

k-armed bandit
Incremental Implementation

Update Rule

$$Q_n = Q_n + \frac{1}{n} [R_n - Q_n]$$

NewEstimation \leftarrow OldEstimate + StepSize[Target – OldEstimate]

Finite Markov Decision Process

Tracking a Nonstationary Problem

Weighted-average method gradually converges to some value

but ...

What if the probability distribution for compensation is not fixed?

The problems that we will actually address are more often that the probability distribution is nonstationary

Let's set Step size as $\alpha \in (0,1]$

Finite Markov Decision Process

Tracking a Nonstationary Problem

Convergence condition of step-size

$$\sum_{n=1}^{\infty} a_n(a) = \infty \text{ and } \sum_{n=1}^{\infty} a_n^2(a) < \infty$$

What if the probability distribution for compensation is not fixed?

$$a_n(a) = \frac{1}{n} \text{ is meets two conditions}$$

$$\sum_{n=1}^{\infty} \frac{1}{n} \text{ is harmonic series} \qquad \sum_{n=1}^{\infty} \frac{1}{n^2} \text{ is basel problem}$$

Finite Markov Decision Process

Tracking a Nonstationary Problem

$$\sum_{n=1}^{\infty} a_n(a) = \infty \text{ and } \sum_{n=1}^{\infty} a_n^2(a) < \infty$$

$\alpha \in (0,1]$ is not meets these two conditions

Finite Markov Decision Process

Tracking a Nonstationary Problem

$$\sum_{n=1}^{\infty} a_n(a) = \infty \text{ and } \sum_{n=1}^{\infty} a_n^2(a) < \infty$$

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\ &= \alpha R_n + (1 - \alpha)Q_n \\ &= \alpha R_n + (1 - \alpha)[\alpha R_{n-1} + (1 - \alpha)Q_{n-1}] \\ &= \alpha R_n + (1 - \alpha)\alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1} \\ &= \alpha R_n + (1 - \alpha)\alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} + \cdots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1 \\ &= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} R_i, \end{aligned}$$

Finite Markov Decision Process

Tracking a Nonstationary Problem

$$\begin{aligned}Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\&= \alpha R_n + (1 - \alpha)Q_n \\&= \alpha R_n + (1 - \alpha)[\alpha R_{n-1} + (1 - \alpha)Q_{n-1}] \\&= \alpha R_n + (1 - \alpha)\alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1} \\&= \alpha R_n + (1 - \alpha)\alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} + \cdots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1 \\&= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} R_i,\end{aligned}$$

$$\text{So, } (1 - \alpha)^n + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} = 1 = \text{weighted average}$$

$$\sum_{i=1}^n \alpha(1 - \alpha)^{n-i} = 1 \text{ means old reward multiplied low weight}$$

Finite Markov Decision Process

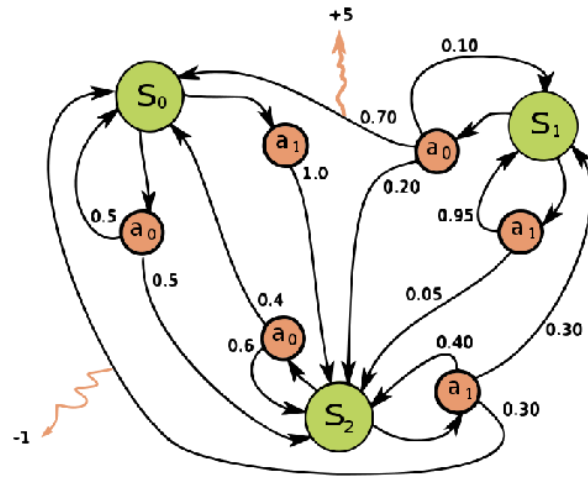
Tracking a Nonstationary Problem

$$(1 - a)^n + \sum_{i=1}^n a(1 - a)^{n-i} = 1 = \text{weighted average}$$

What if the action agent choose now is a factor in determining the next state with the reward?

Finite Markov Decision Process

Finite Markov Decision Process [FMDP]



Finite Markov Decision Process

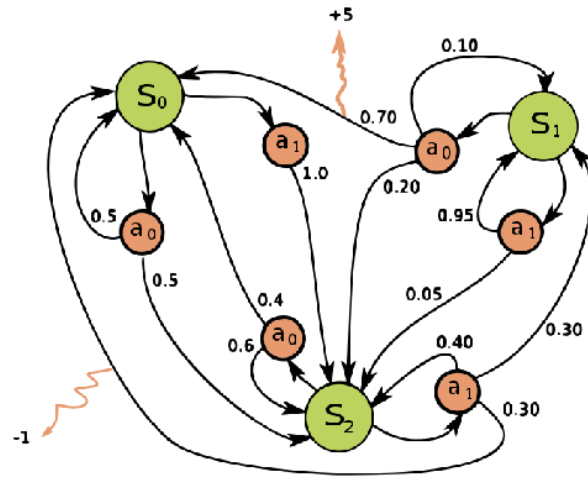
In the Finite MDP, the set of states, actions and rewards consists of one finite number of elements each.

The following sequential events occur by the MDP and the agent

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

Finite Markov Decision Process

Finite Markov Decision Process [FMDP]



Finite Markov Decision Process

Sequential reward after time step t is

$$R_{t+1} + R_{t+2} + R_{t+3} \dots$$

Expected return is

$$G = R_{t+1} + R_{t+2} + R_{t+3} \dots R_T, \text{ when } T \text{ is final step}$$

Maximize Expected return!!

Finite Markov Decision Process

What is Episodes?

When the interaction between the agent and the environment can be divided into several subsequences, a single subsequence is called as **episode**

Problems that can be expressed as units of Episode are called **Episodic tasks**

Episodic tasks have the following characteristics:

Possible to distinguish a set of non-terminal states from a set of all states

Problems that cannot be expressed as units of Episode are called **Contouring tasks**

Finite Markov Decision Process

What is Return?

Recall

$$G = R_{t+1} + R_{t+2} + R_{t+3} \dots R_T, \text{ when } T \text{ is final step}$$

In Continuing tasks

$$T = \infty, \text{ so the Return could be } \infty$$

Solution (Discounting method)

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

Where, γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate

Finite Markov Decision Process

Unified Notation for Return

The generalization of return can encompass both Episodic tasks and contouring tasks.

$$G_t = \sum_{K=t+1}^T \gamma^{k-t-1} R_k \quad \text{including the possibility that } T = \infty \text{ or } \gamma = 1 \text{ (but not both)}$$

$$T \neq \infty, \gamma = 1$$

$$G = R_{t+1} + R_{t+2} + R_{t+3} \dots R_T, \text{ when } T \text{ is final step}$$

$$T = \infty, \gamma \in (0,1]$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$$

Finite Markov Decision Process

Policy and Value Functions

Policy

Returns the probability of selecting each action at a state.

For example, if $S_t = s$, the probability of selecting $A_t = a$ is expressed as $\pi(a|s)$.

State – value function for policy π :

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S}.$$

Action – value function for policy π :

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

Finite Markov Decision Process

A Fundamental Property of Value Functions

$$\begin{aligned}v_{\pi}(s) &\doteq \mathbb{E}_{\pi}[G_t \mid S_t = s] \\&= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_{\pi}[G_{t+1} | S_{t+1} = s'] \right] \\&= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_{\pi}(s') \right], \quad \text{for all } s \in \mathcal{S}.\end{aligned}$$

Bellman Equation for V_{π}

Finite Markov Decision Process

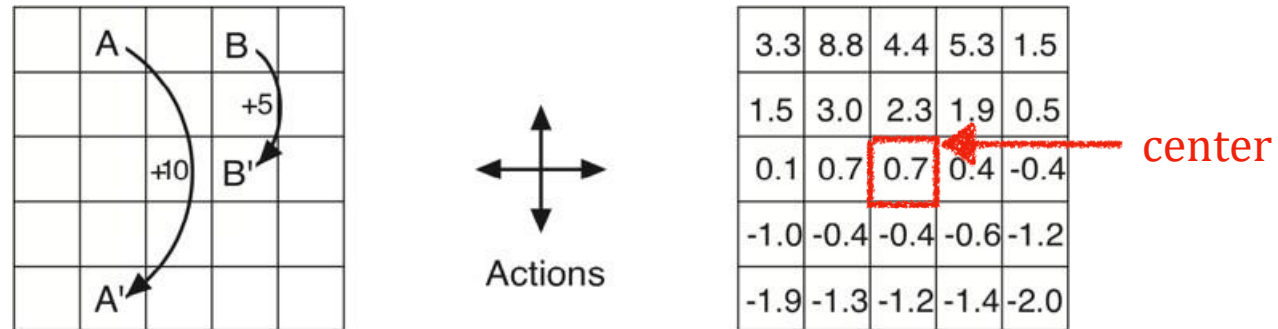
A Fundamental Property of Action-Value Functions

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

Bellman Equation for q_{π}

Finite Markov Decision Process

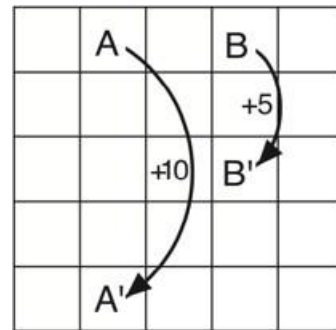
Grid world Example



If $\pi(a | s) = 0.25$ for all $a \in A$ and $s \in S$ and $\gamma = 0.9$
show $v\pi(\text{center}) = 0.7$

Finite Markov Decision Process

Grid world Example



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

$$\begin{aligned}v_{\pi}(\text{center}) &= 0.25 \cdot (0 + 0.9 \cdot 2.3) + 0.25 \cdot (0 + 0.9 \cdot 0.4) + 0.25 \cdot (0 + 0.9 \cdot -0.4) + 0.25 \cdot (0 + 0.9 \cdot 0.7) \\&= 0.5175 + 0.09 - 0.09 + 0.1575 \\&= 0.675 \\&\approx 0.7\end{aligned}$$

Finite Markov Decision Process

Optimal Policies and Optimal Value Functions

Optimal Policy

When a policy returns a value that is always greater than or equal to all other policies in all states, it is called an objective policy and is written as π_*

Optimal Value Function

The value function that follows the Optimal policy is called the Optimal value function.

$$v_*(s) = v_{\pi_*}(s) = \max_{\pi} v_{\pi}(s)$$

$$q_*(s, a) = q_{\pi_*}(s, a) = \max_{\pi} q_{\pi}(s, a)$$

Finite Markov Decision Process

Bellman Optimality Equation

For any given state, the value by the objective policy means that the expected value for the best action is for the best action.

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')]. \end{aligned}$$

Finite Markov Decision Process

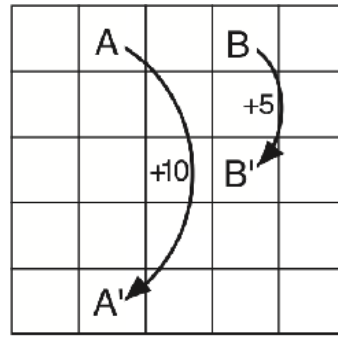
Bellman Optimality Equation

For any given state, the value by the objective policy means that the expected value for the best action is for the best action.

$$\begin{aligned} q_*(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]. \end{aligned}$$

Finite Markov Decision Process

Optimal Policy Estimation



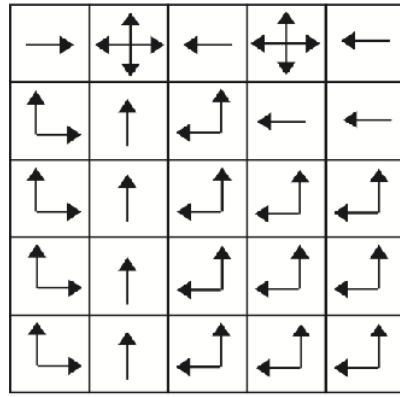
Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

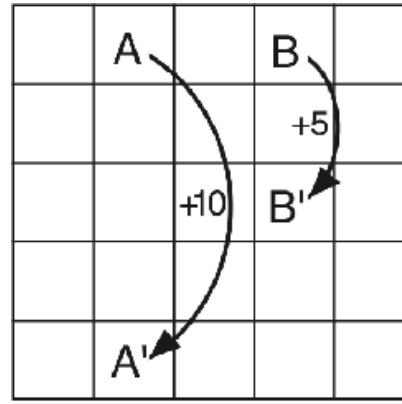
v_*

Finite Markov Decision Process

Estimated Optimal Policy



π_*



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

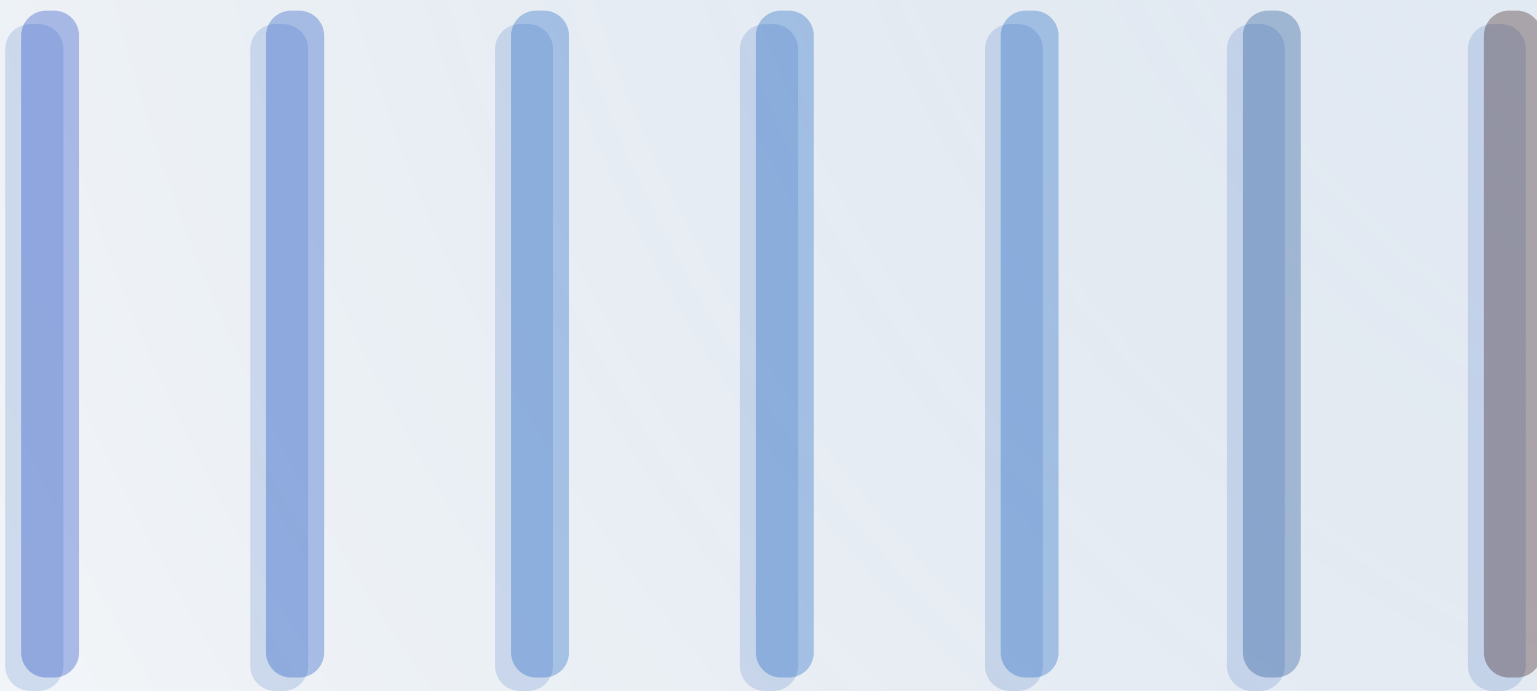
v_*

Finite Markov Decision Process

Next...

How do we calculate the value function following which policy?

How do I get a better policy?



Q-Learning



Q-Learning

Q-Learning Introduce

Reinforcement Learning



Bellman optimize equation \rightarrow Value iteration

Q-Learning

Q-Learning

Q-Learning Introduce

Q-Learning

Off-policy methods

Policy generating Episode is separate from policy evaluation & improvement target

Q-function Update

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

Bellman optimizing equation

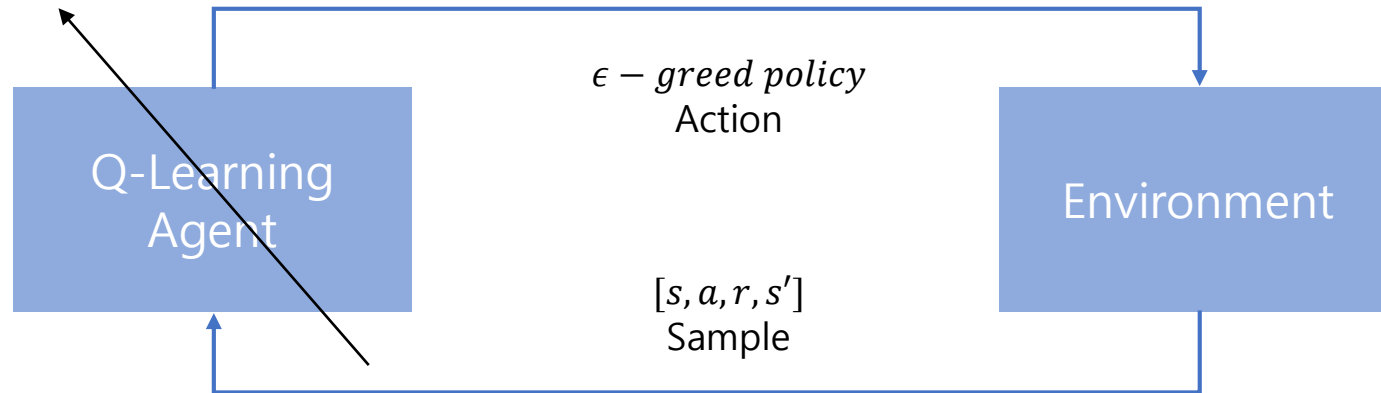
$$q^*(s, a) = E[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') | S_t = s, A_t = a]$$

Q-Learning

Q-Learning Introduce

Q-Learning Agent

Q – function update bellman optimize equation



Q-Learning

Q-Learning

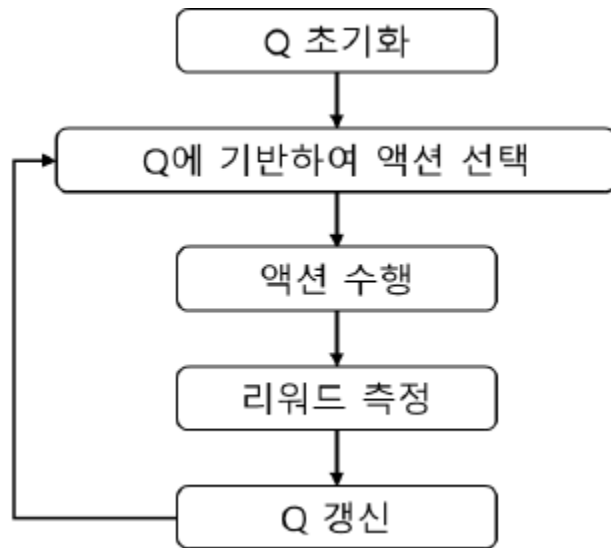
Definition

Markov decision-based reinforcement learning technique to find the optimal policy by calculating the future value (Q) of the behavior in a particular state.

Q-Learning

Q-Learning

Learning Procedure



Perform action, measure/update based on policy by reward

1. Initialize value table Q
2. Policy based action selecting /performing
3. Observe new state and reward
4. Update next state's max reward
5. Set new state, and repeat

Q-Learning

Q-Learning

Components

Part	Components	Explanation
Policy	<ul style="list-style-type: none">– Maximum rewards– Observation of future rewards	<ul style="list-style-type: none">– Select the highest Q-based action– $\pi(s) = \operatorname{argmax} Q(s,a)$
Bellman Equation	<ul style="list-style-type: none">– Policy iteration– Recursive function	<ul style="list-style-type: none">– Repeat finding optimal policies– Current highest rewards, future rewards
Q-Learning Algorithm	<ul style="list-style-type: none">– Table-based– Iteration approximation	<ul style="list-style-type: none">– Repeat Bellman Equation– Repeat-based Q-function approximation

Q-Learning

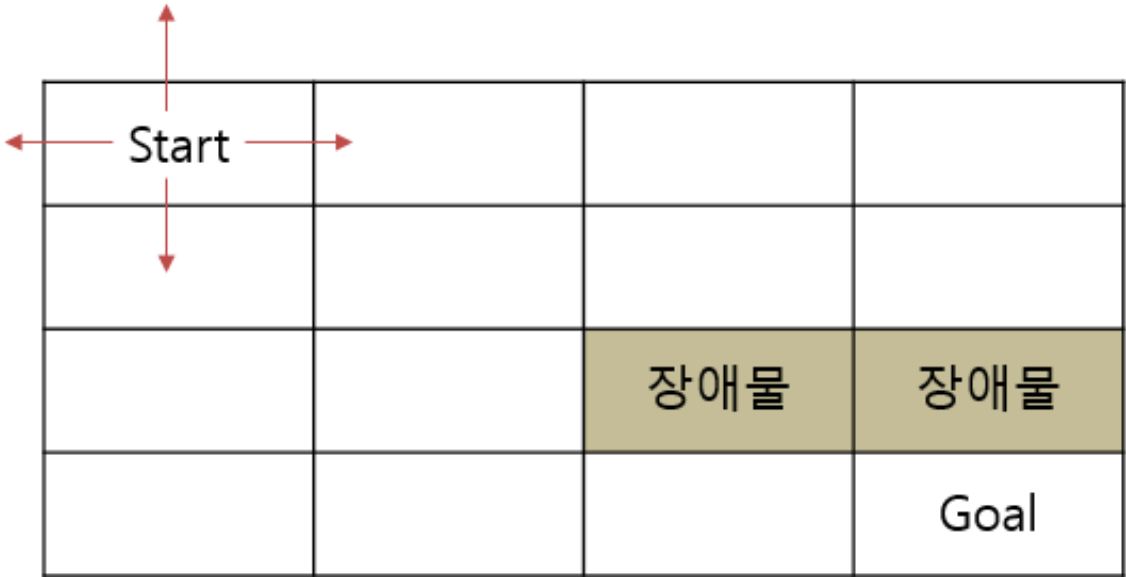
Q-Learning
Example



Human

Q-Learning

Q-Learning
Example



Computer: ???

Q-Learning

Q-Learning

Example



In this case, there are four things a computer can do at the start point (1,1), moving up, down, left, and right.

Moving up, down, left, and right is referred to as **action** and defines the current position (1, 1) as **state**.

Q-Learning

Q-Learning

Example

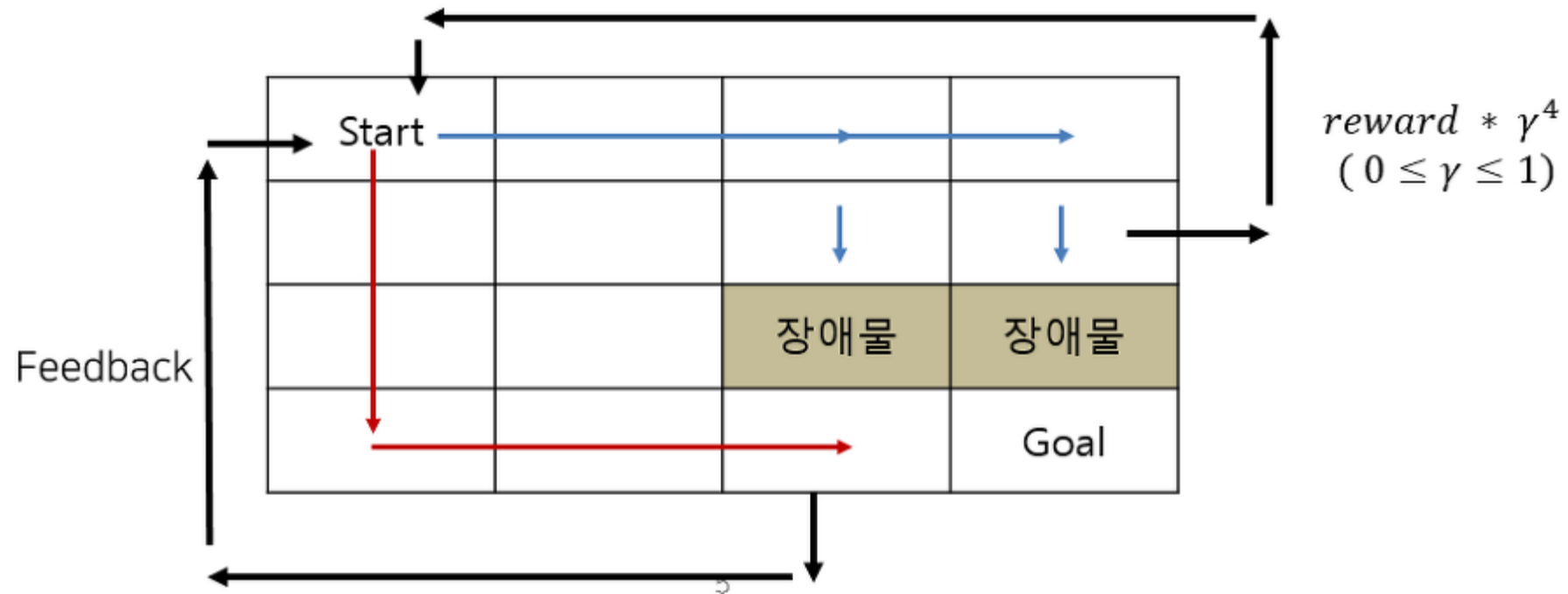


Introduce **Discount factor**(할인율) to give feedback that this action will be better for the future.

Q-Learning

Q-Learning

Example



Q-Learning

Q-Learning

Example

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

$$\pi^* = \operatorname{argmax}_{\pi} V^{\pi}(s) \quad \text{for all } s$$


Q-Learning

Q-Learning

Example

State에 따른 optimal $V^*(s)$ 와 $\pi^*(s)$ 를 다음과 같이 쓸 수 있음

$$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$
$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))] \rightarrow Q(s, a)$$


$$V^*(s) = \max_a Q(s, a)$$
$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

Q-Learning

Q-Learning

Example

Q learning

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

즉시 reward + 다음 state에서의 가장 큰 q값 * discount factor

Q-Learning

Q-Learning

Example

For each (s, a) initialize the table entry $\hat{Q}(s, a)$ to zero

Repeat (for each episode)

Observe the initial state s

Repeat (for each step of episode)

Select an action a from state s (e.g. ϵ -greedy) and execute it

Receive immediate reward r

Observe the new state s'

Update the table entry for $\hat{Q}(s, a)$ as follows:

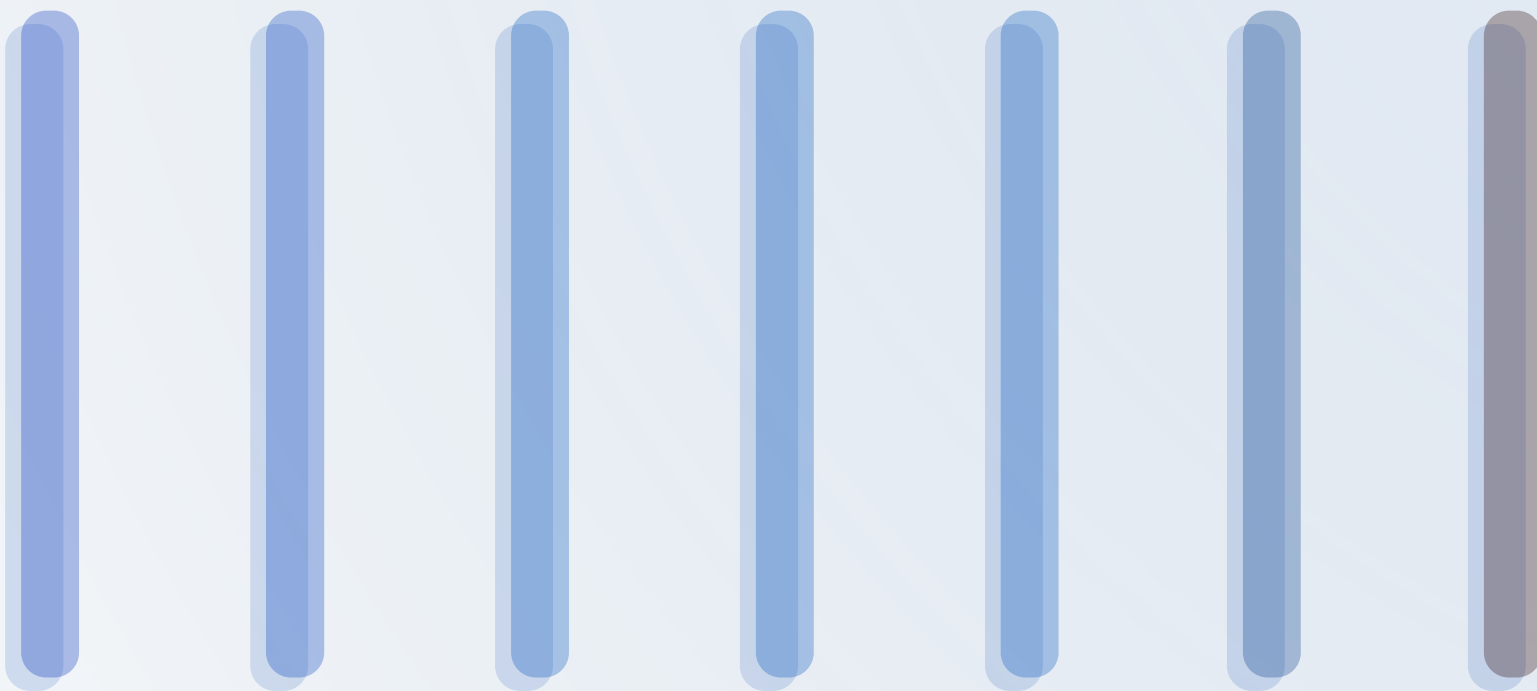
$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

$$s \leftarrow s'$$

Until s is terminal (goal state)

Exploration (탐험)





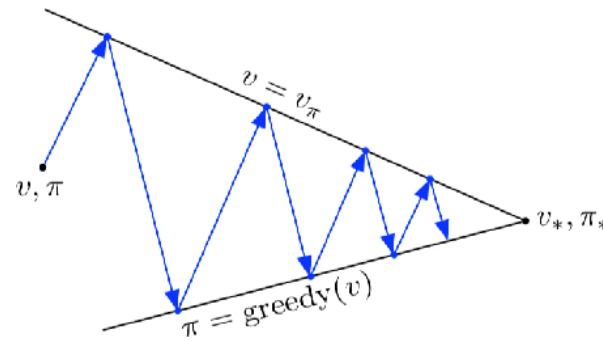
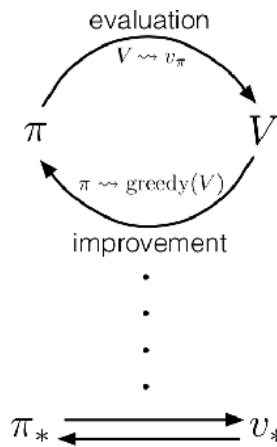


Dynamic Programming

Dynamic Programming

Why Dynamic Programming

To systematize and structure the use of value function to explore good policy.



Bootstrapping

Estimate the value of the current state based on the estimate of the value of the successive states

Dynamic Programming

Policy Evaluation

Calculate the state-value function v_π for any policy π
can be seen as a kind of prediction problem

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_k(s') \right] \quad \text{where all } s \in \mathcal{S}. \end{aligned}$$

When $k \rightarrow \infty$, v_k is converge to v_π
These algorithms are called iterative policy validation.

Dynamic Programming

Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π , the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')] \leftarrow \text{SWEEP : one routine around all the states}$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Dynamic Programming

Policy Improvement

Finding a better policy through a given value function is called a policy improvement

π' : *greedy policy of given π*

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

where $\arg \max_a x_a$ denotes the value of a at which the expression that follows is maximized

Dynamic Programming

Policy Improvement

$$\begin{aligned} \text{Proof of } V_{\pi}(s) \leq V_{\pi'}(s) \quad & v_{\pi}(s) \leq q_{\pi}(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_{\pi}(S_{t+2}) \mid S_{t+1}] \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_{\pi}(S_{t+2}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_{\pi}(S_{t+3}) \mid S_t = s] \\ &\vdots \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s] \\ &= v_{\pi'}(s). \end{aligned}$$

Dynamic Programming

Policy Iteration

Find **optimal policy** and **optimal value** by repeating policy evaluation and policy improvement is **policy iteration**

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

where $\xrightarrow{\text{E}}$ denotes a policy *evaluation* and $\xrightarrow{\text{I}}$ denotes a policy *improvement*.

Dynamic Programming

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

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

2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

old-action $\leftarrow \pi(s)$

$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

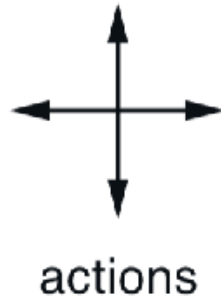
If *old-action* $\neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Dynamic Programming

Example

Grid World



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions

Dynamic Programming

Example

Grid World

$k = 0$

v_k for the
random policy

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

greedy policy
w.r.t. v_k

	↕↕↕	↕↕↕	↕↕↕
↔↔↔	↕↕↕	↕↕↕	↕↕↕
↔↔↔	↕↕↕	↕↕↕	↕↕↕
↔↔↔	↕↕↕	↕↕↕	

← random
policy

Dynamic Programming

Example

Grid World

$$k = 1$$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

	←	↕	↕
↑	↕	↕	↕
↕	↕	↕	↓
↕	↕	→	


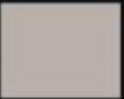
Dynamic Programming

Example

Grid World

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

	←	←	↕
↑	↖	↕	↓
↑	↕	↘	↓
↕	→	→	

Dynamic Programming

Example

Grid World

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

	←	←	↙
↑	↖	↙	↓
↑	↖	↘	↓
↙	→	→	

Dynamic Programming

Example

Grid World

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

	←	←	↙
↑	↖	↙	↓
↑	↖	↘	↓
↖	→	→	

Dynamic Programming

Example

Grid World

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

	←	←	↙
↑	↖	↙	↓
↑	↖	↘	↓
↖	→	→	

Starting from $k=3$, the 'greedy policy' is converged into the 'optimal policy'

Dynamic Programming

Example

Grid World

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

	←	←	↙
↑	↖	↙	↓
↑	↗	↘	↓
↘	→	→	

Starting from $k=3$, the 'greedy policy' is converged into the 'optimal policy'

Dynamic Programming

Example

Grid World

Policy Iteration Example

Dynamic Programming

Value Iteration

Value Iteration is the method to reflect the greedy selection in the policy improvement to the policy evaluation phase

The following methods can significantly reduce the number of repetitions.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_k(s')], \quad \text{for all } s \in S. \end{aligned}$$

Dynamic Programming

Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

```
|  $\Delta \leftarrow 0$   
| Loop for each  $s \in \mathcal{S}$ :  
|    $v \leftarrow V(s)$   
|    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$   
|    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
```

until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$$

Dynamic Programming

Example

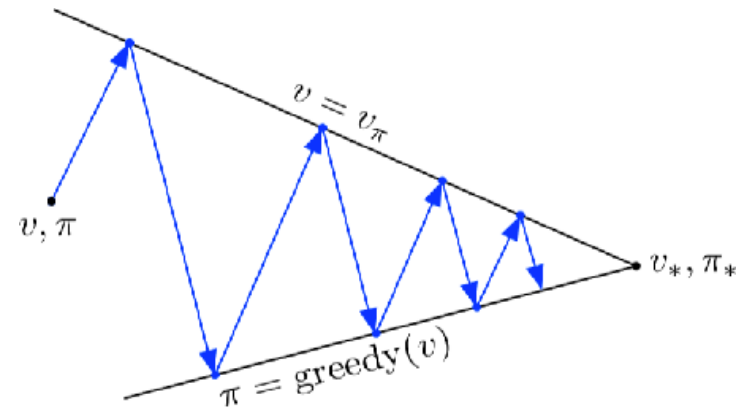
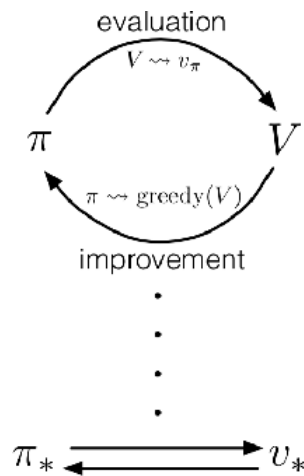
Grid World

Value Iteration Example

Dynamic Programming

Generalized Policy Iteration

The process in which Policy Evaluation and Policy Implementation interact with each other is called Generalized Policy Iteration (GPI)



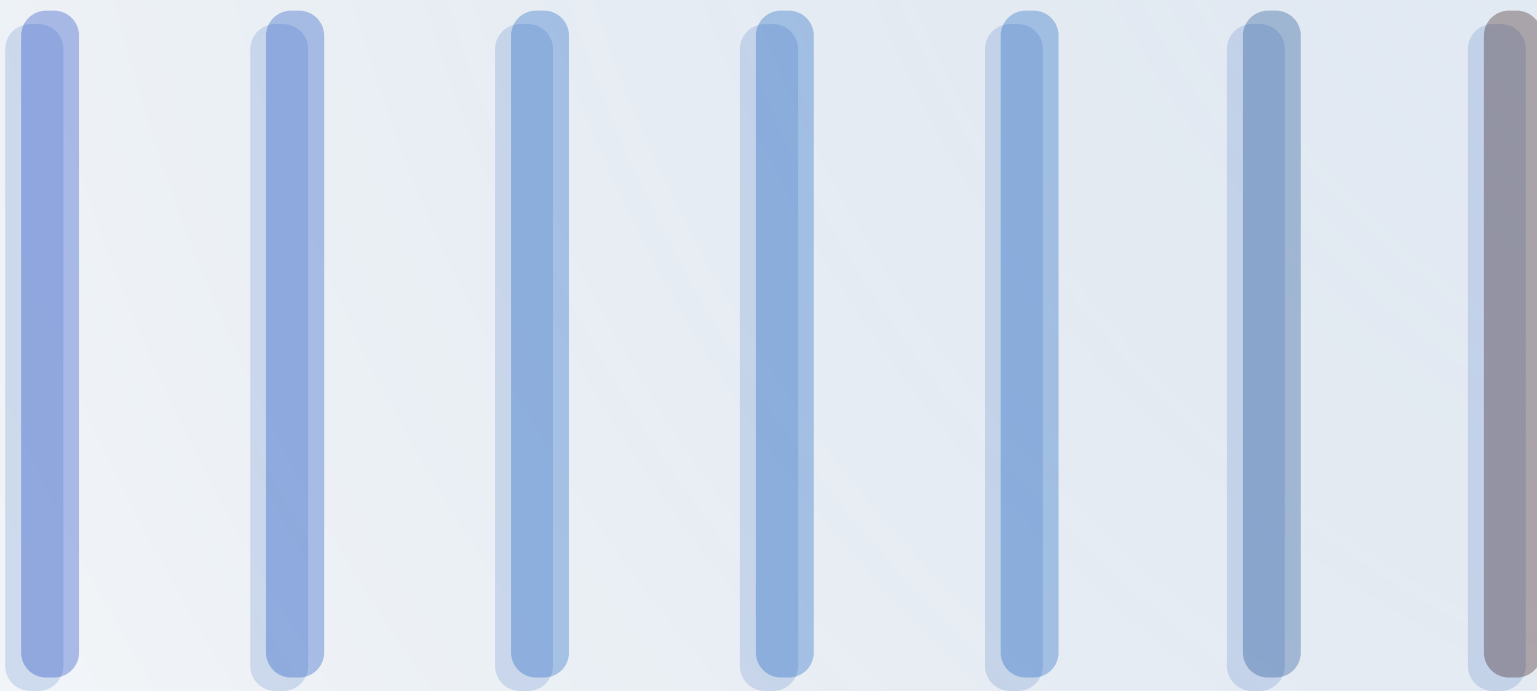
Dynamic Programming

Dynamic Programming

Dynamic programming requires significantly higher calculations and accurate models

What if the information from real-world experience interacted with the environment is reflected in live time?

Model-free RL Methods



Monte Carlo Methods



Monte Carlo Methods

Monte Carlo Methods

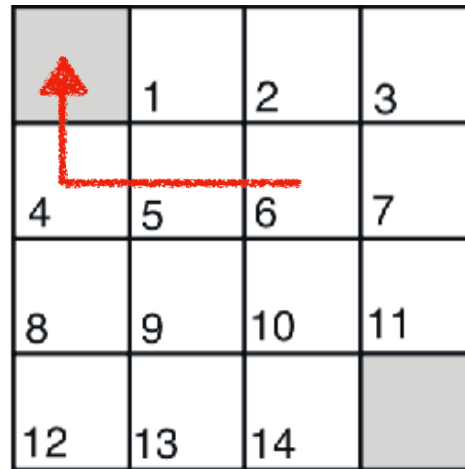
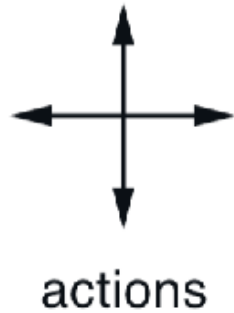
Monte Carlo methods are not assumed to be fully aware of the environment.

Monte Carlo methods require only real experience.

After experiencing one episode, calculate the average return for each state or state-action pair

Monte Carlo Methods

Monte Carlo Methods Introduction



$$R_t = -1$$

on all transitions

Update after experience whole one episode

Monte Carlo Methods

Monte Carlo Methods

Introduction

First-visit MC method

Get average return for only the first state visited in one episode

Every-visit MC method

Get average return for every state visited in one episode

Monte Carlo Methods

Monte Carlo Methods

Introduction

First-visit MC method

Get average return for only the first state visited in one episode

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

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

Monte Carlo Methods

Monte Carlo Methods

For Action Value function

Since the model does not exist, there is no information on the transition between states.

So, estimates of state value function alone are not sufficient.

Both First-visit MC method and Every-visit MC method are valid

Monte Carlo Methods

Monte Carlo Methods

Introduction

A simple method for ensuring convergence

Evaluation and improvement on an episode-by-episode basis

Exploring Starts

Monte Carlo Methods

Monte Carlo Methods Introduction

Exploring Starts

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
 $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
 $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability > 0

Generate an episode from S_0, A_0 , following π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$

Monte Carlo Methods

Monte Carlo Methods Introduction

Exploring Starts

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
 $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
 $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ ~~such that all pairs have probability > 0~~
Generate an episode from S_0, A_0 , following π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
 $G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$

Almost Impossible

Monte Carlo Methods

Monte Carlo Control
Without Exploring Starts

What if the Agent are to explore itself?

On-policy methods

The policy that generates the episode and the policy evaluation & improvement target are the same

Off-policy methods

Policy generating Episode is separate from policy evaluation & improvement target

Monte Carlo Methods

Monte Carlo Control Without Exploring Starts

On-policy methods

On-policy first-visit MC control (for ϵ -soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\epsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ϵ -soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, \dots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg \max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} \frac{1 - \epsilon + \epsilon/|\mathcal{A}(S_t)|}{\epsilon/|\mathcal{A}(S_t)|} & \text{if } a = A^* \\ \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{if } a \neq A^* \end{cases}$$

Same Policy

S_t : Number of selective actions same as exercise at k – armed bandit

Monte Carlo Methods

Monte Carlo Control
Without Exploring Starts

What if the Agent are to explore itself?

On-policy methods

The policy that generates the episode and the policy evaluation & improvement target are the same

Off-policy methods

Policy generating Episode is separate from policy evaluation & improvement target

Monte Carlo Methods

Monte Carlo Control Without Exploring Starts

Off-policy methods

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \in \mathbb{R}$ (arbitrarily)

$C(s, a) \leftarrow 0$

$\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$ (with ties broken consistently)

Loop forever (for each episode):

$b \leftarrow$ any soft policy

Generate an episode using b : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

$W \leftarrow 1$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)

If $A_t \neq \pi(S_t)$ then exit For loop

$W \leftarrow W \frac{1}{b(A_t|S_t)}$

Monte Carlo Methods

Monte Carlo Control Without Exploring Starts

On-policy methods

It can't find the optimal policy because it have to constantly explore

So it separate its policy two variation

Target Policy

Policy to be learned

Behavior Policy

Policy that determines the behavior
(Generate Episodes)

Monte Carlo Methods

Monte Carlo Control Without Exploring Starts

On-policy methods

Most off-policy methods use importation sampling

Importance sampling is a method to estimate the value of an expected value by using two different probability distributions.

To understand Importance sampling, we have to know about Importance-sampling rate first

Monte Carlo Methods

Trajectory : A set of contiguous timestamps, from a single episode, or a single part of a continuous problem

Monte Carlo Control

Without Exploring Starts

Importance-sampling rate

When there is given trajectory $A_t, S_{t+1}, A_{t+1}, \dots, S_t$

The probability of occur that trajectory by some policy π is same as below

$$\begin{aligned} & \Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ &= \pi(A_t | S_t) p(S_{t+1} | S_t, A_t) \pi(A_{t+1} | S_{t+1}) \cdots p(S_T | S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k | S_k) p(S_{k+1} | S_k, A_k), \end{aligned}$$

Monte Carlo Methods

Monte Carlo Control

Without Exploring Starts

Importance-sampling rate

When a given trajectory is created by the behavior policy, the relative probability of it occurring trajectory in the target policy is expressed as follows equation and this is called the **import-sampling rate**

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

Monte Carlo Methods

Monte Carlo Control

Without Exploring Starts

Ordinary Importance-sampling

The method of estimating value with following equation is called ordinal import-sampling

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|} \quad \rho_{t:T-1} \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

G_t : The return after t up through T(t)

$\tau(s)$: The total number of visits on state s

Monte Carlo Methods

Monte Carlo Control

Without Exploring Starts

Incremental Implementation

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \quad n \geq 2,$$

For calculation efficiency, induced incubation of weighted importation-sampling is as below

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n} [G_n - V_n], \quad n \geq 1,$$

and

$$C_{n+1} \doteq C_n + W_{n+1},$$

where $C_0 \doteq 0$ (and V_1 is arbitrary and thus need not be specified).

Monte Carlo Methods

Example

Grid World

Monte Carlo
With State Value

$$V_t = \frac{1}{t} \sum_{j=1}^t G_j$$

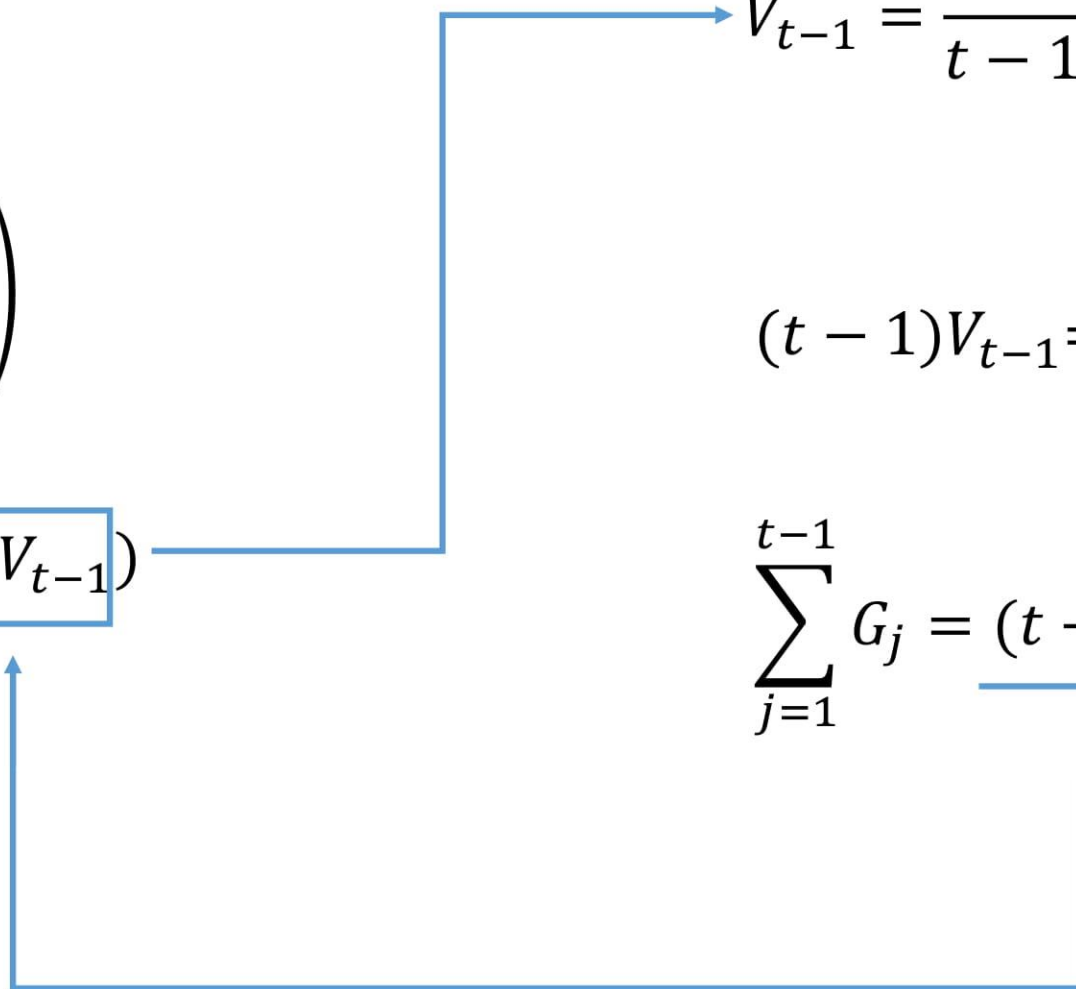
$$= \frac{1}{t} \left(G_t + \sum_{j=1}^{t-1} G_j \right)$$

$$= \frac{1}{t} (G_t + \boxed{(t-1)V_{t-1}})$$

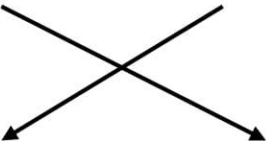
$$V_{t-1} = \frac{1}{t-1} \sum_{j=1}^{t-1} G_j$$

$$(t-1)V_{t-1} = \sum_{j=1}^{t-1} G_j$$

$$\sum_{j=1}^{t-1} G_j = \underline{(t-1)V_{t-1}}$$



$$V_t = \frac{1}{t} (G_t + (t-1)V_{t-1})$$

$$= \frac{1}{t} G_t + V_{t-1} - \frac{1}{t} \cdot V_{t-1}$$


$$V_{t-1} + \frac{1}{t} G_t - \frac{1}{t} \cdot V_{t-1}$$

$$= V_{t-1} + \frac{1}{t} (G_t - V_{t-1})$$

$$= V_{t-1} + a(G_t - V_{t-1})$$

$$V_t = V_{t-1} + a(G_t - V_{t-1})$$

↓
새로 업데이트 될
가치함수

↓
이전 가치함수

↓
실제 받은
리워드

↓
이전 가치함수 예측값

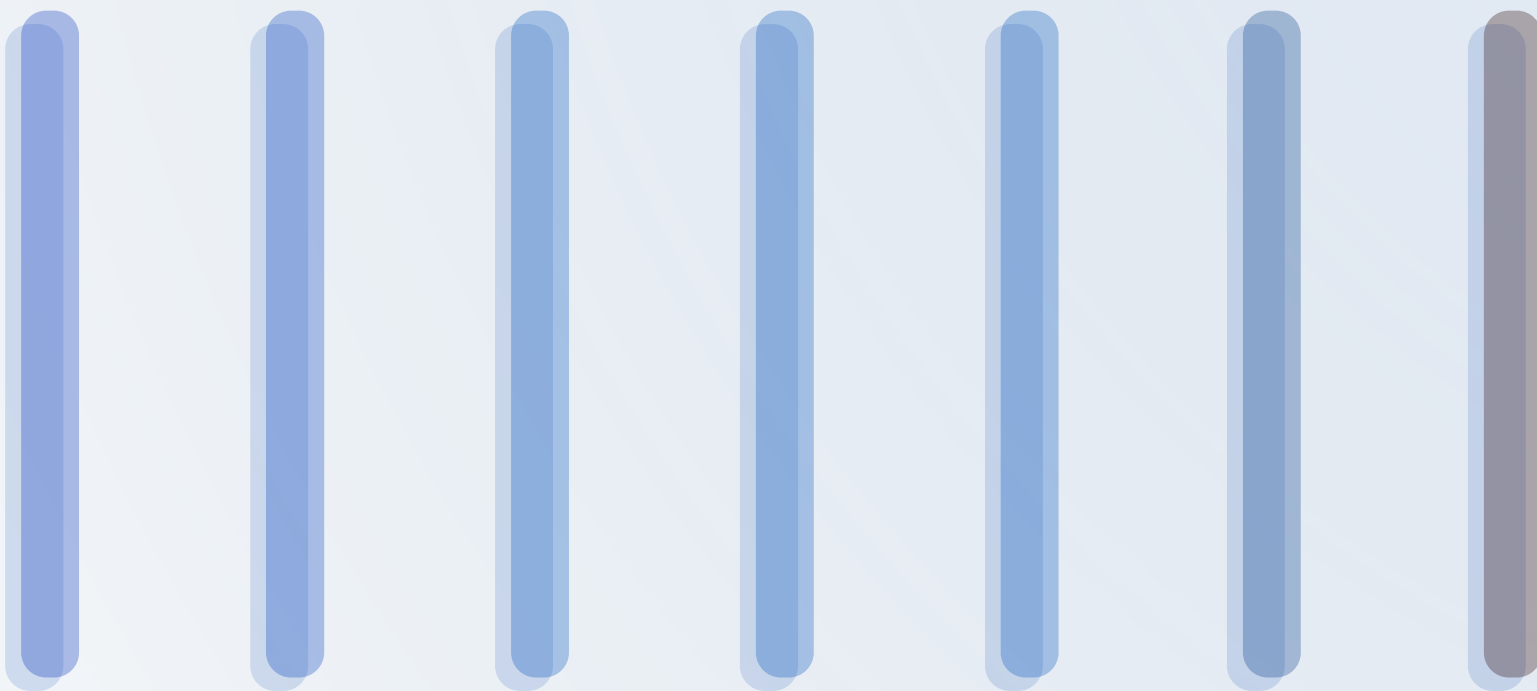
$$V_{\pi}(s_t) \leftarrow V_{\pi}(s_t) + a(G_t - V(s_t))$$

Monte Carlo Methods

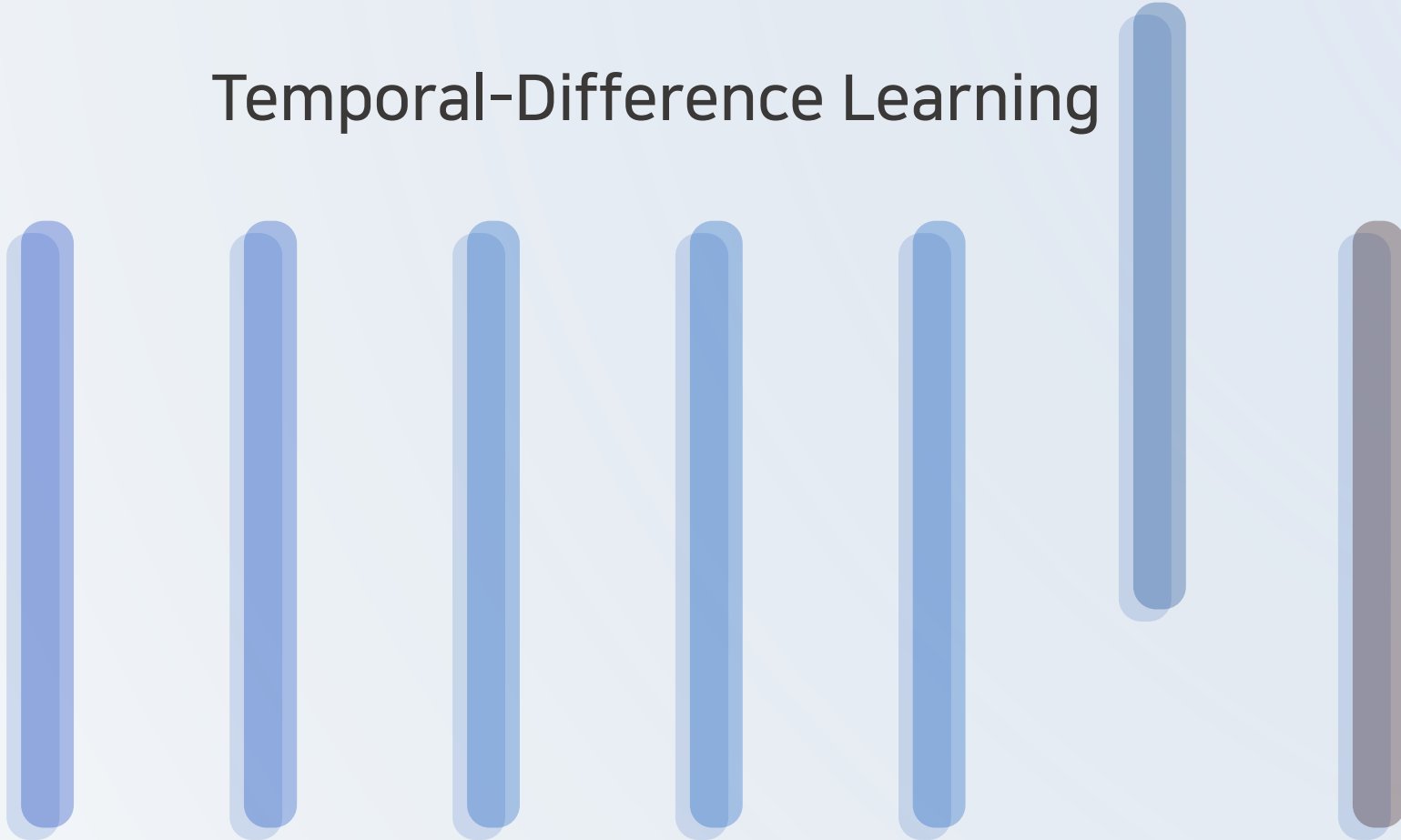
Dynamic Programming

Monte Carlo Methods updates the policy on an episode-by-episode basis.

What if the episode is very long or solve Continuing task problems?



Temporal-Difference Learning



Temporal-Difference Learning

Temporal-Difference

Introduction

Like MC, TD learns from real experience without model.

TD learns through bootstrapping without waiting for final outcome like DP.

If the every-visit MC method in the nonstationary environment was as follows,

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

the TD is expressed as follows

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Temporal-Difference Learning

Temporal-Difference

Introduction

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

Algorithm parameter: step size $\alpha \in (0, 1]$

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

$A \leftarrow$ action given by π for S

 Take action A , observe R, S'

$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

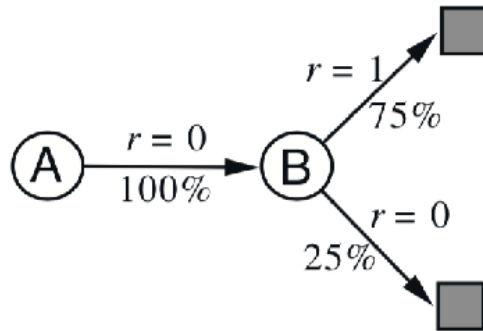
$S \leftarrow S'$

 until S is terminal

Temporal-Difference Learning

Temporal-Difference

Example



A, 0, B, 0

B, 1

B, 1

B, 1

B, 1

B, 1

B, 1

B, 0

Eight episodes on the right were collected by the Unknown Markov forward process on the left.

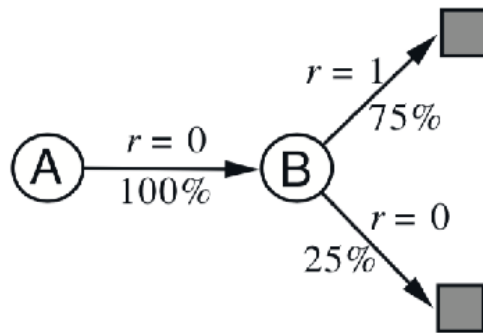
How much is the $V(A)$ calculated by MC and TD (0) when

Step size = 1, Gamma=1 and $V(B) = 3/4$, $V(A) = 0$?

Temporal-Difference Learning

Temporal-Difference

Example



A, 0, B, 0

B, 1

B, 1

B, 1

B, 1

B, 1

B, 1

B, 0

$$\text{TD}(0): V(A) = V(A) + 1 * (0 + 1 * V(B) - V(A)) = 3/4$$

$$\text{MC}: V(A) = V(A) + 1 * (G(A) - V(A)) = 0$$

Although it can be seen that $V(A) = V(B)$ is transferred from A to B at a 100% probability, the MC does not reflect this because it is learned in only one episode unit.

Temporal-Difference Learning

Sarsa: On-policy TD Control

Example

Learn action-value function instead of state-value function like a TD revision method

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

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

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

 Loop 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

Temporal-Difference Learning

Example

Grid World

Sarsa

Temporal-Difference Learning

Q-learning: Off-policy TD Control

Example

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop 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

Temporal-Difference Learning

Example

Grid World

Q-Learning

Temporal-Difference Learning

Double Q-learning: Off-policy TD Control

Example

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right].$$

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using the policy ε -greedy in $Q_1 + Q_2$

 Take action A , observe R, S'

 With 0.5 probability:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left(R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A) \right)$$

 else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left(R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A) \right)$$

$S \leftarrow S'$

 until S is terminal

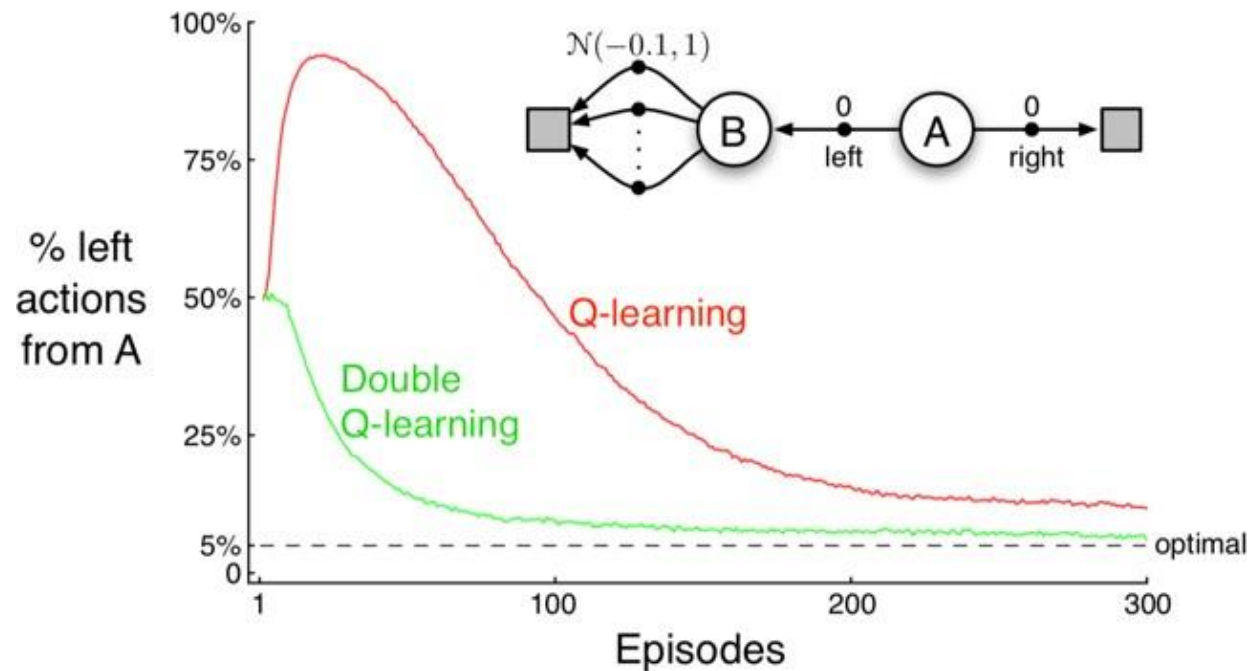
Double Q-learning
uses two Q
functions

This function
reduces the
positive bias

Temporal-Difference Learning

Double Q-learning: Off-policy TD Control

Introduce



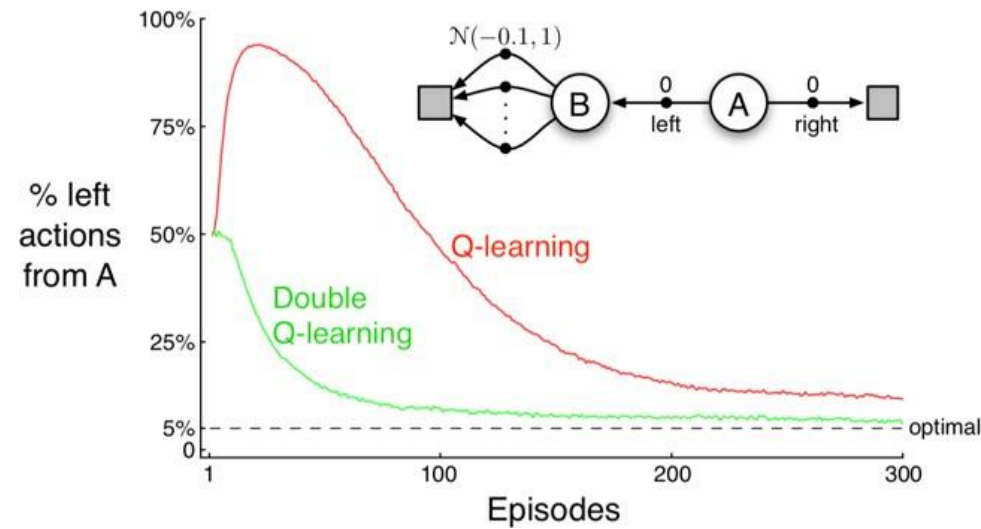
Let's call A the initial position. The expected value of $V(B)$ is -0.1 , so moving to the left is not a good decision

But Q-learning's optimism continues to make the wrong choices early on, as shown in the picture

Temporal-Difference Learning

Double Q-learning: Off-policy TD Control

Introduce



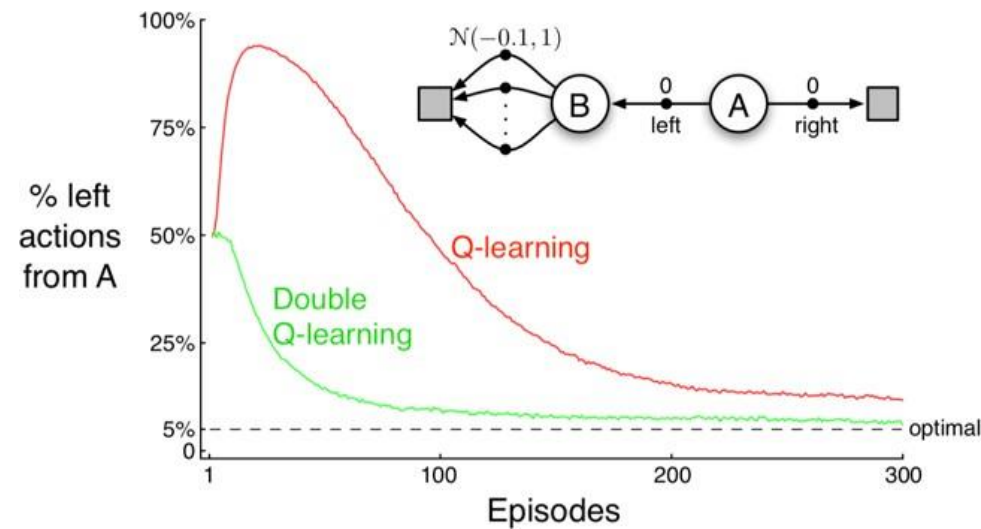
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

Positive Bias Occur

Temporal-Difference Learning

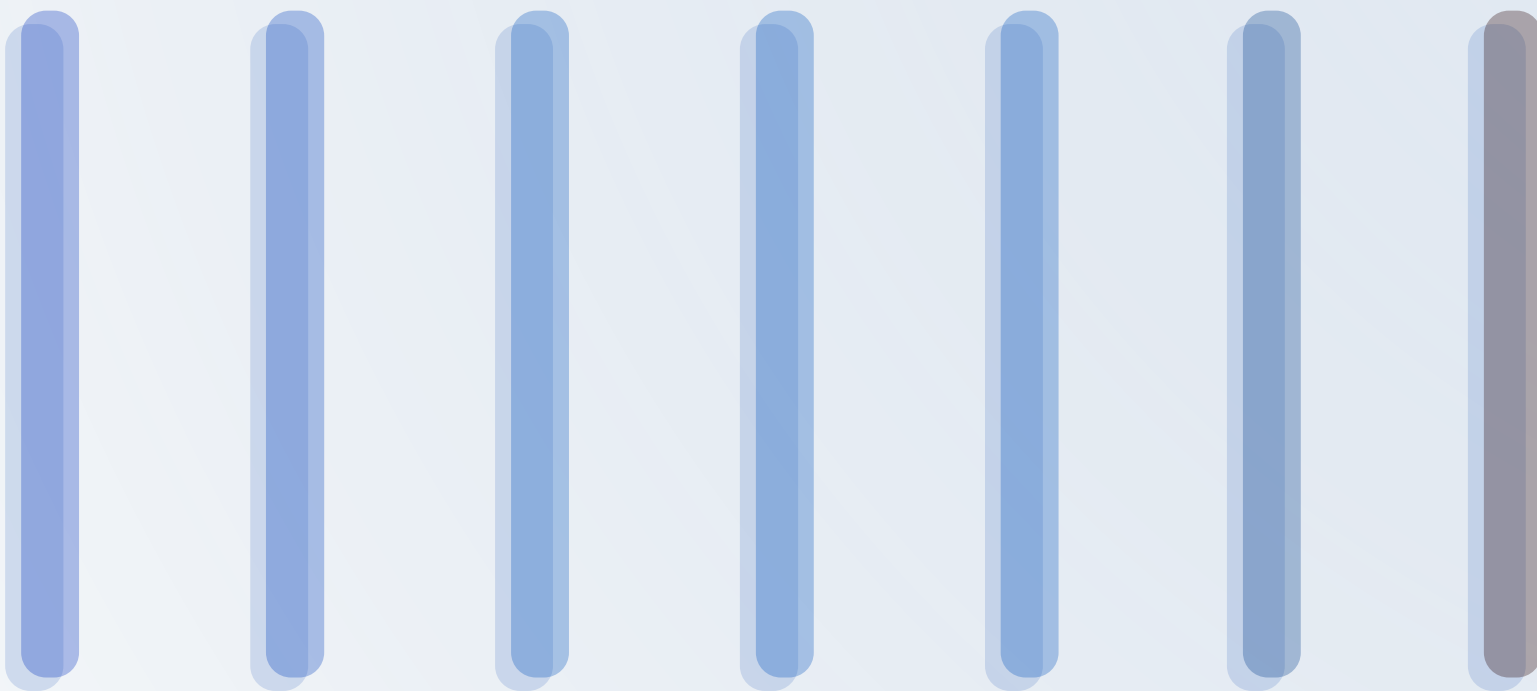
Double Q-learning: Off-policy TD Control

Introduce



$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right].$$

To Solve Positive Bias : Separate Select and Evaluate Q function





Dynamic Q-Learning

Dynamic Q-Learning

Introduce

In order to the Reinforcements learning agents to work well in matters of real world complexity...

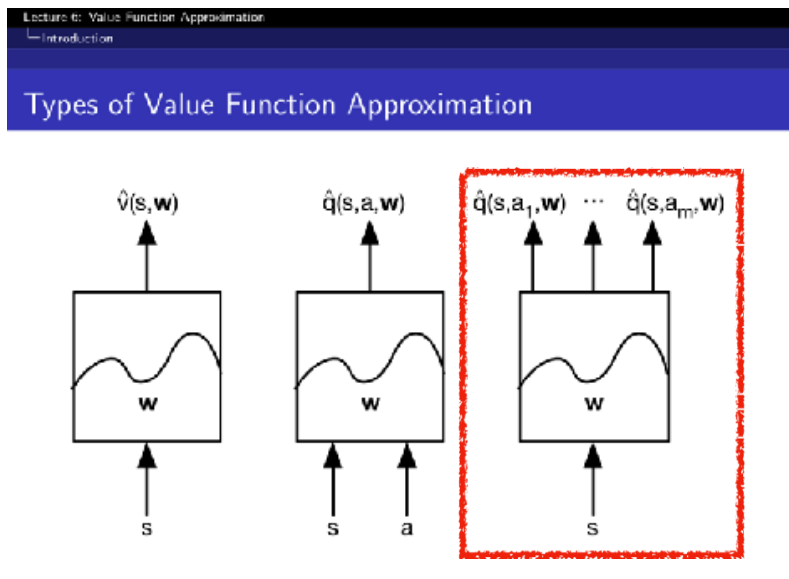
It is important to be able to obtain the representation from the sensor inputs of considerable dimensions

The obtained presentation should generalize the past experience so that it can be applied well even in new situations

Dynamic Q-Learning

Dynamic Q-Learning

Introduce



Deep Convolutional Neural Network is showing great performance as a non-linear function application.

Why don't we use the CNN structure to create an action-value function by inputting raw sensor data?

Dynamic Q-Learning

Dynamic Q-Learning

Introduce

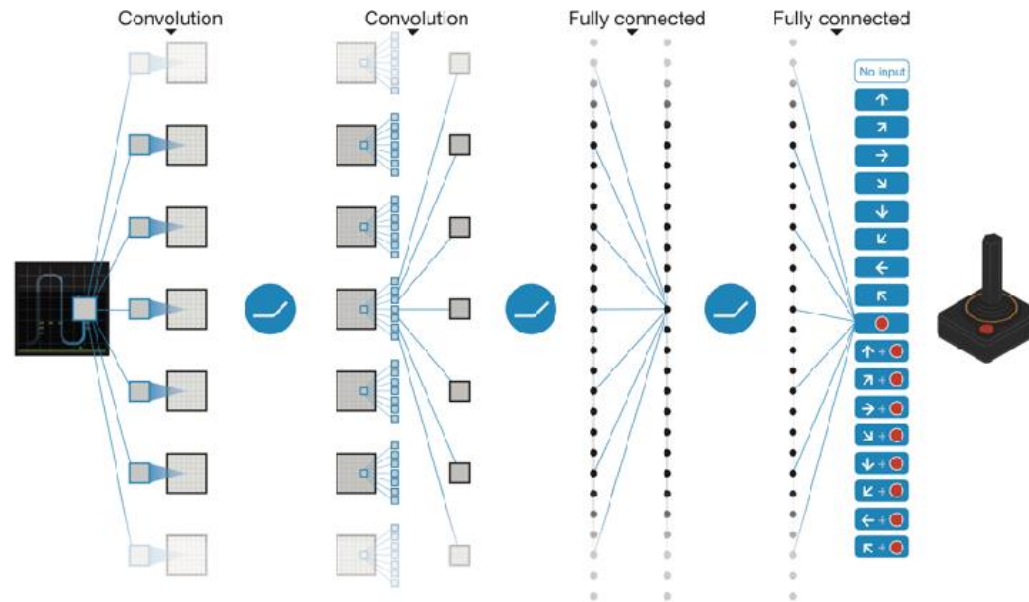


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ , followed by three convolutional layers (note: snaking blue line

symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0, x)$).

Dynamic Q-Learning

Dynamic Q-Learning

Architecture

Input: 84x84x4 (by preprocessing map ϕ)

32 convolutional filters of 8x8 with stride 4 followed by a rectifier non-linearity

64 convolutional filters of 4x4 with stride 2 followed by a rectifier non-linearity

64 convolutional filters of 3x3 with stride 1 followed by a rectifier non-linearity

Fully connected layer with 512 nodes + a rectifier non-linearity

Fully connected linear layer with a single output for each valid action

Dynamic Q-Learning

Dynamic Q-Learning

Architecture

Non-stationary targets

Dynamic Q-Learning

Dynamic Q-Learning

Architecture
Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

Dynamic Q-Learning

Example

Cart-Pole

DQN

