강화학습 기초

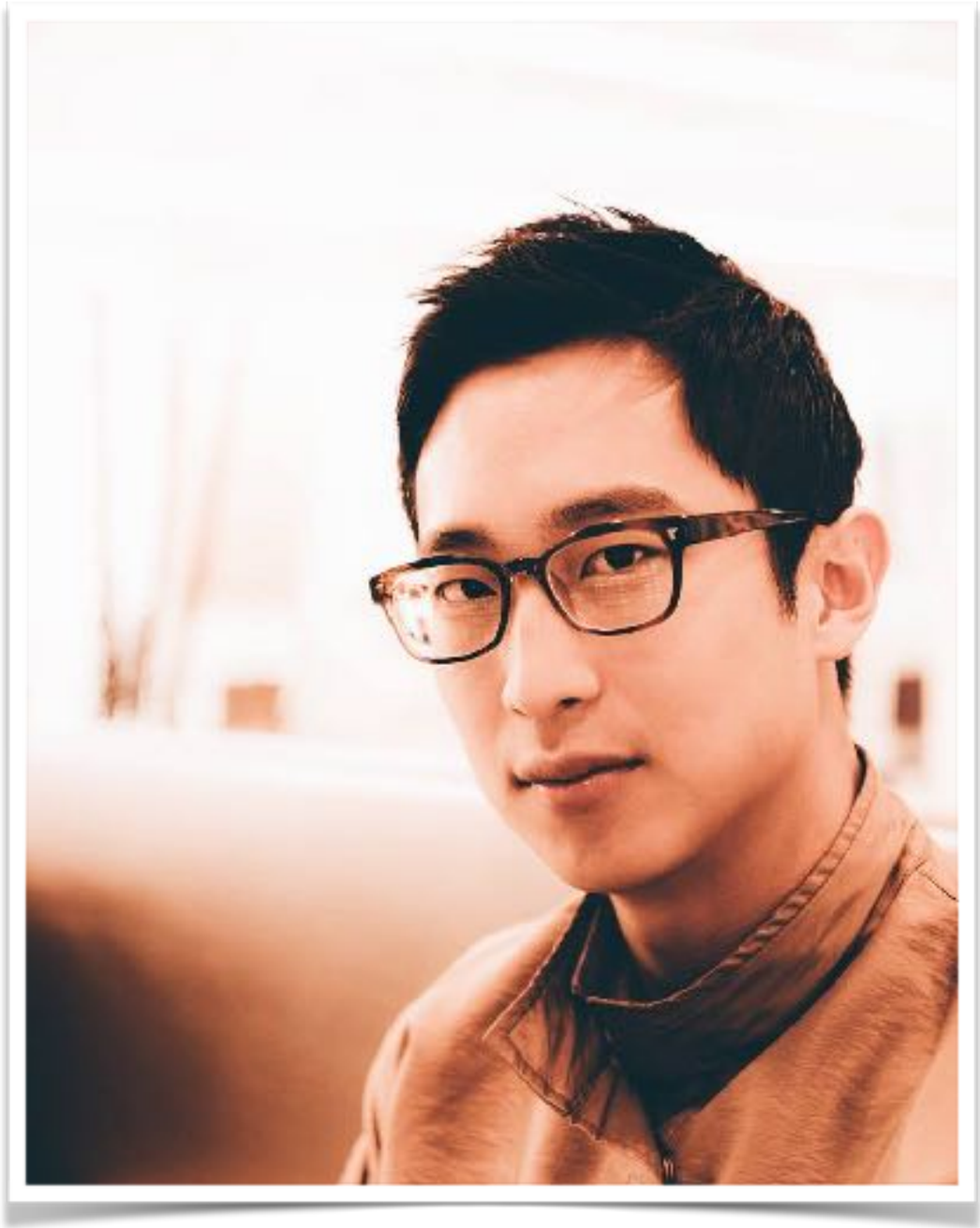
Reinforcement Learning Basics

박진우(Curt Park)

The 9th KIAS CAC Summer School
2018.06.28 - 29

발표자 소개

- **소속:** 모두의 연구소
- **연구주제:**
  - 다중 인물에 대한 합성 얼굴 이미지 생성
  - 강화학습을 통한 게임 에이전트 학습
- **기타활동:**
  - 모두를 위한 컨벡스 최적화
  - Sutton&Barto 이론부터 구현까지



이 발표는?

- **범위**
  - 고전강화학습 + 딥러닝(약간)
- **목표**
  - Classic RL에서 Deep RL에 이르기까지의 전반적 내용 파악

목차

목차

Reinforcement Learning

The nature of learning

- 환경과의 상호작용
  - 일련의 행위(action)를 하는 과정에서 환경과의 상호작용에서 얻어지는 정보(행위에 대한 결과)를 바탕으로 우리는 목표를 성취할 수 있는 방법에 대해 점차 깨우쳐가게 된다.

The nature of learning

- 환경과의 상호작용
  - 일련의 행위(action)를 하는 과정에서 환경과의 상호작용에서 얻어지는 정보(행위에 대한 결과)를 바탕으로 우리는 목표를 성취할 수 있는 방법에 대해 점차 깨우쳐가게 된다.

이미지 출처:
http://m.befe.co.kr/webzine/webzineView.befe?bbs_seq=46487
http://www.caoquefuma.com/2017/10/atualidade-em-xeque-o-voto.html

The nature of learning

- 환경과의 상호작용

  - 상호작용을 통한 학습은 대부분의 learning and intelligence 이론들의 기저에 있는 발상

➡ 상호작용과 학습에 대한 계산적 관점의 접근 (computational approach)

➡ 상호작용을 통한 목표지향적 학습 (goal-directed learning from interaction)

The nature of learning

- 환경과의 상호작용

  - 상호작용을 통한 학습은 대부분의 learning and intelligence 이론들의 기저에 있는 발상

➡ 상호작용과 학습에 대한 계산적 관점의 접근 (computational approach)

➡ 상호작용을 통한 목표지향적 학습 (goal-directed learning from interaction)

강화학습 (Reinforcement Learning)!

Reinforcement Learning?

- 강화학습이란, 주어진 어떤 상황(situation)에서 보상(reward)을 최대화 할 수 있는 행동(action)에 대해 학습하는 것

- 학습의 주체(agent)가 상황에 가장 적합한 행동을 찾기까지는 수많은 시행착오가 필요

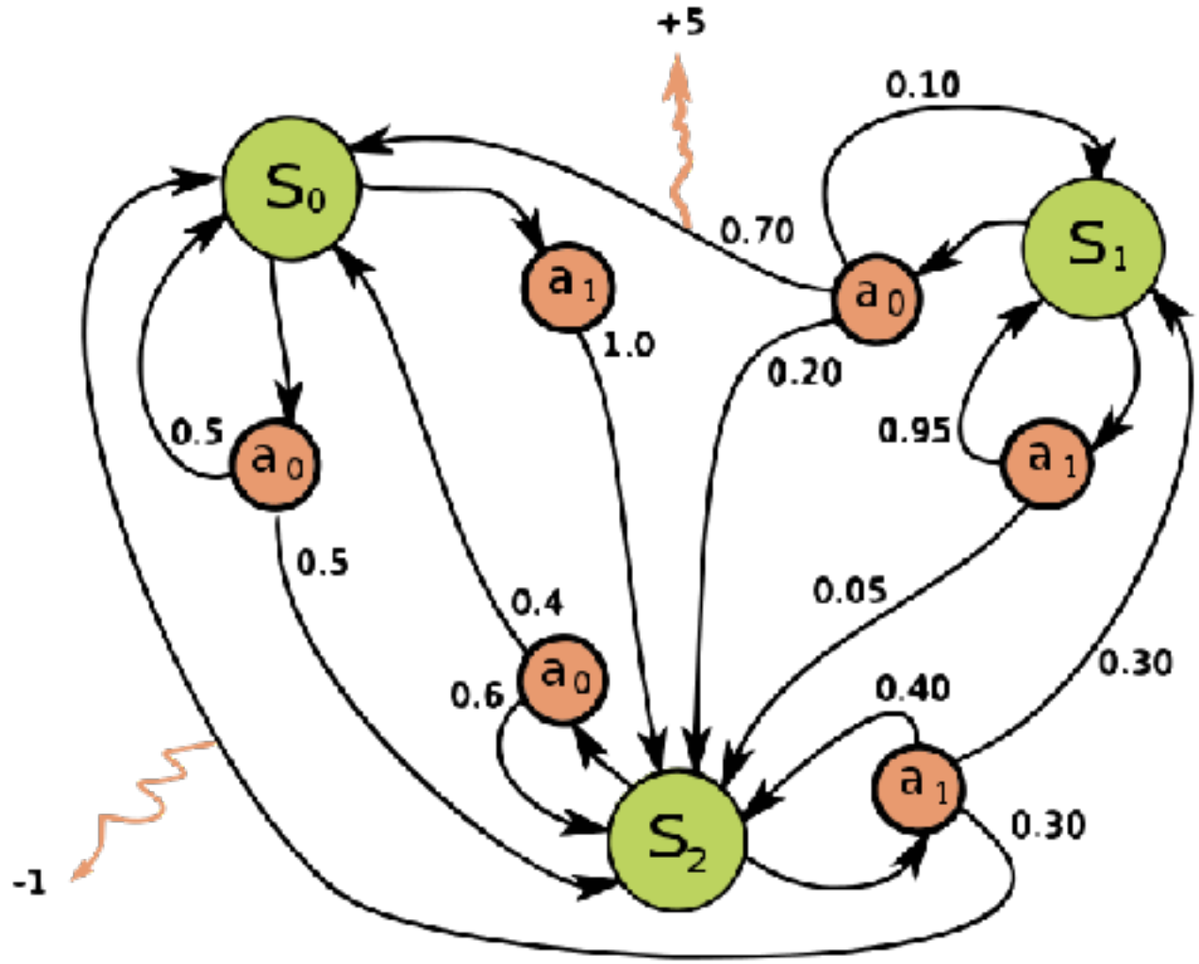
➡ **Trial-and-error search**

- 복잡한 상황에서는 현재 선택한 행동이 미래의 순차적인 보상에 영향

➡ **Delayed reward**

Reinforcement Learning?

- 강화학습 문제는 Markov Decision Process(MDP)로 표현

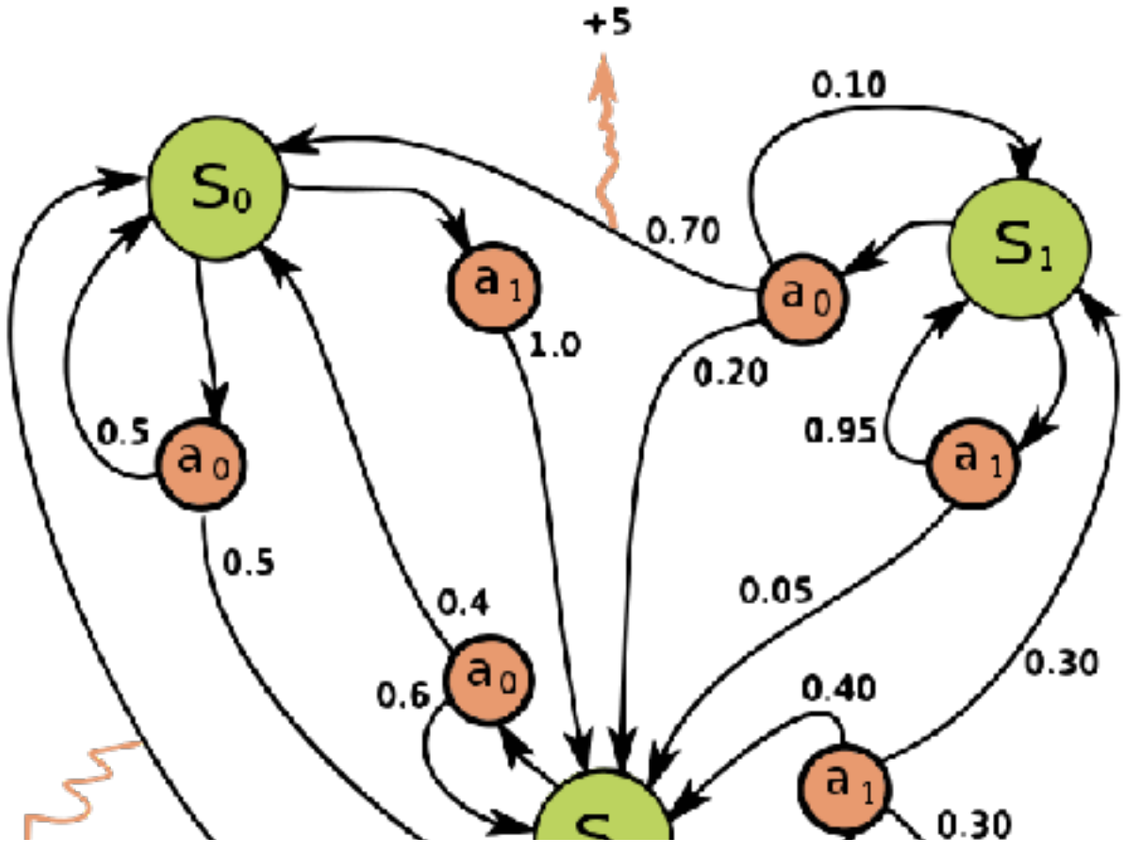


- MDP는 sensation, action, goal의 세 가지 개념을 포함

이미지 출처:
https://en.wikipedia.org/wiki/Markov_decision_process

Reinforcement Learning?

- 강화학습 문제는 Markov Decision Process(MDP)로 표현



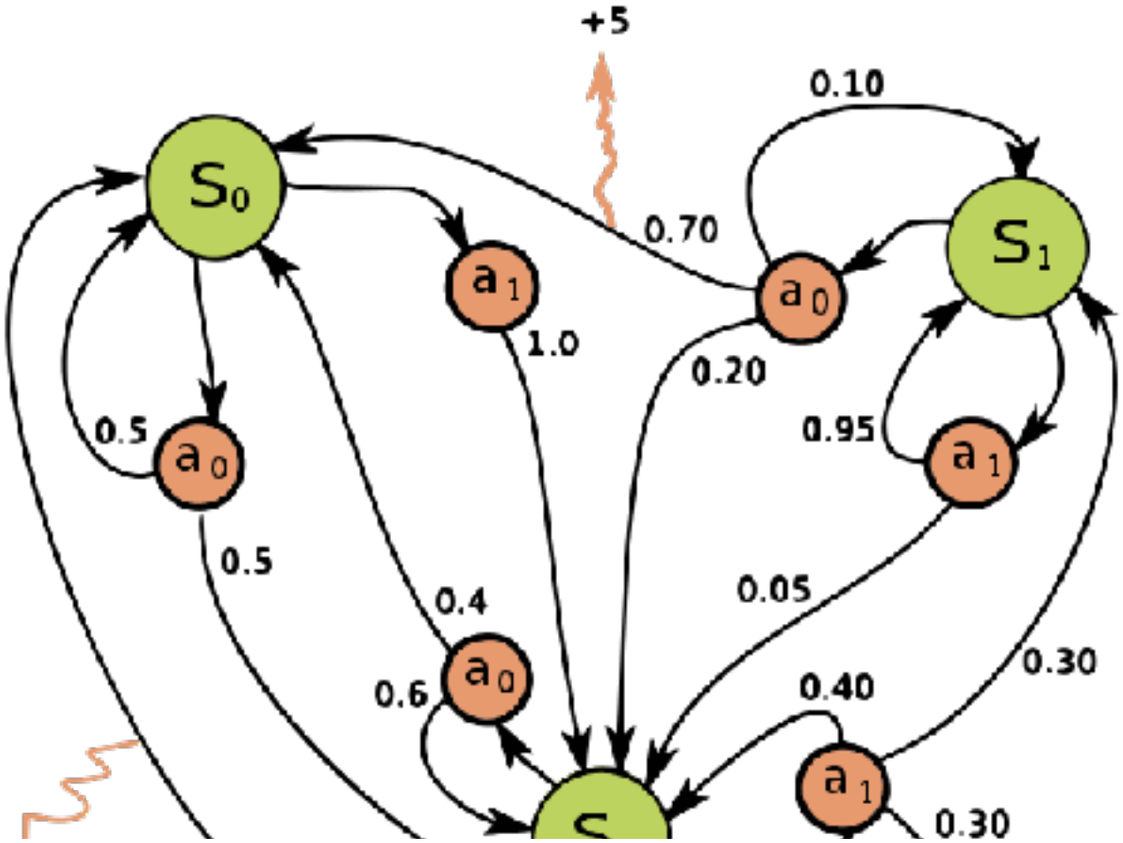
Agent는 환경이 어떤 상태(state)인지 인지할 수 있어야 한다.

- MDP는 sensation, action, goal의 세 가지 개념을 포함

이미지 출처:
https://en.wikipedia.org/wiki/Markov_decision_process

Reinforcement Learning?

- 강화학습 문제는 Markov Decision Process(MDP)로 표현



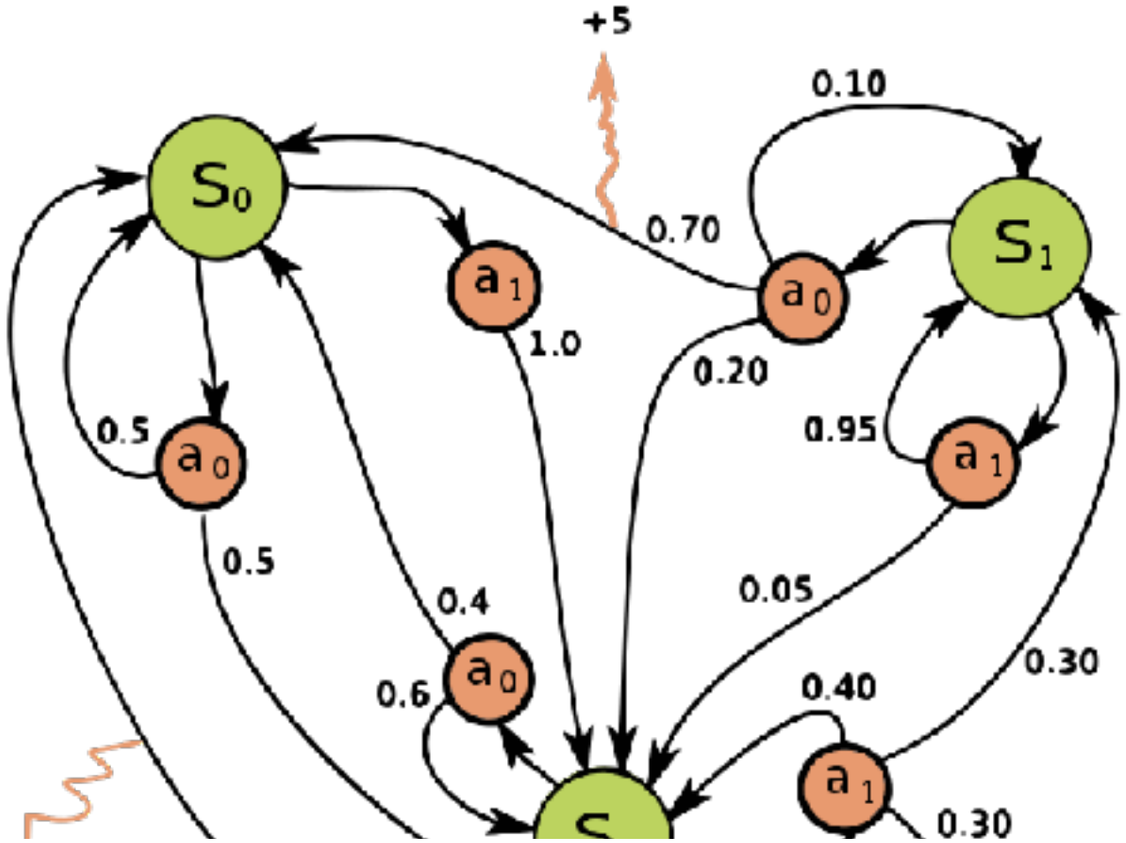
Agent는 주어진 상태(state)에 따라 행동(action)을 결정

- MDP는 sensation, action, goal의 세 가지 개념을 포함

이미지 출처:
https://en.wikipedia.org/wiki/Markov_decision_process

Reinforcement Learning?

- 강화학습 문제는 Markov Decision Process(MDP)로 표현



강화학습 문제에는 목표(goal)가 있어야 함

- MDP는 sensation, action, goal의 세 가지 개념을 포함

이미지 출처:
https://en.wikipedia.org/wiki/Markov_decision_process

Reinforcement Learning?

- 학습의 세 가지 종류
  - 지도학습 (Supervised Learning)
  - 비지도학습 (Unsupervised Learning)
  - 강화학습 (Reinforcement Learning)

Reinforcement Learning?

- 지도학습에서는,
  - 어떤 도메인의 전문가로부터 적절히 분류된(labeled) 학습 데이터를 제공받는다.
  - 학습데이터를 통해 학습함으로써 새로운 입력에 대해서도 적절한 출력을 보이리라 기대한다.
- 강화학습에서는,
  - 잘 분류된(labeled) 데이터가 아닌 환경과의 상호작용을 통해 얻은 보상(reward)로부터 학습한다.

Reinforcement Learning?

- 비지도학습에서는,
  - 미분류(unlabeled) 데이터의 숨겨진 구조(hidden structure)를 찾는 것을 목표로 한다.
- 강화학습에서는,
  - 보상을 최대화하는 것을 목표로 한다.

Reinforcement Learning?

- 탐험(Exploration) 과 이용(Exploitation)

  - 더 높은 보상을 받기 위해서는 주어진 상황에서의 더 적절한 행동을 선택(exploit)해야 한다.

  - 이때 각 action들의 가치에 대해 알기 위해서는 사전에 탐험(explore)을 할 필요가 있다.

  - 탐험을 위해서는 지금 당장 최선이라고 믿어지는 action을 포기할 수도 있어야한다.

  ➡ Exploration-exploitation dilemma

Reinforcement Learning?

- 탐험(Exploration) 과 이용(Exploitation)

  - 더 높은 보상을 받기 위해서는 주어진 상황에서의 더 적절한 행동을 선택(exploit)해야 한다.

  - 이때 각 action들의 가치에 대해 알기 위해서는 사전에 탐험(explore)을 할 필요가 있다.

  - 탐험을 위해서는 지금 당장 최선이라고 믿어지는 action을 포기할 수도 있어야한다.

  ➡ Exploration-exploitation dilemma

  ➡ **강화학습을 지도학습, 비지도학습과 구분짓는 또 한가지 차이점**

Examples

- 새끼 가젤의 첫 걸음마
  - 새끼 가젤은 생후 30분만에시속 20마일의 속도로 뛸 수 있게 된다.

이미지 출처:
http://www.burrard-lucas.com/photo/kenya/baby_gazelle.html

Examples

- 로봇 청소기
  - 무선 청소로봇이 청소를 위해 방에 들어갈지 혹은 충전소로 돌아갈지 결정하는 것.

이미지 출처:
http://appsarang.co.kr/xe/?mid=game&document_srl=2419&listStyle=viewer

Elements of RL

- Policy
- Reward signal
- Value function
- Model

Policy는 현재의 상태(state)에 대해 어떤 행동(action)을 결정하는 역할을 한다.

간단하게는 lookup table이 될 수도 있고, 복잡하게는 상당한 계산비용을 필요로하는 탐색과정이 될 수도 있다.

보통 policy는 확률적(stochastic)인 경우가 많다.

Elements of RL

- Policy
- Reward signal
- Value function
- Model

Agent가 어떤 행동(action)을 할 때마다 환경은 agent에게 보내주는 숫자.

Reward signal에 대한 정의가 바로 목표에 대한 정의라고 봐도 무방하다.

보통 reward signal은 state와 action에 대한 확률함수(stochastic function)이다.

Elements of RL

- Policy
- Reward signal
- Value function
- Model

Value function은 reward signal보다 좀 더 장기적인 관점에서의 가치를 의미한다.

어떤 state에 대한 value라고 하면, 그 state를 시작으로 agent가 얻게 되리라 예상되는 reward에 대한 총합이라고 할 수 있다.

판단에 앞서 평가나 선택을 함에 있어서는 (가장 높은) value가 우선적으로 염두된다.

Elements of RL

- Policy
- Reward signal
- Value function
- Model

Model은 환경의 behavior를 모방하는 무언가이다.

어떤 state와 action이 주어졌을때 model은 (마치 환경처럼) 그 결과로써 reward와 다음의 state를 반환한다.

Multi-Armed Bandits

Introduction

- 다시한번, 강화학습의 특징?
  - 학습정보를 통해 올바른 행동을 지시하기 보다는 선택가능한 행동들에 대한 가치를 평가

- Multi-armed bandit problem?
  - 오직 하나의 상태에서 독립된 여러개의 행동 중 하나를 선택하는 특수한 강화학습 문제

Introduction

- 다시한번, 강화학습의 특징?

  - 학습정보를 통해 올바른 행동을 지시하기 보다는 선택가능한 행동들에 대한 가치를 평가

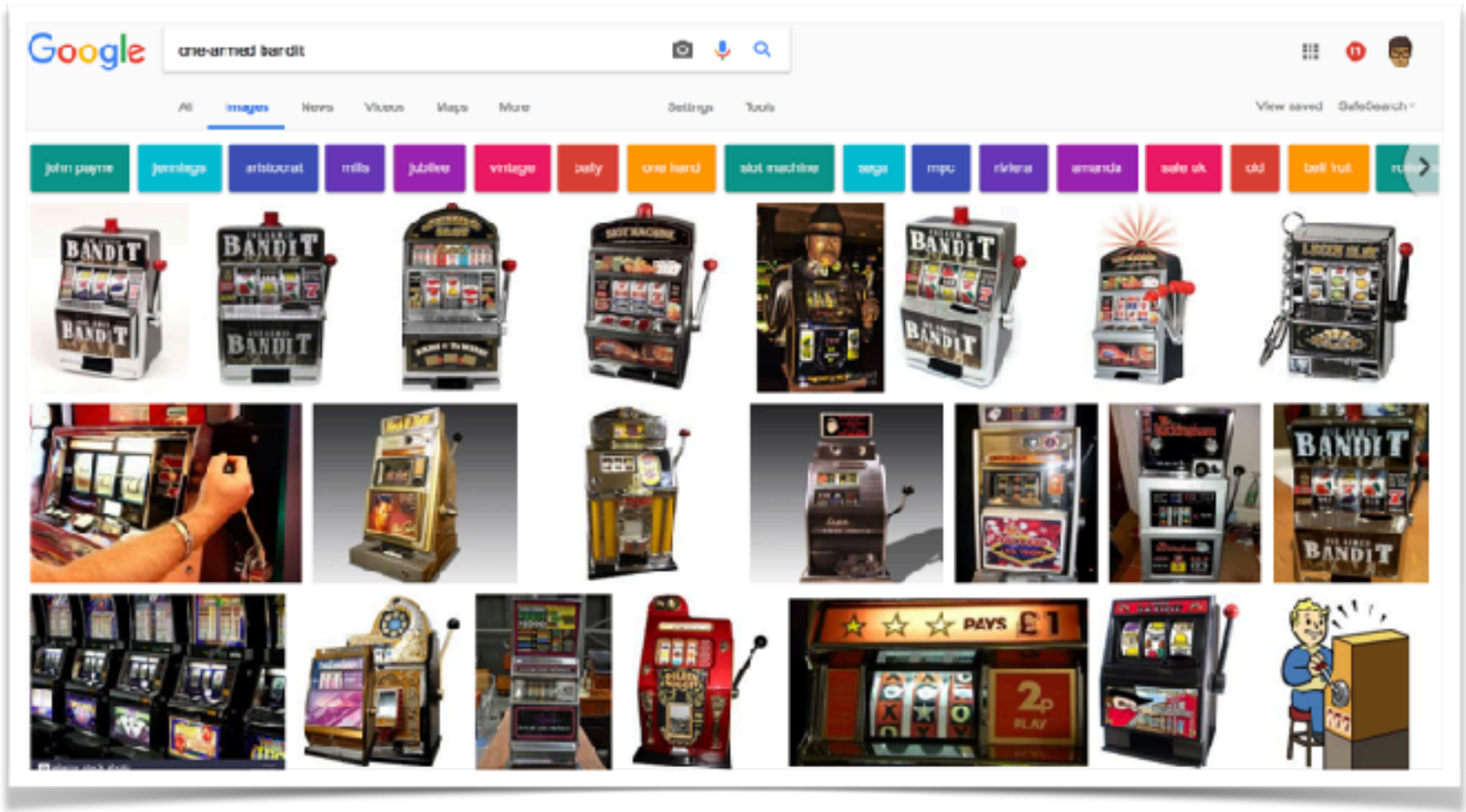
- Multi-armed bandit problem?

  - 오직 하나의 상태에서 독립된 여러개의 행동 중 하나를 선택하는 특수한 강화학습 문제

➡ **몇 가지 solution methods를 살펴봅시다.**

One-armed bandit?

- ‘외팔이 강도’ 라는 뜻.

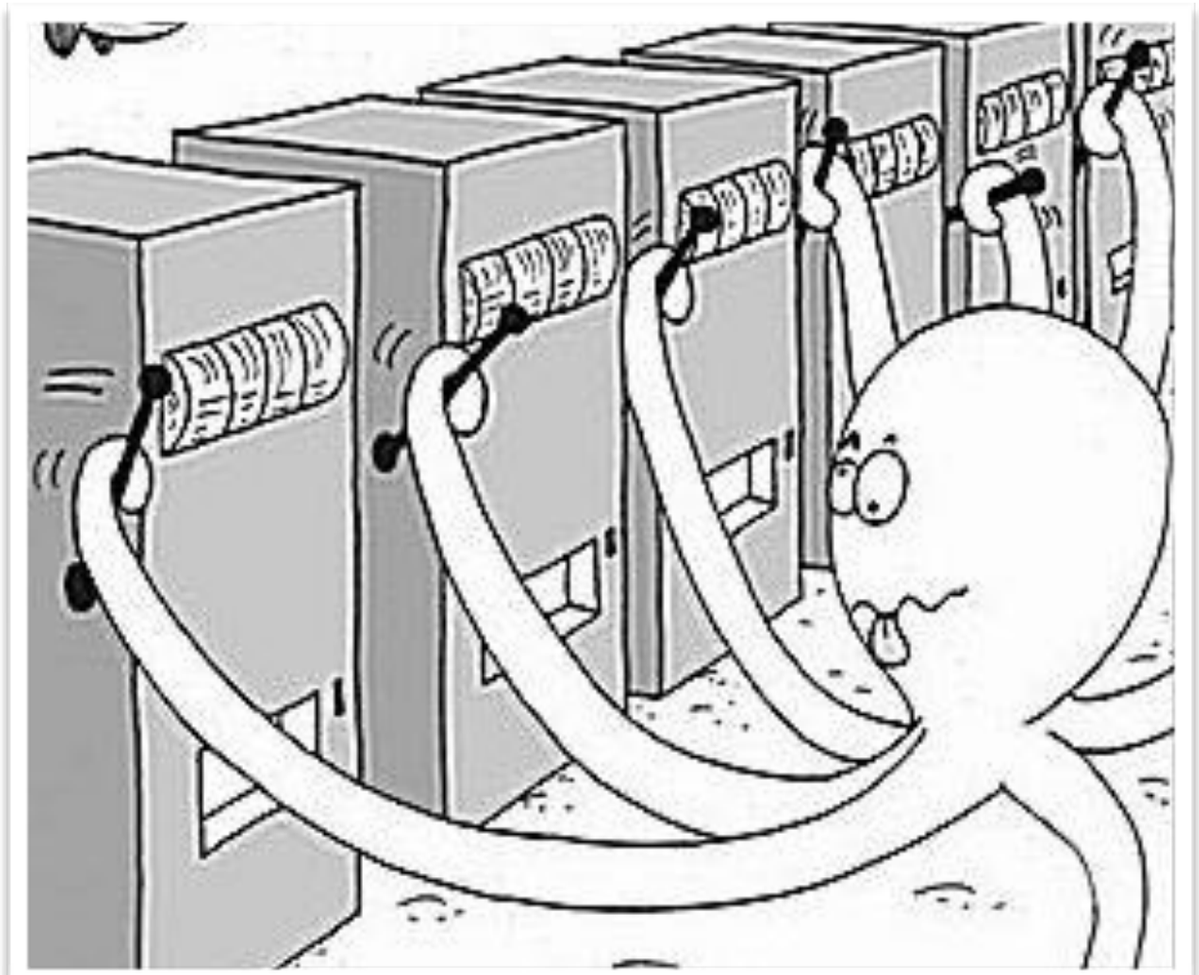


- 도박용 슬롯머신의 생김새를 본따 비꼬아 표현한 말

이미지 출처: 구글 이미지 검색

k-armed bandit?

- K개의 슬롯머신이 내 눈앞에 있는 상황



이미지 출처: Microsoft research

k-armed bandit problem

- 문제정의

  1. 매 시도마다 k개의 슬롯머신 중에서 하나를 선택한다. 이는 매번 k개의 액션을 선택할 수 있는 것과 같다.
  2. 선택된 슬롯머신은 고정된 확률 분포(stationary probability distribution)를 통해 수치적인(numerical) 보상을 반환한다.

- 목표

  - 위 절차를 어떤 횟수만큼 반복하여 보상의 총합에 대한 기댓값이 가장 높은 슬롯머신을 찾는 것

Value Function

- 정의

  - t 시점에서 어떤 행동을 선택했을때의 보상에 대한 기댓값

$$q_*(a) = E[R_t | A_t = a]$$

➡ 보상을 최대화 할 수 있으리라 기대되는 행동에 집중

➡ Action value가 가장 높은 행동을 선택하는 것이 목표

Value Estimation

- 안타깝게도 대부분의 경우, 우리는 (optimal) action value에 대한 정보를 사전에 가지고 있지 않다.

- 실험을 반복하며 얻은 정보를 토대로 어떤 추정(estimate)을 할 수 있다.

- 그 추정함수를 아래와 같이 표현하고, Q함수라 부른다.

$$Q_t(a)$$

➡ 추정이 action value에 최대한 가까워진다면 객관적으로 좋은 행동을 선택할 수 있으리라 기대할 수 있다.

Terminologies

- **Greedy actions**: 매 시도에서 Q함수의 값이 가장 높은 action 들의 집합
- **Exploiting**: Greedy actions 중 한가지 action을 선택하는 것
- **Exploring**: 탐험을 위해 non-greedy actions 중 한가지 action을 선택하는 것

Terminologies

- **Greedy actions**: 매 시도에서 Q함수의 값이 가장 높은 action 들의 집합

- **Exploiting**: Greedy actions 중 한가지 action을 선택하는 것

- **Exploring**: 탐험을 위해 non-greedy actions 중 한가지 action을 선택하는 것

➡ **Exploitation과 exploration은 trade-off 관계에 해당한다.**

➡ **Exploitation과 exploration 사이에서 적절한 균형을 잡는 것이 중요하다!**

Action-value Methods

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

- $1_{predicate}$: predicate이 true이면 1을 반환하고, false면 0을 반환.
- 분모가 0인 경우에는 $Q_t(a)$을 임의의 상수값으로 정의. (가령, 0)

- 위와 같은 방식을 **Sample-average method**라 부른다.

Action Selection

$$A_t = argmax_a \, Q_t(a)$$

$argmax_a \, Q_t(a)$ 함수는 $Q_t(a)$을 최대화시키는 action a를 반환.

- 위와 같은 방식을 **Greedy action selection**이라 부른다.
- Exploring을 위해 매 시도마다 ϵ만큼의 확률로 *non-greedy action selection*을 할 수 있다. (**ϵ-greedy method**)

Exercise

- 현재 $Q_t(a) = q_*(a)$를 만족하고 10개의 action에 대해 ϵ=0.1로 ϵ-greedy action selection을 하고 있다. 이때, optimal action이 선택될 확률은?

Exercise

- 현재 $Q_t(a) = q_*(a)$를 만족하고 10개의 action에 대해 ϵ=0.1로 ϵ-greedy action selection을 하고 있다. 이때, optimal action이 선택될 확률은?

1. Greedy action selection에 의해 1- ϵ의 확률로 optimal action이 선택된다: 1 - ϵ = 1 - 0.1 = 0.9

2. Non-greedy action selection에 의해 ϵ의 확률로 10개 중 임의의 action이 선택된다. 이중 optimal action이 선택될 확률은:
0.1 * 1/10 = 0.01

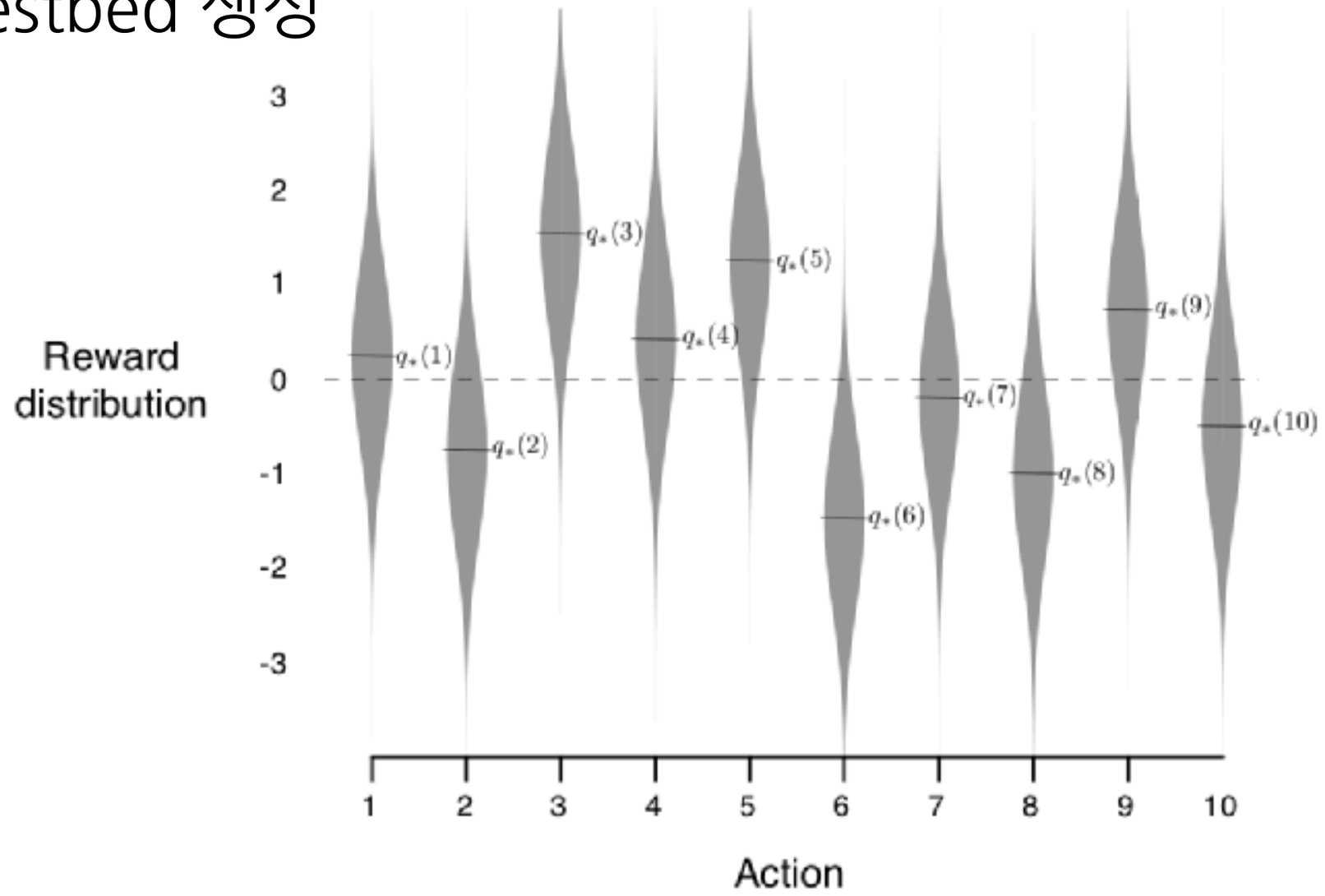
즉, optimal action이 선택될 확률은: 0.9 + 0.01 = 0.91

The 10-armed Testbed

- Greedy method와 ϵ-greedy method의 성능비교를 위한 간단한 실험
- 2,000개의 10-armed bandit 문제를 랜덤하게 생성하여 각 time step마다의 average reward와 optimal action selection의 비율을 그래프로 출력

The 10-armed Testbed

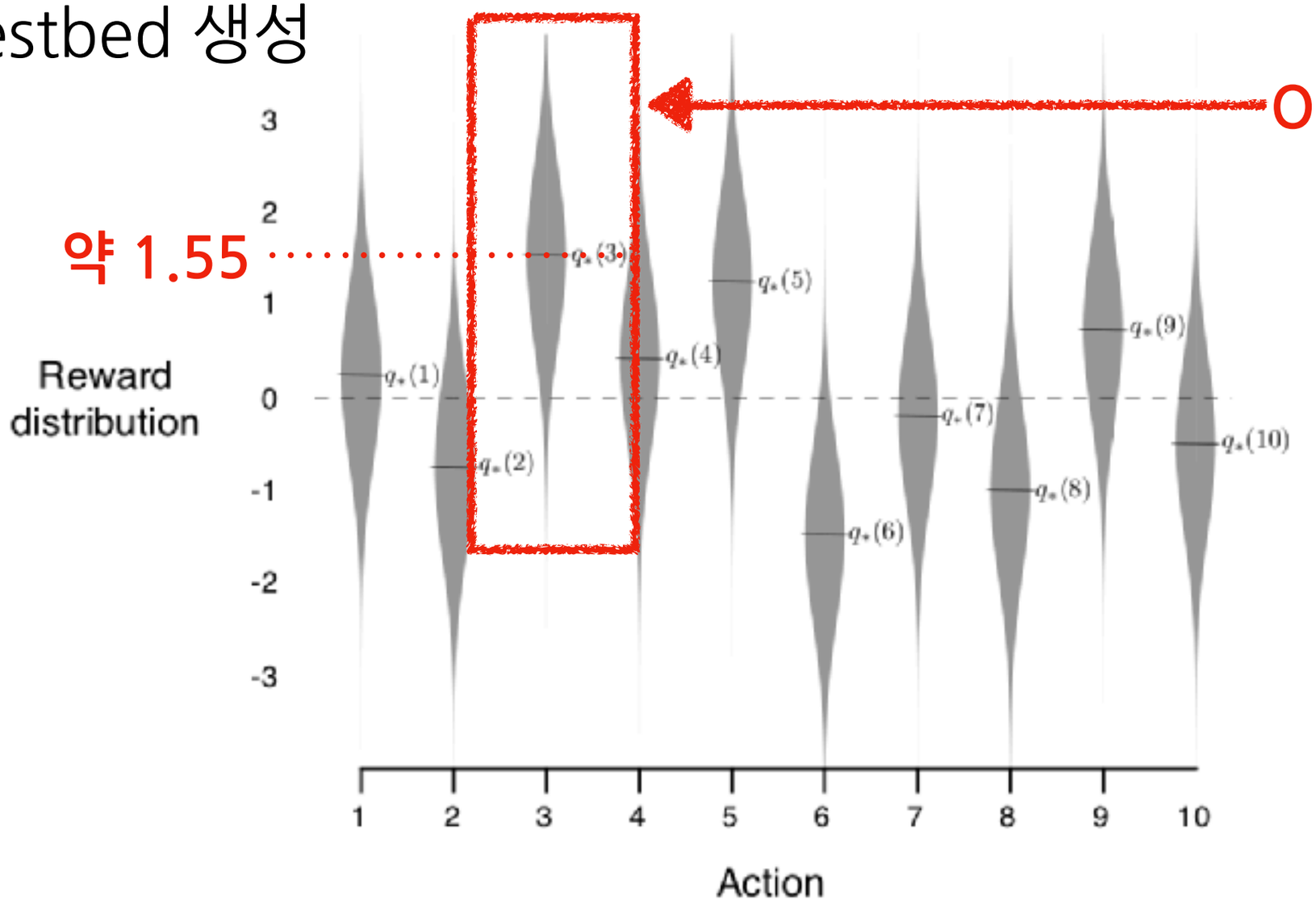
- mean=0, var=1의 가우시안 분포를 이용하여 10개의 testbed 생성



이미지 출처: [1]

The 10-armed Testbed

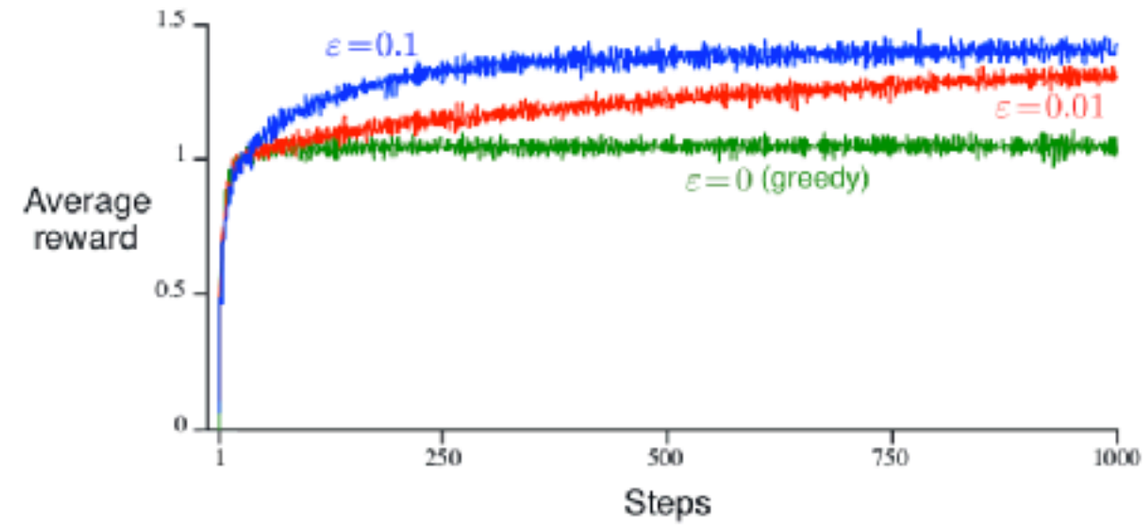
- mean=0, var=1의 가우시안 분포를 이용하여 10개의 testbed 생성

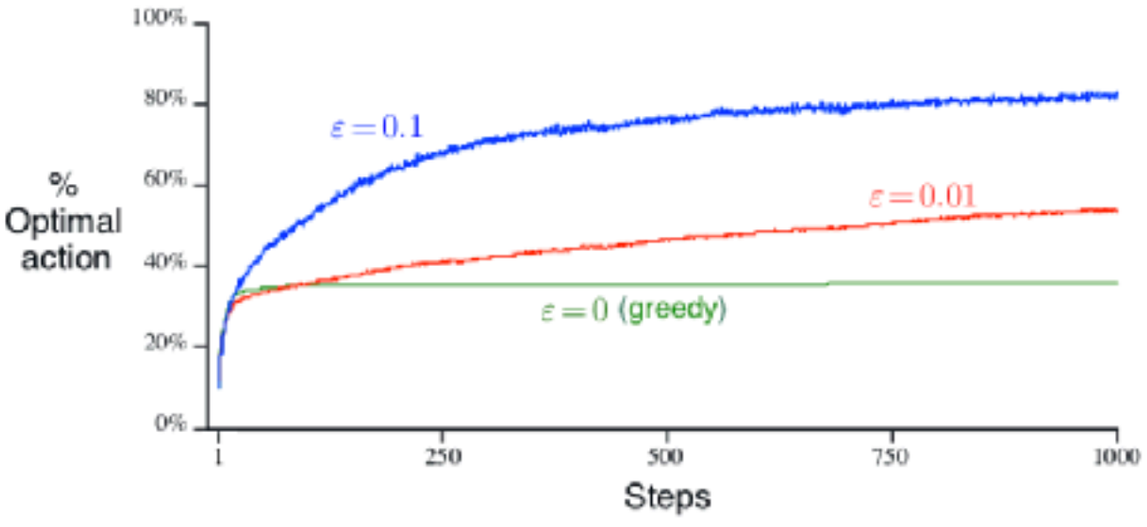


이미지 출처: [1]

The 10-armed Testbed

- 결과

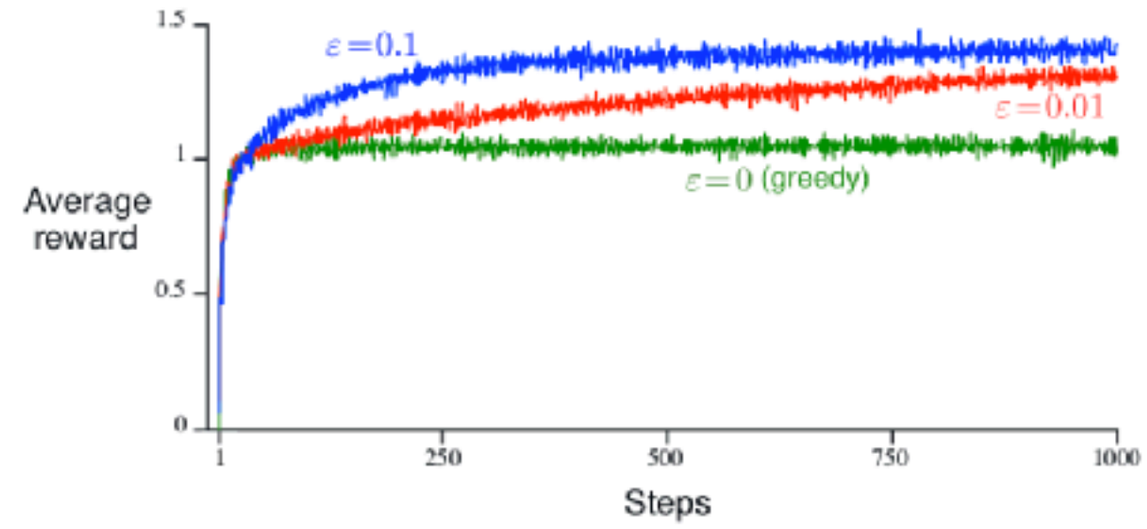


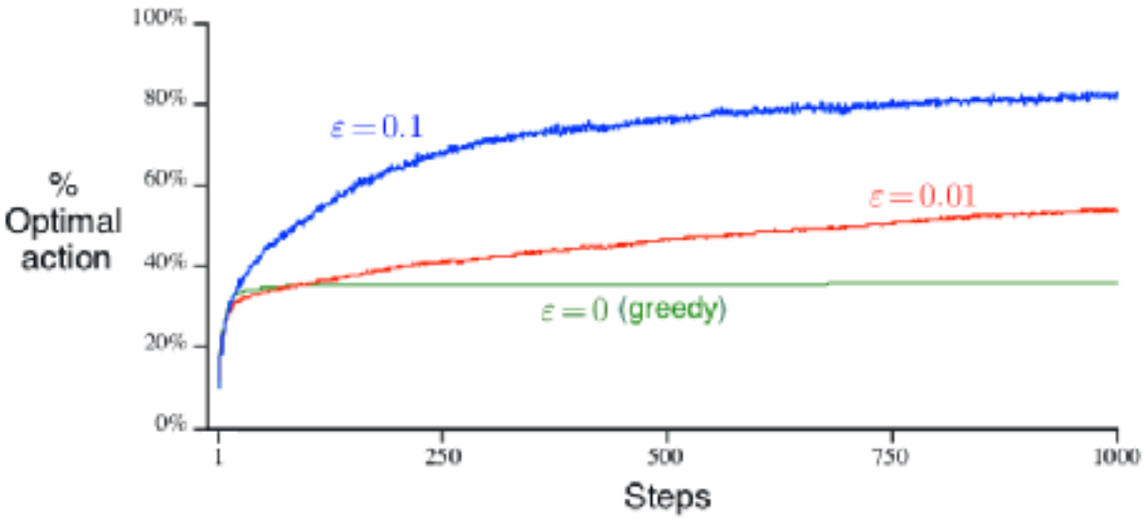


이미지 출처: [1]

The 10-armed Testbed

- 결과



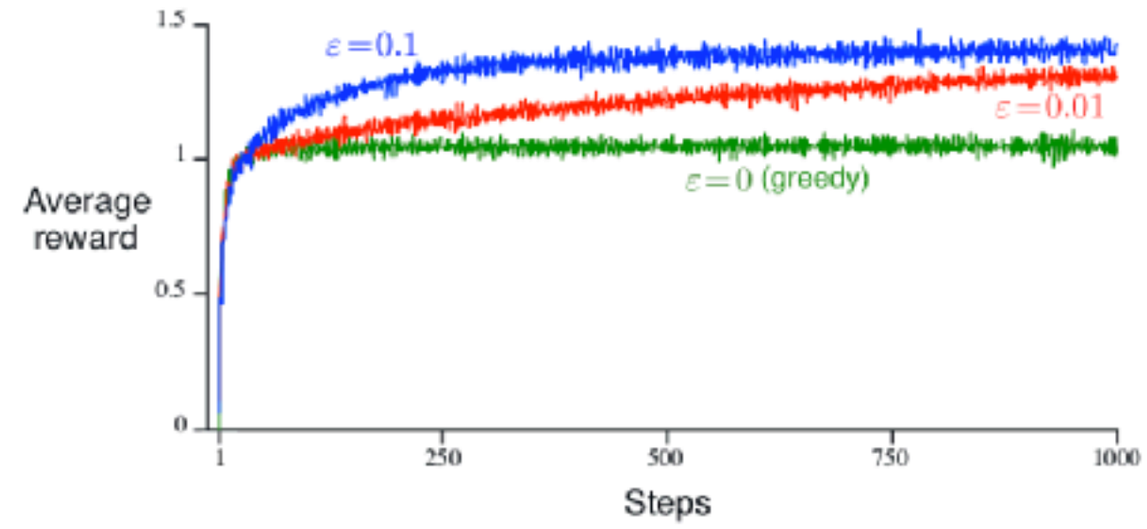


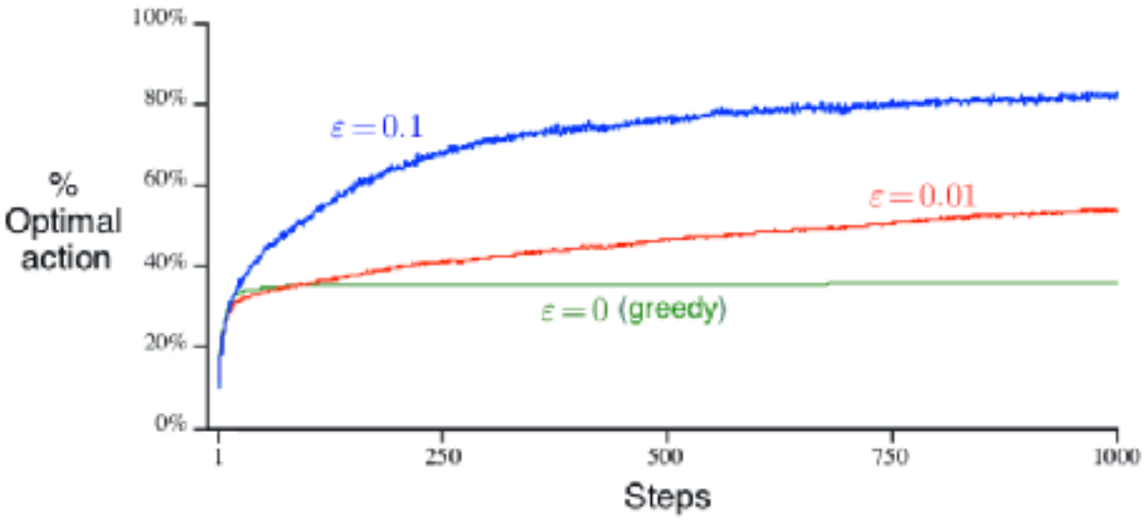
Exploring이 전혀 없음

이미지 출처: [1]

The 10-armed Testbed

- 결과



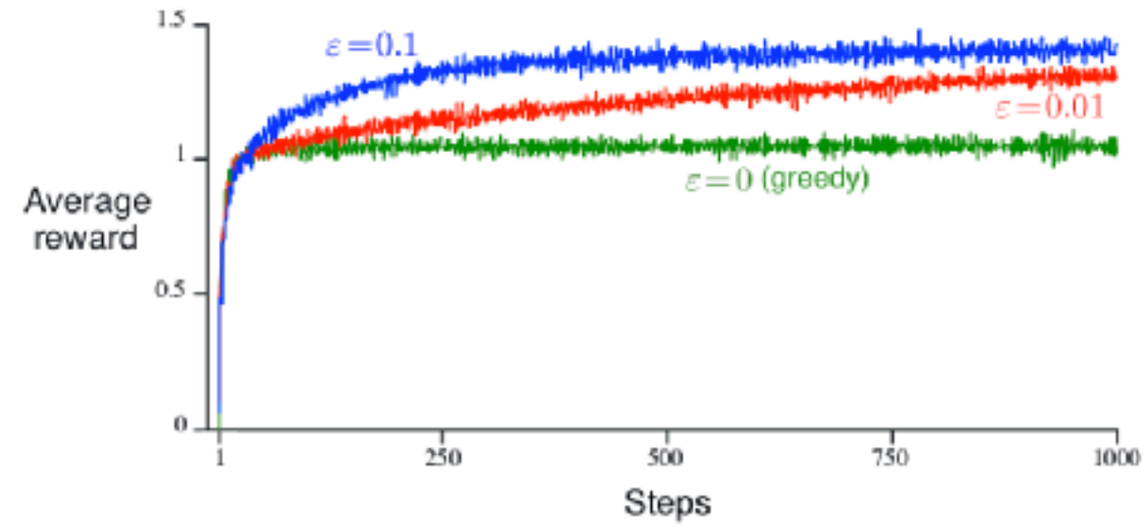


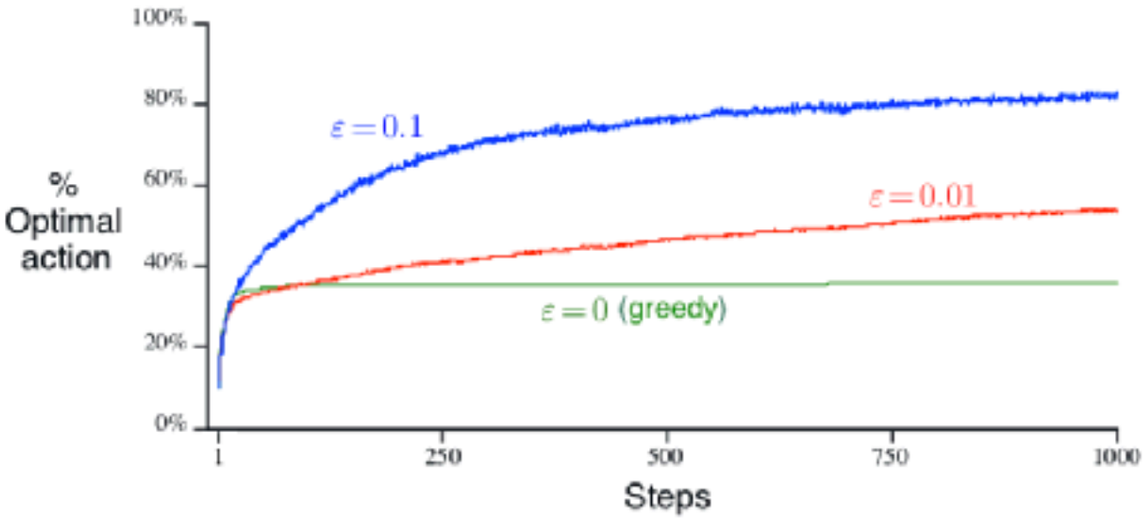
ϵ 이 0.01일 때보다 exploring이 많기 때문에 optimal action의 초반 탐색이 빠름

이미지 출처: [1]

The 10-armed Testbed

- 결과





ϵ 이 0.1일 때보다
optimal action의
초반 탐색이 느리지만,
장기적인 관점에서의
성능은 더 좋으리라 예상됨

이미지 출처: [1]

Incremental Implementation

- Sample average method의 더 효율적인 계산방법이 있을까?
- n번째 선택된 어떤 action에 대한 action value estimation:

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

Incremental Implementation

- Sample average method의 더 효율적인 계산방법이 있을까?
- n번째 선택된 어떤 action에 대한 action value estimation:

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

문제점!

1. 모든 time step마다 발생된 reward에 대해 기록해 두어야 한다.

2. 매번 각 action에 대해 발생된 모든 reward에 대한 합을 구해야 한다.

Incremental Implementation

- 간단한 수식유도로 해결가능

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

Incremental Implementation

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

- O(1)의 시간/공간 복잡도로 Q함수의 업데이트가 가능하다.
- 다음과 같은 방식으로도 해석할 수 있다.

$$NewEstimation \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

Incremental Implementation

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

- O(1)의 시간/공간 복잡도로 Q함수의 업데이트가 가능하다.
- 다음과 같은 방식으로도 해석할 수 있다.

$$NewEstimation \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

이후 등장하는 많은 update rule들은 이 아이디어를 따르게 된다!

Tracking a Nonstationary Problem

Sample-average method는 어떤 값으로 점차 수렴한다.

- 보상에 대한 확률분포가 고정되어있지 않다면?
- 우리가 실제로 다루게 될 문제들은 그 확률분포가 유동적(nonstationary)인 경우가 더 많다.

Tracking a Nonstationary Problem

Weighted-average method는 어떤 값으로 점차 수렴한다.

- 보상에 대한 확률분포가 고정되어있지 않다면?
- 우리가 실제로 다루게 될 문제들은 그 확률분포가 유동적(nonstationary)인 경우가 더 많다.

➡ **Step-size를 임의의 상수 $\alpha \in (0,1]$로 둔다면?**

Tracking a Nonstationary Problem

- Step-size의 수렴조건

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

- Stationary bandit problem에서 사용했던 $\alpha_n(a) = \frac{1}{n}$는 위 두 조건을 모두 만족한다.

  보통 $\sum_{n=1}^{\infty} \frac{1}{n}$는 harmonic series로, $\sum_{n=1}^{\infty} \frac{1}{n^2}$는 Basel problem으로 불린다.

Tracking a Nonstationary Problem

- Step-size의 수렴조건

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

- Stationary bandit problem에서 사용했던 $\alpha_n(a) = \frac{1}{n}$ 는 위 두 조건을 모두 만족한다.

  보통 $\sum_{n=1}^{\infty} \frac{1}{n}$ 는 harmonic series로, $\sum_{n=1}^{\infty} \frac{1}{n^2}$ 는 Basel problem 으로 불린다.

  임의의 상수 $\alpha \in (0,1]$는 위 수렴조건을 만족하지 않는다.

Tracking a Nonstationary Problem

- Step-size $\alpha \in (0,1]$가 암시하는 것

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

➡ 이때, $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$이므로 이는 weighted average와 같다. (No vanishing, No exploding)

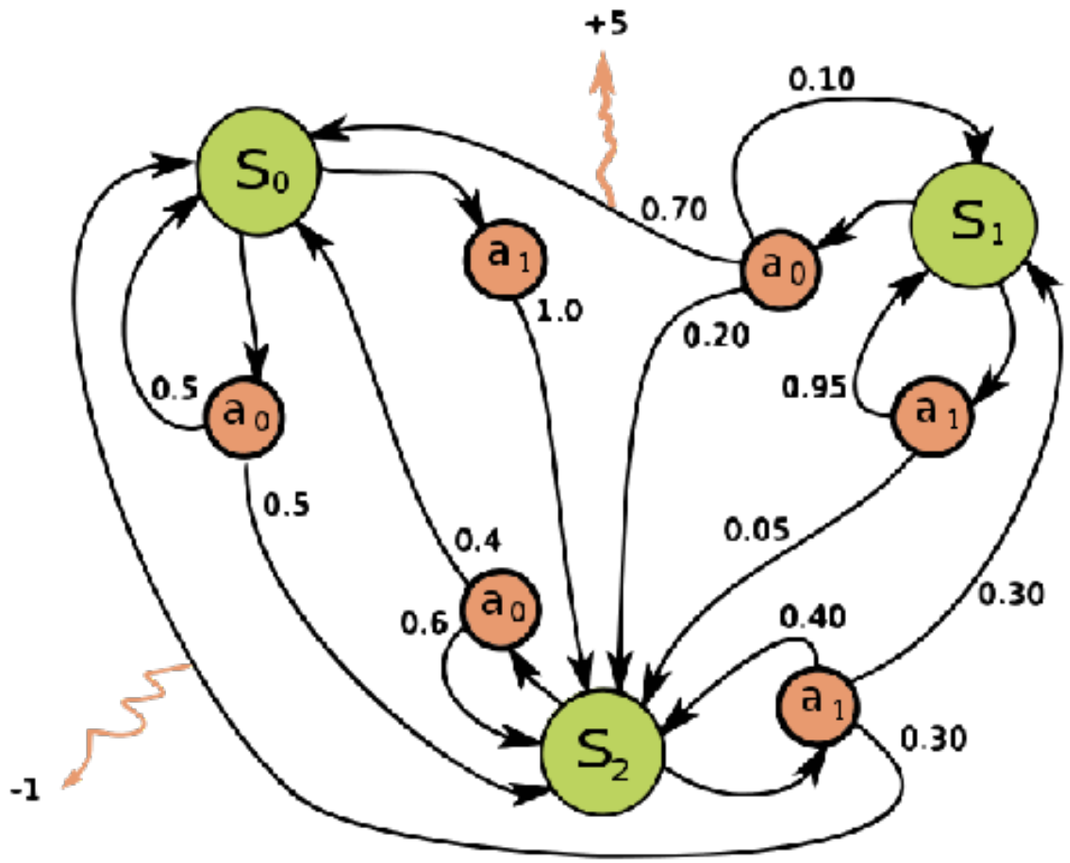
➡ $\sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$ 는 오래된 보상에 낮은 가중치가 곱해짐을 시사한다.

만약에 지금 선택한 행동이 보상과 더불어
그 다음의 상황을 결정하는 요인이 된다면?

Finite Markov Decision Processes

Introduction

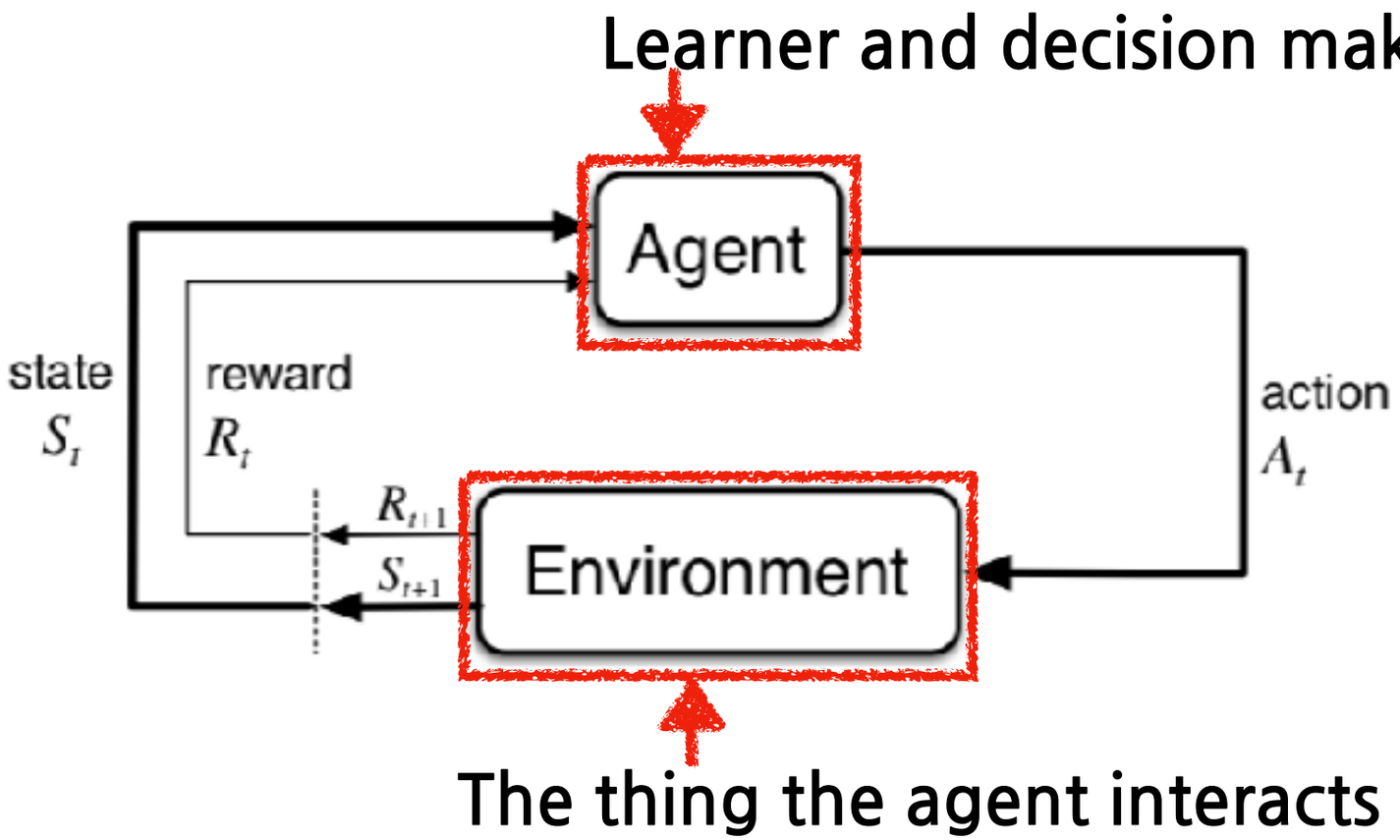
- MDP는 목표를 이루기 위해 상호작용으로부터 학습하는 문제를 표현한다.



- MDP는 sensation, action, goal의 세 가지 개념을 포함
- 즉각적인 보상과 지연(delayed) 보상간의 trade-off를 내포

이미지 출처:
https://en.wikipedia.org/wiki/Markov_decision_process

The Agent-Environment Interface



- Finite MDP에서 state, action, reward의 집합은 각각 유한한 갯수의 원소로 이루어져 있다.

- MDP와 agent는 다음과 같은 순차적인 사건을 발생시킨다.

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

이미지 출처: [1]

Return

- Time step t 이후의 순차적인 reward을 $R_{t+1}, R_{t+2}, R_{t+3}, \ldots,$ 라고 할때, 가장 간단한 형태의 return은 다음과 같다.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$ where T is a final time step.

- 일반적으로 expected return을 최대화하는 방향으로 학습이 이루어진다.

Episodes

- 에이전트와 환경간의 상호작용(interaction)이 자연스럽게 여러개의 subsequence로 나뉘어 질 수 있을때, 하나의 subsequence를 **episode**라고 부른다.
- Episode의 단위로 표현할 수 있는 문제를 **Episodic tasks**라 명명한다. Episodic tasks는 다음과 같은 특징이 있다.

  - 모든 states의 집합(denoted $\mathcal{S}^+$)에서 non-terminal states의 집합(denoted $\mathcal{S}$)을 구분할 수 있다.

- Episode의 단위로 표현할 수 없는 문제를 **continuing tasks**라고 명명한다.

Return

- **Recall**: 가장 간단한 형태의 return은 다음과 같다.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \text{ where T is a final time step.}$$

- Continuing tasks의 경우 $T = \infty$ 이므로 위의 return은 ∞로 발산할 수 있다.

- Discounting이라는 개념을 도입하여 continuing tasks를 위한 return을 정의할 수 있다.

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

where γ is a parameter, $0 \leq \gamma \leq 1$, called the **discount rate**.

Unified Notation for Return

- 다음과 같이 return을 일반화시킴으로써 Episodic tasks와 continuing tasks를 모두 아우를 수 있다.

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k,$$

including the possibility that $T = \infty$ or $\gamma = 1$ (but not both).

- $T \neq \infty, \gamma = 1$ 일때, 이는 다음과 같다.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T .$$

- $T = \infty, \gamma \in [0,1]$ 일때, 이는 다음과 같다.

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

Policy and Value Functions

- Policy: 어떤 state에서 각 action들을 선택할 확률을 반환한다. 예를 들어, $S_t = s$일때 $A_t = a$를 선택할 확률은 $\pi(a|s)$로 표현한다.

- State-value function for policy π :

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

- Action-value function for policy π :

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

A Fundamental Property of Value Functions

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

A Fundamental Property of Value Functions

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

Bellman Equation for v_π

A Fundamental Property of Action-Value Functions

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

A Fundamental Property of Action-Value Functions

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

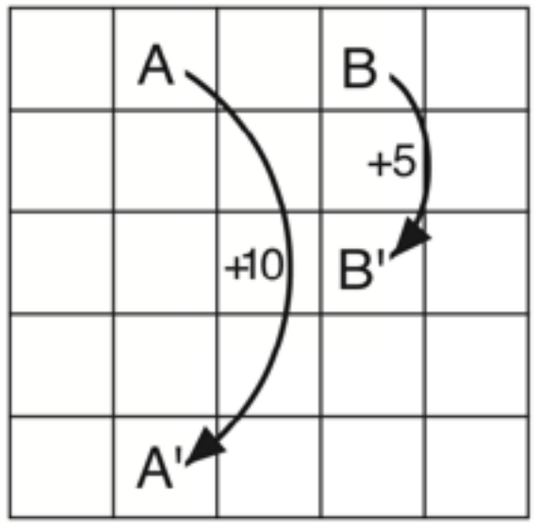
$$= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]$$

Bellman Equation for q_π

Exercise

- Gridworld

밖으로 떨어지면
reward: -1



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

center

$\pi(a|s) = 0.25$ for all $a \in A$ and $s \in S$이고 $\gamma = 0.9$일때,
$v_\pi(center) = 0.7$ 임을 보여라. (소수점 2째 자리에서 반올림)

이미지 출처: [1]

Exercise

- Gridworld

$$v_\pi(center) = 0.25 \cdot (0 + 0.9 \cdot 2.3)$$
$$+0.25 \cdot (0 + 0.9 \cdot 0.4)$$
$$+0.25 \cdot (0 + 0.9 \cdot -0.4)$$
$$+0.25 \cdot (0 + 0.9 \cdot 0.7)$$
$$= 0.5175 + 0.09 - 0.09 + 0.1575$$
$$= 0.675$$
$$\approx 0.7$$

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

이미지 출처: [1]

Optimal Policies and Optimal Value Functions

- **Optimal Policy:** 어떤 policy가 모든 state에서 다른 모든 policy들보다 항상 더 크거나 같은 value를 반환할때 이를 optimal policy라고 부르고 π_*로 쓴다.
- **Optimal Value Functions:** Optimal policy를 따르는 value function을 optimal value function이라고 한다.

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

$$= max_\pi \, v_\pi(s)$$

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

$$= max_\pi \, q_\pi(s,a)$$

Bellman Optimality Equation

- 주어진 어떤 state에 대해 optimal policy에 의한 value는 best action에 대한 기댓값과 동일함을 의미한다.

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

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

Bellman Optimality Equation

- 주어진 어떤 state에 대해 optimal policy에 의한 value는 best action에 대한 기댓값과 동일함을 의미한다.

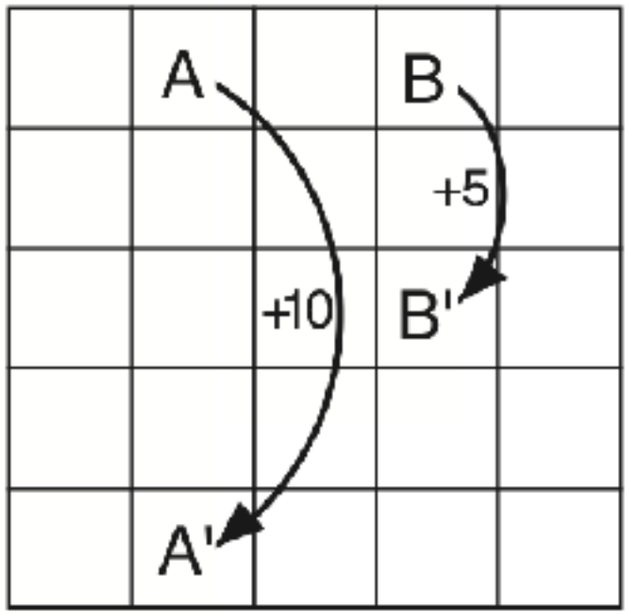
$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a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

π_*는 value를 최대화하는 policy이기 때문

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

Exercise

- **Gridworld:** optimal policy를 추정해보자.



Gridworld

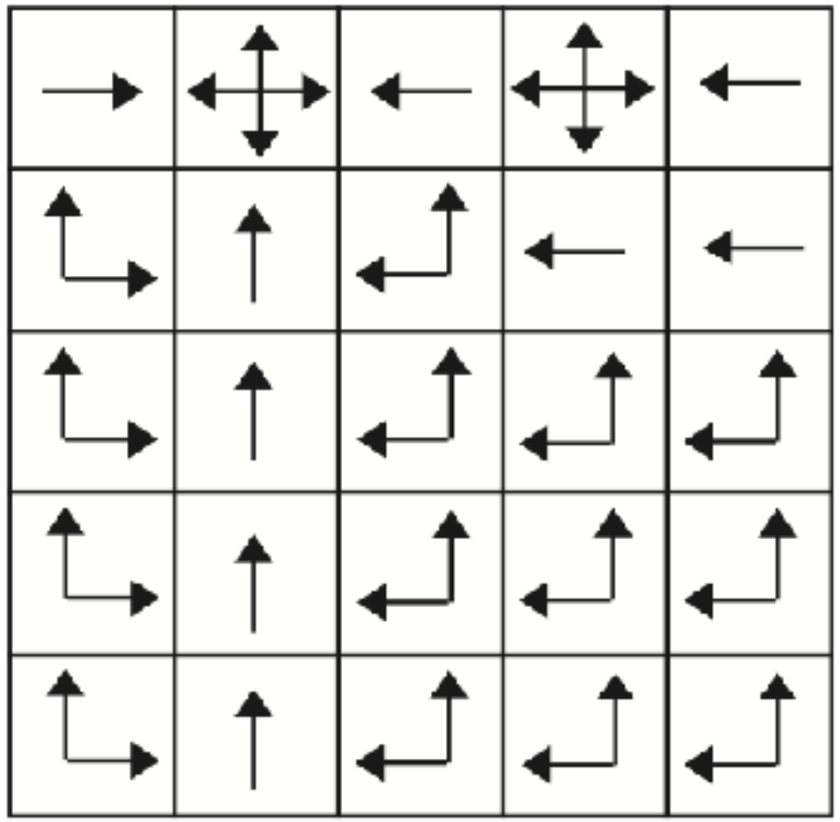
22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*

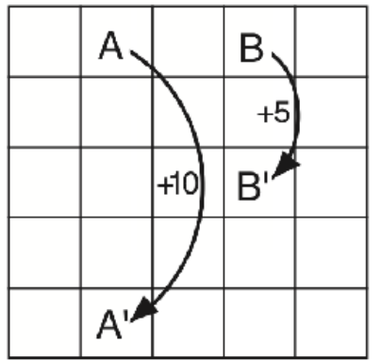
이미지 출처: [1]

Exercise

- **Gridworld:** 추정된 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_*

이미지 출처: [1]

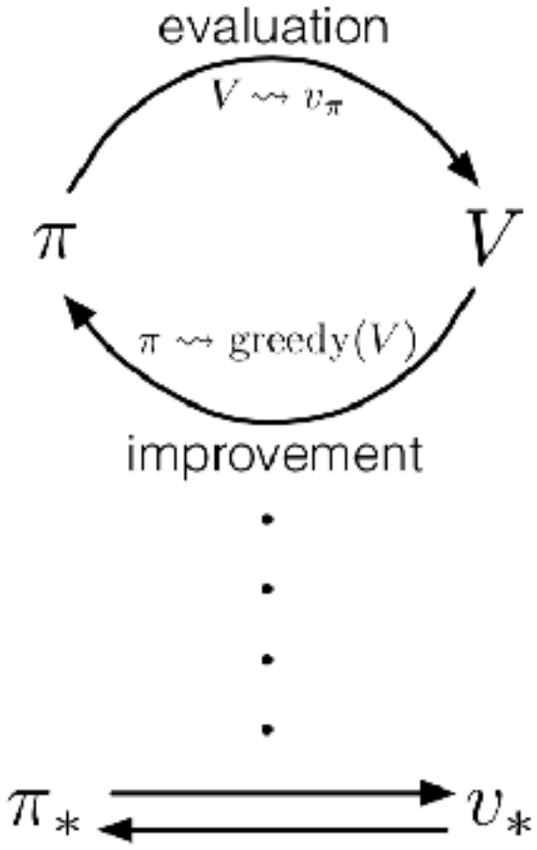
어떤 policy를 따르는 value function을 계산하는 방법은?

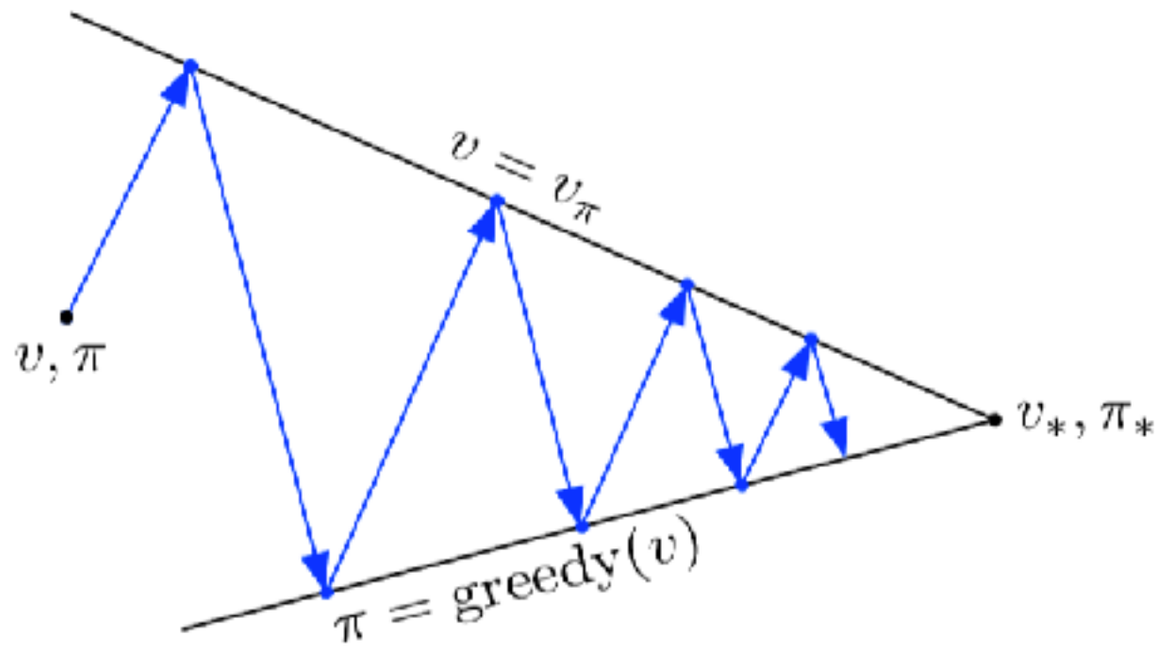
더 나은 policy를 구하는 방법은?

Dynamic Programming

Key Idea

- 좋은 policy의 탐색을 위해 value function의 사용을 체계화, 구조화 하는것.





- Bootstrapping: 잇따른 state들의 value에 대한 추정을 기반으로 현재 state의 value를 추정

이미지 출처: [1]

Policy Evaluation

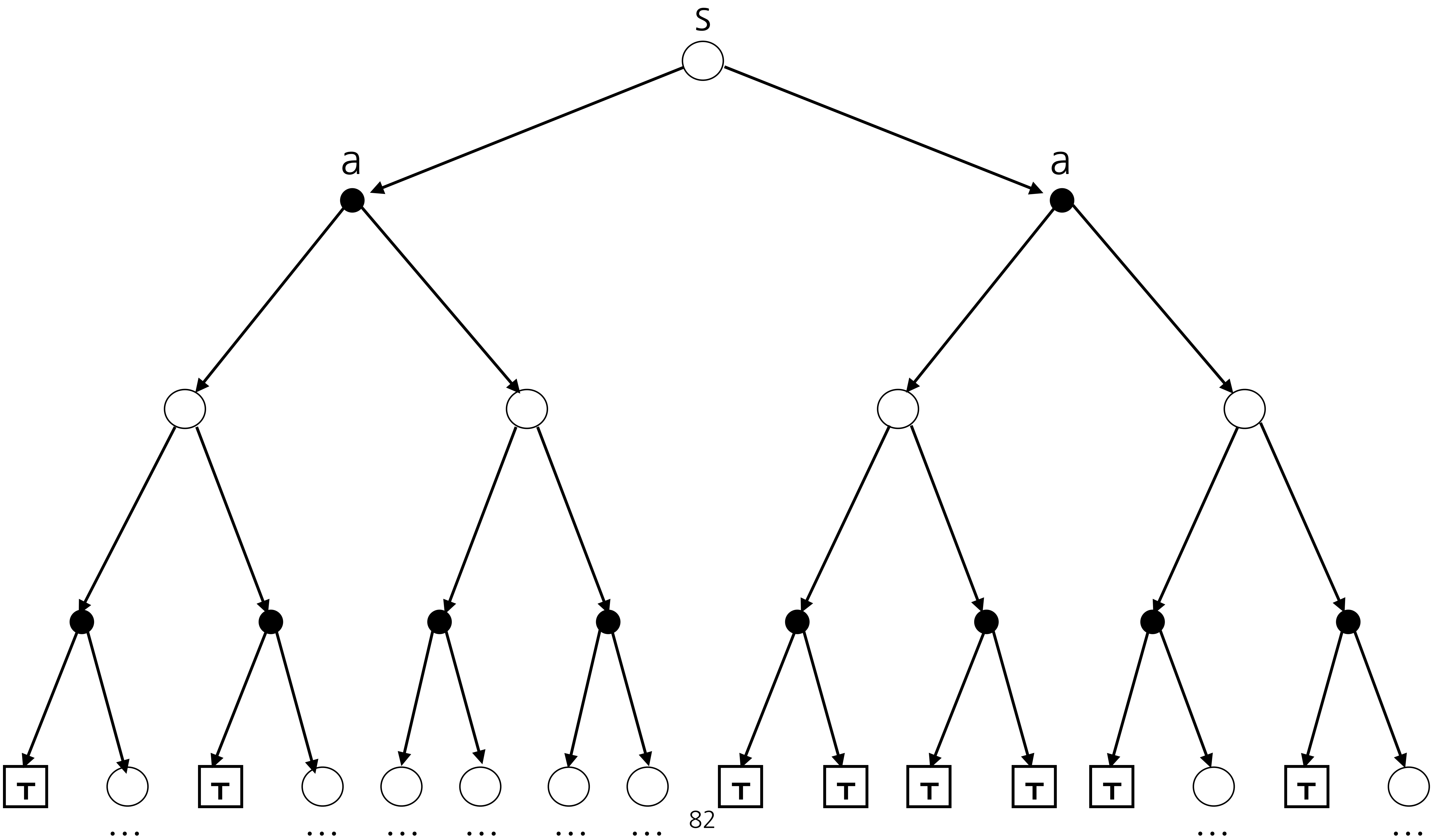
- 임의의 policy π에 대한 state-value function v_π를 계산하는 방법. 일종의 prediction problem으로 볼 수 있다.

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

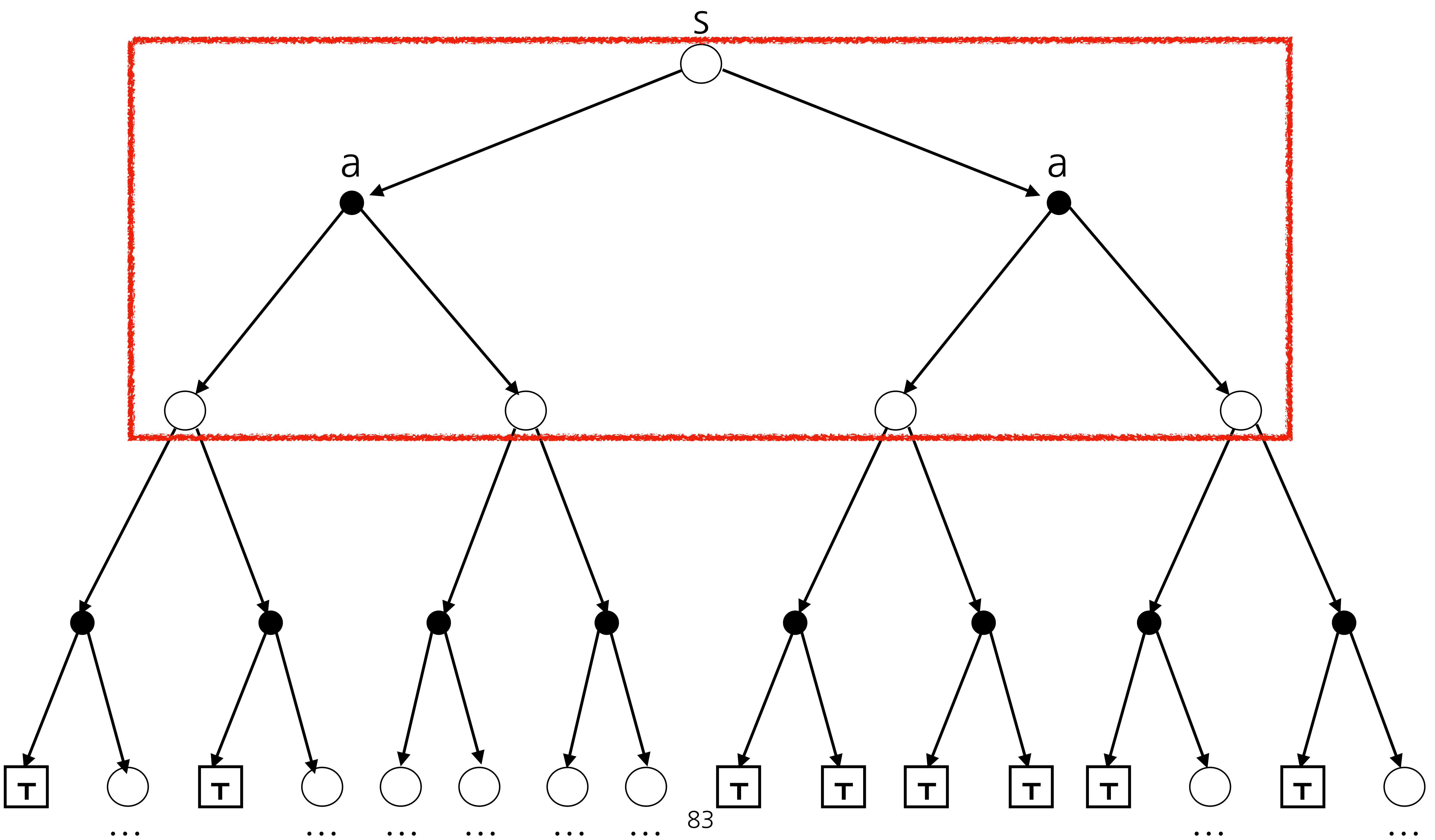
where all $s \in S$.

- $k \to \infty$일때, v_k는 v_π로 수렴하게 된다. 이러한 알고리즘을 iterative policy evaluation이라 한다.

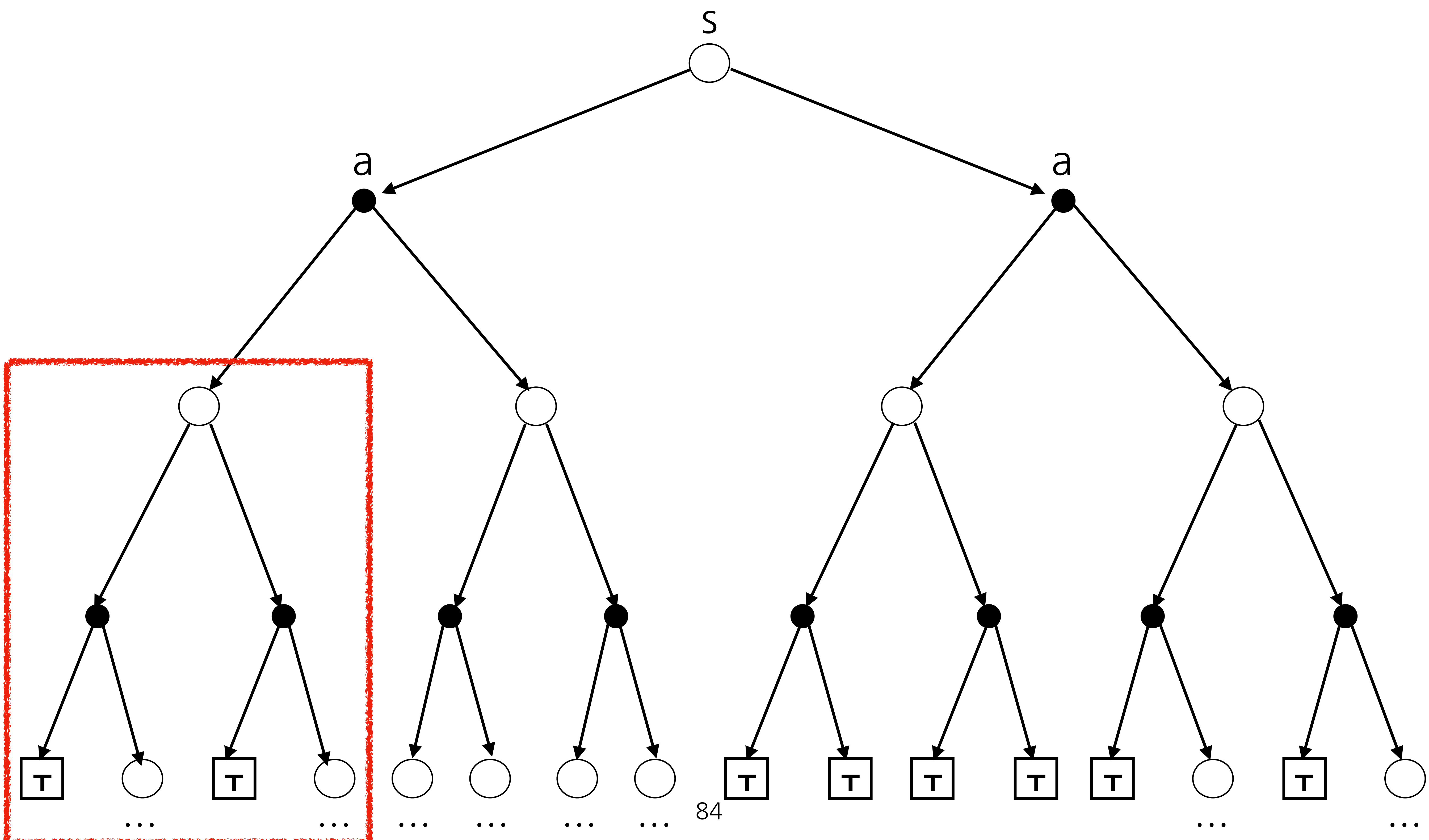
Policy Evaluation



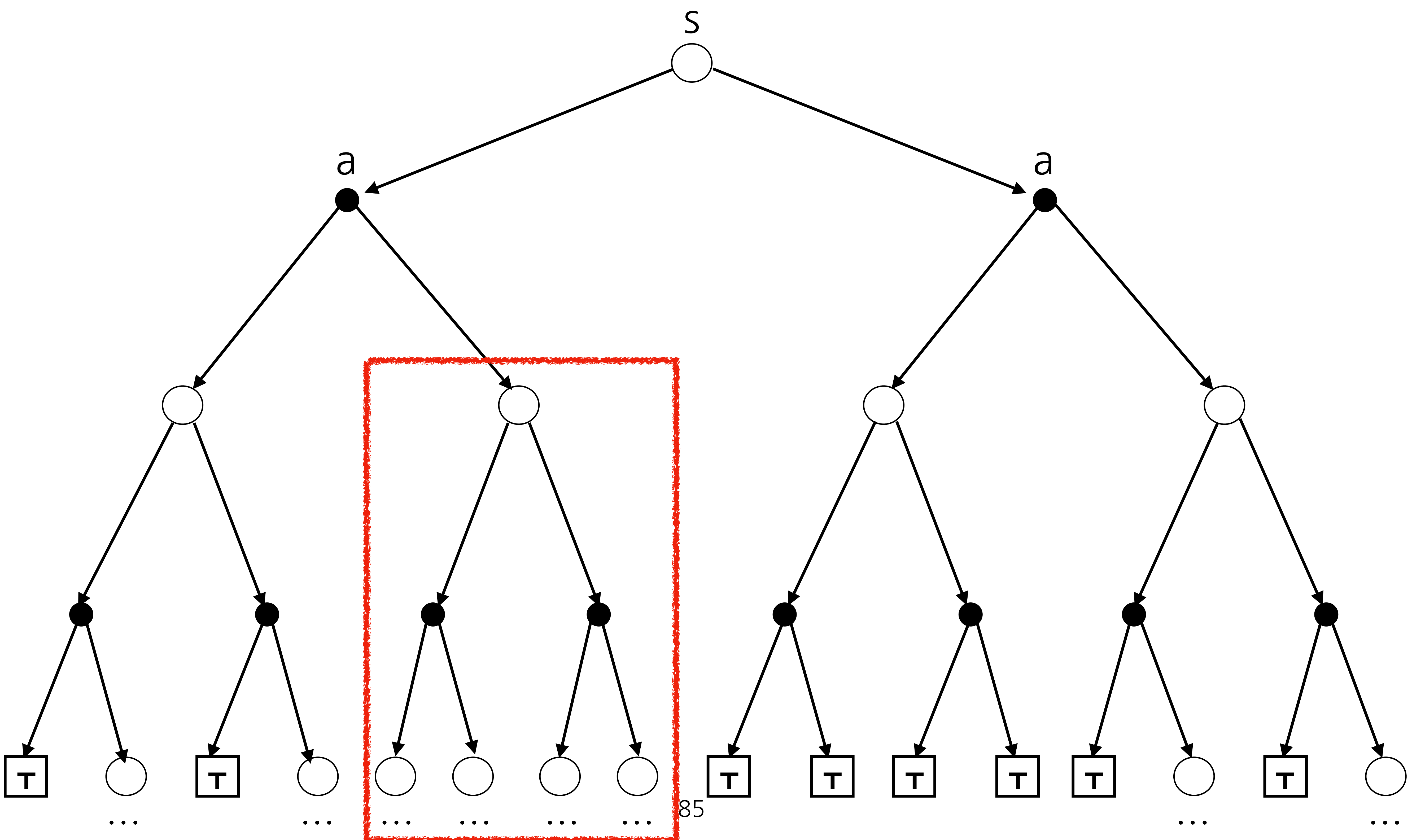
Policy Evaluation



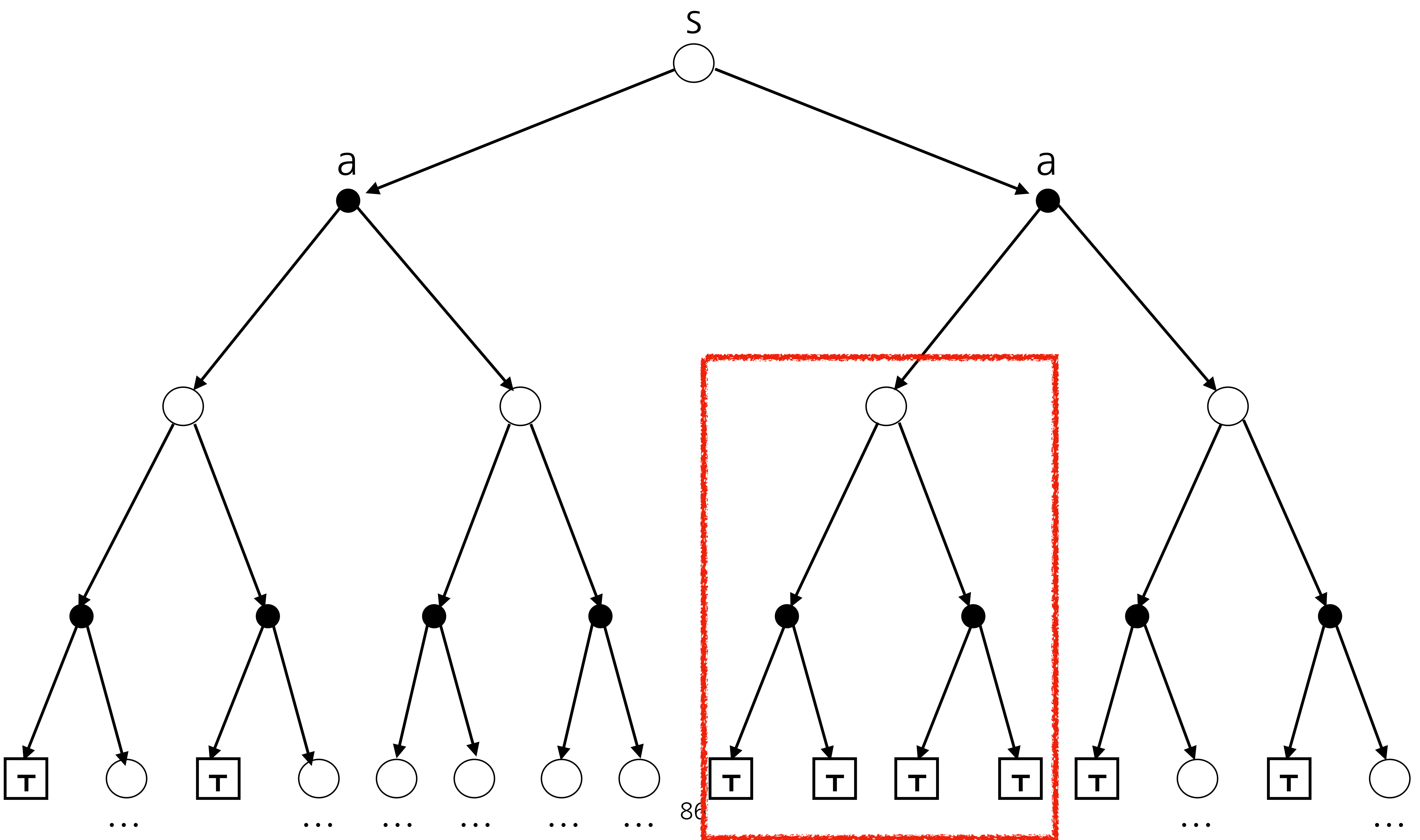
Policy Evaluation



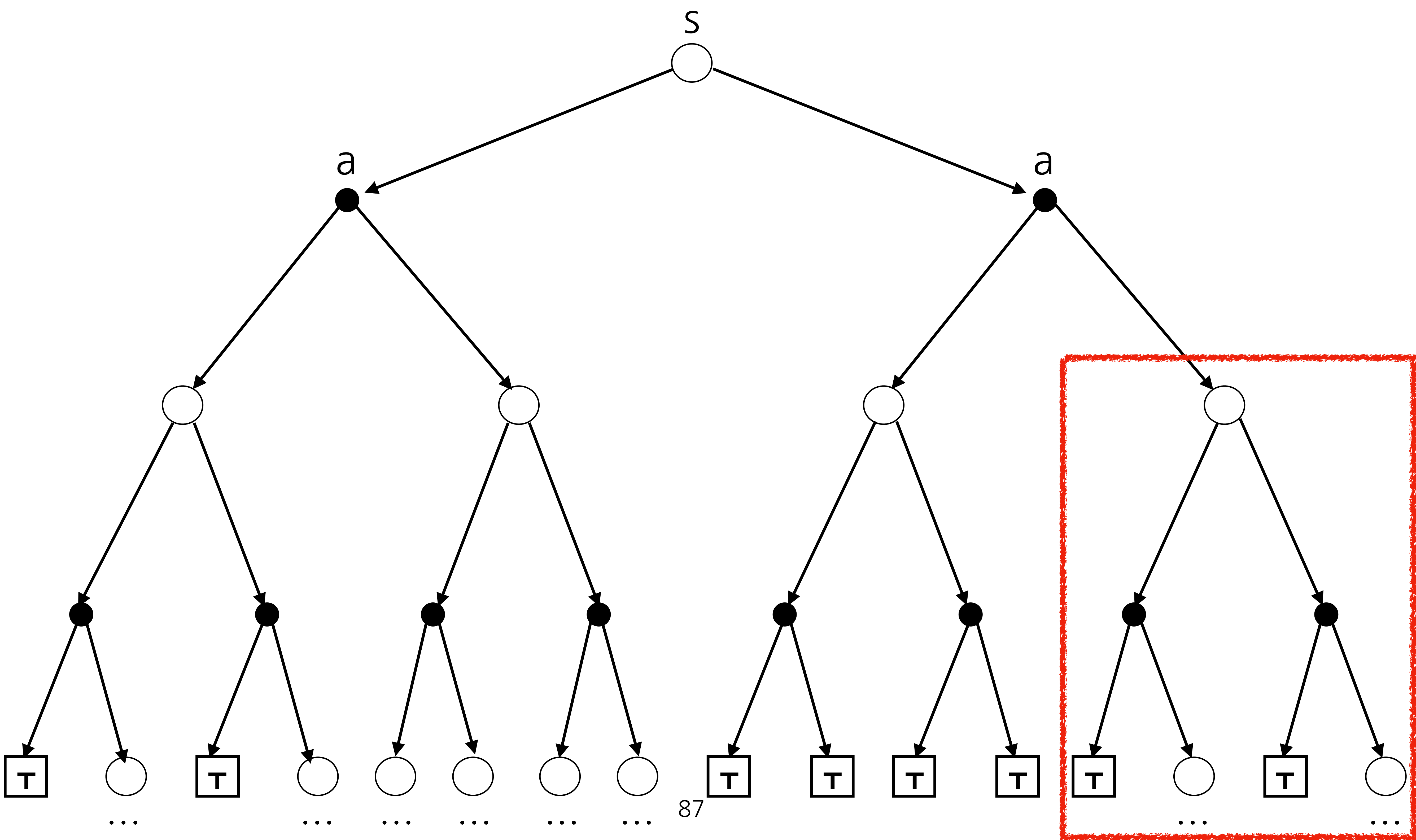
Policy Evaluation



Policy Evaluation



Policy Evaluation



Policy Evaluation

- 다음은 Iterative policy evaluation에 대한 pseudocode다.

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 $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$: ←——모든 state를 한 번 일주하는 것을 **sweep**이라 한다.
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

- 어떤 sweep에 대한 $max_{s \in S} | v_{k+1}(s) - v_k(s) |$가 θ보다 작음이 확인될때 알고리즘은 종료된다.

이미지 출처: [1]

Policy Improvement

- 주어진 value function을 통해 더 좋은 policy를 찾는 것을 policy improvement라고 한다.

- π에 대한 greedy policy π'는 다음과 같이 정의된다.

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

where $argmax_a$ denotes the value of a at which the expression that follows is maximized.

Policy Improvement

- $v_\pi(s) \le v_{\pi'}(s)$ 임은 다음의 과정을 통해 증명된다.

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\le \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\right] \\
&\vdots \\
&\le \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Policy Iteration

- Policy evaluation과 policy improvement를 반복하며 optimal policy와 optimal value에 차츰 접근하는 방법을 **Policy Iteration**이라고 한다.

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

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

Policy Iteration

- 다음은 policy iteration에 대한 pseudocode다.

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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

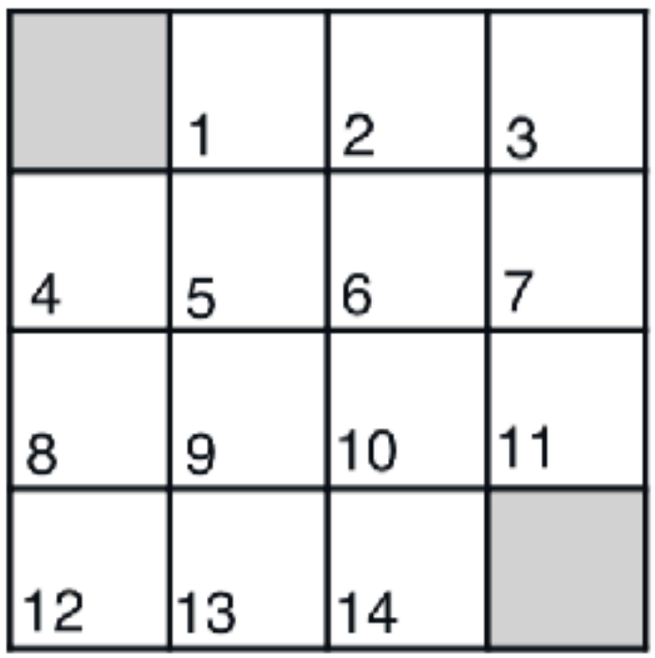
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$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

이미지 출처: [1]

Example

- Gridworld

actions

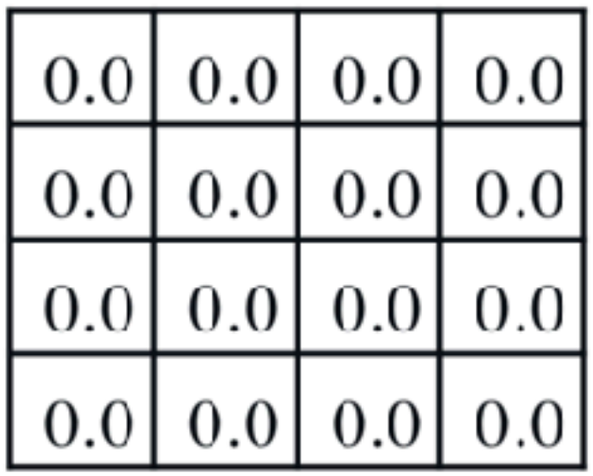
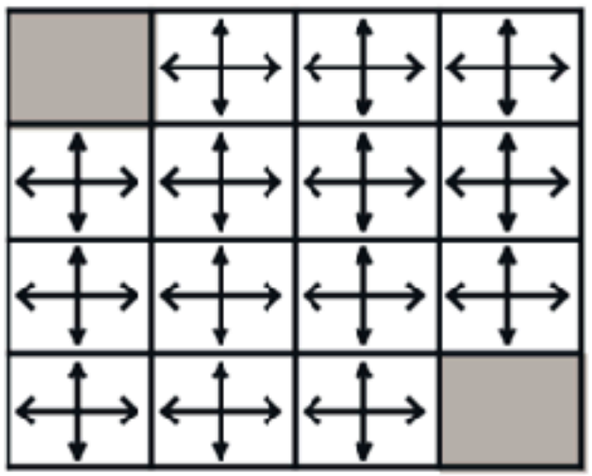


$$R_t = -1$$
on all transitions

이미지 출처: [1]

Example

- Gridworld

	v_k for the random policy	greedy policy w.r.t. v_k
$k = 0$		 random policy

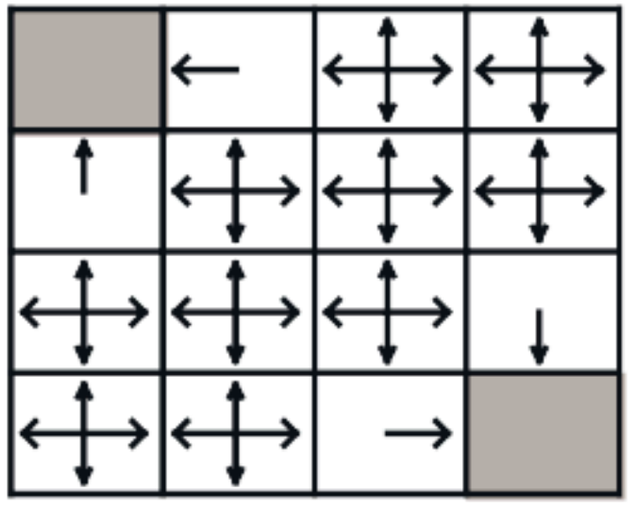
이미지 출처: [1]

Example

- Gridworld

$k = 1$

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



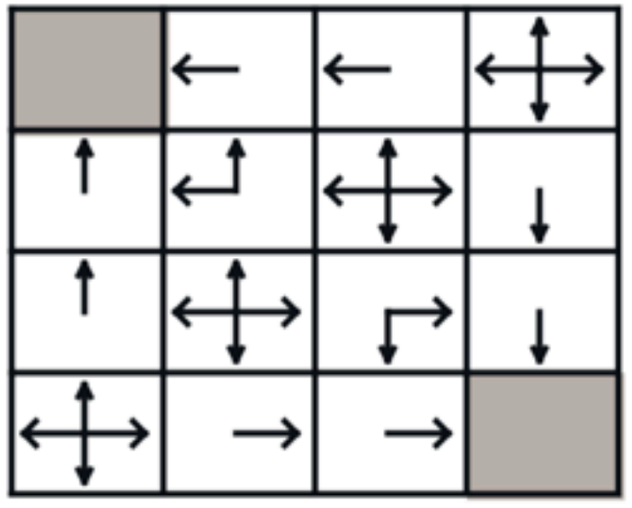
이미지 출처: [1]

Example

- Gridworld

$k = 2$

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



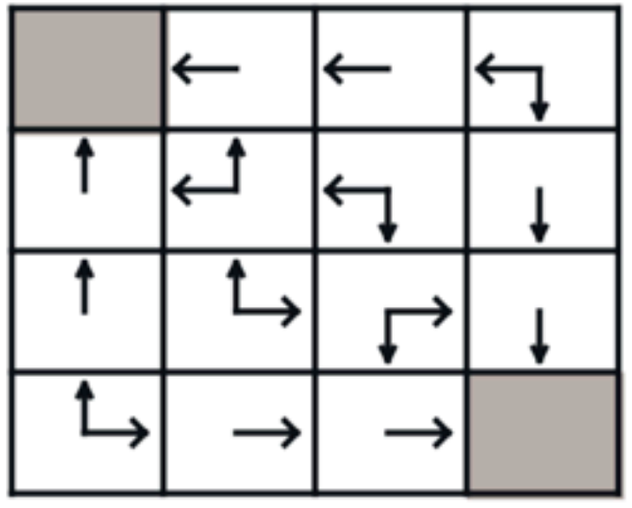
이미지 출처: [1]

Example

- Gridworld

$k = 3$

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



Optimal Policy!

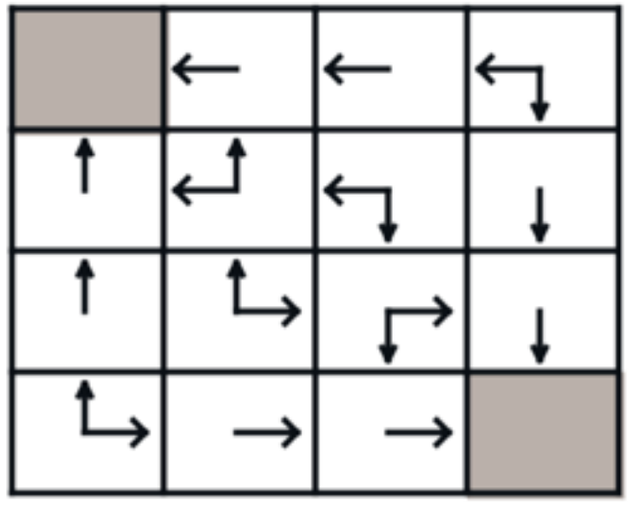
이미지 출처: [1]

Example

- Gridworld

$k = 10$

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



Optimal Policy!

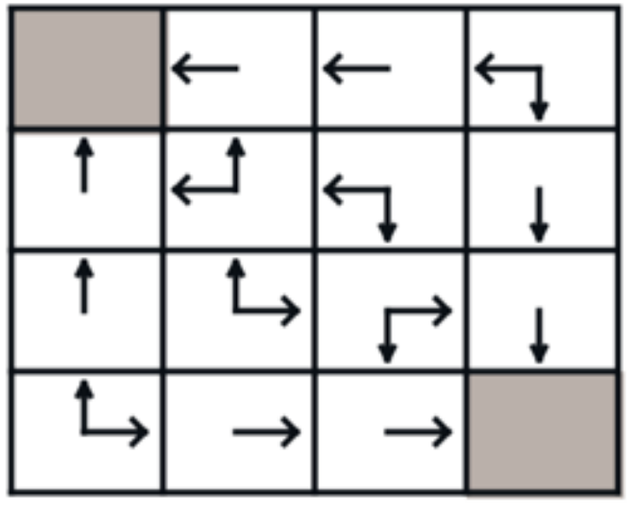
이미지 출처: [1]

Example

- Gridworld

$k = \infty$

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



Optimal Policy!

k=3 부터 greedy policy는 optimal policy로 수렴한다.

이미지 출처: [1]

Demo: Policy Iteration

https://github.com/Curt-Park/KIAS_RL_Basics

Value Iteration

- Policy improvement의 greedy selection을 policy evaluation 단계에 반영하는 방법을 **Value Iteration**이라고 한다. 아래의 방법을 통해 반복의 횟수를 상당히 줄일 수 있다.

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

for all $s \in S$.

Value Iteration

- 다음은 value iteration에 대한 pseudocode다.

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 $V(terminal) = 0$

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

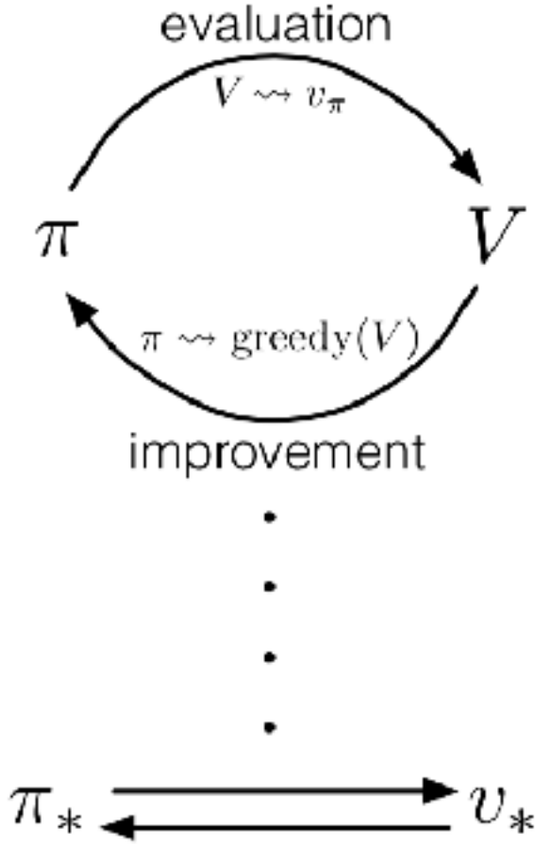
이미지 출처: [1]

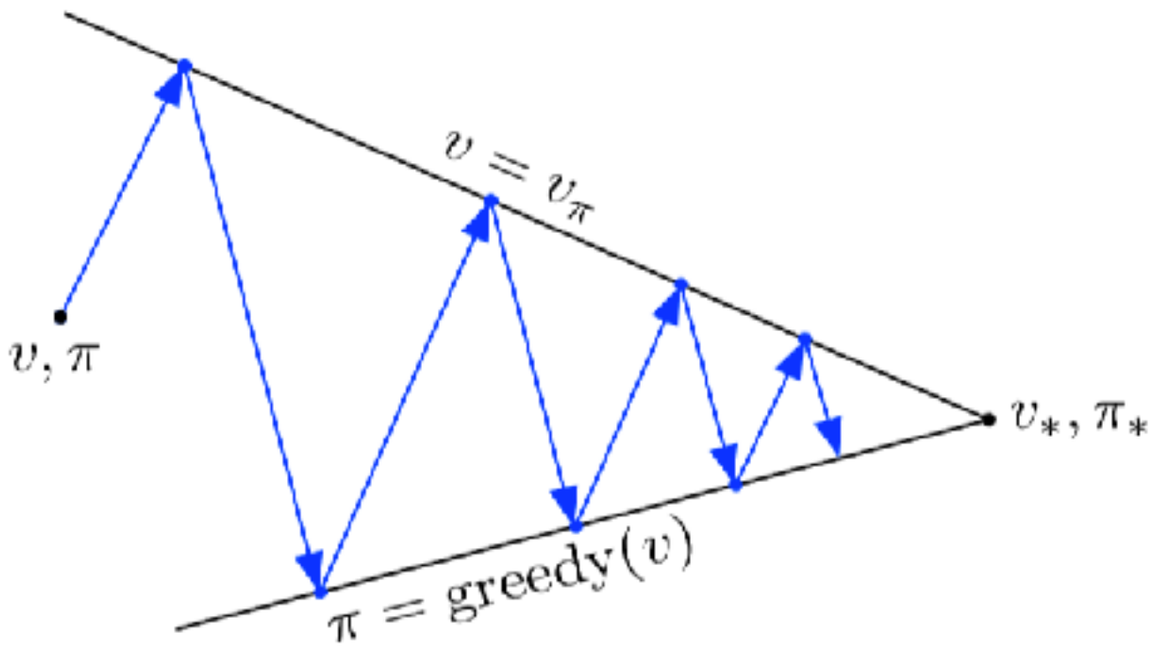
Demo: Value Iteration

https://github.com/Curt-Park/KIAS_RL_Basics

Generalized Policy Iteration

- Policy evaluation과 policy improvement이 서로 상호작용하는 과정을 Generalized Policy Iteration (GPI)라 부른다. 이에 대한 기대효과는 아래와 같다.





이미지 출처: [1]

Dynamic programming은
상당히 높은 계산량과 정확한 model을 요구한다.

실제 경험을 통해 환경과 상호작용하여 얻은 정보를
그때그때 반영한다면?

Dynamic programming은
상당히 높은 계산량과 정확한 model을 요구한다.

실제 경험을 통해 환경과 상호작용하여 얻은 정보를
그때그때 반영한다면?

Model-free RL Methods

Monte Carlo Methods

Introduction

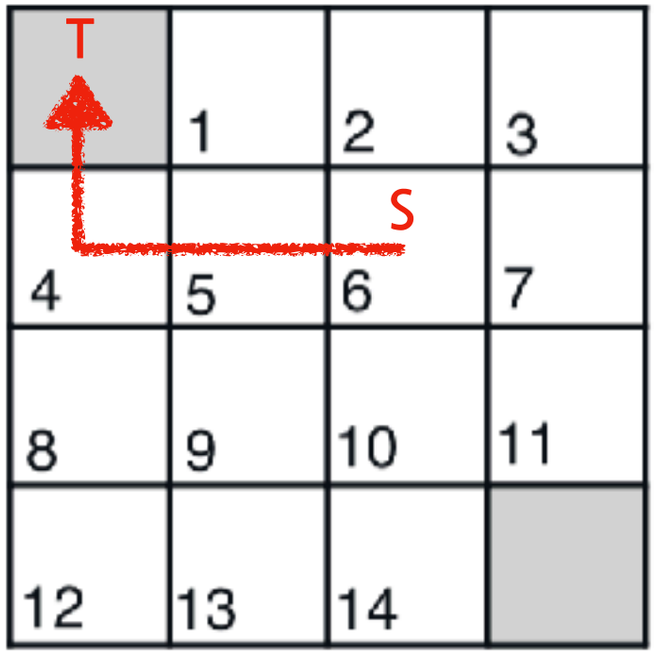
- Monte Carlo methods는 환경을 완전히 알고있다고 전제하지 않는다.
- Monte Carlo methods는 실제 경험만을 요구한다.

➡ 하나의 episode를 경험한 뒤, 각 state 또는 state-action pair에 대한 average return을 계산한다.

Introduction

- 한 episode를 끝까지 경험해본 뒤 업데이트

actions



$$R_t = -1$$
on all transitions

이미지 출처: [1]

Monte Carlo Prediction
For State Value function

- **First-visit MC method**: 하나의 에피소드에서 처음으로 방문한 state에 대해서만 average return을 구한다.

- **Every-visit MC method**: 하나의 에피소드에서 모든 방문하는 state에 대해서 average return을 구한다.

Monte Carlo Prediction
For State Value function

- **First-visit MC method**: 하나의 에피소드에서 처음으로 방문한 state에 대해서만 average return을 구한다.

- **Every-visit MC method**: 하나의 에피소드에서 모든 방문하는 state에 대해서 average return을 구한다.

Monte Carlo Prediction
For State Value function

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: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

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

Append G to $Returns(S_t)$

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

이미지 출처: [1]

Monte Carlo Prediction
For Action Value function

- 모델이 존재하지 않으므로 state간의 transition에 대한 정보도 주어져있지 않다. 즉, state value function의 추정만으로는 부족함이 있다.

- First-visit MC method와 every-visit MC method 둘 다 유효하다.

Monte Carlo Prediction
For Action Value function

- 수렴을 보장하는 간단한 방법
  - Evaluation and improvement on an episode-by-episode basis
  - Exploring starts

Monte Carlo Prediction
For Action Value function

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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \text{argmax}_a \, Q(S_t, a)$

이미지 출처: [1]

Monte Carlo Prediction
For Action Value function

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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

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

Append G to $Returns(S_t, A_t)$

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

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

이미지 출처: [1]

Exploring Starts를 유심히 살펴보자.

Monte Carlo Control
Without Exploring Starts

- Motivation

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):
다소 비현실적인 조건
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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

이미지 출처: [1]

Monte Carlo Control
Without Exploring Starts

- Agent가 직접 탐험을 한다면 어떨까?

- 두 가지 방법
  - **On-policy methods**: Episode를 생성하는 policy와 policy evaluation & improvement의 대상이 동일.
  - **Off-policy methods**: Episode를 생성하는 policy와 policy evaluation & improvement의 대상이 별개.

Monte Carlo Control
Without Exploring Starts

- Agent가 직접 탐험을 한다면 어떨까?
- 두 가지 방법
  - **On-policy methods**: Episode를 생성하는 policy와 policy evaluation & improvement의 대상이 동일.
  - **Off-policy methods**: Episode를 생성하는 policy와 policy evaluation & improvement의 대상이 별개.

Monte Carlo Control
Without Exploring Starts

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

Algorithm parameter: small $\varepsilon > 0$

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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

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

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

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

이미지 출처: [1]

Monte Carlo Control
Without Exploring Starts

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

Algorithm parameter: small $\varepsilon > 0$

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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

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

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

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

동일한 policy 사용

K-armed bandit에서의 exercise와 동일

* $|A(S_t)|$는 S_t에서 선택가능한 action의 갯수를 뜻함.

이미지 출처: [1]

Monte Carlo Control
Without Exploring Starts

- Agent가 직접 탐험을 한다면 어떨까?

- 두 가지 방법
  - **On-policy methods**: Episode를 생성하는 policy와 policy evaluation & improvement의 대상이 동일.
  - **Off-policy methods**: Episode를 생성하는 policy와 policy evaluation & improvement의 대상이 별개.

Off-policy Prediction via Importance sampling

- On-policy methods의 단점
  - 지속적으로 탐험을 해야하기 때문에 optimal policy를 학습할 수 없다.
- Policy를 두 개로 분리하면 어떨까?
  - **Target policy**: 학습의 대상이 되는 policy
  - **Behavior policy**: behavior를 결정하는 policy (에피소드를 생성하는 policy)

Off-policy Prediction via Importance sampling

- On-policy methods의 단점
  - 지속적으로 탐험을 해야하기 때문에 optimal policy를 학습할 수 없다.
- Policy를 두 개로 분리하면 어떨까?
  - **Target policy**: 학습의 대상이 되는 policy
  - **Behavior policy**: behavior를 결정하는 policy (에피소드를 생성하는 policy)

    On-policy는 target policy = behavior policy인 off-policy의 special case로 볼 수 있다.

Off-policy Prediction via Importance sampling

- 대부분의 off-policy 방법은 importance sampling을 이용한다. Importance sampling이란 다른 두 가지의 확률분포를 함께 이용하여 expected value를 추정하는 방법이다.
- Importance sampling을 이해하기 위해서는 우선 Importance-sampling ratio를 살펴봐야 한다.

Off-policy Prediction via Importance sampling

- Importance-sampling ratio
  - 하나의 에피소드에 대한 trajectory $A_t, S_{t+1}, A_{t+1}, \ldots, S_T$ 가 주어졌을때, 어떤 policy π 에 의해 해당 trajectory가 발생할 확률은 다음과 같다.

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

Off-policy Prediction via Importance sampling

- Importance-sampling ratio
  - 주어진 trajectory가 behavior policy에 의해 생성되었다고 할때, target policy에서 이 trajectory가 발생할 상대적인 확률을 다음과 같이 표현하고 이를 **importance-sampling ratio**라고 명명한다.

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

Off-policy Prediction via Importance sampling

- Ordinary Importance-sampling
  - 다음과 같이 value를 추정하는 방식을 ordinary importance-sampling이라고 한다.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}.$$

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

$\mathcal{T}(s)$: The total number of visits on state s

Off-policy Prediction via Importance sampling

- Ordinary Importance-sampling
  - 다음과 같이 value를 추정하는 방식을 ordinary importance-sampling이라고 한다.

무한 횟수의 에피소드에서 behavior policy가 생성하지 않을만한 에피소드가 만들어진다면?

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

➡ Unbounded Variance

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

$\mathcal{T}(s)$: The total number of visits on state s

Off-policy Prediction via Importance sampling

- Weighted Importance-sampling
  - Unbounded variance에 의한 문제를 막기 위해 importance-sampling ratio로 normalizing을 한 형태. Ordinary importance-sampling 보다 더 선호된다.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}},$$

or zero if the denominator is zero.

Incremental Implementation

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

- 계산효율을 위해 weighted Importance-sampling의 incremental implementation을 유도하면 다음과 같다.

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

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).

Off-policy Monte Carlo Control

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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit For loop
$W \leftarrow W \frac{1}{b(A_t|S_t)}$

이미지 출처: [1]

Off-policy Monte Carlo Control

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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit For loop
$W \leftarrow W \frac{1}{b(A_t|S_t)}$

A_t가 target policy의 입장에서 greedy action 이면 loop 종료

A_t가 target policy의 입장에서 greedy action 이므로 분자가 1

$(W = W \frac{0}{b(A_t|S_t)}$이므로)

이미지 출처: [1]

Demo:
Monte Carlo
(for state value)

https://github.com/Curt-Park/KIAS_RL_Basics

Monte Carlo Methods는 에피소드 단위로
policy를 업데이트한다.

만약 에피소드가 굉장히 길거나 혹은
Continuing task가 주어졌을 때는 어떻게 해야 할까?

Temporal-Difference Learning

“If one had to identify one idea as central and novel to reinforcement learning, it would be undoubtedly be temporal-difference (TD) learning.”

- Richard S. Sutton and Andrew G. Barto

Introduction

- TD는 MC와 마찬가지로 model없이 실제 경험으로부터 학습한다.

- TD는 DP처럼 final outcome을 기다리지 않고 bootstrapping을 통해 학습한다.

- Nonstationary environment에서의 every-visit MC method가 다음과 같았다면,

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

TD는 아래와 같이 표현된다. (called one-step TD or TD(0))

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

TD Prediction

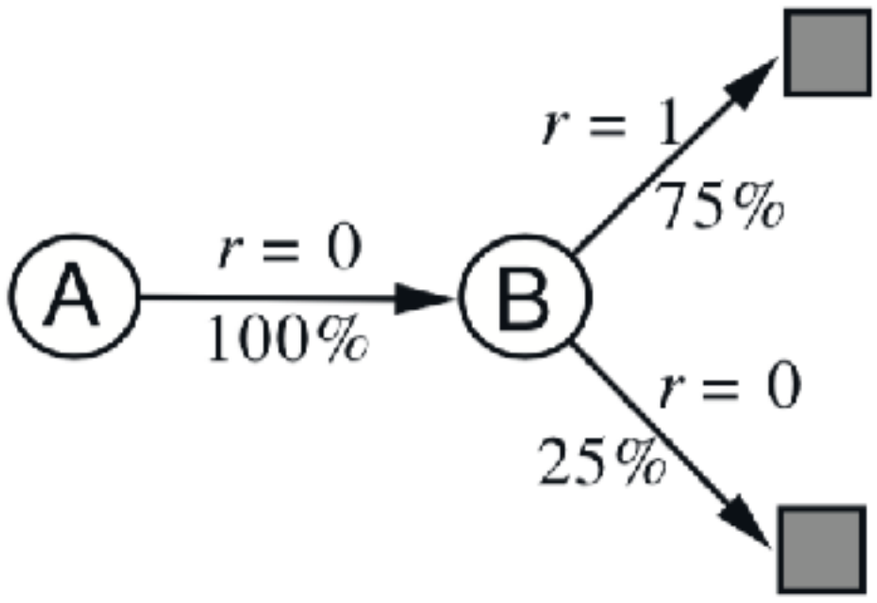
Tabular TD(0) for estimating v_π

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 $V(terminal) = 0$

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'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

이미지 출처: [1]

Exercise

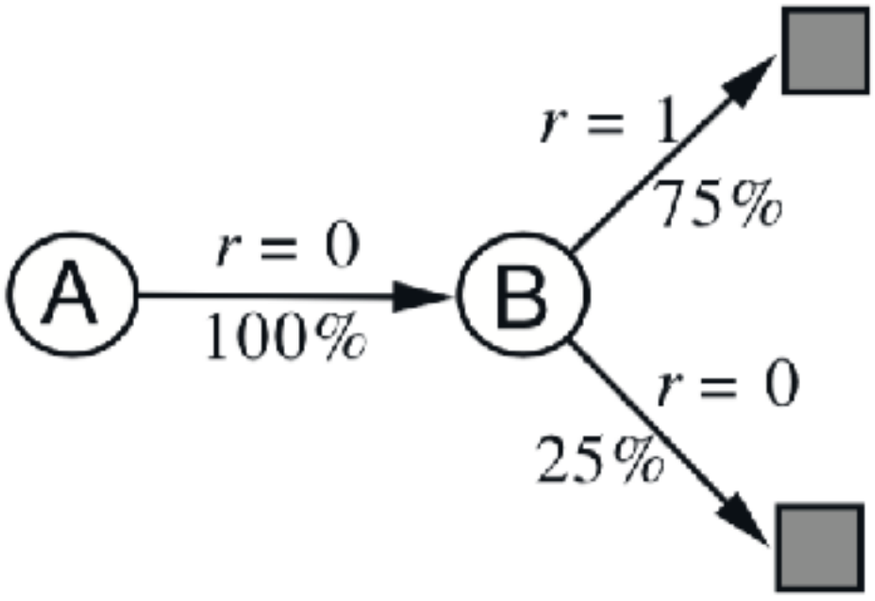


A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

- 좌측의 Unknown Markov reward process에 의해 우측의 8개 에피소드를 수집했다. Step size = 1, gamma=1이고 V(B) = 3/4, V(A) = 0일때 MC와 TD(0)로 계산된 V(A)는 각각 얼마인가?

이미지 출처: [1]

Exercise



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

- TD(0): V(A) = V(A) + 1 * (0 + 1 * V(B) - V(A)) = 3/4
- MC: V(A) = V(A) + 1 * (G(A) - V(A)) = 0

A에서 B로 100%의 확률로 전이 되므로 V(A) = V(B)임을 알 수 있지만, MC는 하나의 에피소드 단위로만 학습하기 때문에 이를 반영하지 못한다.

이미지 출처: [1]

Sarsa: On-policy TD Control

- TD prediction method처럼 state-value function을 학습하는 것이 아니라 action-value function을 학습한다.

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

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 $Q(terminal, \cdot) = 0$

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)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

이미지 출처: [1]

Demo: Sarsa

https://github.com/Curt-Park/KIAS_RL_Basics

Q-learning: Off-policy TD Control

- 초기 강화학습의 발전에 큰 기여를 한 알고리즘 중 하나.

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

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 $Q(terminal, \cdot) = 0$

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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

이미지 출처: [1]

Q-learning: Off-policy TD Control

- 초기 강화학습의 발전에 큰 기여를 한 알고리즘 중 하나.

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

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 $Q(terminal, \cdot) = 0$

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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Behavior policy로 동작

Target policy로 동작

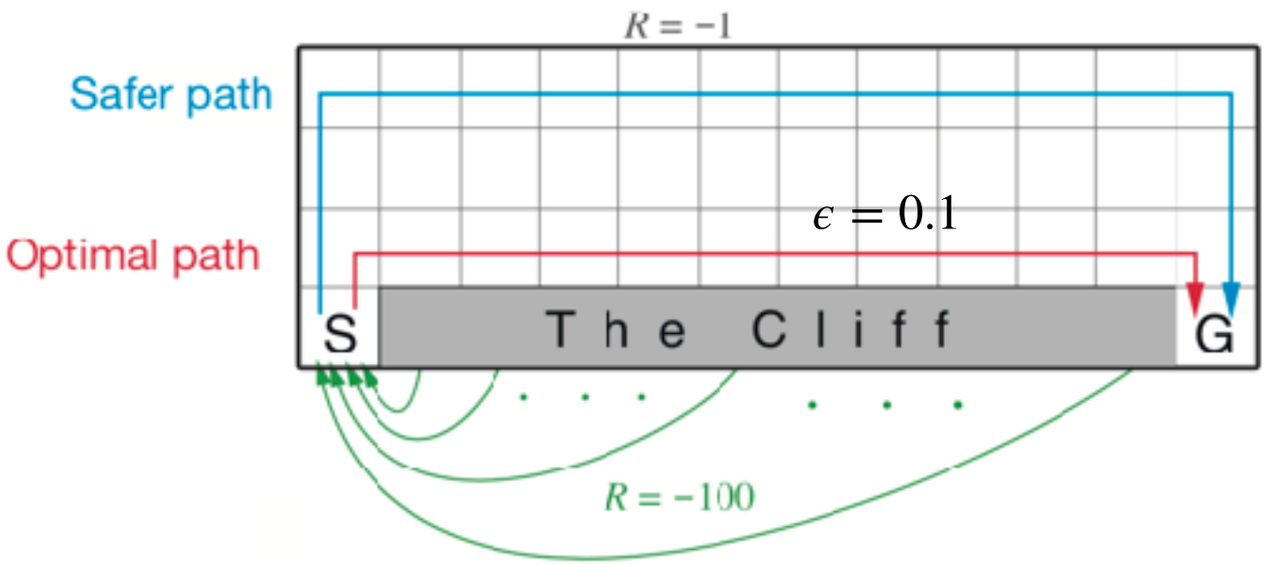
이미지 출처: [1]

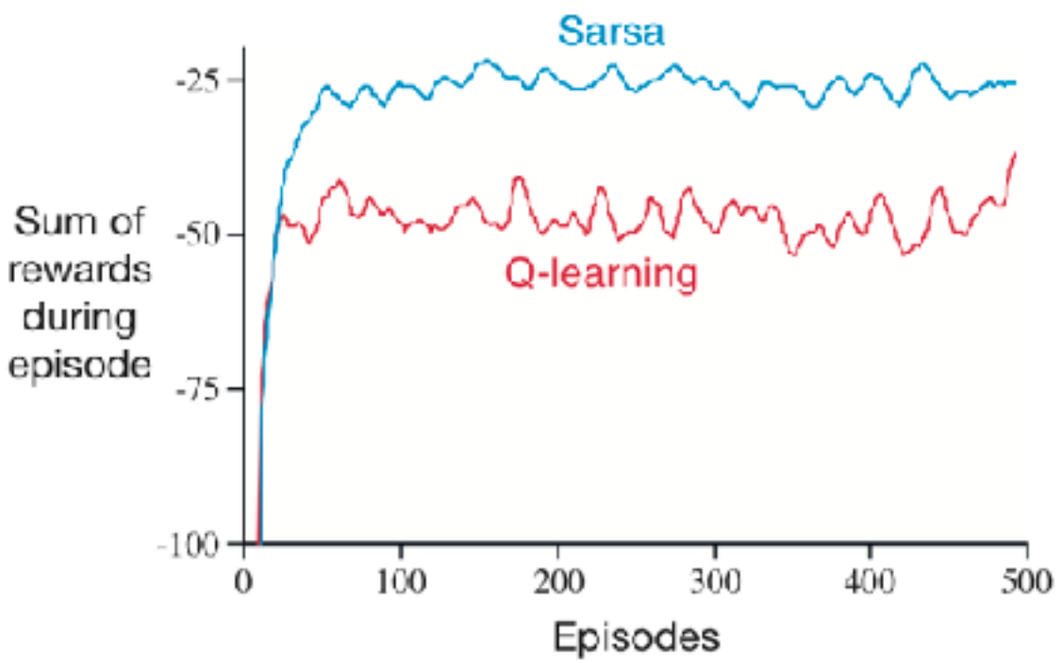
Demo: Q-learning

https://github.com/Curt-Park/KIAS_RL_Basics

Example

- **Cliff Walking**: Sarsa와 Q-learning을 비교하기에 아주 적합한 예제다.





- Sarsa는 Q함수를 업데이트 할 때도 ϵ-greedy 방식을 취하기 때문에 cliff 근처에서는 penalty가 높게 나올 수 밖에 없다. 그러므로 상대적으로 안전한 길로 돌아간다.
- 반면 Q-learning은 high-risk를 무시하는 (낙관적인) 성향을 보인다. 이로 인해 optimal path를 찾는데 성공하지만 학습시 ϵ에 의해 종종 cliff로 빠진다.

이미지 출처: [1]

Double Q-learning

- Double Q-learning은 2개의 Q함수를 사용한다. 이는 positive bias를 줄여주는 역할을 한다.

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

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 $Q(\textit{terminal}, \cdot) = 0$

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 probabilility:

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

    else:

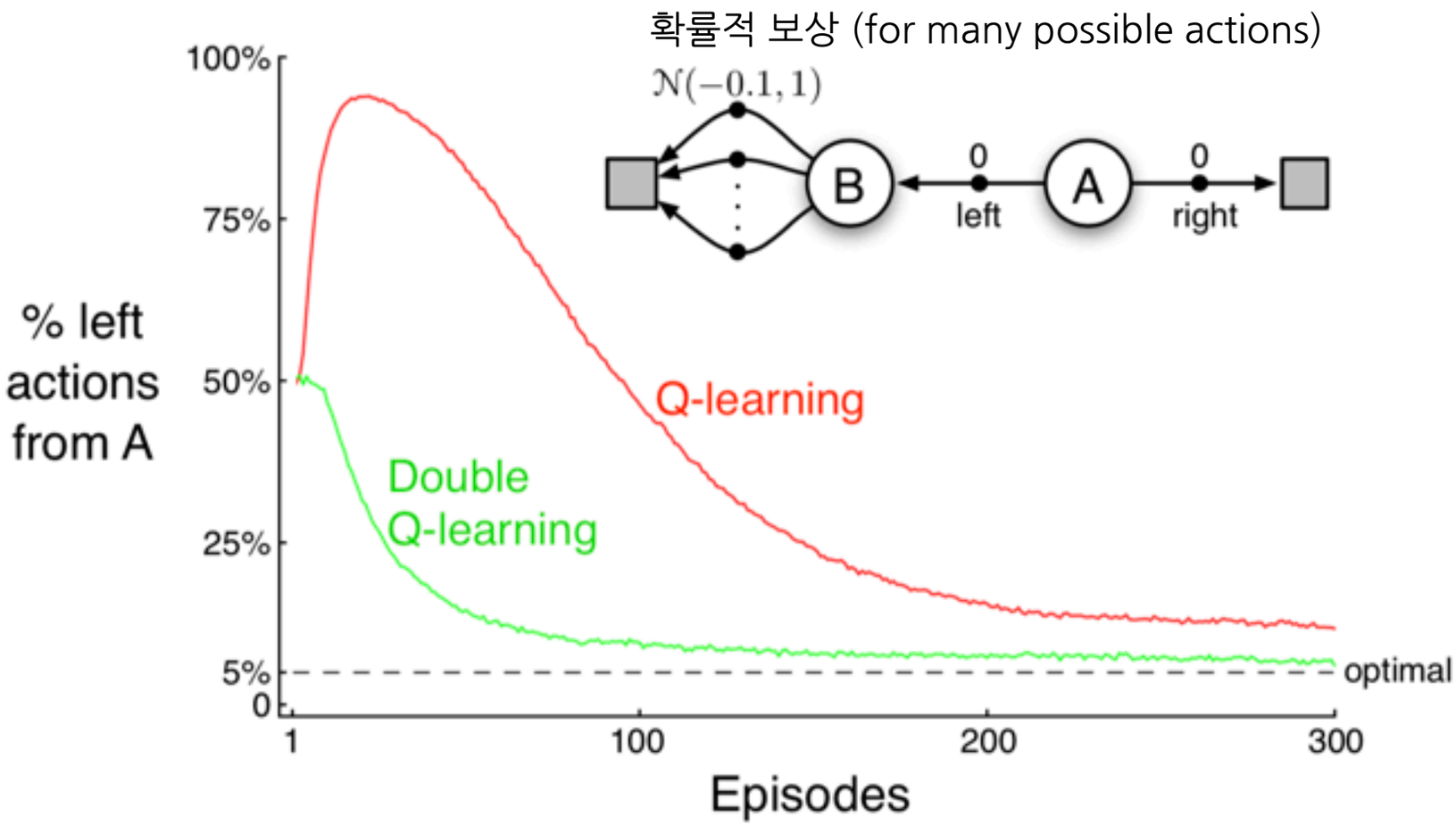
$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

이미지 출처: [1]

Example

- Maximization bias

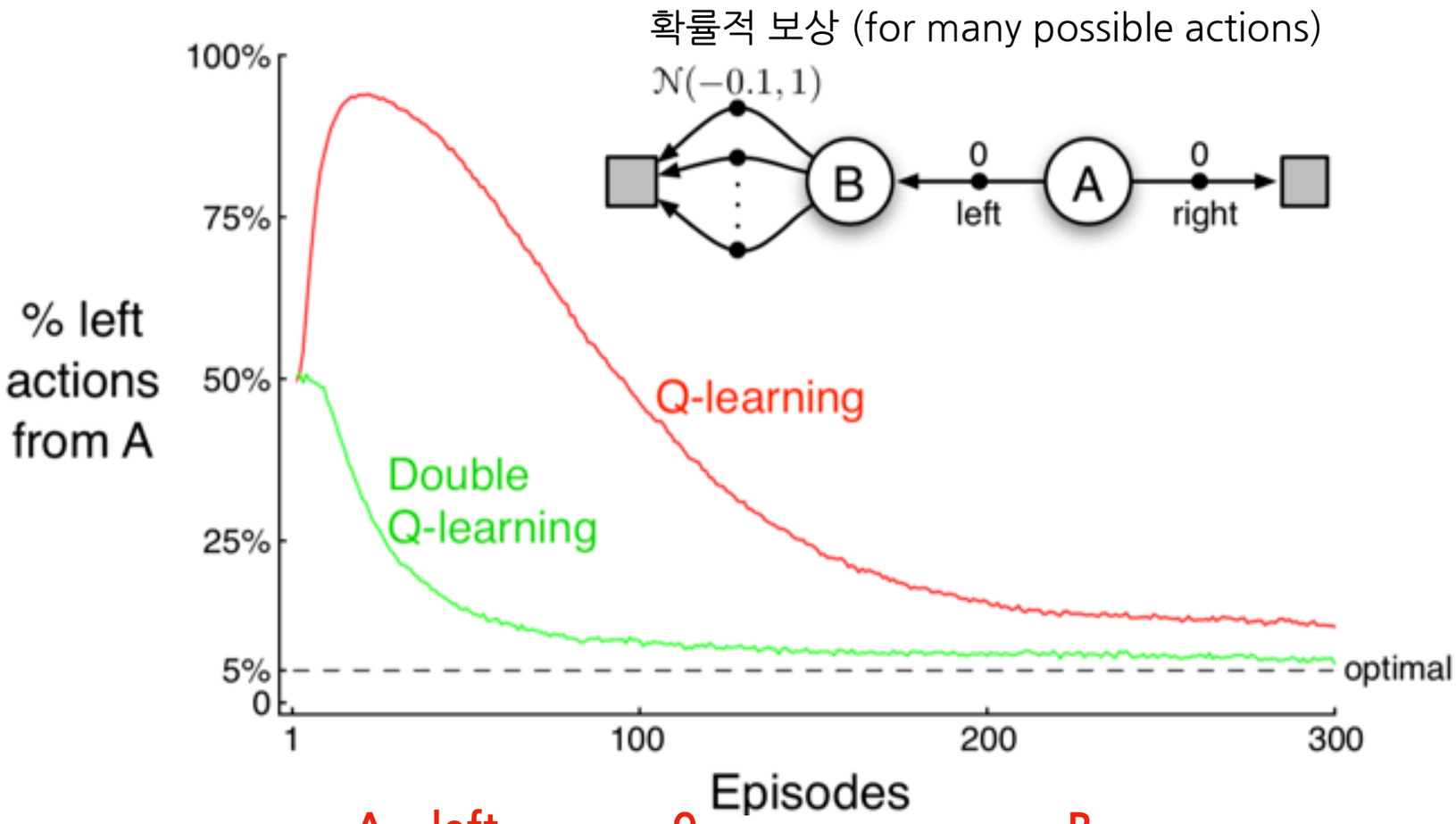


A를 초기위치라고 하자. 이때, V(B)의 기댓값이 -0.1이므로 좌측으로 이동하는 것은 좋은 결정이 아니다. 하지만 Q-learning의 낙관적인 성향때문에 그림에서처럼 초반 계속하여 잘못된 선택을 한다.

이미지 출처: [1]

Example

- Maximization bias



A left 0 B

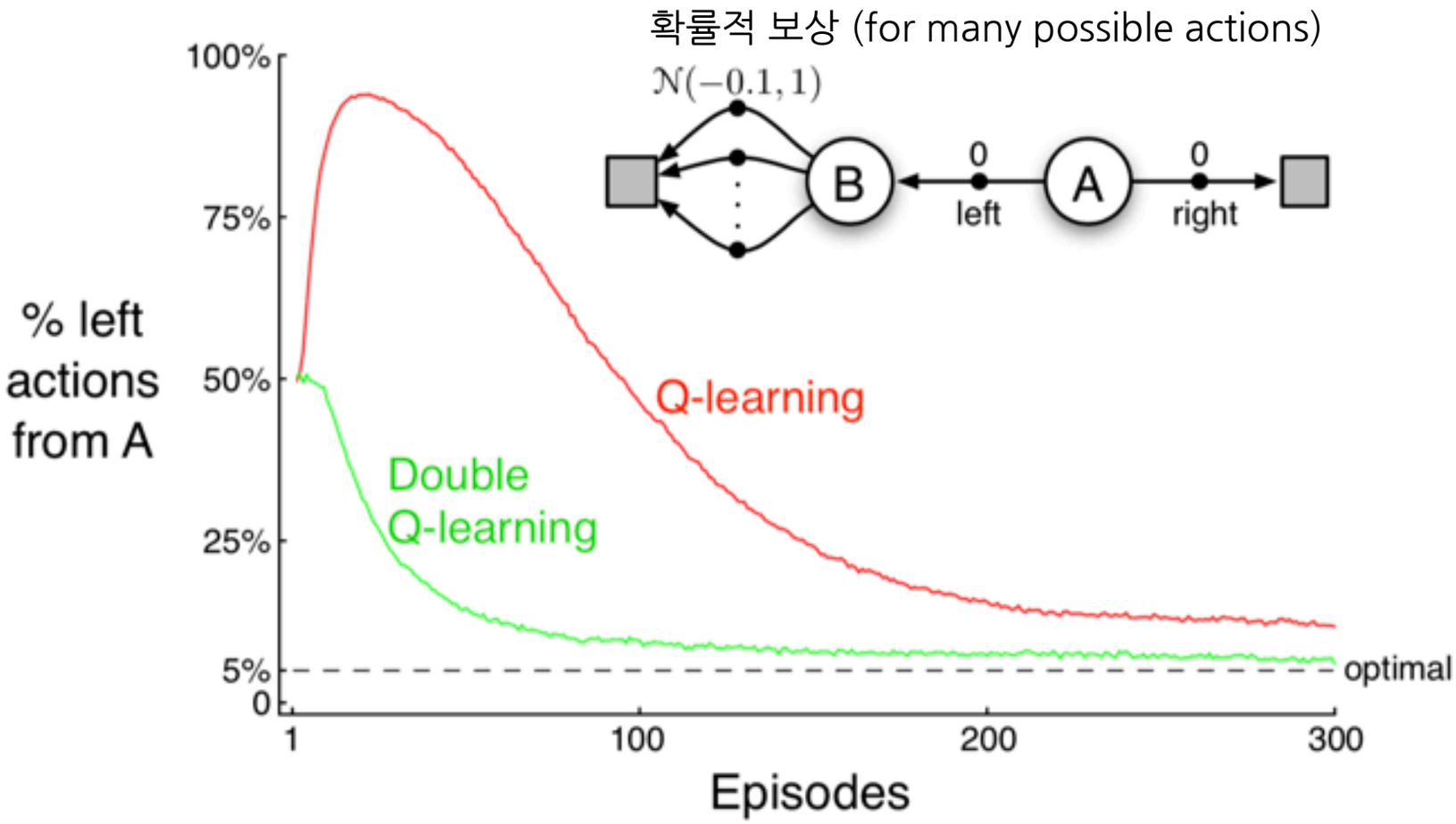
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

바로 이 max함수에 의해 positive bias가 발생한다.

이미지 출처: [1]

Example

- Maximization bias



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

Double Q-learning의 경우 action을 선택하는 Q함수와 이를 평가하는 Q함수가 별도로 존재하기 때문에 bias가 상당히 누그러든다.

이미지 출처: [1]

Double Q-learning

- Double Q-learning과 같은 맥락으로 Double-Sarsa도 존재한다.
- Double Q-learning의 좀 더 일반화된 알고리즘으로는 Multi Q-learning[3]이 있다.

Other Important Topics

지금까지 살펴본 내용

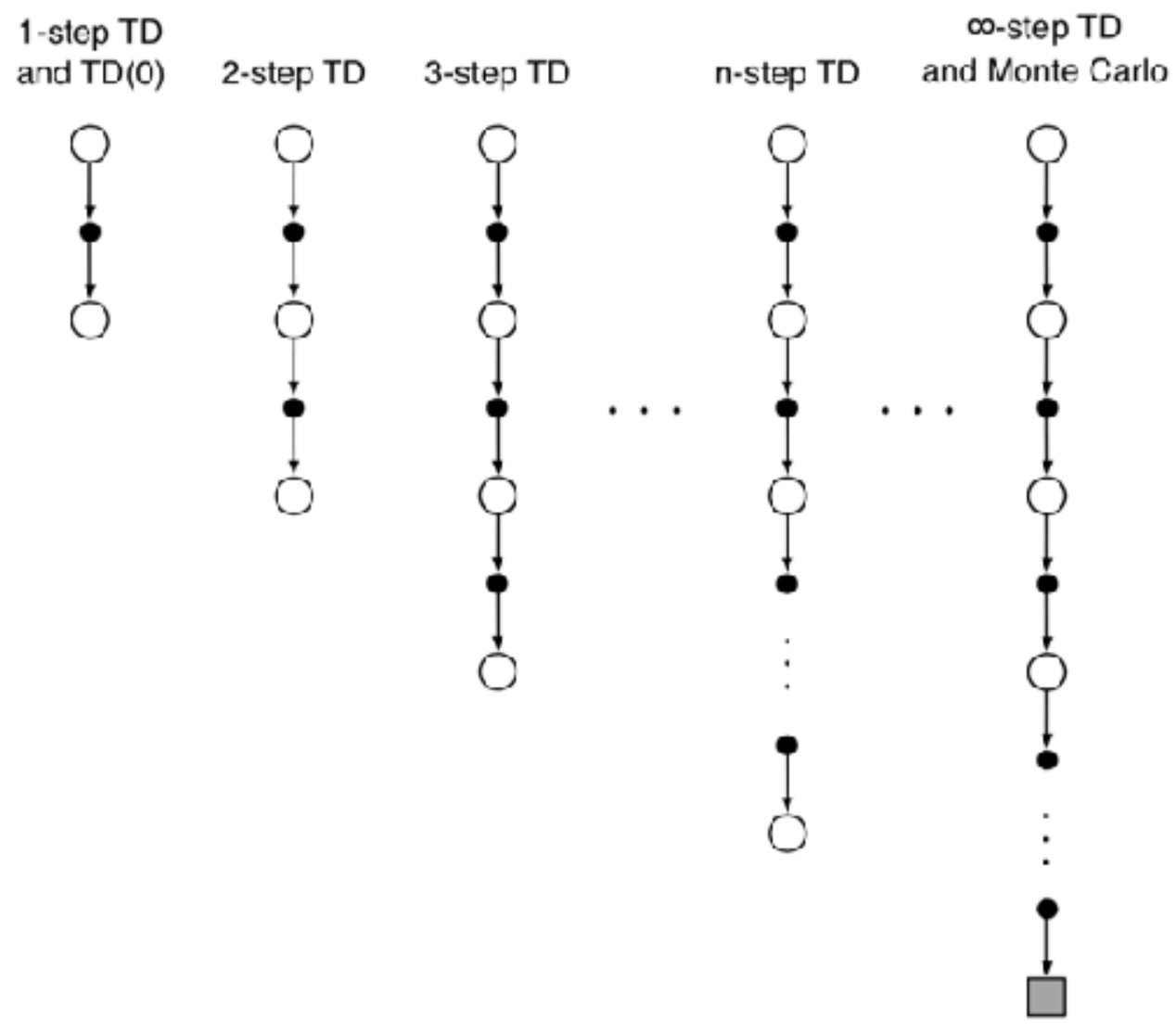
- 교과서[1]의 Chapter 6까지의 내용
  - Introduction
  - Multi-armed Bandits
  - Finite Markov Decision Processes
  - Dynamic Programming
  - Monte Carlo Methods
  - Temporal-Difference Learning

다루지 못한 내용

- 교과서[1]의 Chapter 7부터
  - n-step Bootstrapping
  - Planning and Learning with Tabular Methods
  - Approximate Solution Methods

n-step Bootstrapping

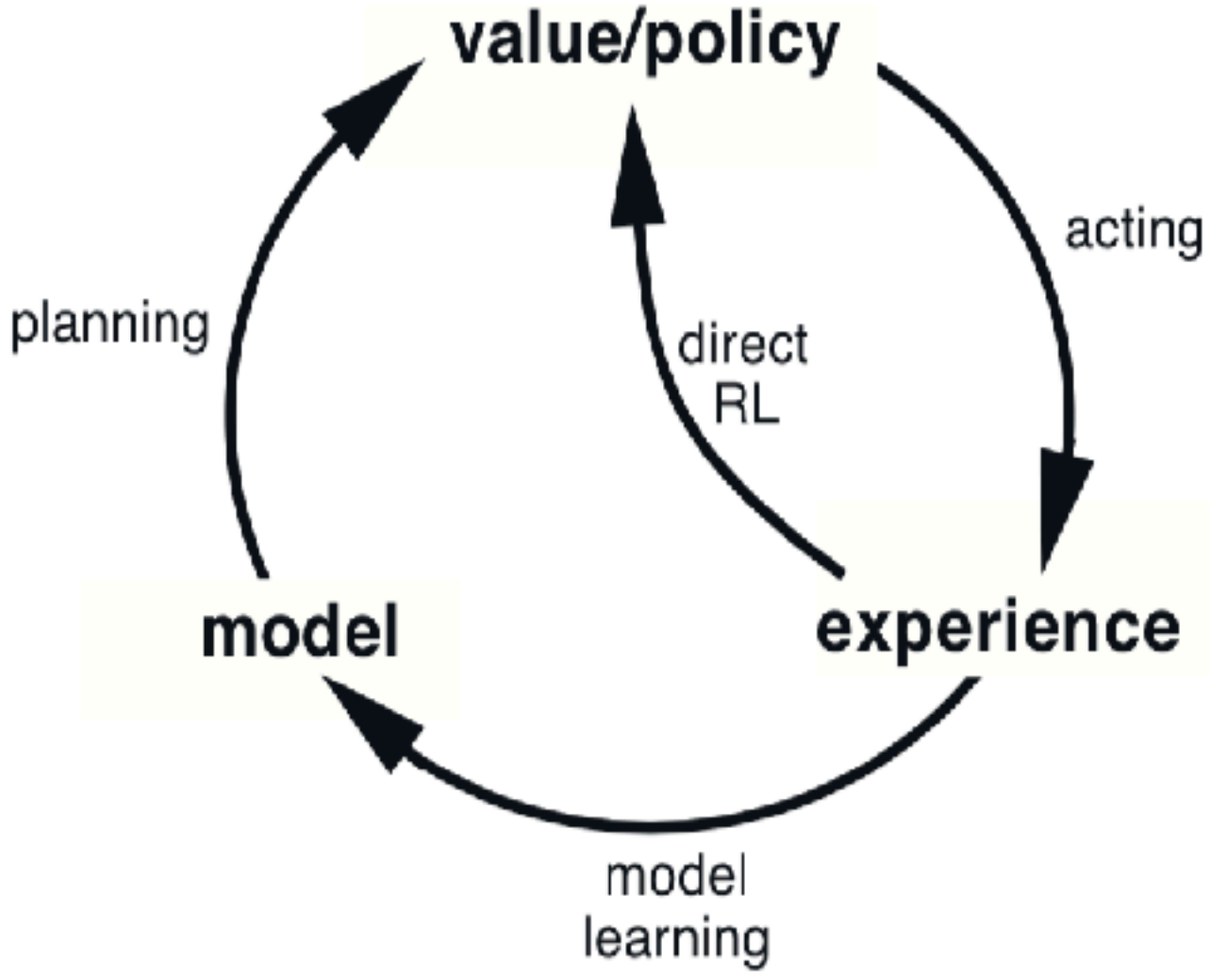
- The backup diagram of n-step TD methods



이미지 출처: [1]

Planning and Learning with Tabular Methods

- Model-free methods와 model-based methods의 통합



이미지 출처: [1]

Approximate Solution Methods

- 다음의 문제들은 Tabular method로 해결하기에 너무 많은(혹은 무한한) 상태를 가지고 있다[6].
  - Backgammon: 10^{20} states
  - 바둑: 10^{170} states
  - 헬리콥터 조종: continuous state space
- 위와 같은 상태를 표현할 수 있는 근사함수가 필요하다.
- Deep Reinforcement Learning에서는 value function에 대한 근사함수로 deep neural network를 사용한다.

DQN

Existing problem

- Reinforcement learning agents가 실제 세계의 복잡도를 가진 문제에서 잘 작동하기 위해서는:

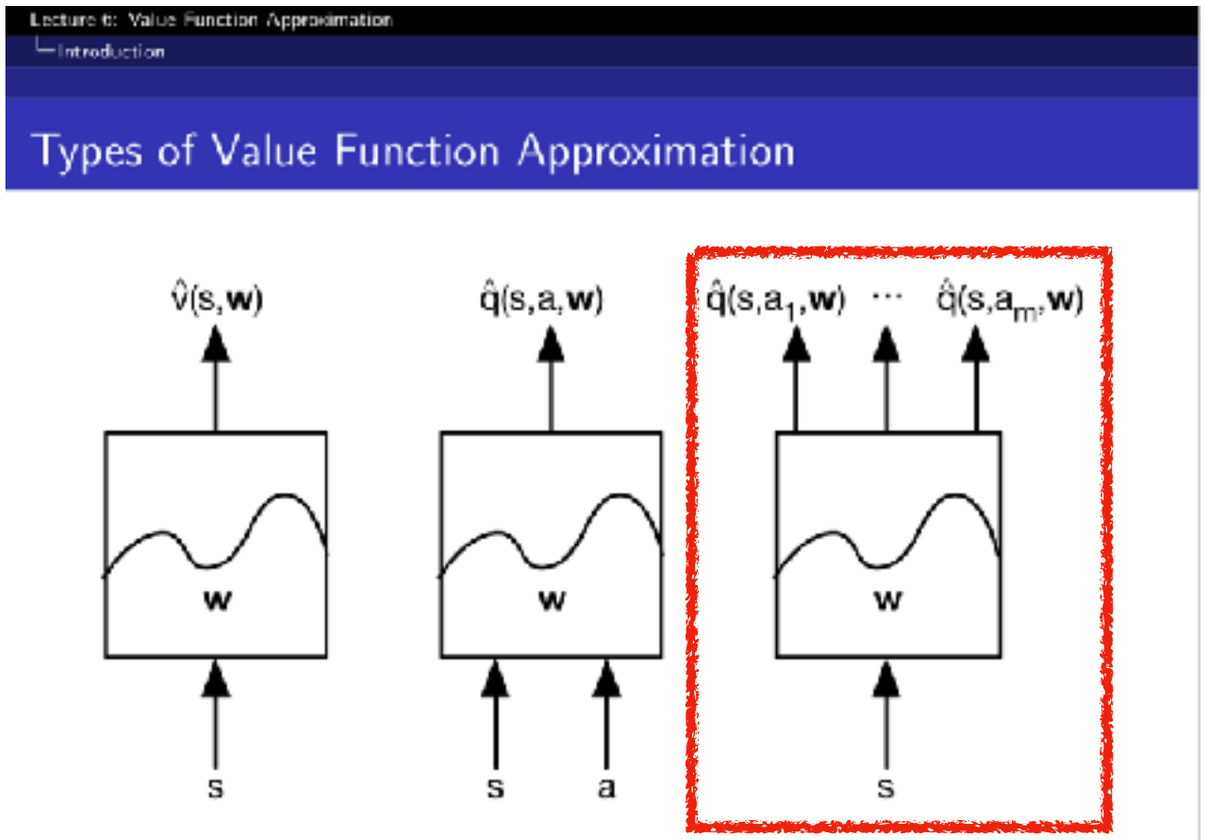
  - 상당한 고차원의 sensory inputs으로부터 representation을 잘 얻어낼 수 있어야 한다.

  - 얻어낸 representation으로 과거의 경험을 일반화하여 새로운 상황에서도 잘 적용할 수 있어야한다.

  ➡ RL의 유용성은 아주 제한적인 도메인(e.g. 저차원의 state-space를 가진 도메인)에 머물러 있다.

Objective

- Deep Convolutional Neural Network가 non-linear function approximator로써 이례적인 성능을 보이고 있다.
- CNN 구조를 이용하여 raw sensory data를 입력으로하는 action-value function의 근사함수를 만들어보면 어떨까?



이미지 출처: [6]

Architecture

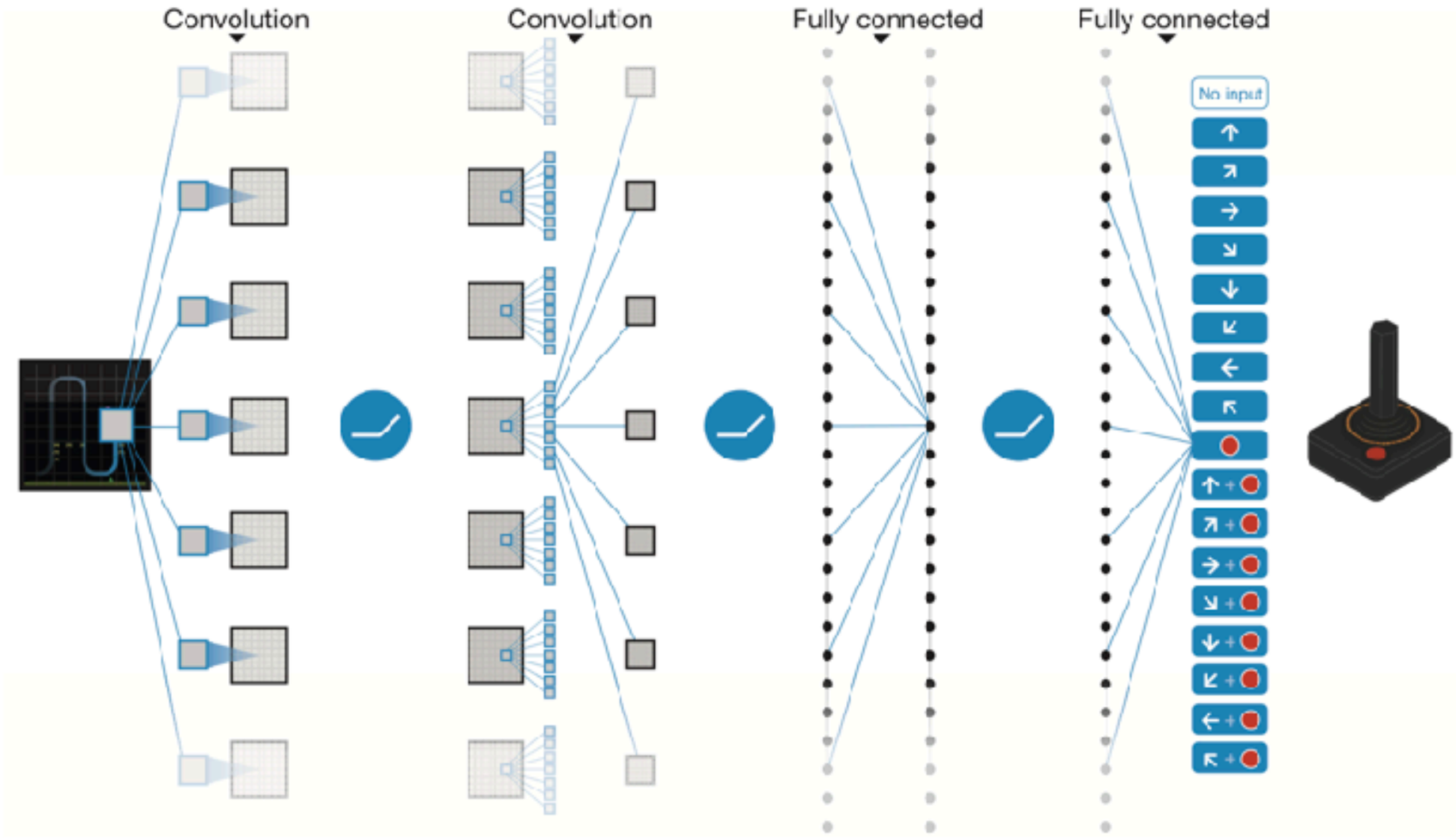


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking 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, $\max(0,x)$).

이미지 출처: [4]

Architecture

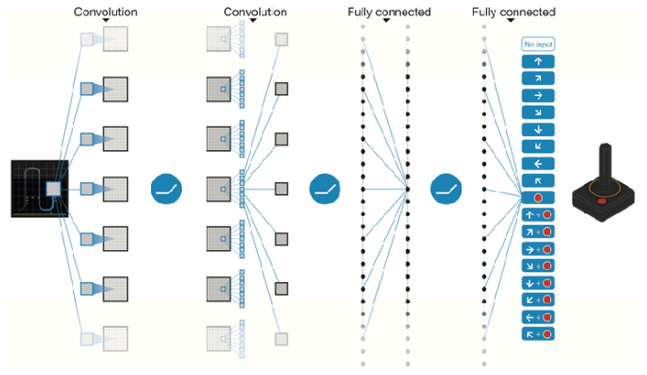


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking 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, $\max(0,x)$).

- 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

Challenges

- 강화학습에서 action-value(Q) function을 나타내기 위해 non-linear function approximator를 사용하였을 경우 수렴이 보장되지 않는 것으로 알려져 있다.

Convergence of Prediction Algorithms

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

이미지 출처: [6]

Challenges

- 강화학습에서 action-value(Q) function을 나타내기 위해 non-linear function approximator를 사용하였을 경우 수렴이 보장되지 않는 것으로 알려져 있다.

- 다음과 같은 이유들 때문이다[7].
  - Correlation between samples
  - Non-stationary targets

Challenges

- 강화학습에서 action-value(Q) function을 나타내기 위해 non-linear function approximator를 사용하였을 경우 수렴이 보장되지 않는 것으로 알려져 있다.

- 다음과 같은 이유들 때문이다[7].

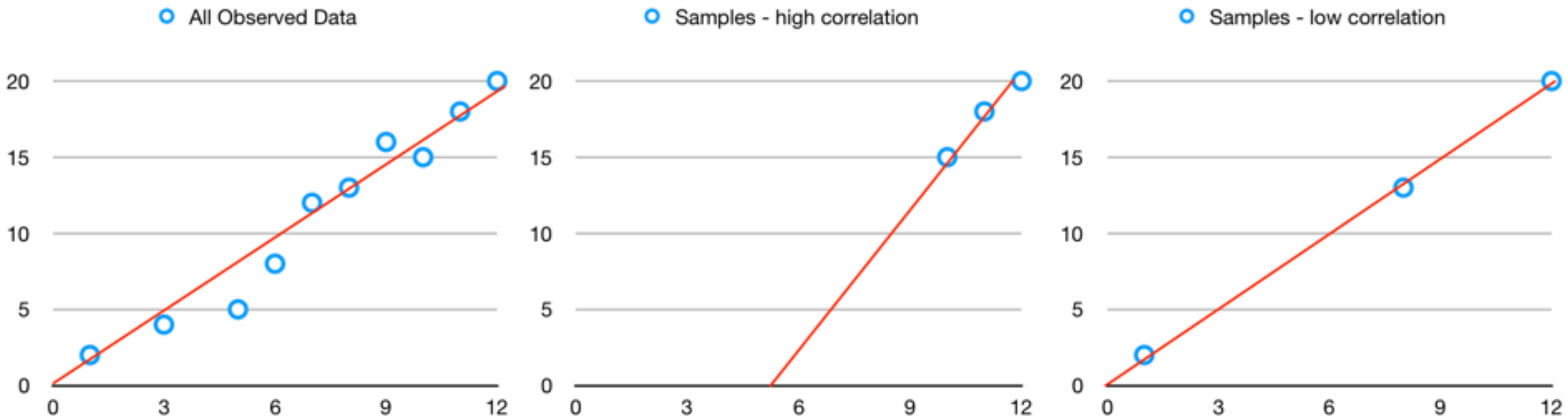
  - Correlation between samples

  - Non-stationary targets

Challenges

- **Correlation between samples**

강화학습에서의 학습데이터는 시간의 흐름에 따라 순차적으로 수집되고,
이 순차적인 데이터는 근접한 것들끼리 높은 correlation을 띄게된다.



만약에 이 순차적인 데이터를 그대로 입력으로 활용하게 되면 입력이미지
들 간의 높은 correlation에 의해 학습이 불안정해질 것이다.

Challenges

- **Correlation between samples (Neural Network perspective)**

  네트워크의 마지막 hidden layer를 통해 입력 s에 대한 representation vector $x(s)$를 얻을 수 있다고 할때, 여기에 어떤 action a에 대한 weight w_a를 내적하여 $Q(s,a)$를 얻을 수 있다.

$$Q(s,a;\theta) = x(s)^T w_a$$

  이때, objective function(loss function)은 parameter w_a에 대해 다음과 같은 quadratic form으로 표현된다.

$$L(w_a) = \frac{1}{2}(Q^*(s,a) - Q(s,a;\theta))^2$$

$$= \frac{1}{2}(Q^*(s,a) - x(s)^T w_a)^2$$

Challenges

- Correlation between samples (Neural Network perspective)

$$L(w_a) = \frac{1}{2}(Q^*(s,a) - Q(s,a;\theta))^2$$
$$= \frac{1}{2}(Q^*(s,a) - x(s)^T w_a)^2$$

w_a에 대한 stochastic gradient descent update는 다음과 같다.

$$\nabla w_a Q(s,a;\theta) = x(s)\,.$$
$$\Delta w_a = \alpha\big(Q^*(s,a) - Q(s,a;\theta)\big)x(s)\,.$$

where $\alpha \in (0,1)$ is a step-size parameter.

만약 입력되는 state가 비슷하다면(highly correlated) 그에 대한 representation인 $x(s)$ 또한 비슷할 것이고, w_a에 대한 업데이트가 다소 편향될 것이다.

Challenges

- 강화학습에서 action-value(Q) function을 나타내기 위해 non-linear function approximator를 사용하였을 경우 수렴이 보장되지 않는 것으로 알려져 있다.

- 다음과 같은 이유들 때문이다[7].

  - Correlation between samples

  - Non-stationary targets

Challenges

- **Non-stationary targets**

  MSE(Mean Squared Error)를 이용하여 optimal action-value function을 근사하기 위한 loss function을 다음과 같이 표현할 수 있다.

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma max_{a'} Q(s', a'; \theta_i) - Q(s, a; \theta_i)\right)^2\right],$$

  where θ_i are the parameters of the Q-network at iteration i.

  이는 Q-learning target 를 근사하는 $y_i = r + \gamma max_{a'} Q(s', a'; \theta_i)$ 를 구하려는 것과 같다. 문제는 $Q(s, a; \theta_i)$ 가 Q함수에 대해 의존성을 갖고 있으므로 Q함수를 업데이트하게 되면 target y_i 또한 움직이게 된다는 것이다. 이 현상으로 인한 학습의 불안정해진다.

Challenges

- 강화학습에서 action-value(Q) function을 나타내기 위해 non-linear function approximator를 사용하였을 경우 수렴이 보장되지 않는 것으로 알려져 있다.

- 다음과 같은 이유들 때문이다[7].

  - Correlation between samples ← experience replay (replay memory)

  - Non-stationary targets ← fixed Q-targets

Solutions!

Replay Memory

1.Agent의 경험(experience) $e_t = (s_t, a_t, r_t, s_{t+1})$를 time-step 단위로 data set $D_t = \{e_1, \ldots, e_t\}$에 저장해 둔다.

2.저장된 data set으로부터 uniform random sampling을 통해 minibatch를 구성하여 학습을 진행한다. $\left((s, a, r, s') \sim U(D)\right)$

- Minibatch가 순차적인 데이터로 구성되지 않으므로 입력 데이터 사이의 correlation을 상당히 줄일 수 있다.

- 과거의 경험에 대해 반복적인 학습을 가능하게 한다[6].

- 논문의 실험에서는 replay memory size를 1,000,000으로 설정한다.

Fixed Q-targets

- $Q(s,a;\theta)$와 같은 네트워크 구조이지만 다른 파라미터를 가진(독립적인) target network $\hat{Q}(s,a;\theta^-)$를 만들고 이를 Q-learning target y_i에 이용한다.

$$y_i = r + \gamma max_{a'} \hat{Q}(s', a'; \theta_i^-) .$$

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s,a;\theta_i)\right)^2\right],$$

in which γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i.

- Target network parameters θ_i^-는 매 C step마다 Q-network parameters(θ_i)로 업데이트된다. 즉, C번의 iteration동안에는 Q-learning update시 target이 움직이는 현상을 방지할 수 있다.

- 논문의 실험에서는 C값을 10,000으로 설정한다.

Gradient Clipping

- **Loss function**:

$$\left(r + \gamma max_{a'} Q(s', a'; \theta_i^- - Q(s, a; \theta_i)\right)^2$$

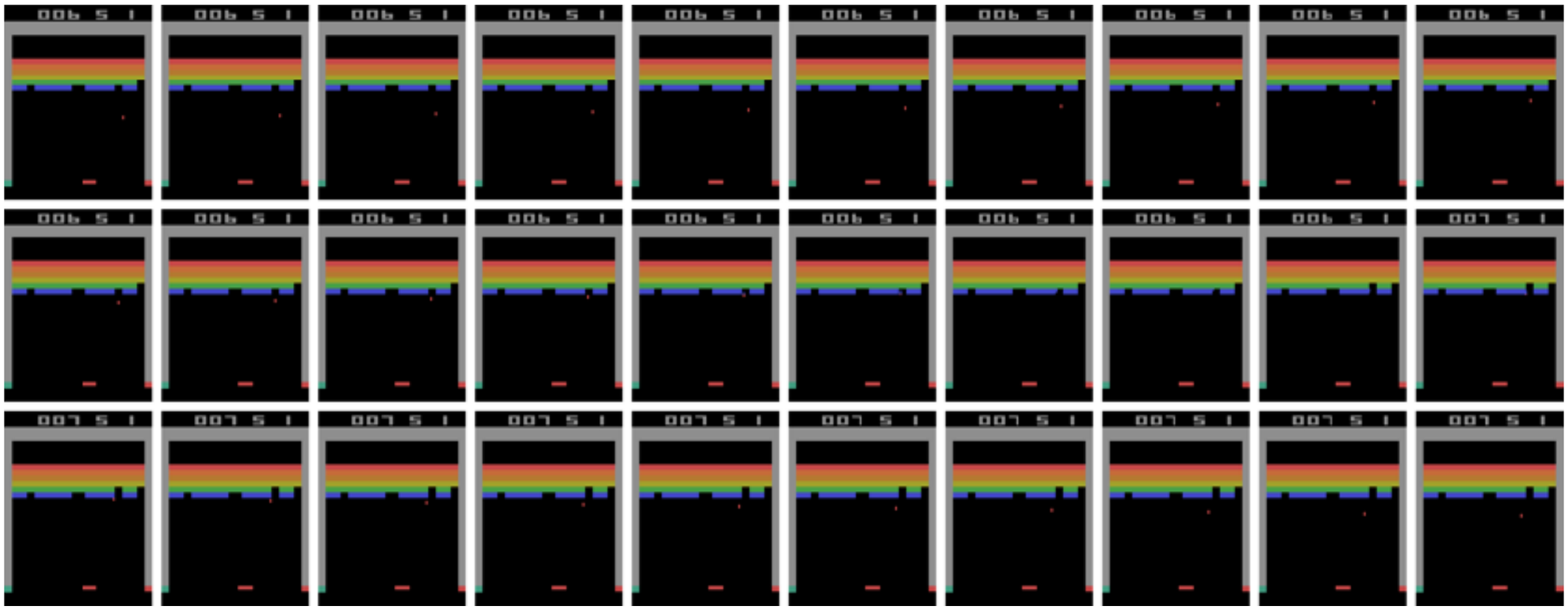
- 위 loss function에 대한 gradient의 절대값이 1보다 클때는 절대값이 1이 되도록 clipping해준다[5].

- Huber loss[10]와 기능적으로 동일하기 때문에 구현시에는 loss function을 Huber loss로 정의하기도 한다[11].

$$\phi_{\mathrm{hub}}(u) = \begin{cases} u^2 & |u| \leq M \\ M(2|u| - M) & |u| > M \end{cases}$$

Data Preprocessing

- Atari 2600은 210x160 pixel의 colour image를 초당 60프레임 정도로 화면에 출력한다. 출력된 화면에 대해 전처리 과정을 거쳐 84x84xm의 입력데이터를 얻는다[9].
(논문에서는 m을 4로 설정)



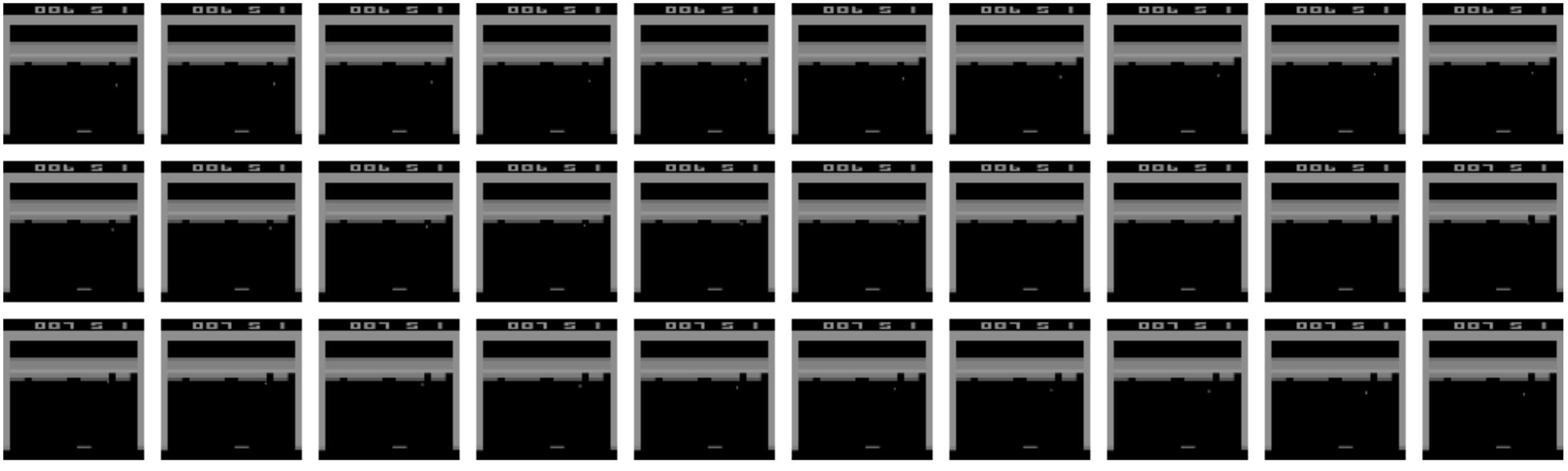
<입력이미지>

이미지 출처: [9]

Data Preprocessing

1. 이미지의 크기를 (210, 160)에서 (84, 84)로 변환

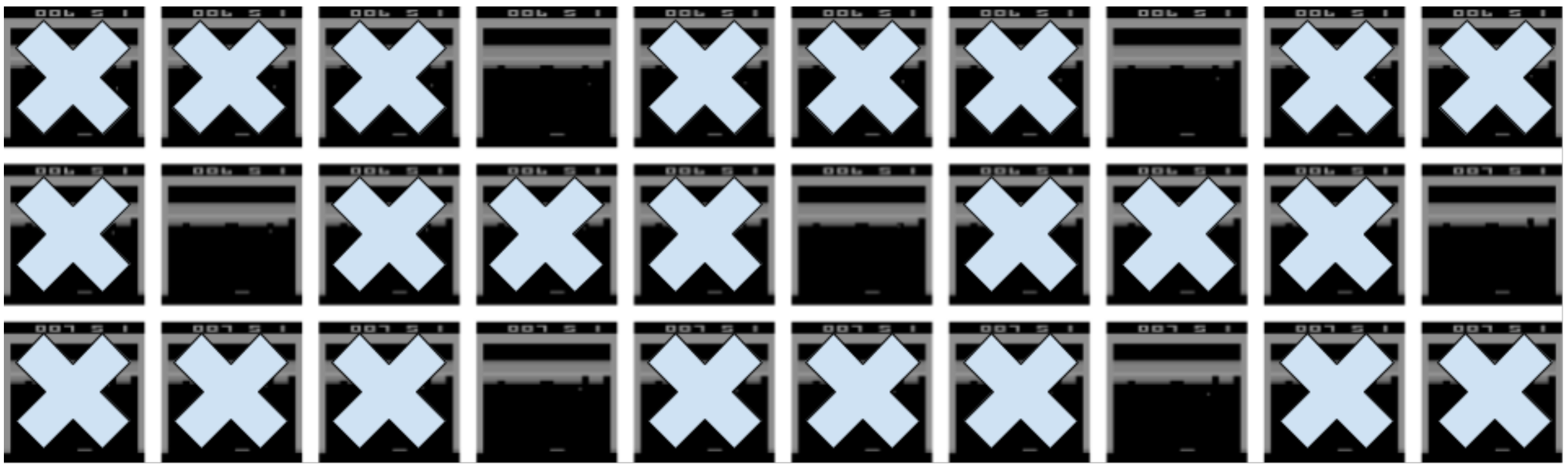
2. RGB 이미지를 grayscale로 변환



이미지 출처: [9]

Data Preprocessing

3. 연속된 이미지들 중 매 k번째에 위치한 이미지들만 선택된다 (Skipped frame)*.

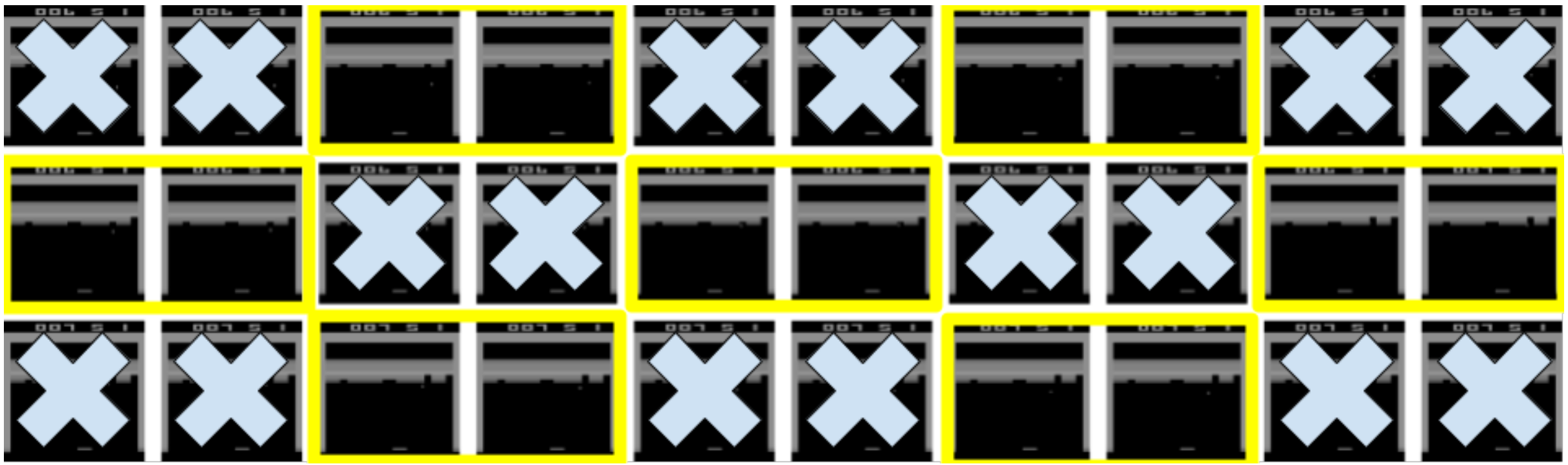


*모든 frame을 전부 입력으로 활용하는 것은 입력 데이터 간의 correlation을 높이게 된다.

이미지 출처: [9]

Data Preprocessing

4. 3에서 선택된 이미지와 그 앞에 연속한 이미지에 대해 pixel-wise(component-wise) maximum을 취해준다*.



*Atari 2600은 화면에 한 번에 표시할 수 있는 sprites가 단 5개 뿐이어서 짝수 프레임, 홀수 프레임에 번갈아서 표시하는 것으로 여러개의 sprites를 화면에 보여줄 수 있었다. 연속된 두 이미지에 대해 component-wise maximum을 취해줌으로써 이를 한 이미지에 모두 표시할 수 있다.

이미지 출처: [9]

Data Preprocessing

5.1~4의 과정을 거친 이미지들을 m개 만큼 쌓으면 네트워크의 입력으로 사용될 수 있는 하나의 상태(state)가 된다*.

* 1~4의 과정들을 거쳐서 얻은 이미지가 $x_1, x_2, \ldots, x_7$라고 할때, 네트워크에 입력되는 상태는 다음과 같다.

$$s1 = (x_1, x_2, x_3, x_4),\ s2 = (x_2, x_3, x_4, x_5),\ \ldots,\ s4 = (x_4, x_5, x_6, x_7)$$

즉, 연속으로 입력되는 상태들간에는 overlapping이 존재한다.

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

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 $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

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

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

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

End For

End For

이미지 출처: [4]

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 $\theta^{-} = \theta$

Initialization

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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

이미지 출처: [4]

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 $\phi_1 = \phi(s_1)$ — Initialization for the episode

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

With probability ε select a random action a_t

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

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 $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

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

End For

End For

이미지 출처: [4]

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

Epsilon-greedy action selection

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 $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

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

End For

End For

이미지 출처: [4]

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

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 $\phi_{t+1} = \phi(s_{t+1})$

Action execution

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 $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

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

End For

End For

이미지 출처: [4]

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

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

Replay memory

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

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

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

End For

End For

이미지 출처: [4]

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

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

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

Gradient descent

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

End For

End For

이미지 출처: [4]

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

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 $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

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

Update fixed Q-targets every C steps

End For

End For

이미지 출처: [4]

Experiments

- Replay memory와 target Q-network의 사용 유무에 따른 퍼포먼스 비교

Extended Data Table 3 | The effects of replay and separating the target Q-network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

DQN agents were trained for 10 million frames using standard hyperparameters for all possible combinations of turning replay on or off, using or not using a separate target Q-network, and three different learning rates. Each agent was evaluated every 250,000 training frames for 135,000 validation frames and the highest average episode score is reported. Note that these evaluation episodes were not truncated at 5 min leading to higher scores on Enduro than the ones reported in Extended Data Table 2. Note also that the number of training frames was shorter (10 million frames) as compared to the main results presented in Extended Data Table 2 (50 million frames).

이미지 출처: [4]

Experiments

- 아래 그래프는 average action value가 점차 수렴함을 보여준다.

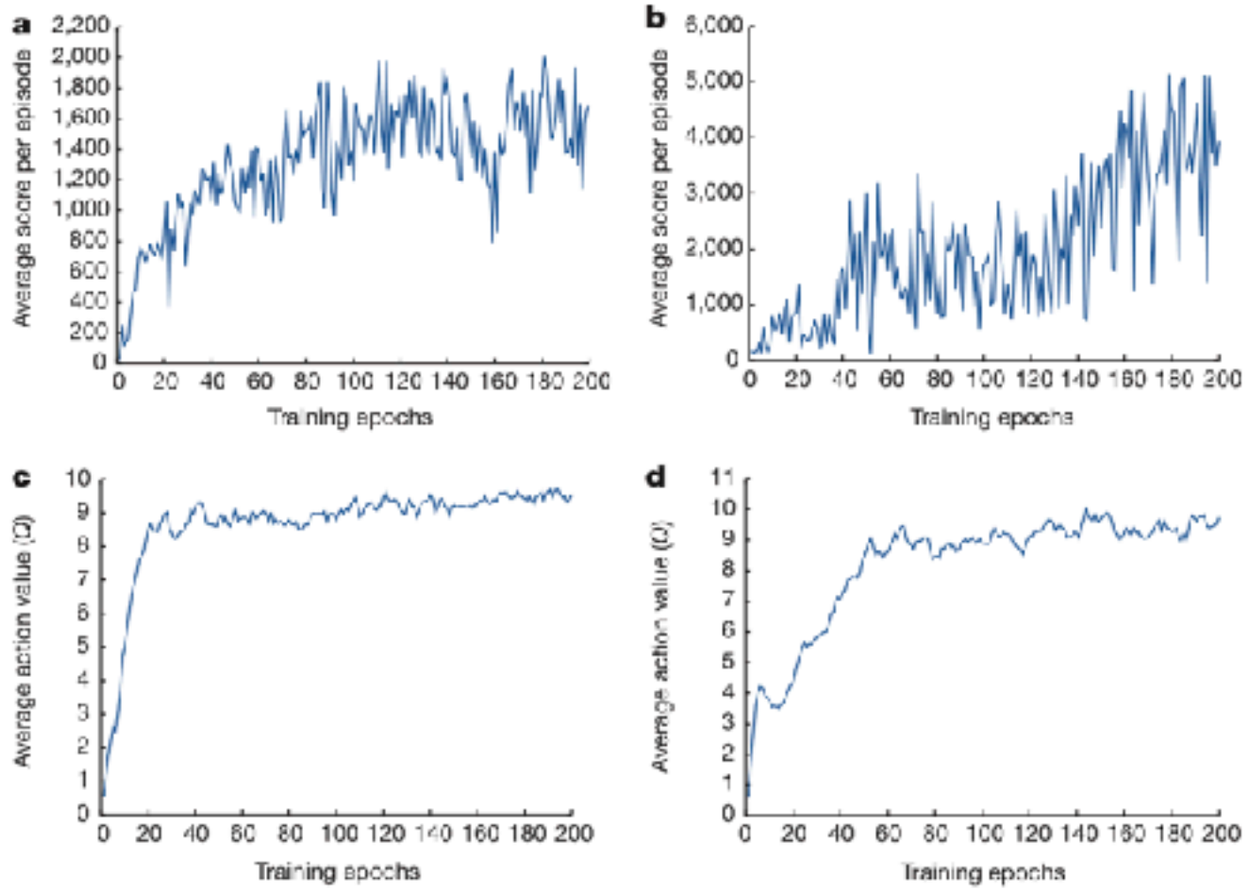
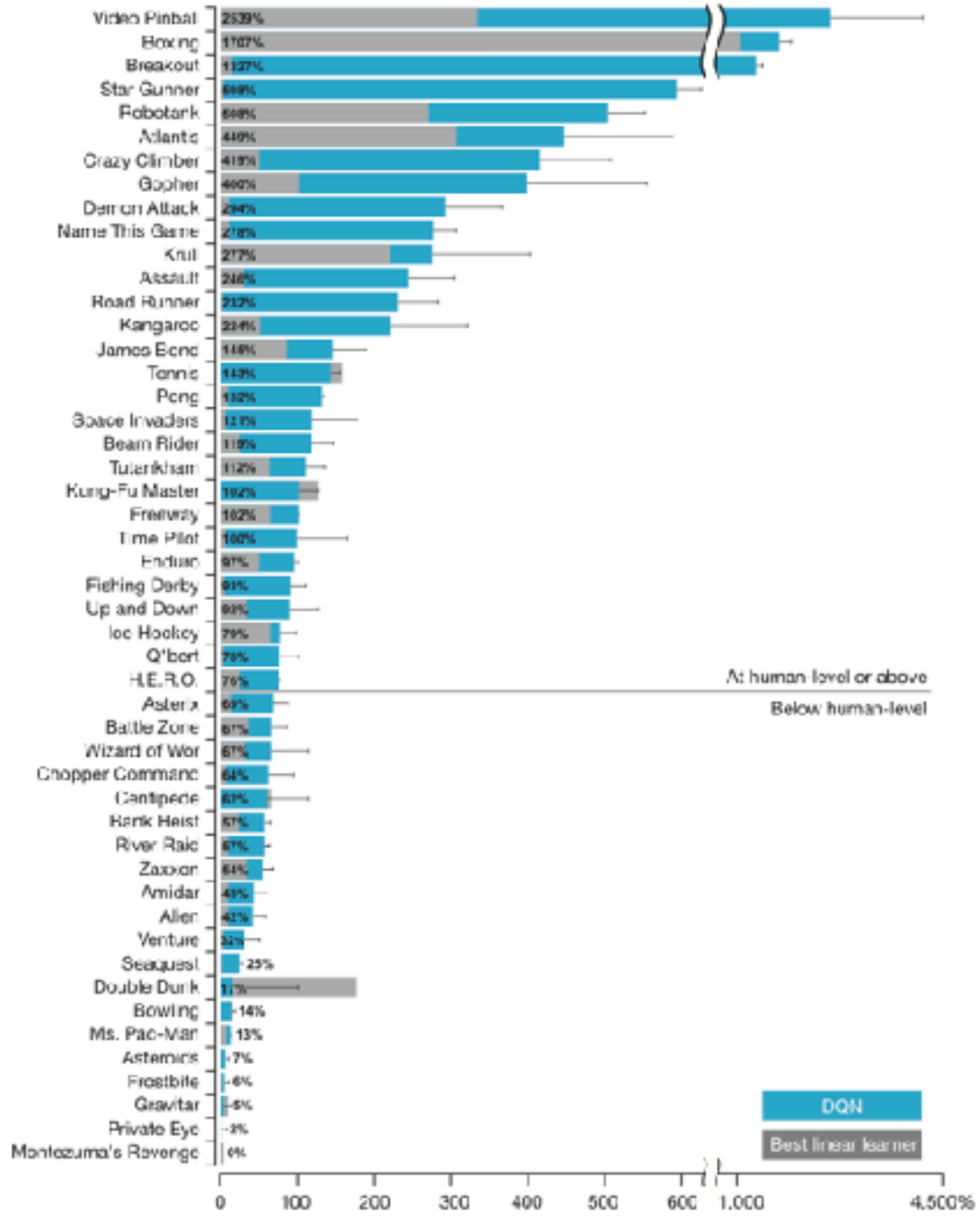


Figure 2 | Training curves tracking the agent's average score and average predicted action-value. **a**, Each point is the average score achieved per episode after the agent is run with ε-greedy policy ($\varepsilon = 0.05$) for 520 k frames on Space Invaders. **b**, Average score achieved per episode for Seaquest. **c**, Average predicted action-value on a held-out set of states on Space Invaders. Each point on the curve is the average of the action-value Q computed over the held-out set of states. Note that Q-values are scaled due to clipping of rewards (see Methods). **d**, Average predicted action-value on Seaquest. See Supplementary Discussion for details.

이미지 출처: [4]

Experiments

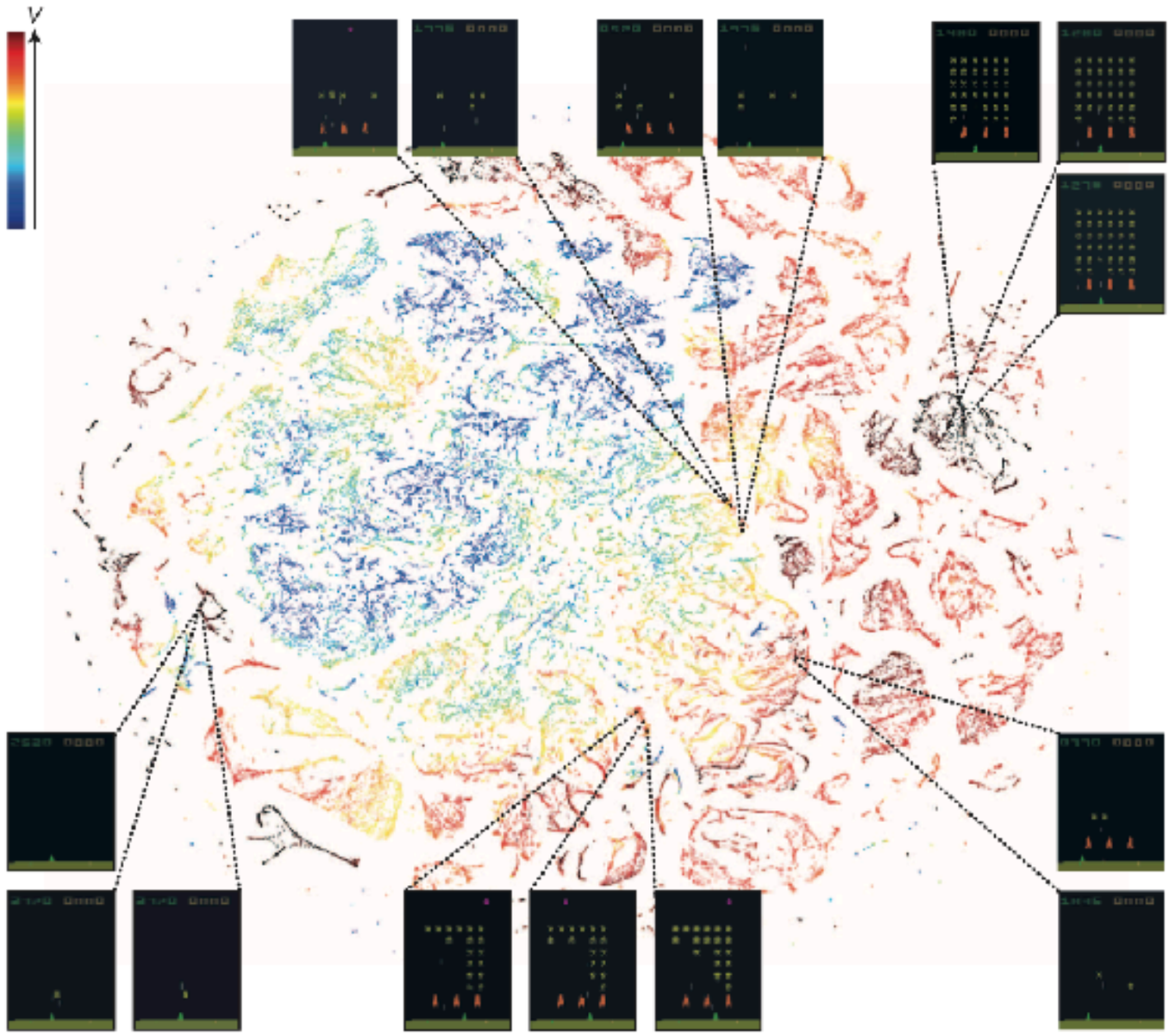
- Professional human games tester와 random play 그리고 DQN의 성능비교 표. 49개의 게임중 75%에 해당하는 29개의 게임에서 인간의 퍼포먼스를 상회한다.



이미지 출처: [4]

Experiments

- 아래 그림은 expected reward가 엇비슷한 서로 다른 states에 대해, 네트워크가 상당히 유사한 representation을 나타냄을 보인다.



이미지 출처: [4]

Demo: DQN

https://github.com/Curt-Park/KIAS_RL_Basics

References

1. Sutton, R. and Barto, A. (2018). Reinforcement Learning: An Introduction. 2nd ed. MIT Press
2. Sutton, R. and Barto, A. (2017). Reinforcement Learning: An Introduction. 2nd ed. MIT Press
3. Ethan D. et al. (2016). Exploring Deep Reinforcement Learning with Multi Q-Learning. Intelligent Control and Automation. Vol.07 No.04. Article ID:72002.
4. Mnih, V., Kavukcuoglu, K., Silver, D. et al. (2015). Human-level control through deep reinforcement learning. Nature, 518 (7540), pp. 529-533.
5. Mnih, V., Kavukcuoglu, K., Silver, D. et al. (2015). Human-level control through deep reinforcement learning. [Code]. Available at: https://sites.google.com/a/deepmind.com/dqn[Accessed 18 May. 2018]
6. Silver, D. (2015). Lecture 6: Value Function Approximation. [Video]. Available at: https://youtu.be/UoPei5o4fps [Accessed 17 May. 2018].
7. Kim, S. (2017). Lecture 7: DQN. [Video]. Available at: https://youtu.be/S1Y9eys2bdg [Accessed 17 May. 2018].
8. Kim, S. (2017). PR-005: Playing Atari with Deep Reinforcement Learning (NIPS 2013 Deep Learning Workshop). [Video]. Available at: https://youtu.be/V7_cNTfm2i8 [Accessed 17 May. 2018].
9. Seita, D. (2016). Frame Skipping and Pre-Processing for Deep Q-Networks on Atari 2600 Games. [Online]. Available at: https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games [Accessed 18 May. 2018].
10. Boyd, S. and Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press, p. 299.
11. Karpathy, A. et al. (2016). A bug in the implementation. [Online] Available at: https://github.com/devsisters/DQN-tensorflow/issues/16 [Accessed 18 May. 2018].

감사합니다.