Week 13

Introduction to Reinforcement Learning

Prof. Ph.D. Woo Youn Kim

Chemistry, KAIST

Goals-modified

12주	주제	Molecular generative model 2
	목표	Understanding difference between GAN and VAE
	내용	GAN, ARAE ARAE: conditional molecular design
13주	주제	Reinforcement learning
	목표	Understanding key principle of deep reinforcement learning
	내용	Bellman equation, Deep Q-learning
14주	주제	No lecture (entrance interview)
	목표	
	내용	

KAIST

Source

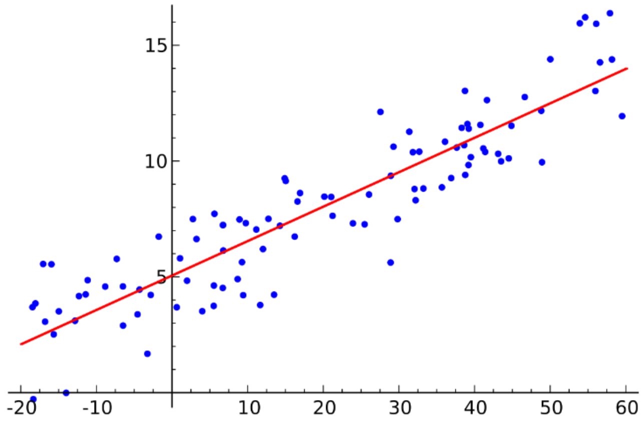
모두를 위한 딥러닝
https://hunkim.github.io/ml/

KAIST EE Jinwoo Shin
http://alinlab.kaist.ac.kr/ee807_2018.html

Types of deep learning

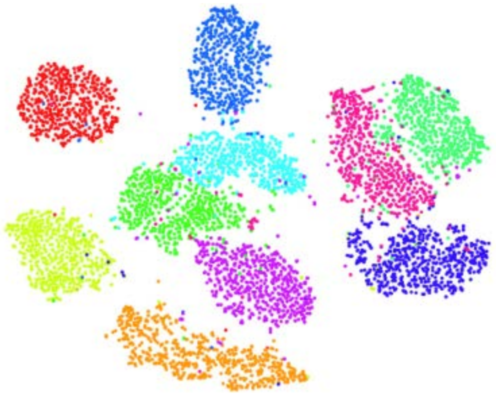
- **Supervised Learning: classification or regression**

  The network makes its guesses, then compare its answers to the known “correct” ones and make adjustments according to its errors.



- **Unsupervised Learning: clustering**

  Searching for a hidden pattern in a data set without known answers.



no labeling

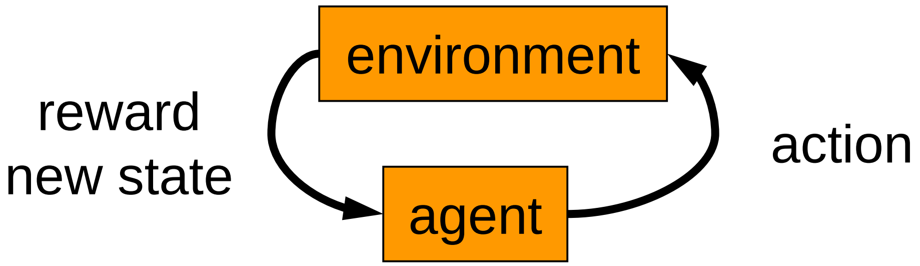
- **Reinforcement Learning: game, robotics, finance, etc.**

  A strategy built on observation.

https://www.youtube.com/watch?v=JFJkpVWTQVM

Reinforcement learning

- More general than supervised/unsupervised learning
- Learn from interaction w/ environment to achieve a goal



Alphago

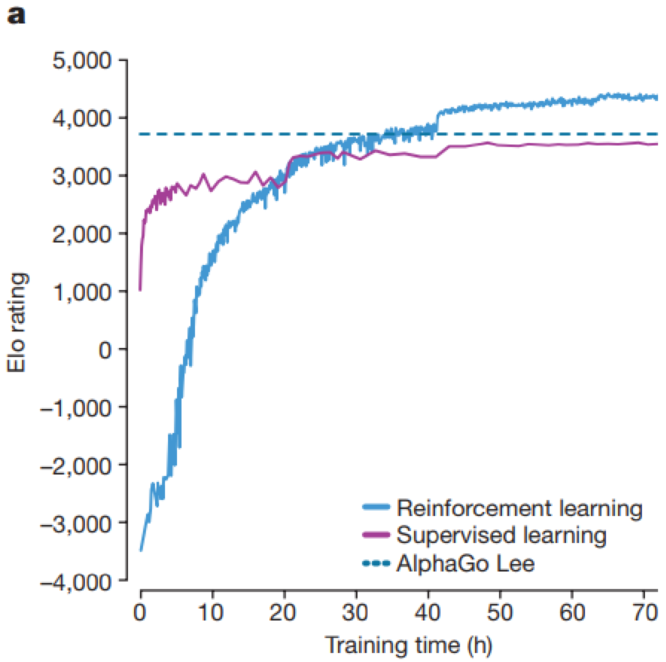
ARTICLE

doi:10.1038/nature24270

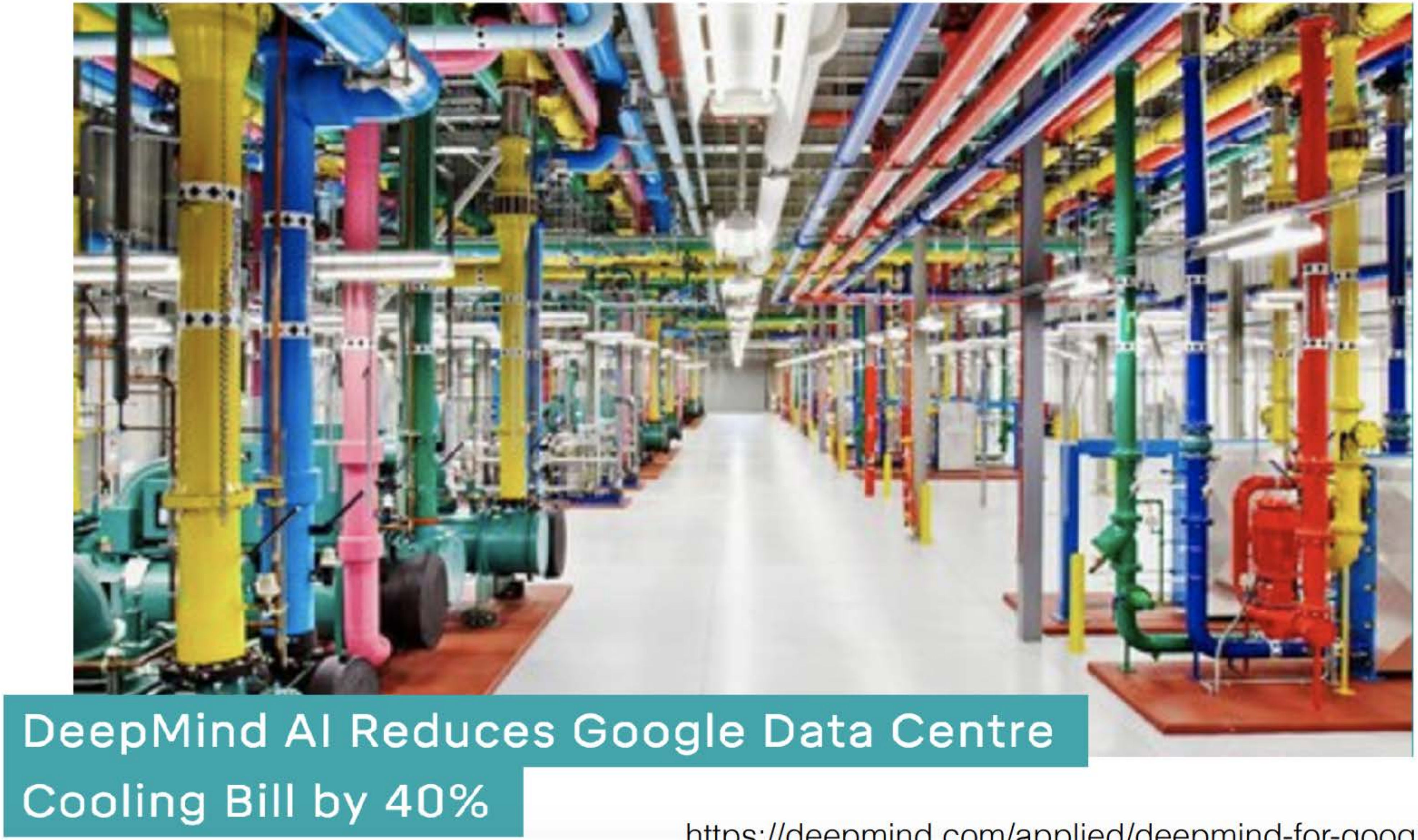
Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.



RL in Reality



https://deepmind.com/applied/deepmind-for-google/

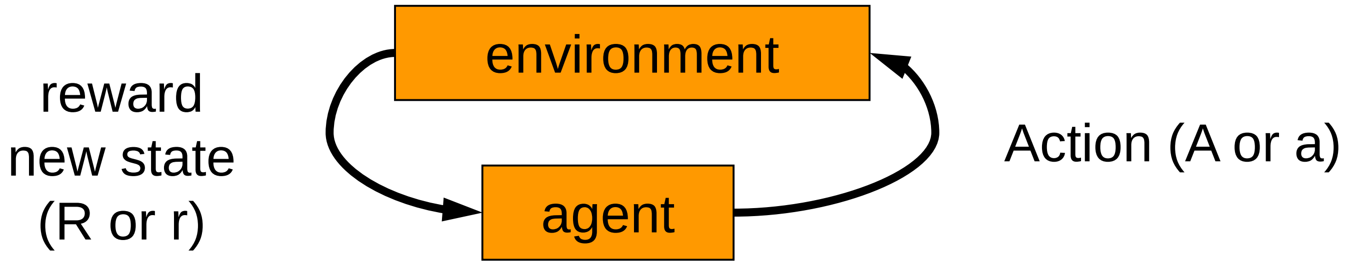
Contents

Markov Decision Process (MDP)

Markov Decision Process (MDP)

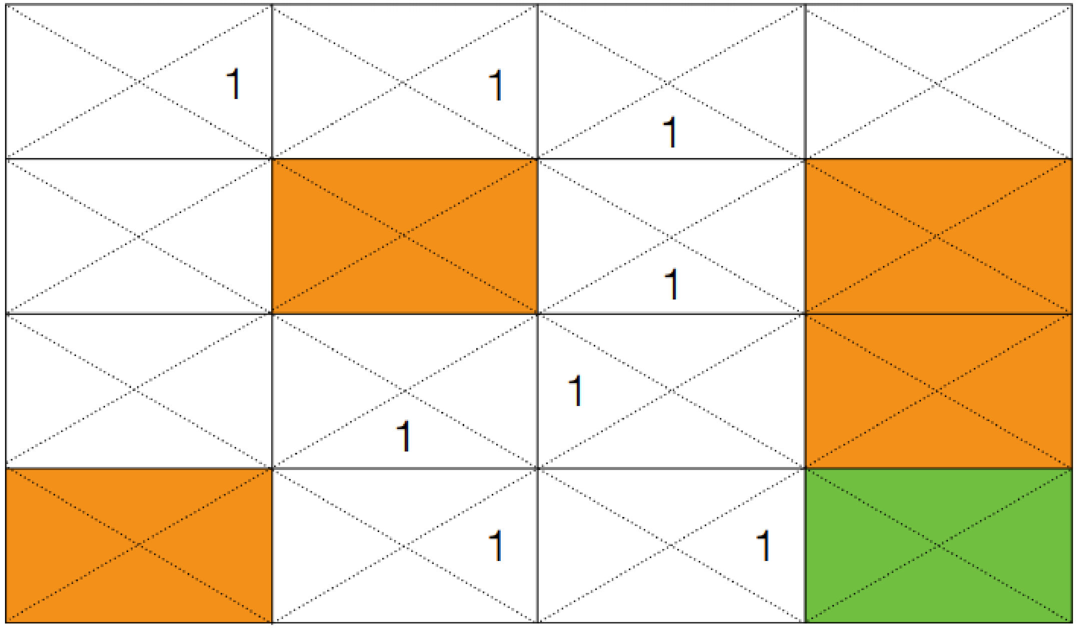
- RL can be formulated by **Markov Decision Process (MDP)**
  - $\mathcal{S}$: a set of states
  - $\mathcal{A}$: a set of actions
  - $\mathcal{P}$: a conditional state transition probability, i.e.,
    $\mathcal{P}(s_t, a_t, s_{t+1}) = \Pr(s_{t+1}|s_t, a_t) = \Pr(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_1, a_1)$
  - $\mathcal{R}$: a reward function, i.e., $r_t = \mathcal{R}(s_t, a_t)$
  - $\gamma \in [0, 1]$: a discount factor

- The agent chooses an action according to policy $\pi(a|s)$



- **Goal:** find optimal policy maximizing **total future reward**

Toy model



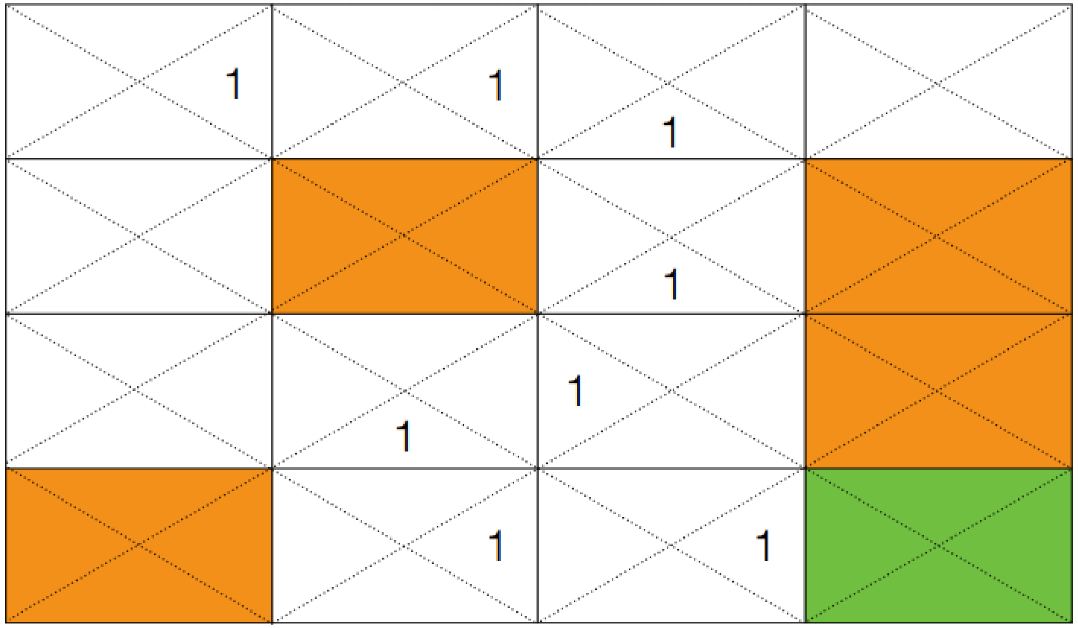
- actions: UP, DOWN, LEFT, RIGHT, UP

- 25% move UP
- 25% move LEFT
- 25% move RIGHT
- 25% move DOWN

- reward +1 at green
- orange is not allowed

- What’s the strategy to achieve max reward?

Toy model



- actions: UP, DOWN, LEFT, RIGHT, UP

- 25% move UP
- 25% move LEFT
- 25% move RIGHT
- 25% move DOWN

- reward +1 at green
- orange is not allowed

- set of states S, set of actions A, initial state S_0, transition model P(s,a,s')
  ex) P([1,1], right, [1,2]) = 0.25

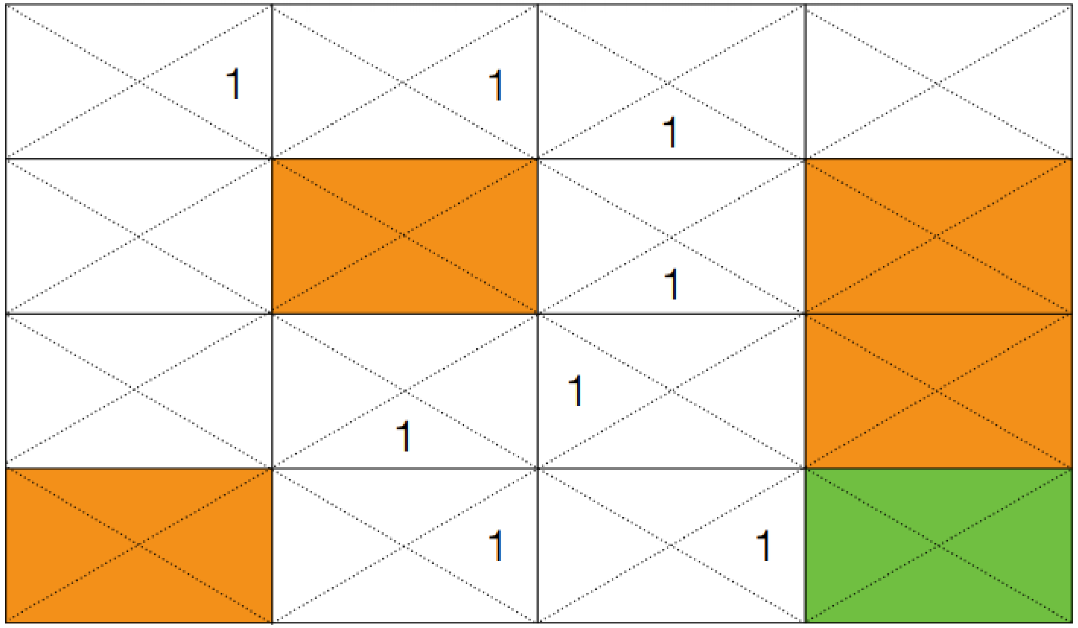
- reward function r(s): r([4,4]) = +1

- policy: mapping from S to A: π(s,a)

- Goal: maximize cumulative reward in the long run

KAIST

Toy model



- actions: UP, DOWN, LEFT, RIGHT, UP

- 25% move UP
- 25% move LEFT
- 25% move RIGHT
- 25% move DOWN

- reward +1 at green
- orange is not allowed

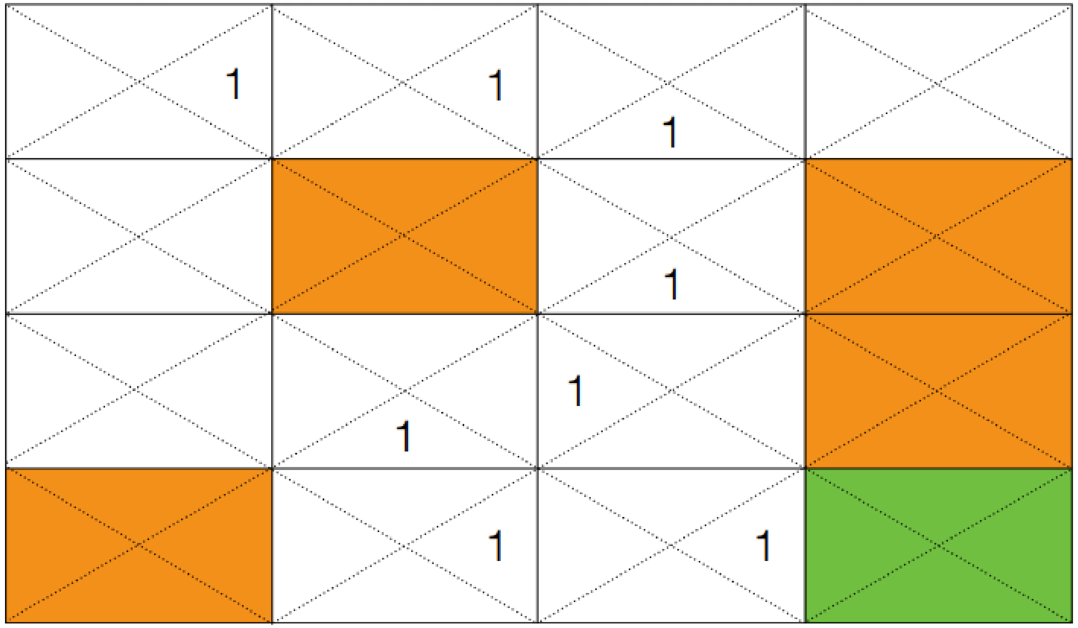
$$\underset{\text{state}}{s_0}, \underset{\text{action}}{a_0}, \underset{\text{reward}}{r_1}, s_1, a_1, r_2, \ldots, s_{n-1}, a_{n-1}, r_n, \underset{\text{Terminal state}}{s_n}$$

$$R = r_1 + r_2 + r_3 + \cdots + r_n$$

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

KAIST

Toy model



- actions: UP, DOWN, LEFT, RIGHT, UP

- 25% move UP
- 25% move LEFT
- 25% move RIGHT
- 25% move DOWN

- reward +1 at green
- orange is not allowed

- additive rewards
  - ✓ $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - ✓ infinite value for continuing tasks
- discounted rewards
  - ✓ $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  - ✓ value bounded if rewards bounded
  - ✓ prefer larger immediate rewards

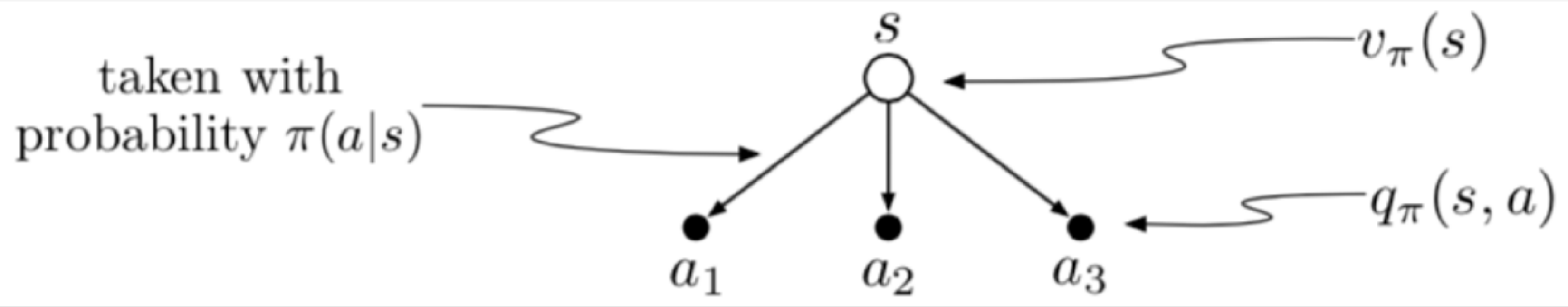
Value functions

- Value functions of a state under a policy π(a|s)
  - State-value function: expected return when starting in *s* and following π

$$v_\pi(s) = \mathbb{E}_{a_1,\ldots\sim\pi}\left[\sum_{t=1}^{\infty}\gamma^{t-1}r_t | s_1 = s\right]$$

  - Action-value function: expected return when starting in *s*, performing *a*, and following π

$$q_\pi(s,a) = \mathbb{E}_{a_2,\ldots\sim\pi}\left[\sum_{t=1}^{\infty}\gamma^{t-1}r_t | s_1 = s, a_1 = a\right]$$



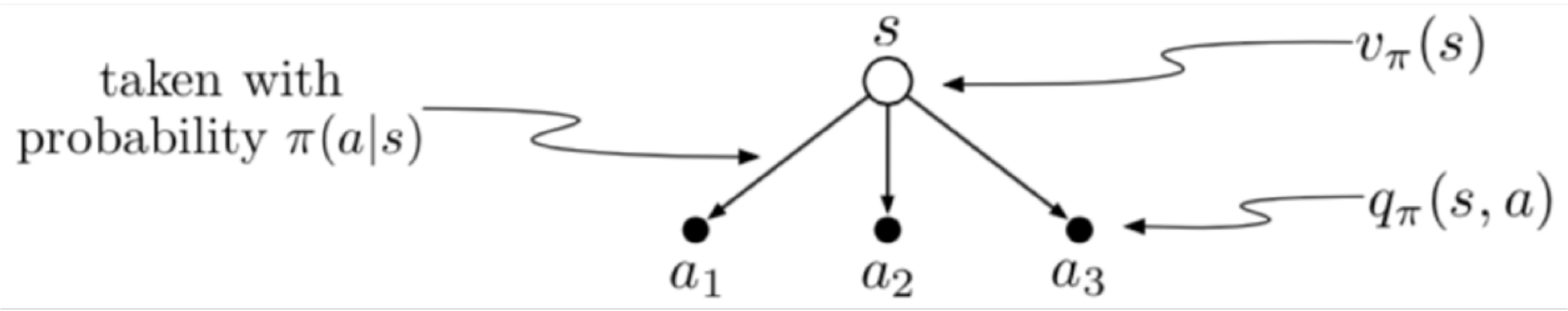
Bellman equation

- Recursion formula by value iteration algorithm

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s') \qquad v_\pi(s) = \sum_a \pi(s,a) q_\pi(s,a)$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_a \pi(s,a) q_\pi(s,a) \qquad v_\pi(s) = \sum_a \pi(s,a) \left[R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s) \right]$$

Bellman (expectation) equation



Optimal value function

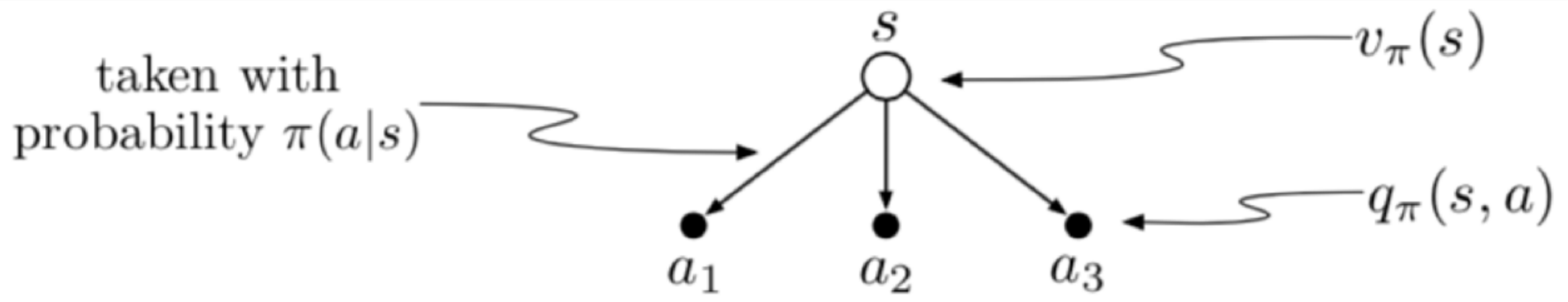
- Optimal value functions:

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

- The optimal policy can be derived from them

$$\pi_*(s) = \arg\max_a q_*(s,a)$$

to achieve our goal: find optimal policy maximizing **total future reward**



Bellman's optimality equation

- Recursion formula by value iteration algorithm

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a v_*(s') \qquad v_*(s) = \max_a q_\pi(s,a)$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \left(\max_a q_*(s,a)\right) \qquad v_*(s) = \max_a \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a v_*(s) \right)$$

Types of RL

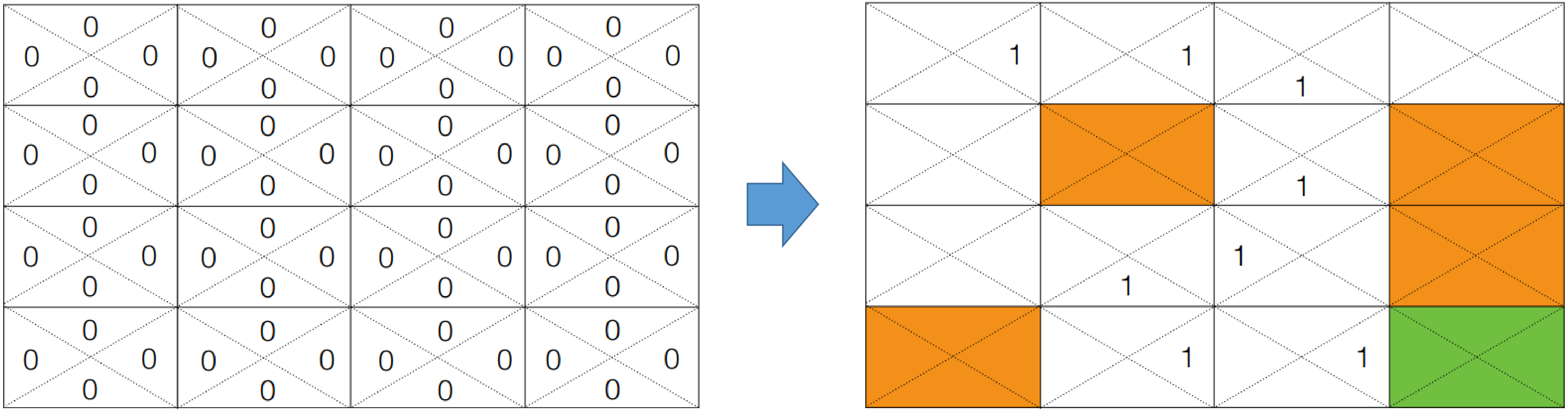
- Value-based *vs.* policy-based algorithms
  - Value-based learns value functions, and then derive policy ➔ Q-learning
  - Policy-based optimizes policy directly from the objective, i.e., $\mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right]$

    ➔ Policy gradient method

Q-learning

Initialize q(s,a)

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \left(\max_a q_*(s,a) \right)$$



Q-learning algorithm

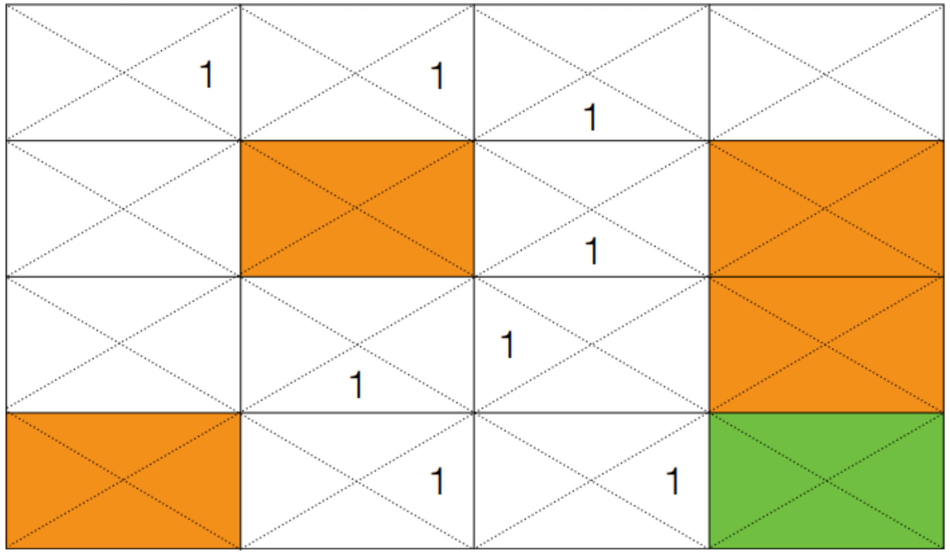
- **Q-learning algorithm** repeats 1-3 until convergence

1. Choose an action from the current state s

2. Observe a reward *R*, a new state s'

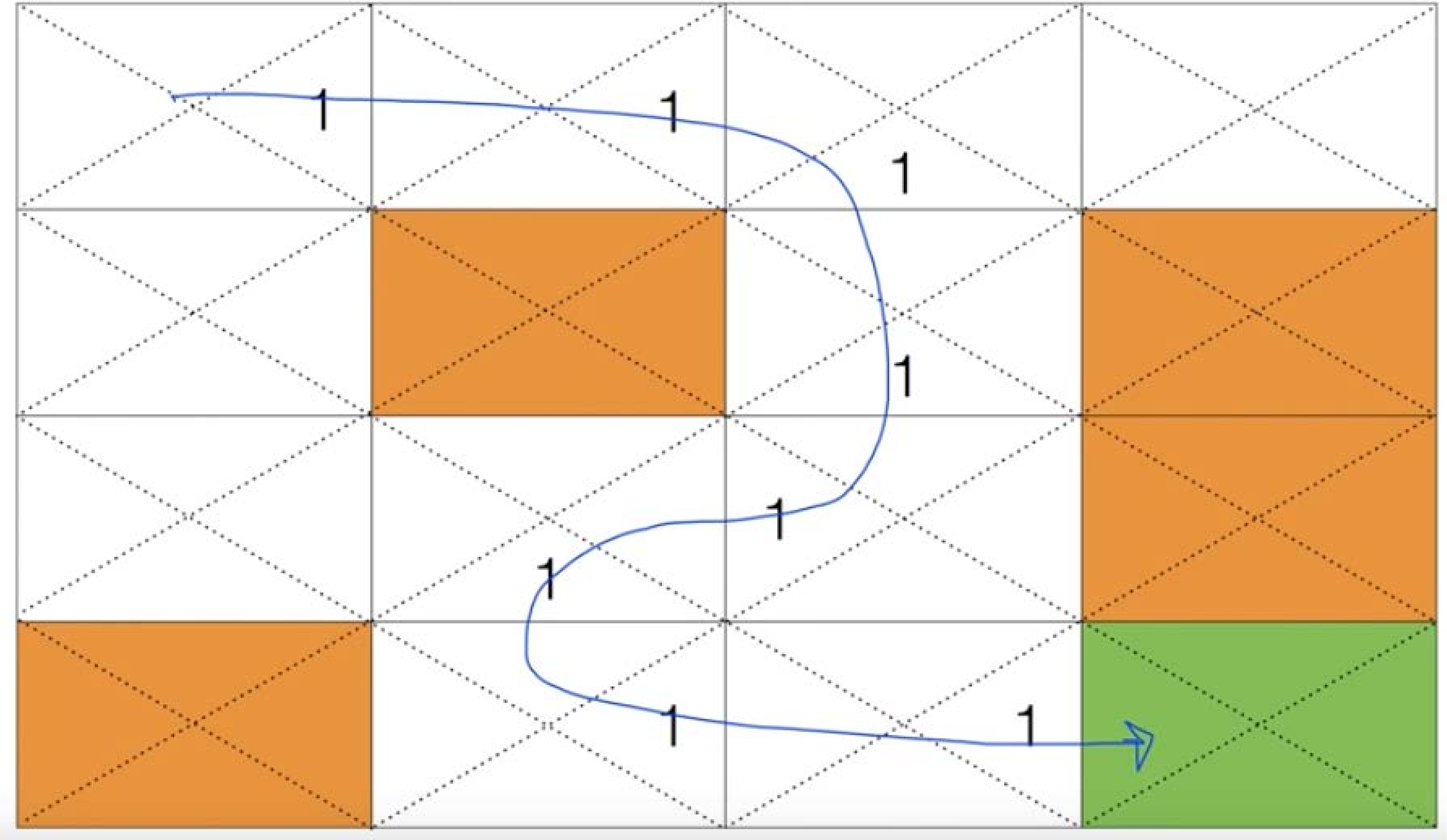
3. Update

$$q_{i+1}(s,a) \leftarrow q_i(s,a) + \underset{\substack{\uparrow \\ \text{learning rate}}}{\alpha} \left[R_s^a + \max_{a'} q_{i+1}(s',a') - q_i(s,a) \right]$$



How to choose the action a?

Exploit vs Exploration



https://www.youtube.com/watch?v=MQ-3QScrFSI&feature=youtu.be

Exploit vs Exploration: ε-greedy

$\varepsilon = 0.1$

if rand(1) < ε :

a = random

else:

$$a = \arg\max_a q(s, a)$$

https://www.youtube.com/watch?v=MQ-3QScrFSI&feature=youtu.be

KAIST

Exploit vs Exploration: decaying ε-greedy

for i = 0, 1000

ε = 0.1 / (i+1)

if rand(1) < ε :

a = random

else:

$$a = \arg\max_a q(s, a)$$

https://www.youtube.com/watch?v=MQ-3QScrFSI&feature=youtu.be

Exploit vs Exploration: random noise

for i = 0, 1000

$$a = \arg\max_a \left(q(s,a) + \mathrm{random}(1)/(i+1) \right)$$

Compared to ε-greedy, this method is likely to take an action with larger values because of q(s,a)

https://www.youtube.com/watch?v=MQ-3QScrFSI&feature=youtu.be

KAIST

Q-learning algorithm

- **Q-learning algorithm** repeats 1-3 until convergence

1. Choose an action from the current state s

   using exploit vs explore algorithm

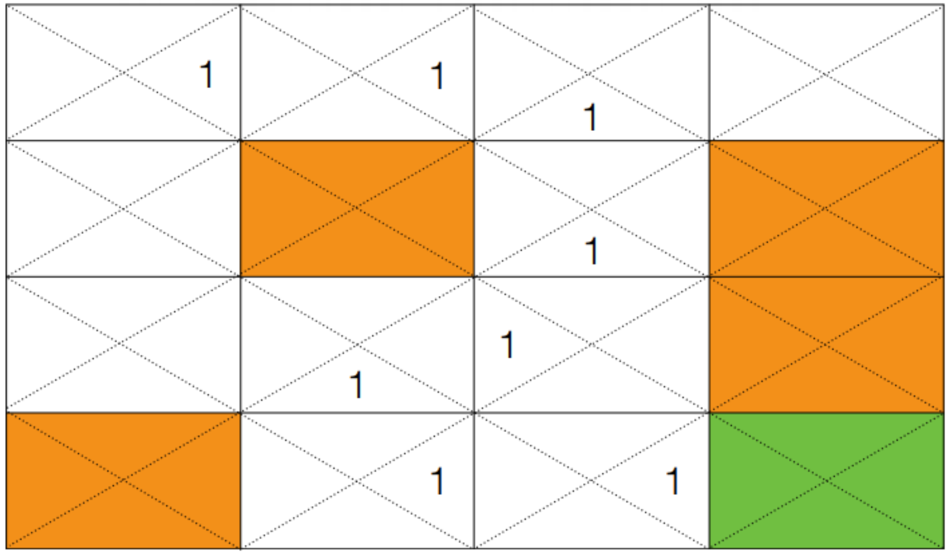
2. Observe a reward *R*, a new state s'

3. Update

$$q_{i+1}(s,a) \leftarrow q_i(s,a) + \alpha \left[R_s^a + \max_{a'} q_{i+1}(s',a') - q_i(s,a) \right]$$

learning rate

Incrementally update the q function



KAIST

Convergence

- Q-learning updates the q-value incrementally to satisfy the Bellman equation for the optimal action-value function

$$q_*(s,a) = \mathrm{E}_{s' \sim Pr(\cdot|s,a)}\left[R_s^a + \gamma \max_a q_*(s,a) \right]$$

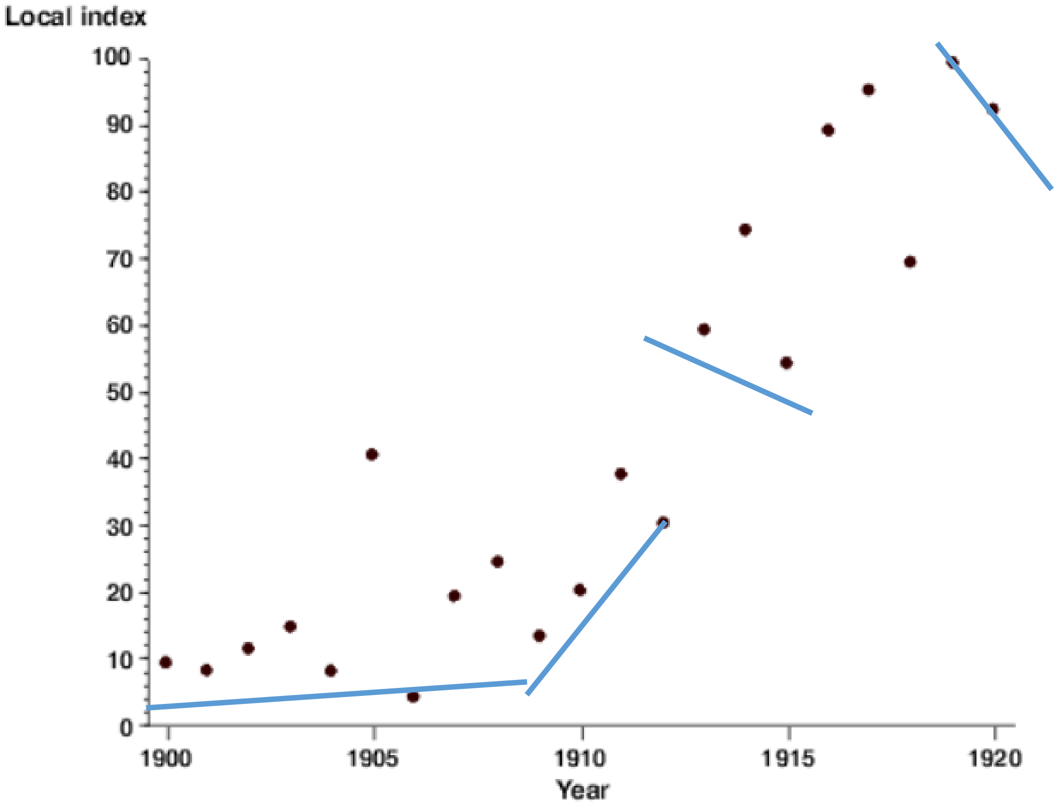
- In principle, it is known that q or v converge to their optimal values by the value iteration algorithm

$$\lim_{i \to \infty} q_i(s,a) \to q_*(s,a)$$

Limitation

- Q-learning is known to be unstable or even to diverge when using nonlinear function approximators such as neural networks because even small updates to may significantly change.

1. Correlations between samples



2. Non-stationary targets

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$

$$\hat{Y} = \hat{Q}(s_t, a_t|\theta) \qquad Y = r_t + \gamma \max_{a'} \hat{Q}_\theta(s_{t+1}, a'|\theta)$$

target q　　　　pred. q

➔ Solution: deep Q-learning

Deep Q-Network (DQN)

LETTER

Human-level control through dee
learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Ve
Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charle
Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

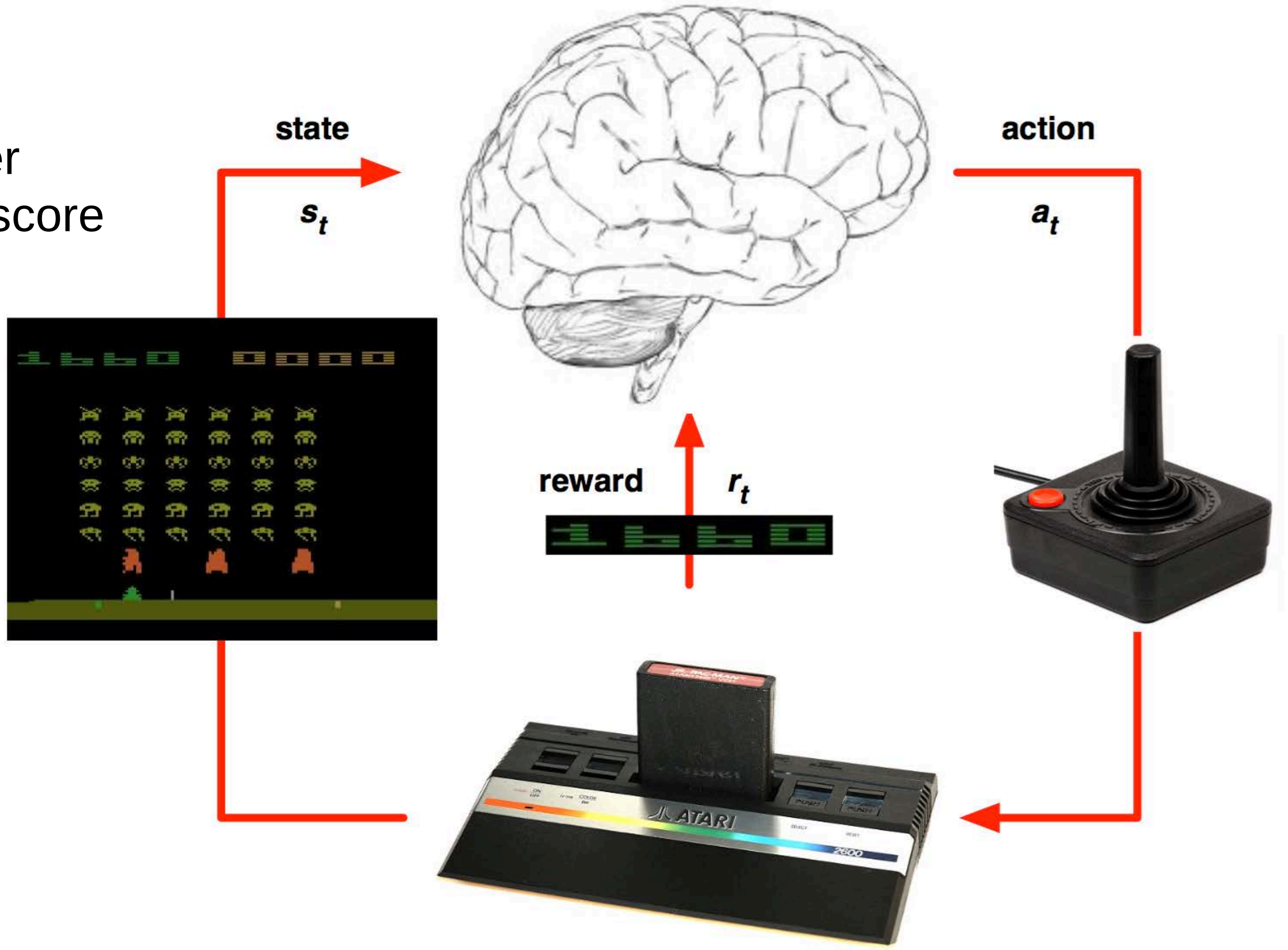
The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted

agent is to select act
reward. More forma
approximate the op

$$Q^*(s,a) = \max_{\pi}$$

Playing game

- State: position
- Action: controller
- Reward: game score



Solution

1. Deep Q-network

2. Capture and replay
   - Correlations between samples

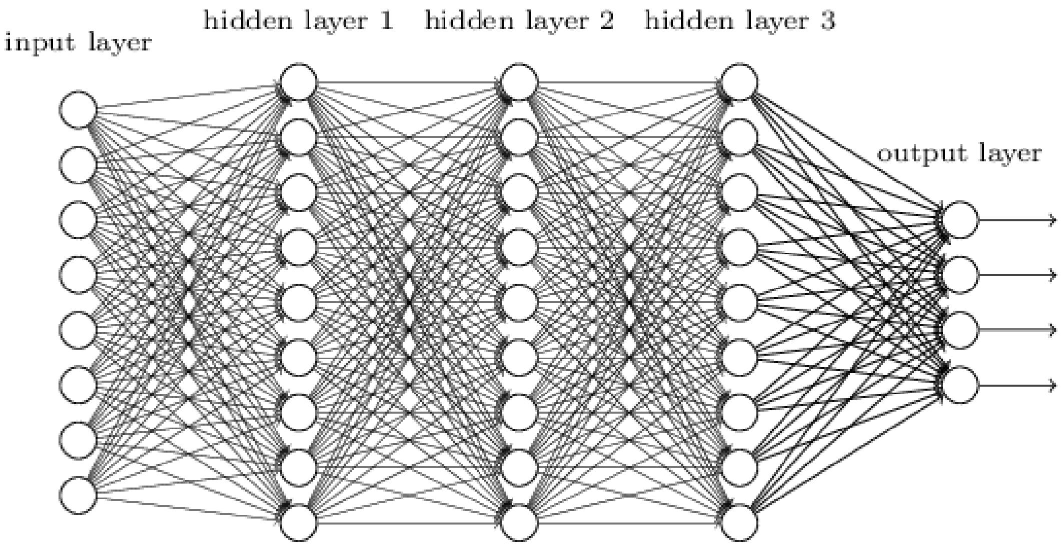
3. Separate networks: create a target network
   - Non-stationary targets

Deep Q-network

- Non-linear parameterization of q function with deep neural network (replace Q table)

$$q(s,a;\theta) \simeq q_*(s,a)$$

input
s,a



output
q value

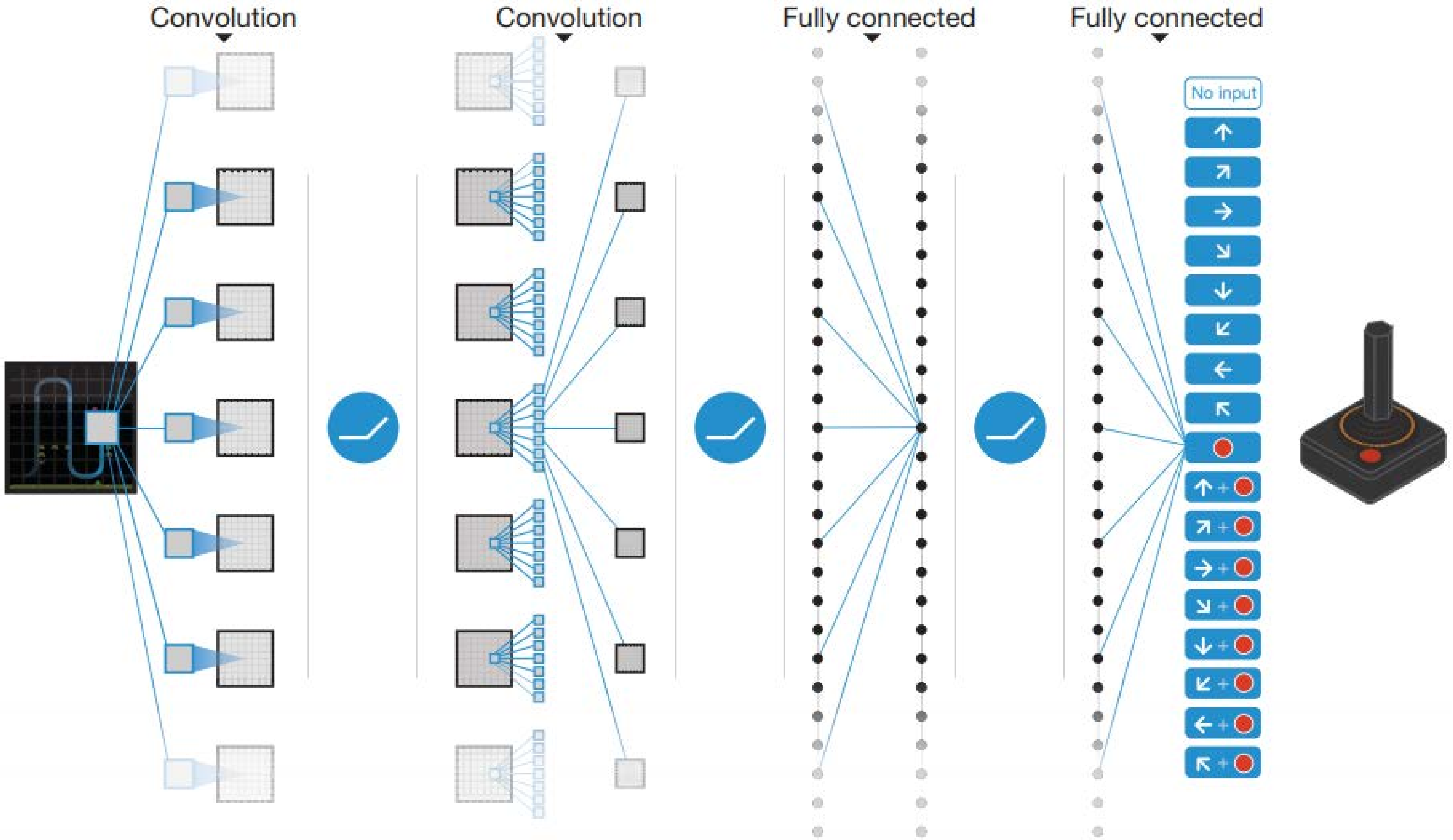
learnable parameter θ
CNN/RNN/GNN

- The update rule for θ

$$\theta \leftarrow \theta + \alpha \left[r + \gamma \max_{a'} q(s', a'; \theta) - q(s,a;\theta) \right] \nabla_\theta q(s,a;\theta)$$

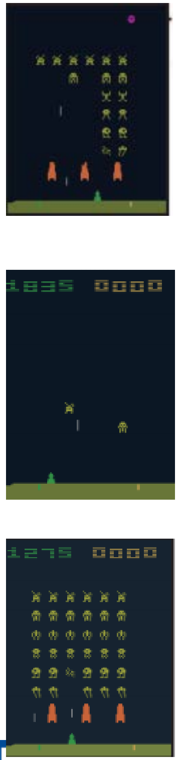
KAIST

Deep Q-network using CNN



Experience replay buffer

- use previous samples stored in buffer *D*
- smoothing data distribution
- remove sequential correlation



Capture →

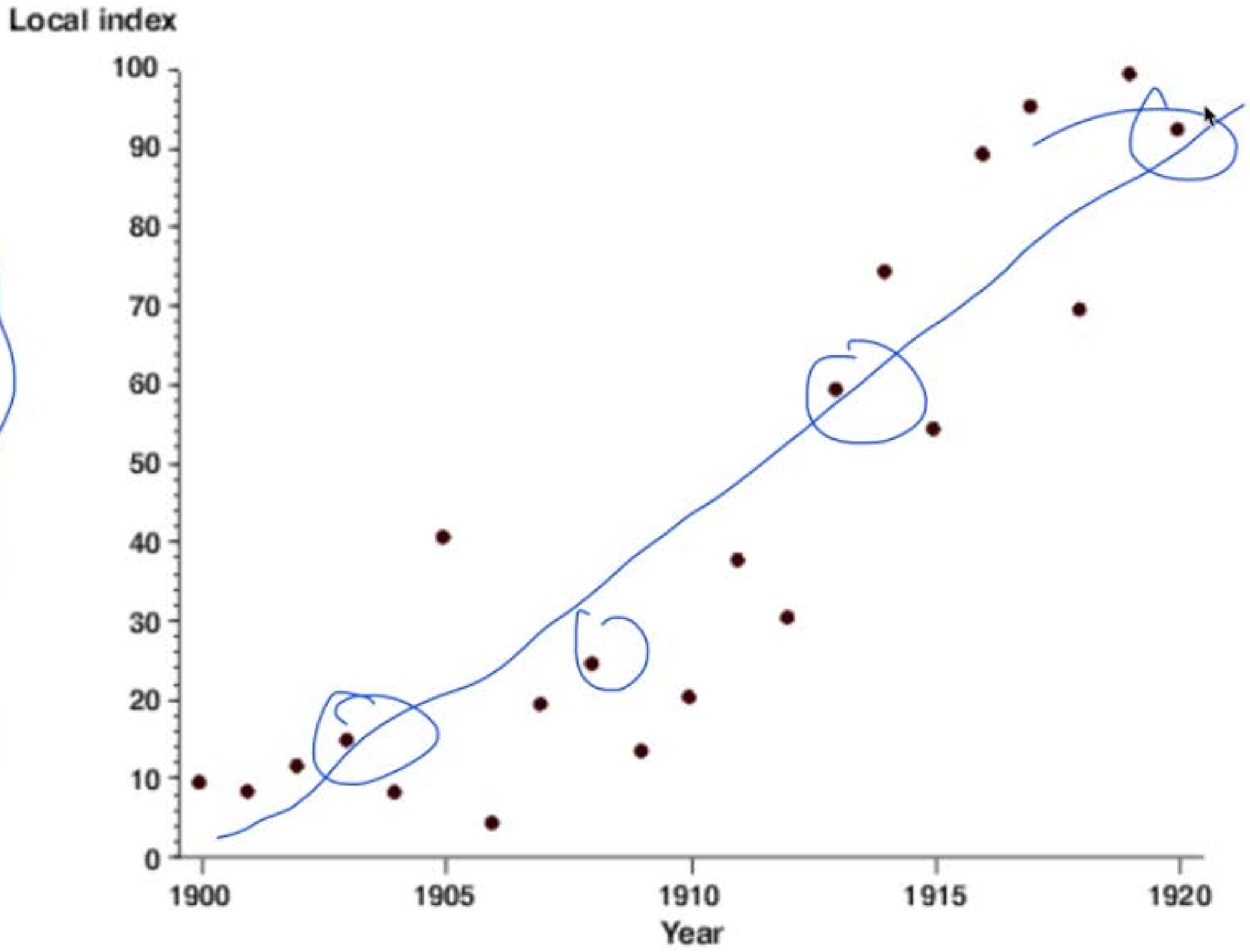
s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

random sample
& Replay →

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$

Experience replay buffer

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$



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

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

  End For

End For

KAIST

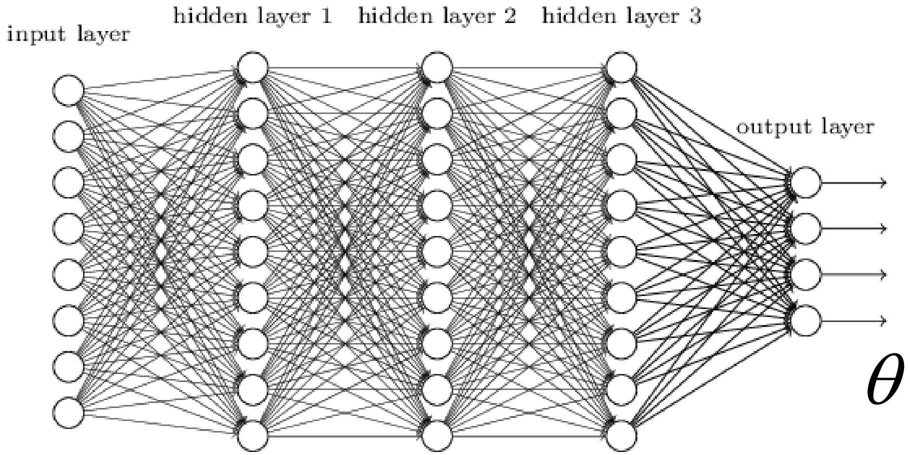
Separate networks

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\bar{\theta}))]^2$$

Every C steps reset

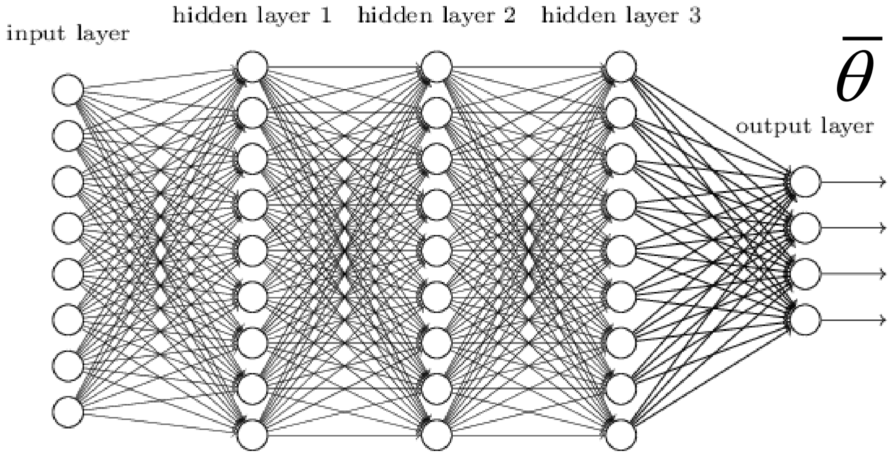
$$\hat{Q} = Q$$

input
s,a



output
target. q value

input
s,a



output
pred. q value

Algorithm 1: deep Q-learning with experience replay.

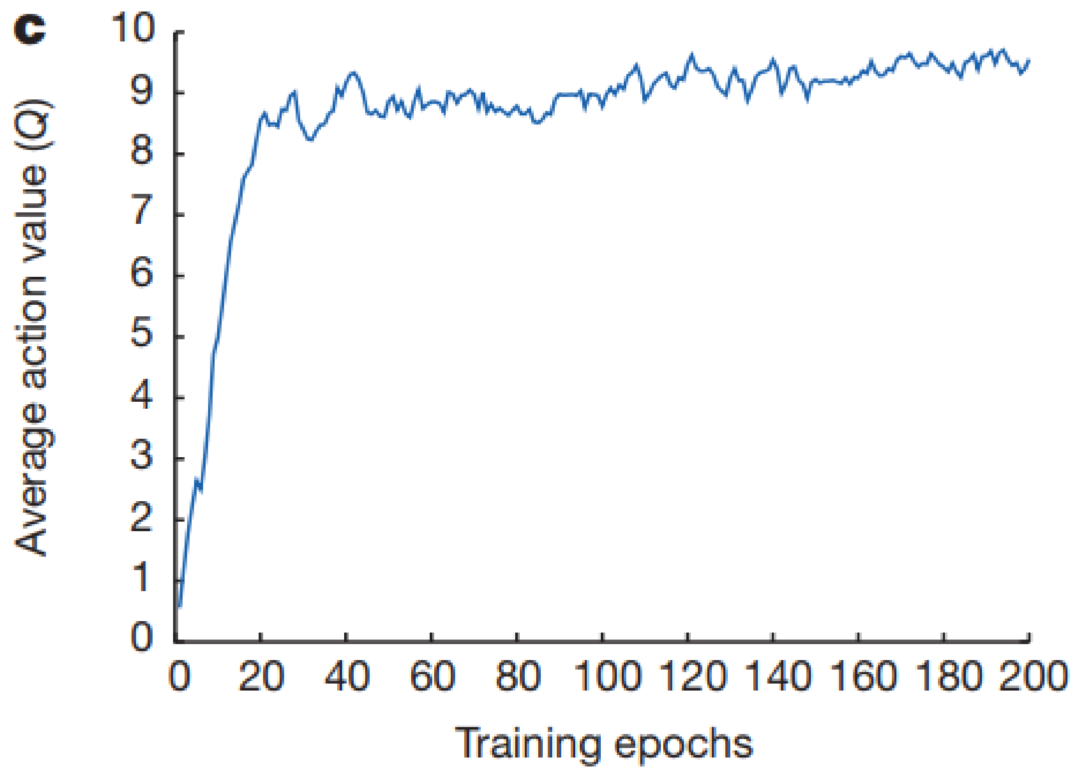
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    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

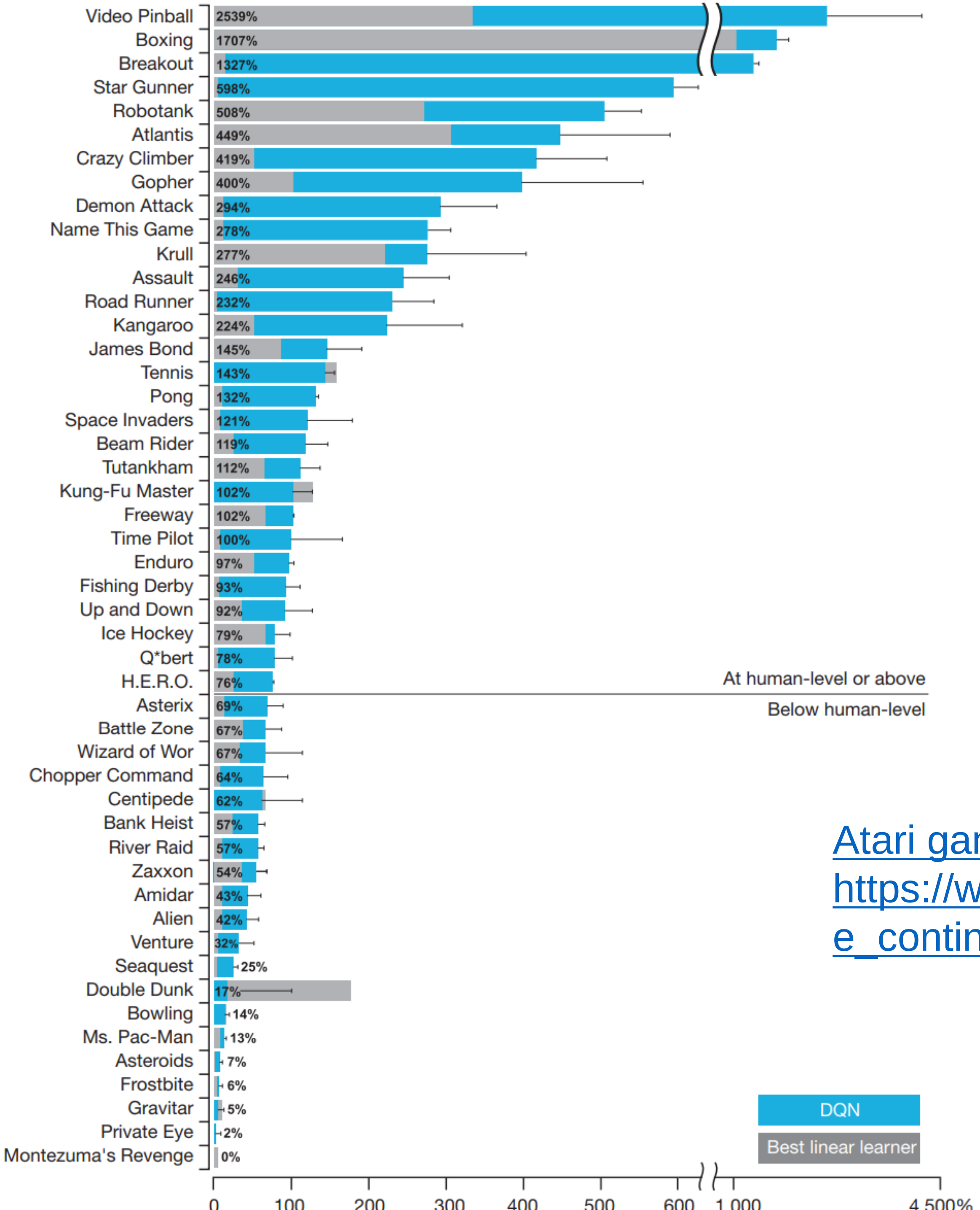
KAIST

Result

- Non-linear parameterization of q function with deep neural network

$$q(s,a;\theta) \simeq q_*(s,a)$$





[Atari game video](https://www.youtube.com/watch?time_continue=2&v=iqXKQf2BOSE)
https://www.youtube.com/watch?time_continue=2&v=iqXKQf2BOSE

Example

NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

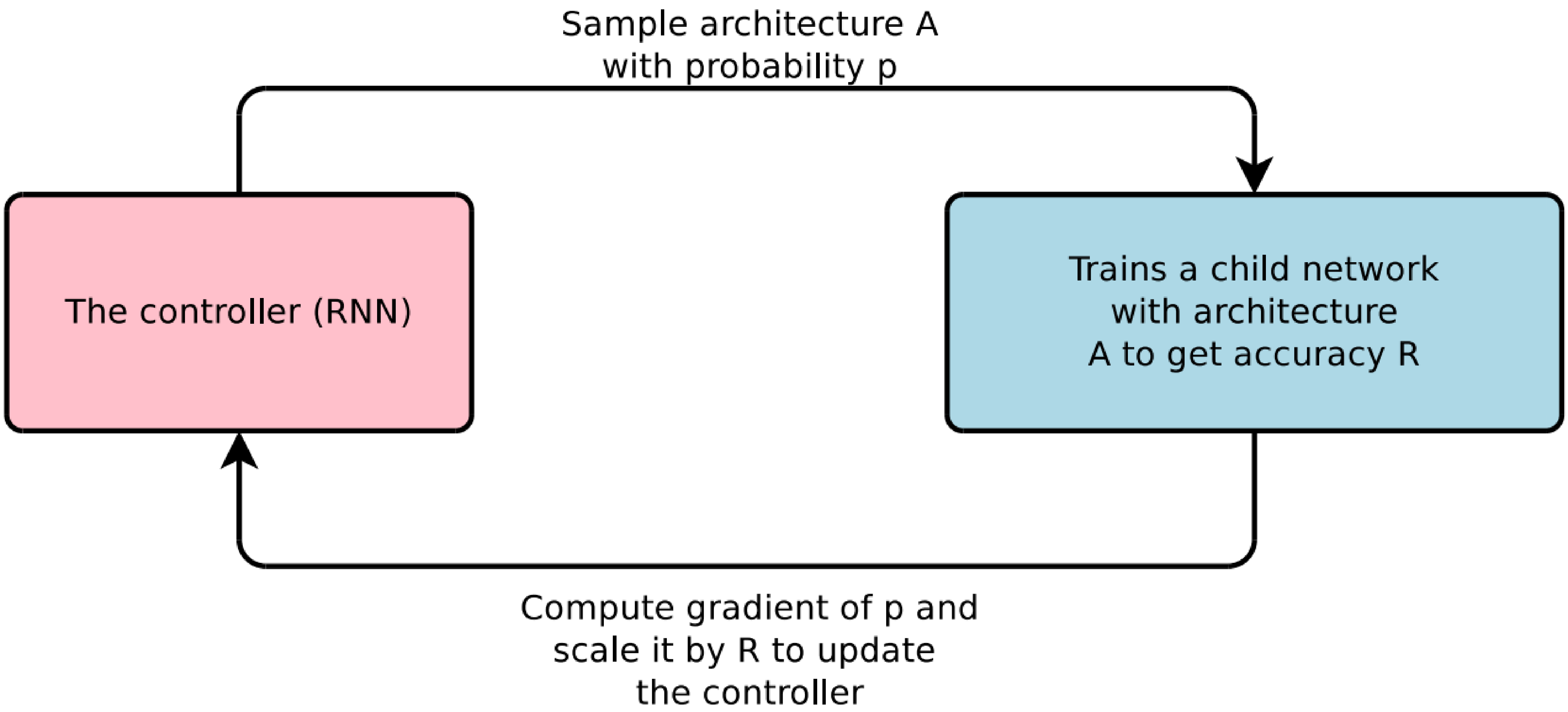
Barret Zoph,[*] Quoc V. Le
Google Brain
{barretzoph,qvl}@google.com

https://arxiv.org/abs/1611.01578

ABSTRACT

Neural networks are powerful and flexible models that work well for many difficult learning tasks in image, speech and natural language understanding. Despite their success, neural networks are still hard to design. In this paper, we use a recurrent network to generate the model descriptions of neural networks and train this RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. On the CIFAR-10 dataset, our method,

An overview of Neural Architecture Search



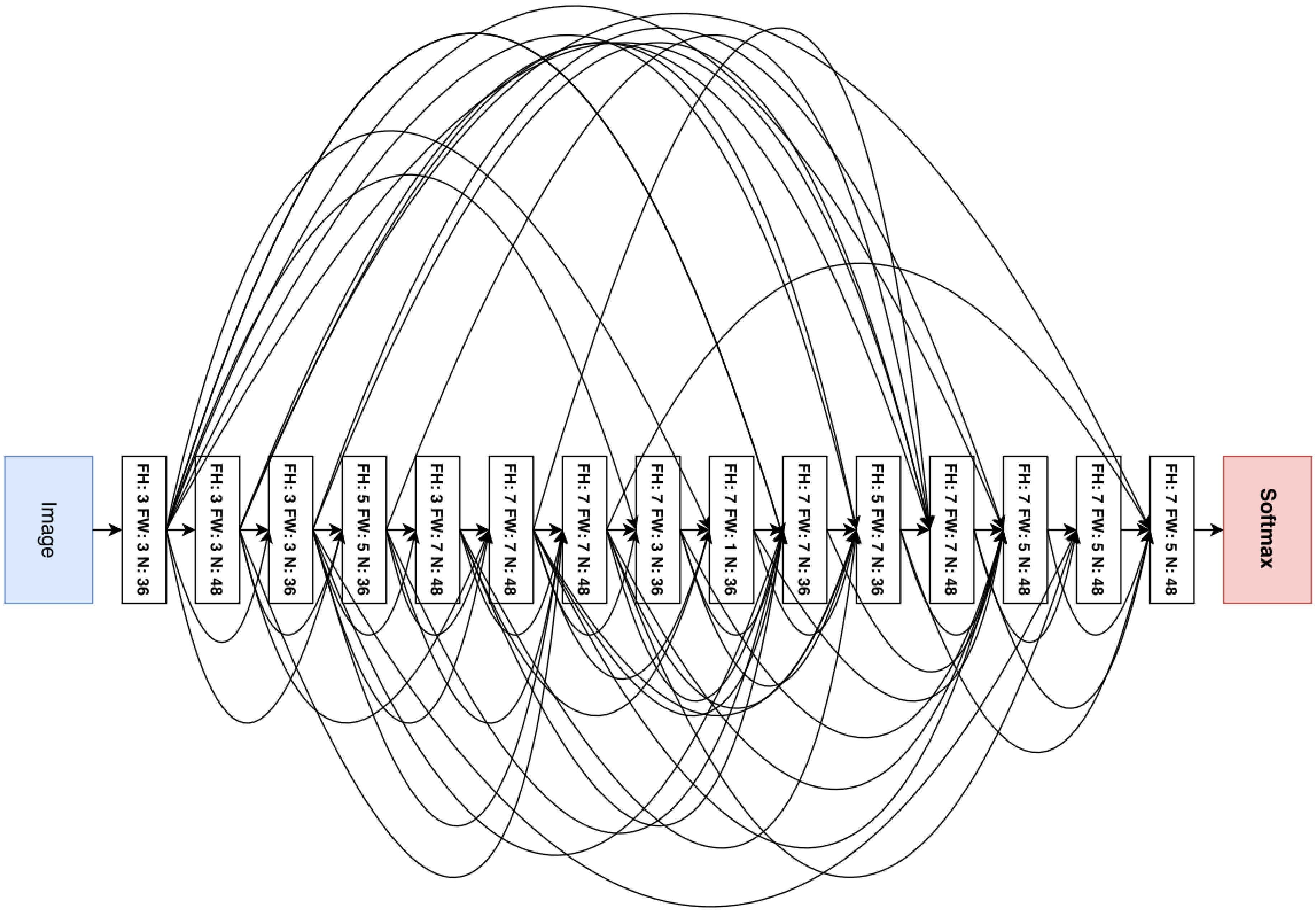


Image
FH: 3 FW: 3 N: 36
FH: 3 FW: 3 N: 48
FH: 3 FW: 3 N: 36
FH: 5 FW: 5 N: 36
FH: 3 FW: 7 N: 48
FH: 7 FW: 7 N: 48
FH: 7 FW: 7 N: 48
FH: 7 FW: 3 N: 36
FH: 7 FW: 1 N: 36
FH: 7 FW: 7 N: 36
FH: 5 FW: 7 N: 36
FH: 7 FW: 7 N: 48
FH: 7 FW: 5 N: 48
FH: 7 FW: 5 N: 48
FH: 7 FW: 5 N: 48
Softmax

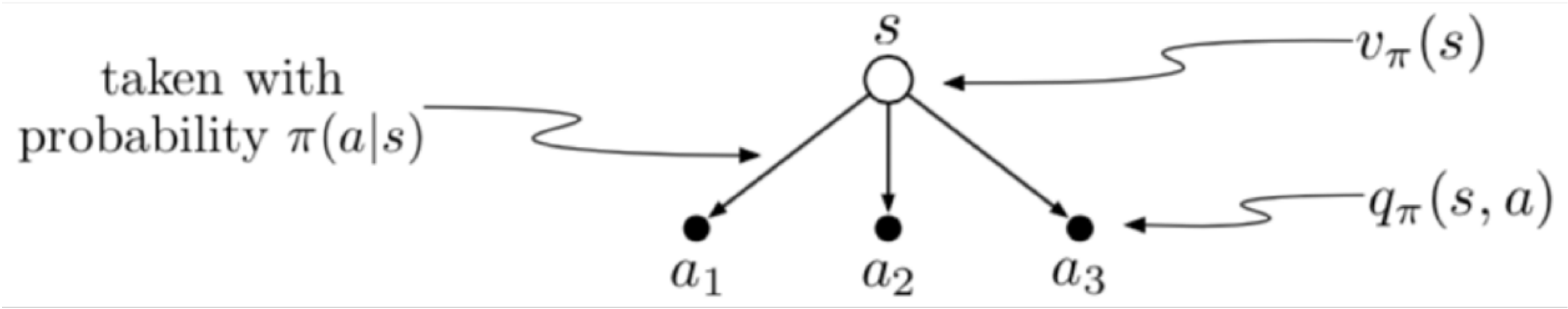
Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

Summary

- Markov decision process

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_a \pi(s,a) q_\pi(s,a) \qquad v_\pi(s) = \sum_a \pi(s,a) \left[R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s) \right]$$

Bellman (expectation) equation



- Non-linear parameterization of q function with deep neural network (replace Q table)

$$q(s,a;\theta) \simeq q_*(s,a)$$

New terms

- Markov Decision Process (MDP)
- Q-learning
- State value function
- Action value function
- Bellman equation
- Bellman's optimal equation
- Exploit vs Explore
- ε-greedy
- Policy gradient method
- Deep Q-Learning (DQN)
- Q-network
- Experience replay