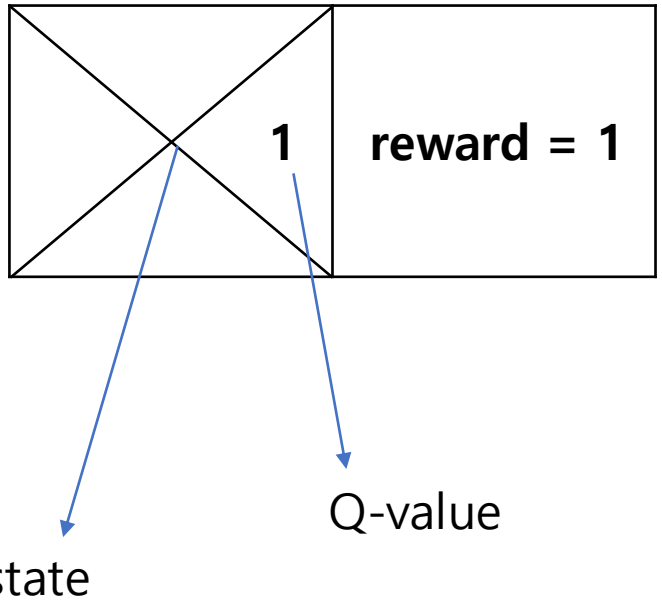
Dueling Network Architectures for Deep Reinforcement Learning

INDEX

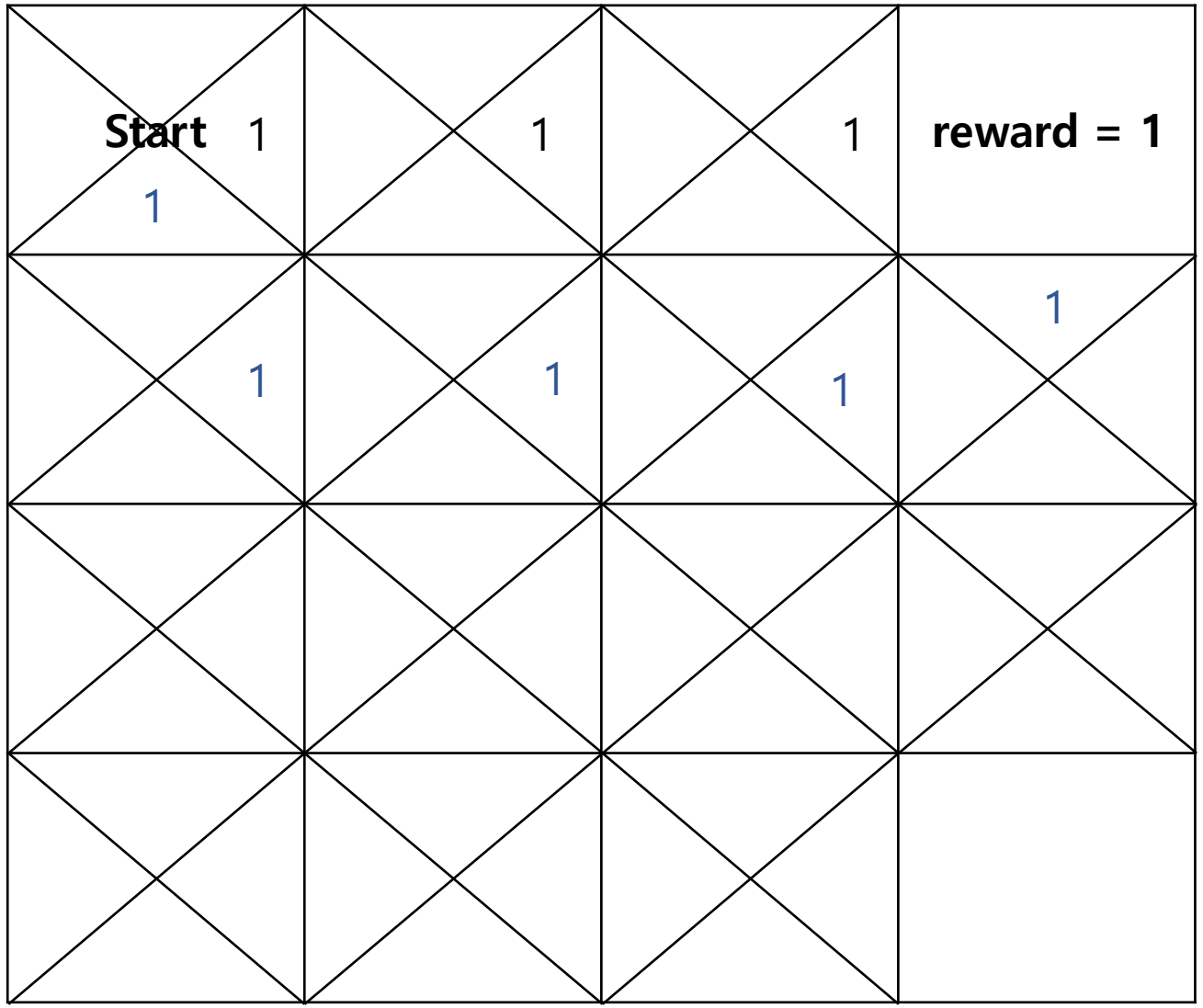
Q-learning

Q-learning

Greedy action : 결과에 대한 보상이 가장 클 것으로 기대되는 action을 고르는 전략



Episode

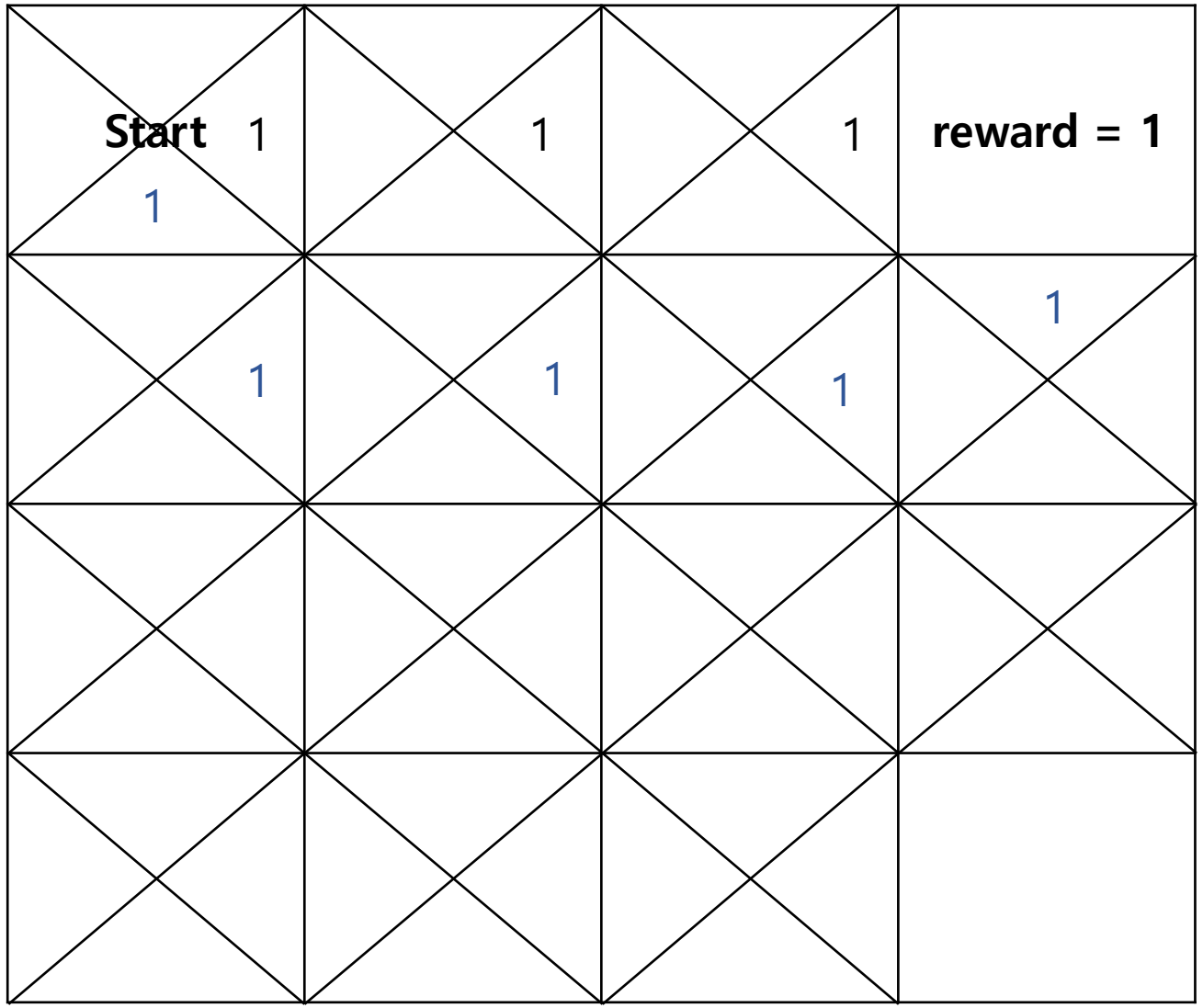


Q-learning

Exploration & Exploitation
탐험 / 착취(이익을 내기위한 이용, 사용)
Trade – off 관계

ε-Greedy
- ε : 0과 1사이의 값
- 너무 greedy한 액션만 해서도 안되고
너무 random하게 움직이면
reward에 도달하지 못할 수 있다.

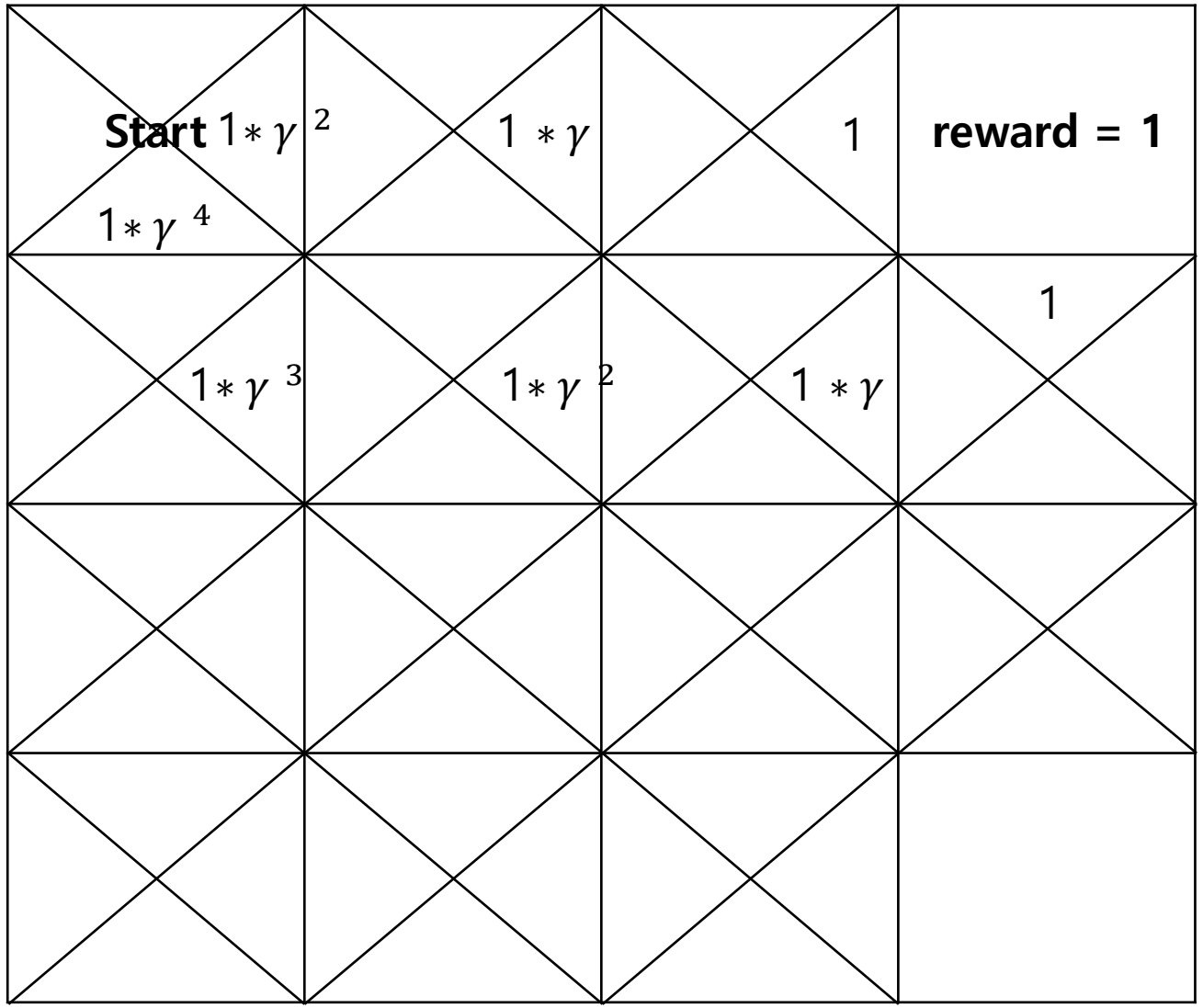
(Decaying)ε-Greedy
- 점점 ε의 값을 줄여나가는 것 (0.9 -> 0.1)
- 처음에는 탐험에 비중,
state가 적당히 갱신되면
states의 score을 따를 수 있도록 한다



Q-learning

"이 action을 취하면 먼 미래에 더 좋다 라는 feedback"

Discount factor(할인율)
- 0에서 1사이의 값
- 주로 감마(γ)로 표기
- reward를 직접 발견한 action에 대해서는 그냥 기록하되, action에 대한 평가를 그 전 action의 state에 기록할 때는 「를 곱해서 기록



Q learning

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

즉시 reward + 다음 state에서의 가장 큰 q값 * discount factor

"episode가 여러 번 반복될 때 score 갱신은?"

Q-update

"온고지신! 고와 신 균형 맞추기"

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

α : 학습 속도 인자

α 가 1이라면
α 가 0.5라면

값이 크면 현재 학습한 값을 조금 더 신경쓰겠다는 것,
작으면 기존에 기록된 값을 더 신경쓰겠다는 것이 된다.

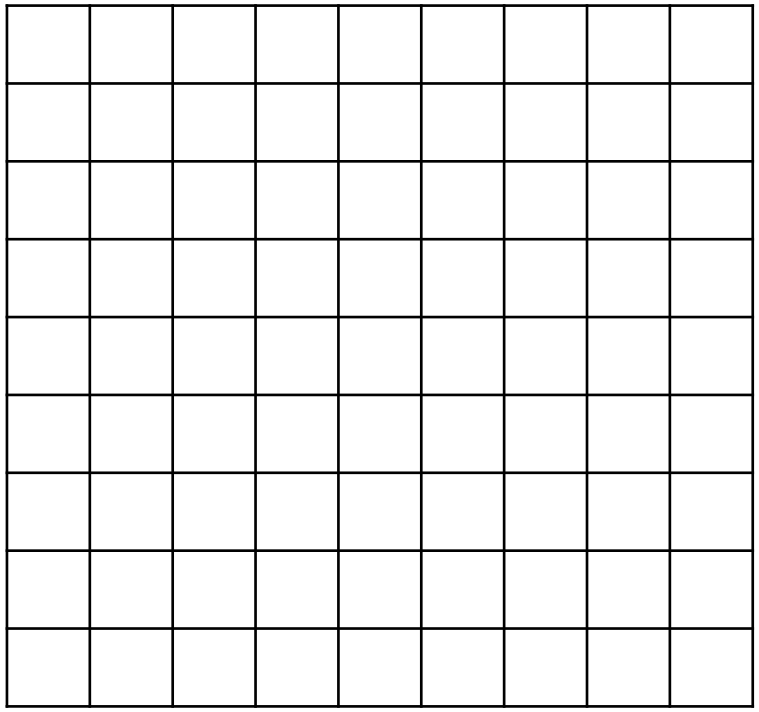
최적 정책 함수(optimal policy function)

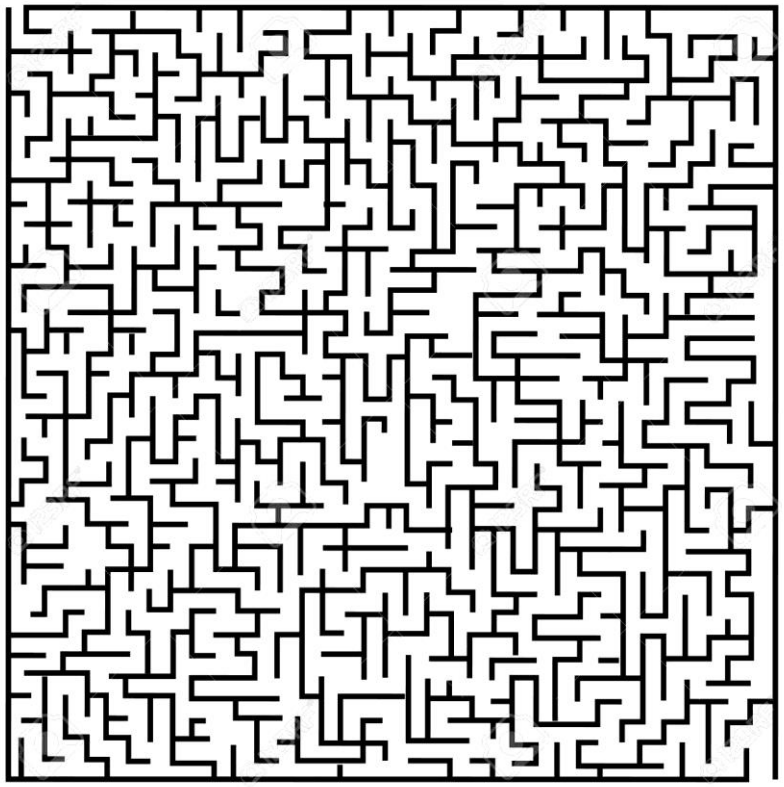
$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \right)$$

Model을 몰라도 s,a,s',r 값만 있으면 Q함수를 구할 수 있다.
즉 어떤 상태에서 어떤 행동을 해서 어떤 상태가 됐고,
어떤 보상을 얻었는지에 대한 데이터만 있으면 수식을 이용해 Q함수를 구할 수 있다.

Deep Q Networks

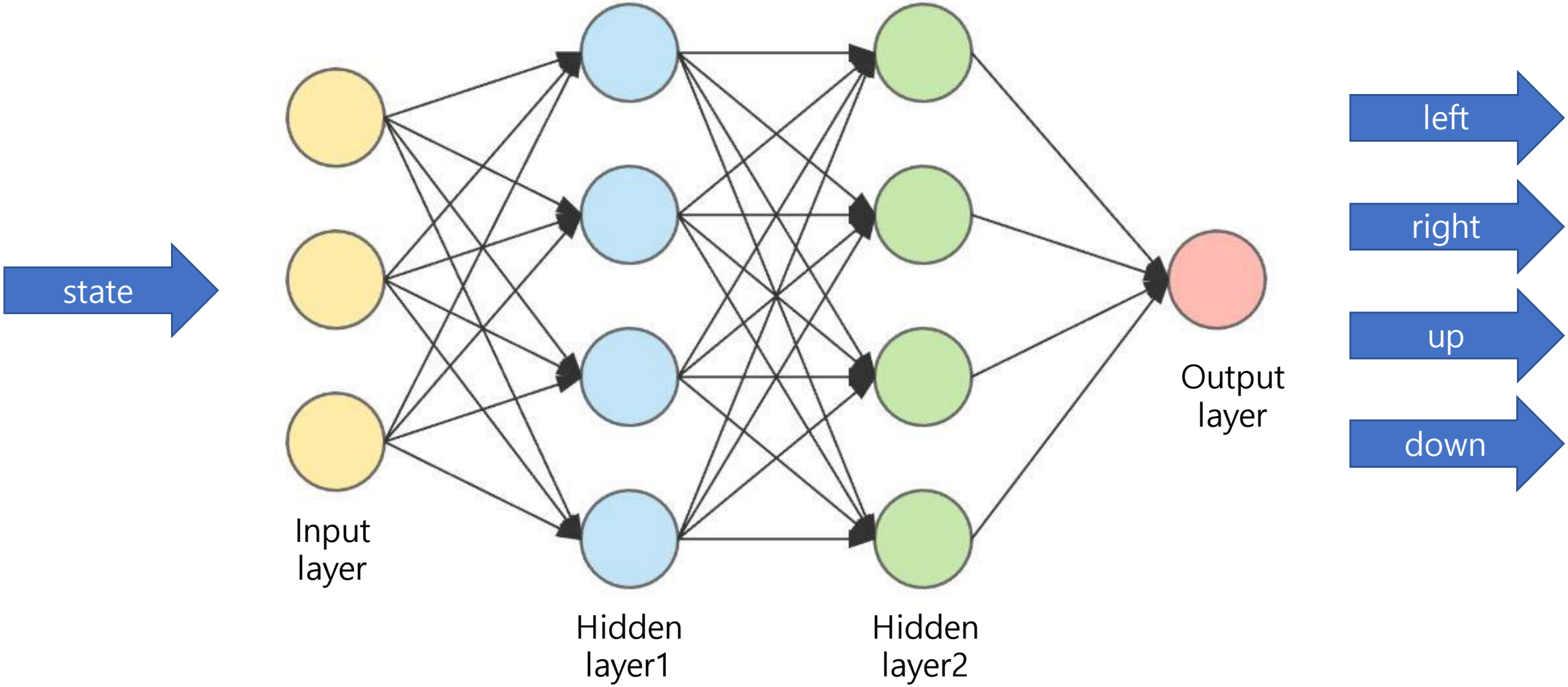
더 복잡한 문제들...





차원의 저주(curse of dimensionality)

Neural Network



Double DQN

Double Q learning

"Q-value가 튀었을 때 그 에러가 바로 퍼지지 않도록 방지책을 두자"

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Bigg(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \Bigg)$$

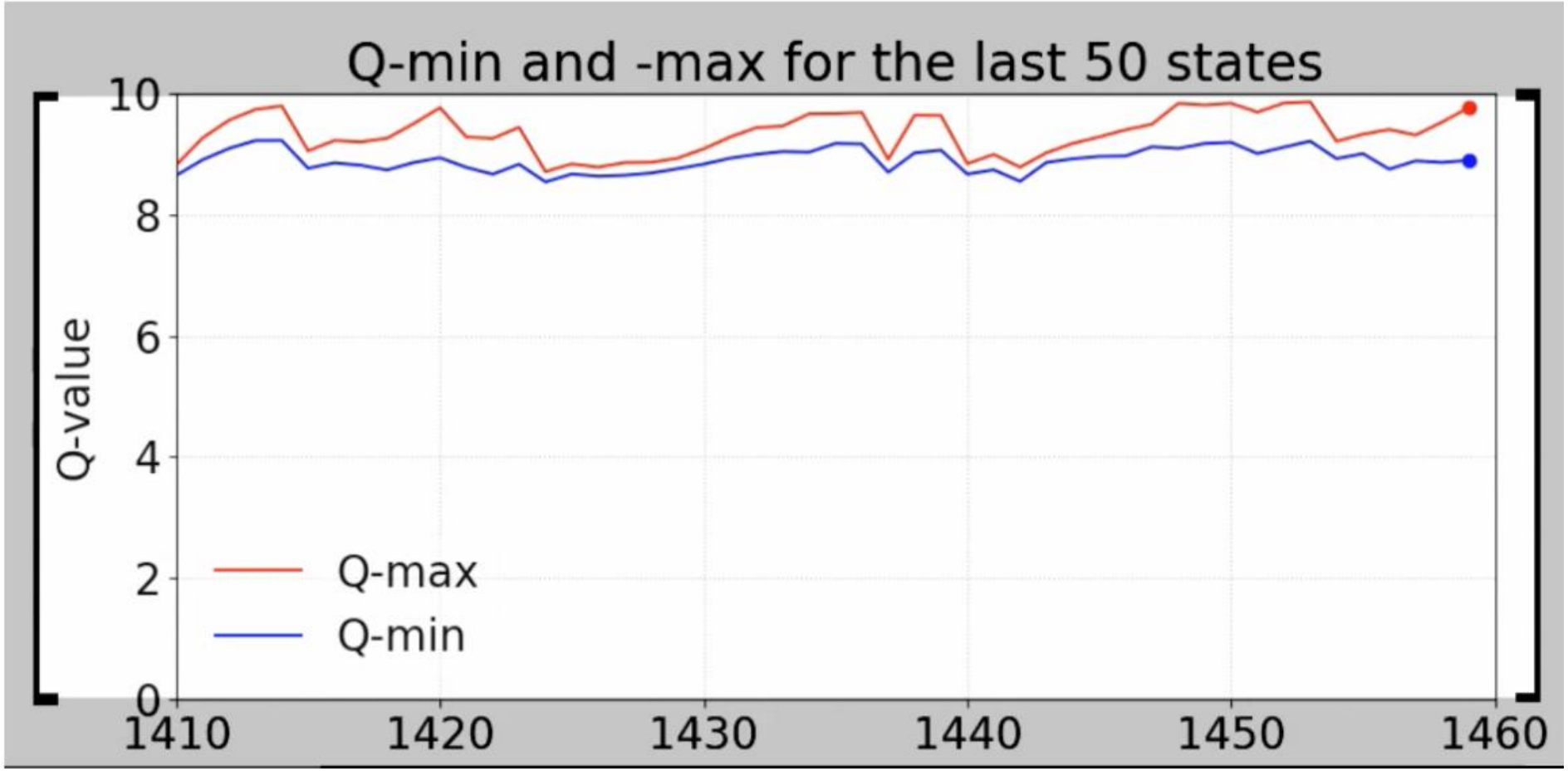
Dueling Deep Q-Network

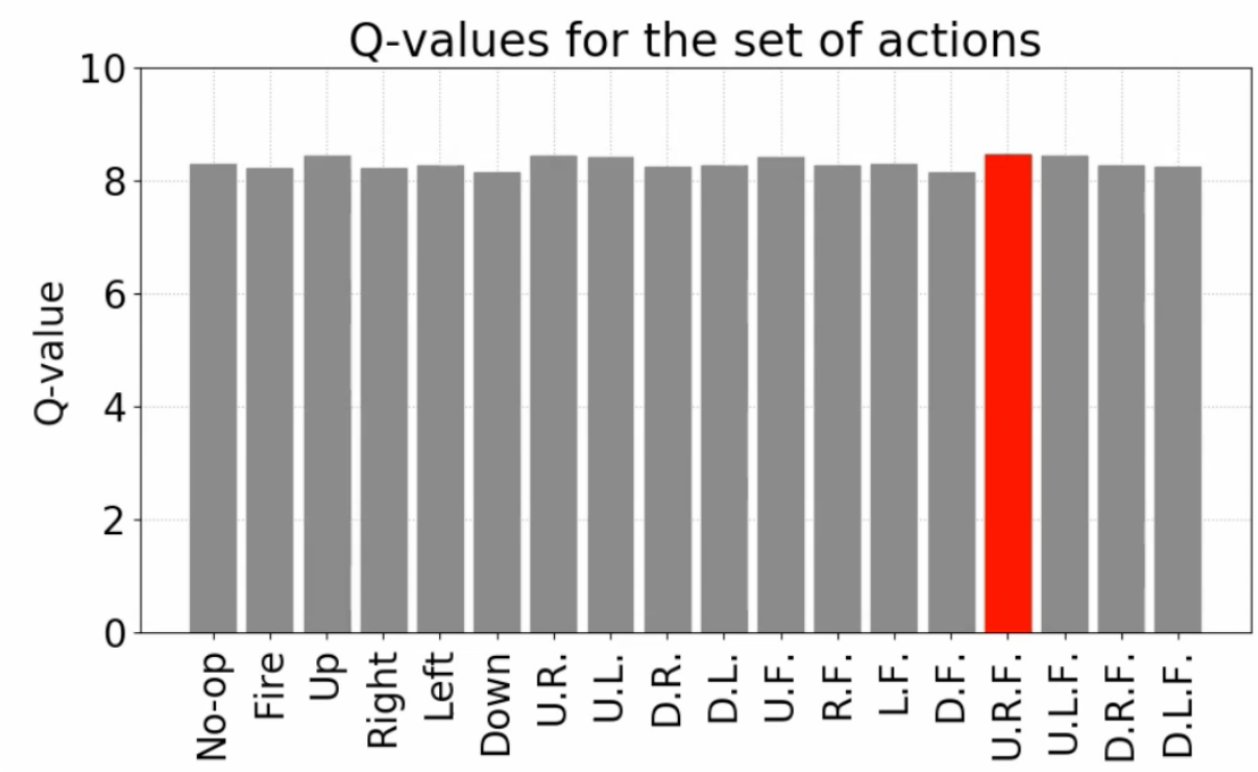
Abstract

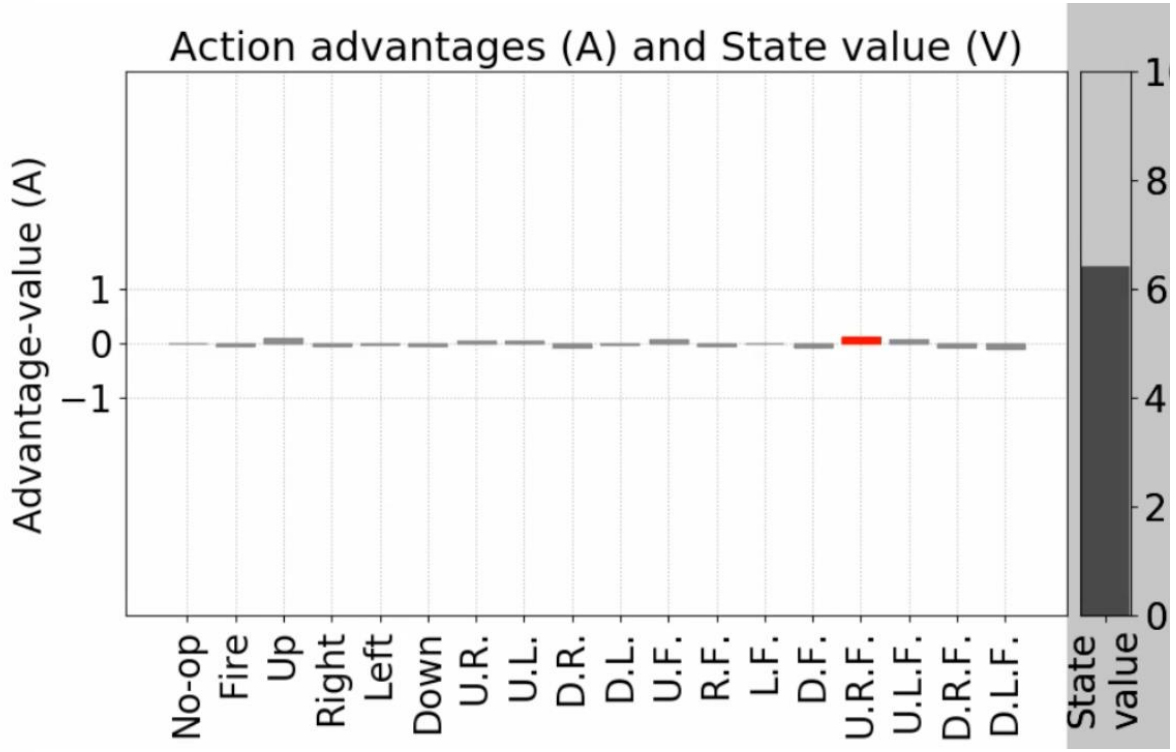
In recent years there have been many successes of using deep representations in reinforcement learning. Still, many of these applications use conventional architectures, such as convolutional networks, LSTMs, or auto-encoders. In this paper, we present a new neural network architecture for model-free reinforcement learning. Our dueling network represents two separate estimators: one for the state value function and one for the state-dependent action advantage function.

- No Operation
- Fire
- Up
- Right
- Left
- Down
- Up-Right
- Up-Left
- Down-Right
- Down-Left
- Up-Fire
- Right-Fire
- Left-Fire
- Down-Fire
- Upper-Right-Fire
- Upper-Left-Fire
- Down-Left-Fire

https://www.youtube.com/watch?v=XjsY8-P4WHM&t=130s


Q-min and -max for the last 50 states
Q-value
10
8
6
4
2
0
1410
1420
1430
1440
1450
1460
Q-max
Q-min


Q-values for the set of actions
Q-value
10
8
6
4
2
0
No-op
Fire
Up
Right
Left
Down
U.R.
U.L.
D.R.
D.L.
U.F.
R.F.
L.F.
D.F.
U.R.F.
U.L.F.
D.R.F.
D.L.F.


Action advantages (A) and State value (V)
Advantage-value (A)
1
0
−1
No-op
Fire
Up
Right
Left
Down
U.R.
U.L.
D.R.
D.L.
U.F.
R.F.
L.F.
D.F.
U.R.F.
U.L.F.
D.R.F.
D.L.F.
State value
10
8
6
4
2
0

$$Q(s,a) = V(s) + A(s,a)$$

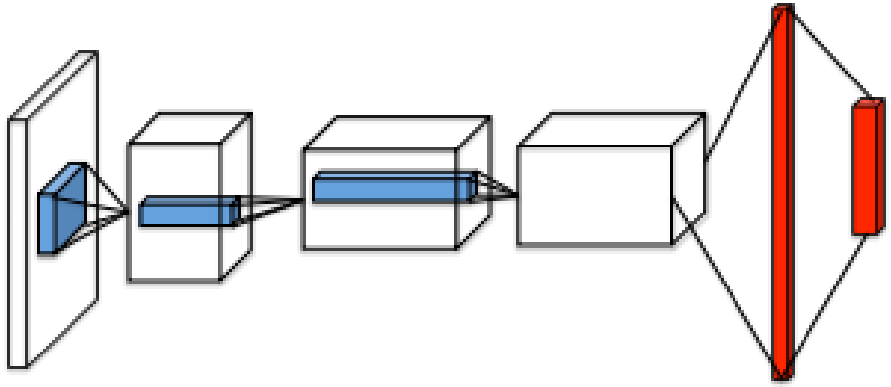
$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q^{\pi}(s,a)\right].$$

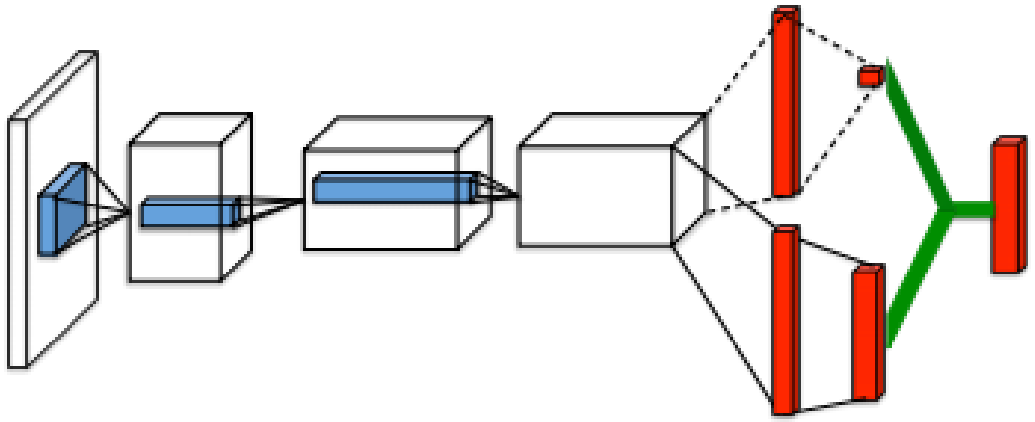
$$\mathbb{E}_{a\sim\pi(s)}\left[A^{\pi}(s,a)\right] = 0.$$

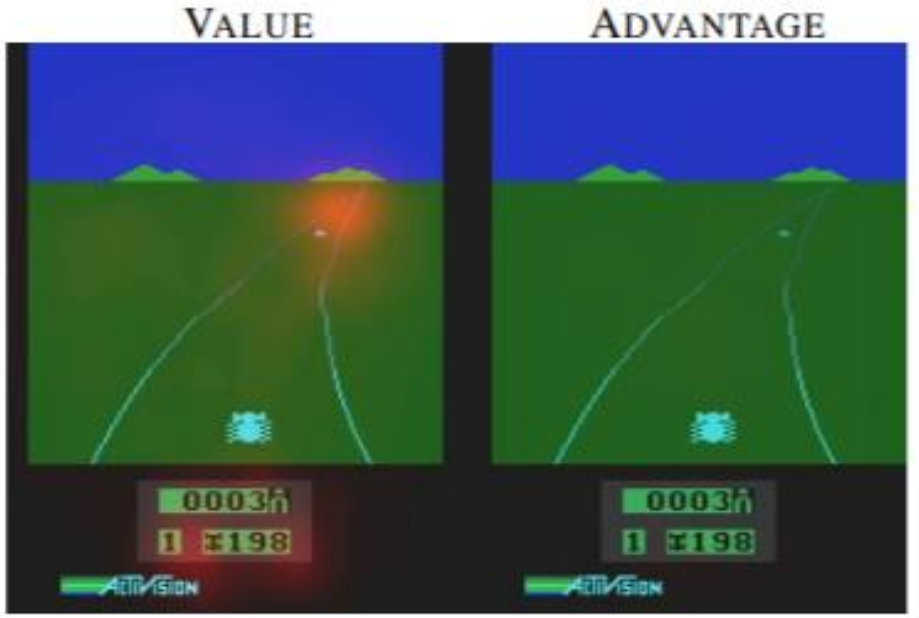
We define another important quantity, the *advantage function*, relating the value and Q functions:

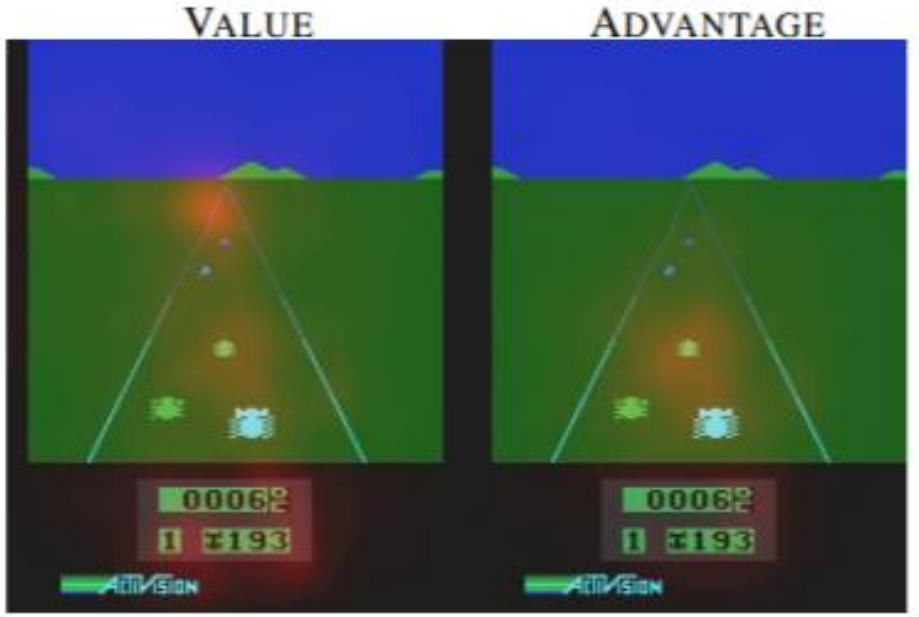
$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{3}$$

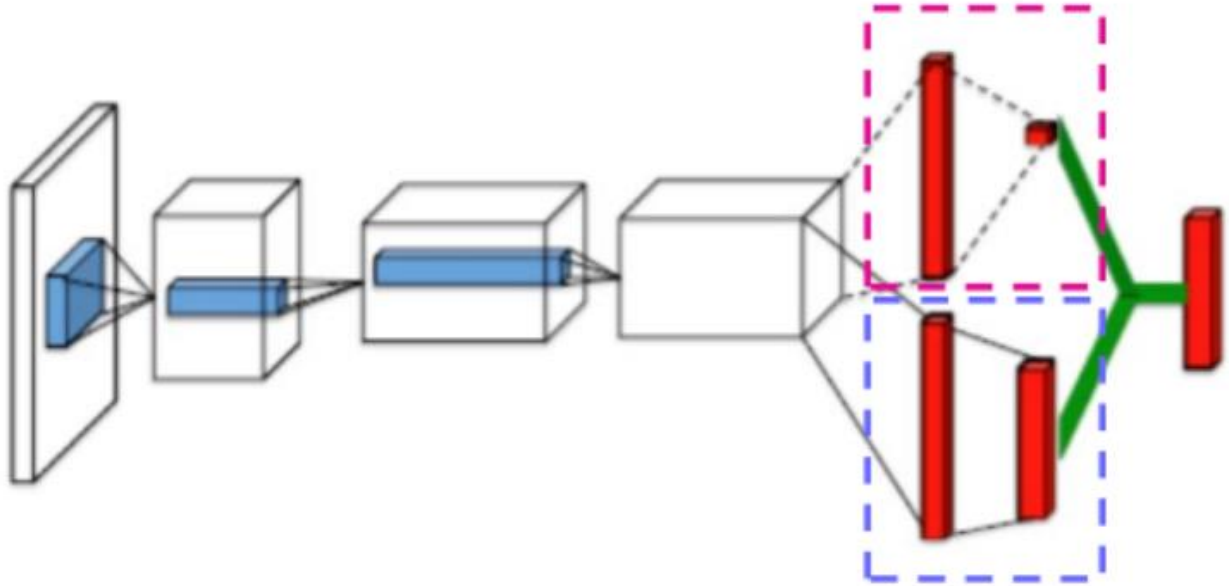
$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$





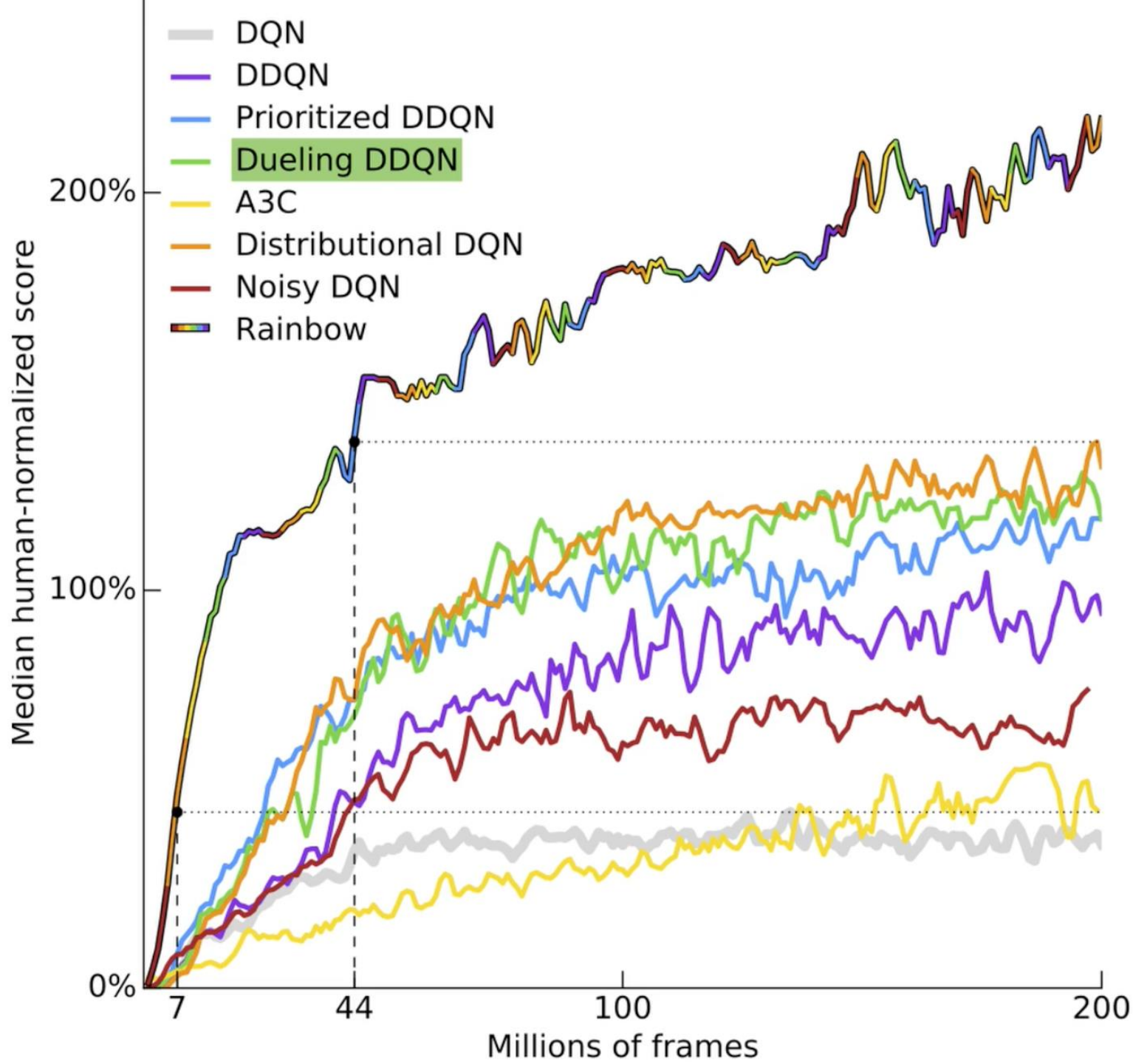

VALUE
ADVANTAGE
0003
1 x198
ACTIVISION
0003
1 x198
ACTIVISION

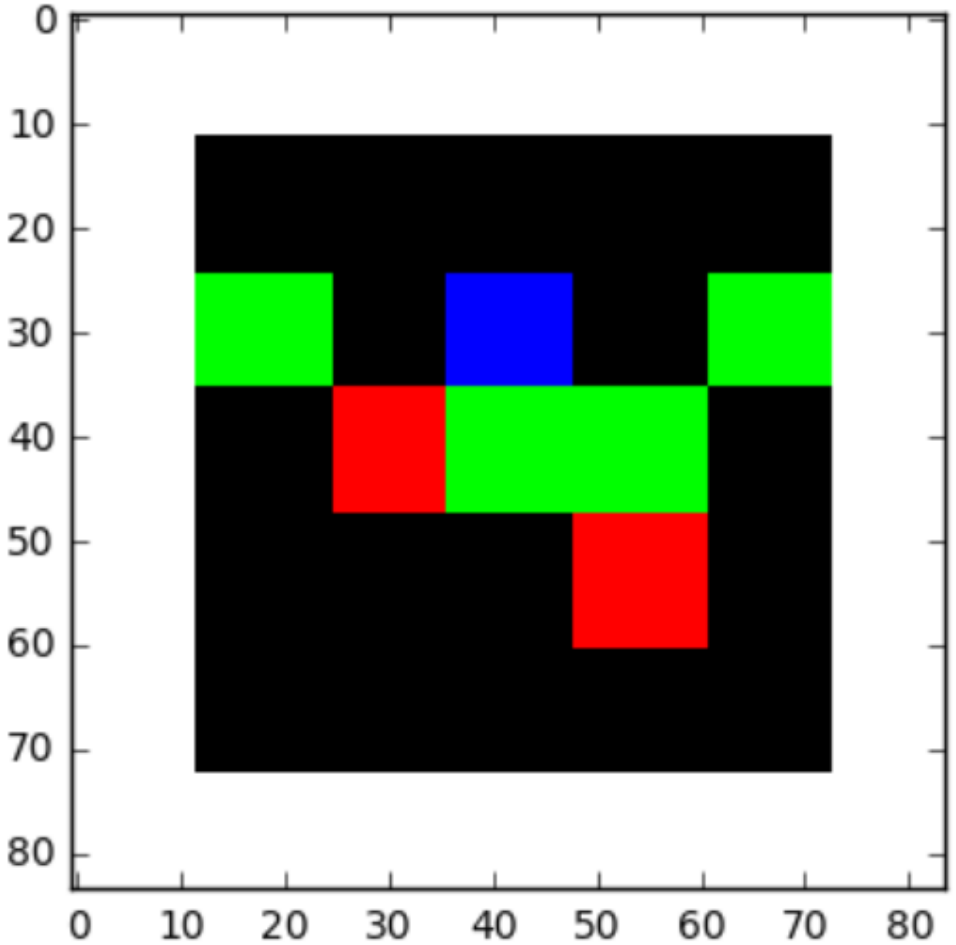

VALUE
ADVANTAGE
0006
1 x193
ACTIVISION
0006
1 x193
ACTIVISION



: state-value funcition(scalar), $\boldsymbol{V(s)}$
state-dependent , action-independent
→ Goodness of state $\boldsymbol{s}$

: advantage function(vector), $\boldsymbol{A(s,a)}$
a state, action dependent
→ Goodness of taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$


DQN
DDQN
Prioritized DDQN
Dueling DDQN
A3C
Distributional DQN
Noisy DQN
Rainbow
200%
100%
0%
Median human-normalized score
7
44
100
200
Millions of frames



Training the network

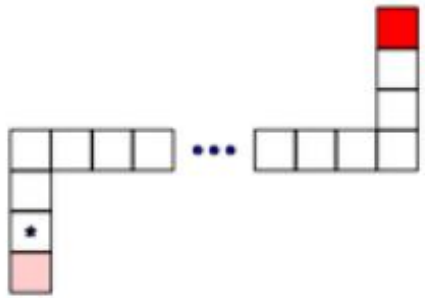
학습 파라미터들 설정하기

```
batch_size = 32 # 각 학습 단계에 대해 얼마나 많은 경험을 사용할지 결정
update_freq = 4 # 학습 단계를 얼마나 자주 수행할 것인가
y = .99 # 타겟 Q 값에 대한 할인 인자
startE = 1 # 무작위 행위의 시작 확률
endE = 0.1 # 무작위 행위의 최종 확률
anneling_steps = 10000. # startE부터 endE까지 몇단계에 걸쳐서 줄일 것인가.
num_episodes = 10000 # 몇개의 에피소드를 할 것인가.
pre_train_steps = 10000 # 학습 시작 전에 몇번의 무작위 행위를 할 것인가.
max_epLength = 50 # 에피소드의 최대 길이 (50 걸음)
load_model = False # 저장된 모델을 불러올 것인가?
path = "./dqn" # 모델을 저장할 위치
h_size = 512 # 이득 함수와 가치 함수로 나뉘기 전에 최종 콘볼루션의 크기
tau = 0.001 # 주요 신경망을 향해 타겟 신경망이 업데이트되는 비율
```

```
# 마지막 콘볼루션 레이어의 출력을 가지고 2로 나눈다.
# streamAC, streamVC 는 각각 1x1x256
self.streamAC,self.streamVC = tf.split(3,2,self.conv4)
# 이를 벡터화한다. streamA 와 streamV는 256 차원씩이다.
self.streamA = tf.contrib.layers.flatten(self.streamAC)
self.streamV = tf.contrib.layers.flatten(self.streamVC)
# 256개의 노드를 곱해서 각각 A와 V를 구하는 가중치
self.AW = tf.Variable(tf.random_normal([256,env.actions]))
self.VW = tf.Variable(tf.random_normal([256,1]))
# 점수화 한다.
self.Advantage = tf.matmul(self.streamA,self.AW)
self.Value = tf.matmul(self.streamV,self.VW)
```

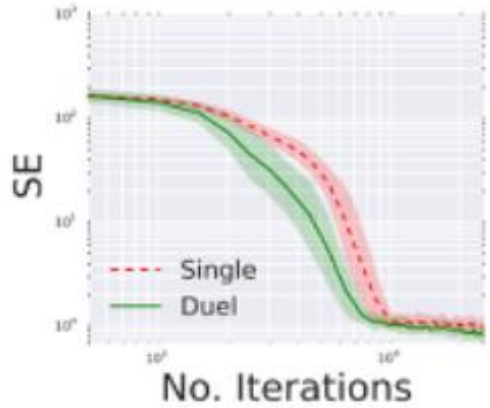
https://gist.github.com/ishuca/6e2fd4c2cc6e49733249856f74c726ff#file-double-dueling-dqn-tutorial-ipynb

CORRIDOR ENVIRONMENT



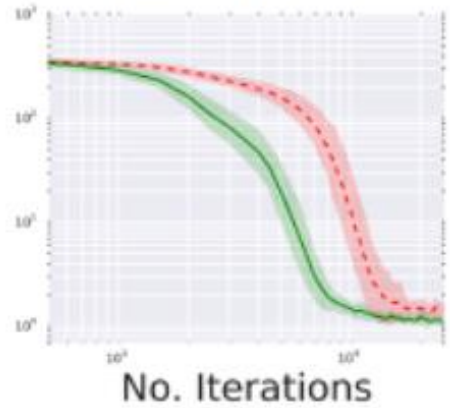
(a)

5 ACTIONS



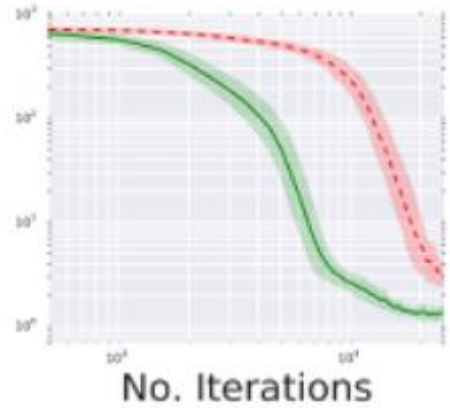
(b)

10 ACTIONS



(c)

20 ACTIONS



(d)

감사합니다.