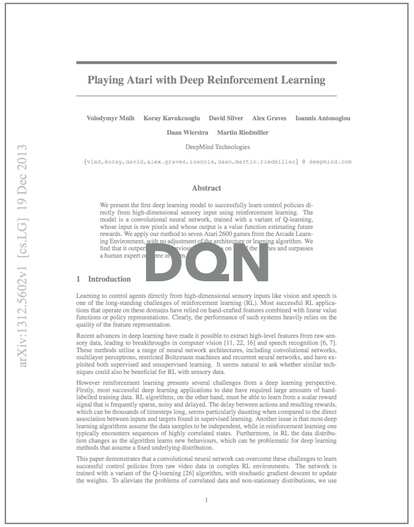
PYCON KOREA 2018

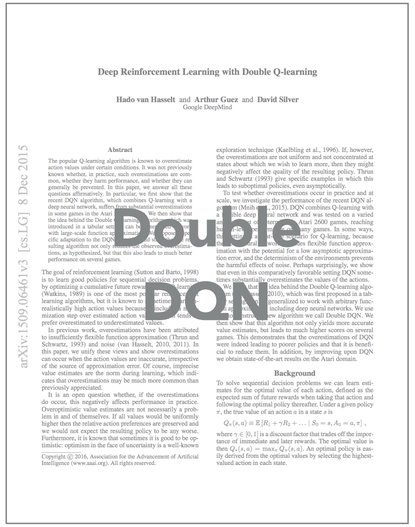
RL Adventure

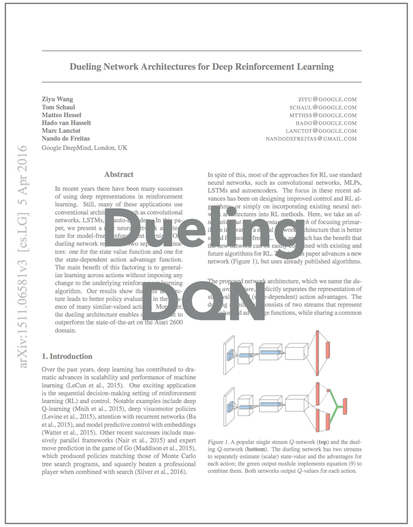
DQN, Double DQN & Duel DQN

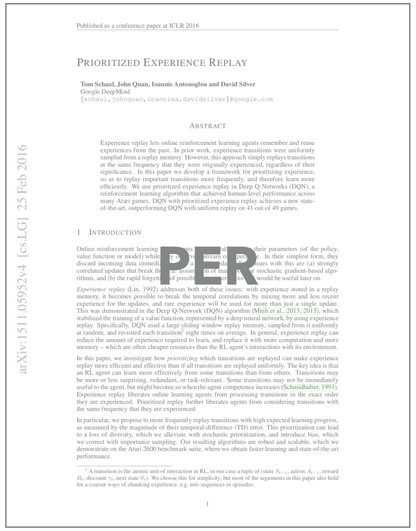
성태경

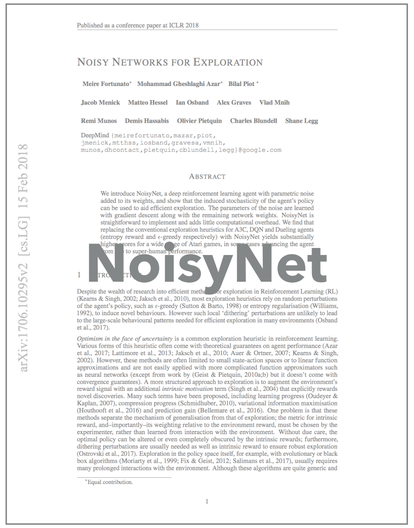
OUTLINE

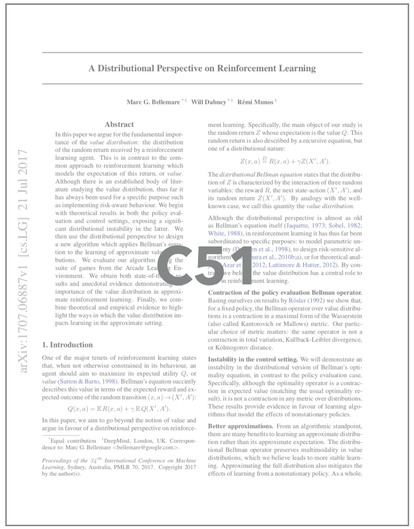

Playing Atari with Deep Reinforcement Learning
Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller
DeepMind Technologies
Abstract
1 Introduction
arXiv:1312.5602v1 [cs.LG] 19 Dec 2013
DQN

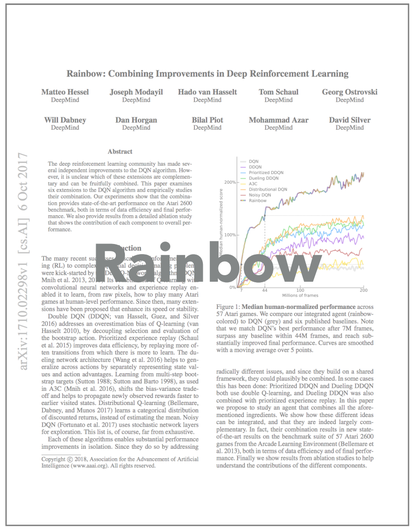

Deep Reinforcement Learning with Double Q-learning
Hado van Hasselt and Arthur Guez and David Silver
Google DeepMind
Abstract
Background
arXiv:1509.06461v3 [cs.LG] 8 Dec 2015
Double DQN


Dueling Network Architectures for Deep Reinforcement Learning
Ziyu Wang
Tom Schaul
Matteo Hessel
Hado van Hasselt
Marc Lanctot
Nando de Freitas
Google DeepMind, London, UK
Abstract
1. Introduction
arXiv:1511.06581v3 [cs.LG] 5 Apr 2016
Dueling DQN


Published as a conference paper at ICLR 2016
Prioritized Experience Replay
Tom Schaul, John Quan, Ioannis Antonoglou and David Silver
Google DeepMind
Abstract
1 Introduction
arXiv:1511.05952v4 [cs.LG] 25 Feb 2016
PER

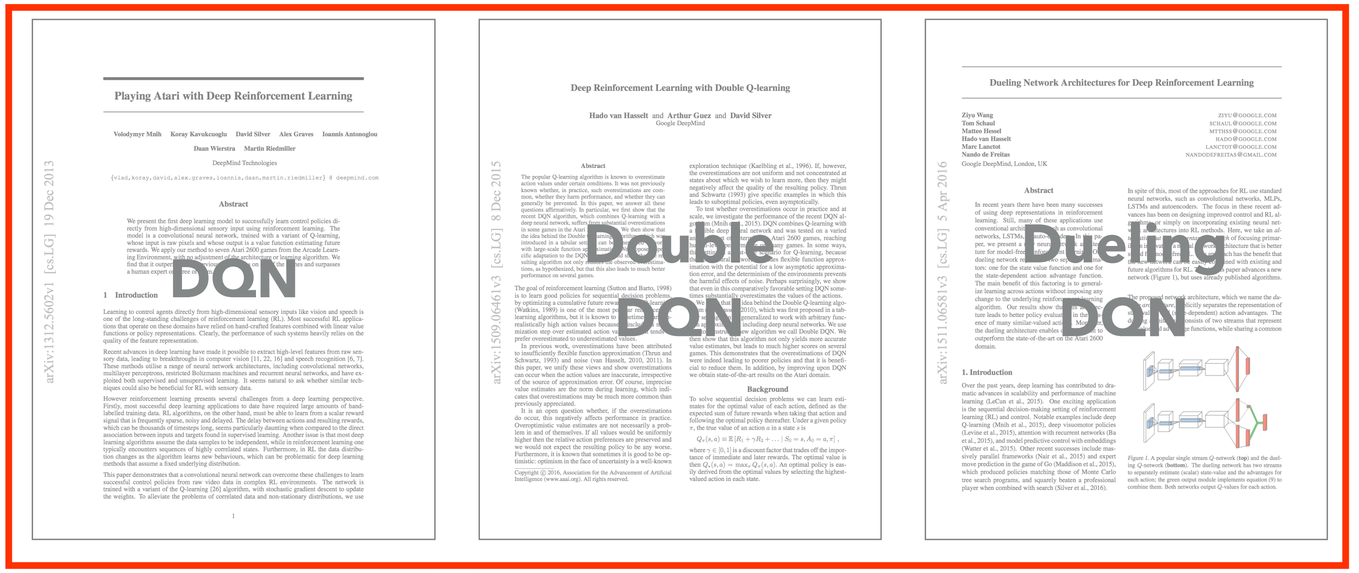

Published as a conference paper at ICLR 2018
Noisy Networks for Exploration
Meire Fortunato* Mohammad Gheshlaghi Azar* Bilal Piot*
Jacob Menick Matteo Hessel Ian Osband Alex Graves Vlad Mnih
Rémi Munos Demis Hassabis Olivier Pietquin Charles Blundell Shane Legg
DeepMind
Abstract
1 Introduction
arXiv:1706.10295v2 [cs.LG] 15 Feb 2018
NoisyNet

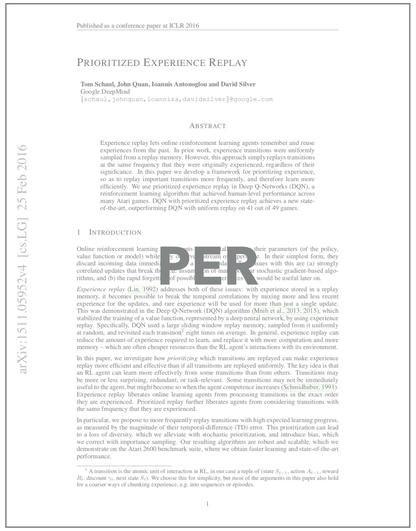

A Distributional Perspective on Reinforcement Learning
Marc G. Bellemare Will Dabney Rémi Munos
Abstract
1. Introduction
arXiv:1707.06887v1 [cs.LG] 21 Jul 2017
C51

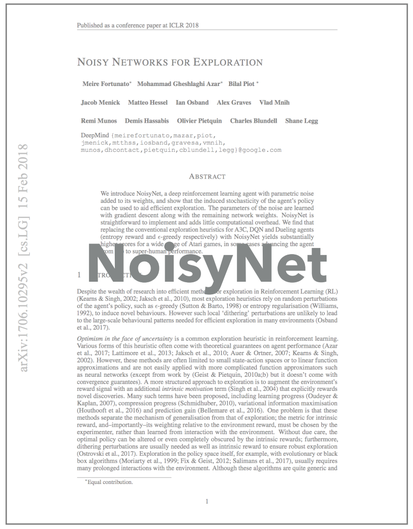

Rainbow: Combining Improvements in Deep Reinforcement Learning
Matteo Hessel Joseph Modayil Hado van Hasselt Tom Schaul Georg Ostrovski
Will Dabney Dan Horgan Bilal Piot Mohammad Azar David Silver
DeepMind
Abstract
1 Introduction
arXiv:1710.02298v1 [cs.AI] 6 Oct 2017
Rainbow

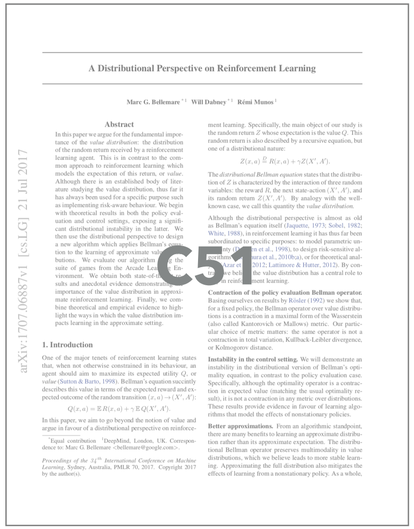
+ 구현

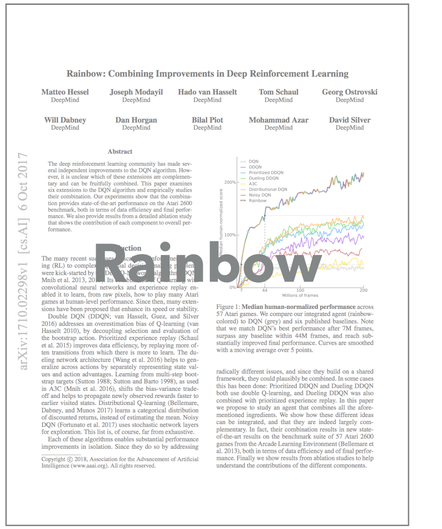
OUTLINE


Playing Atari with Deep Reinforcement Learning
DQN
Deep Reinforcement Learning with Double Q-learning
Double DQN
Dueling Network Architectures for Deep Reinforcement Learning
Dueling DQN


Prioritized Experience Replay
PER


Noisy Networks for Exploration
NoisyNet

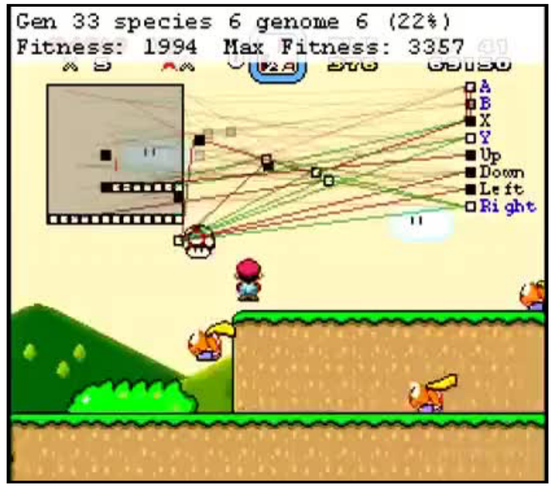

A Distributional Perspective on Reinforcement Learning
C51


Rainbow: Combining Improvements in Deep Reinforcement Learning
Rainbow

+ 구현

RL APPLICATIONS

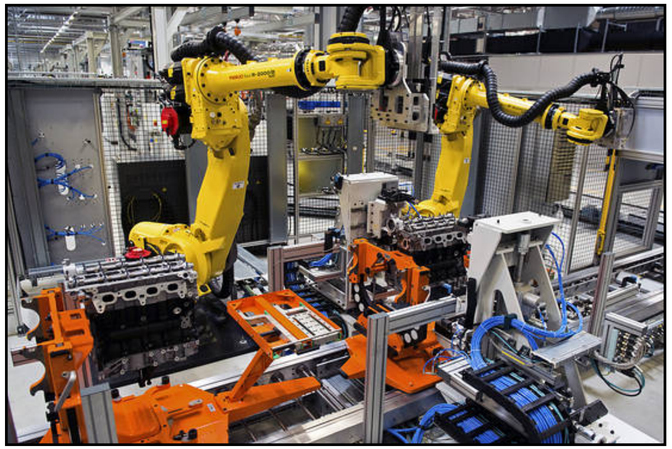
[Atari]



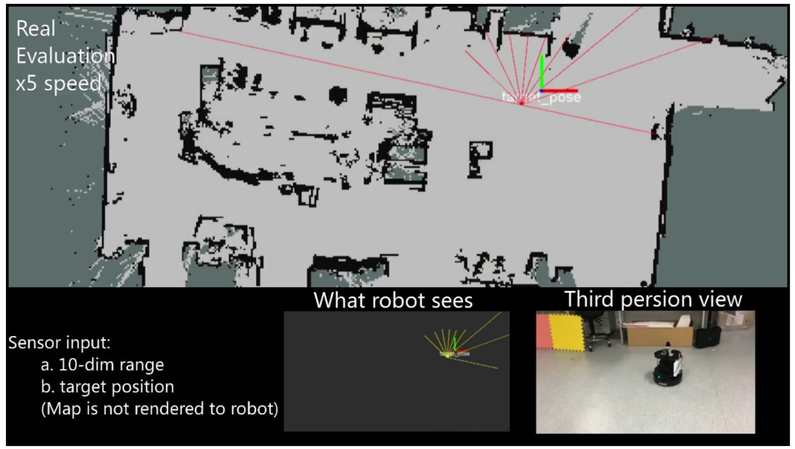
[Mario]

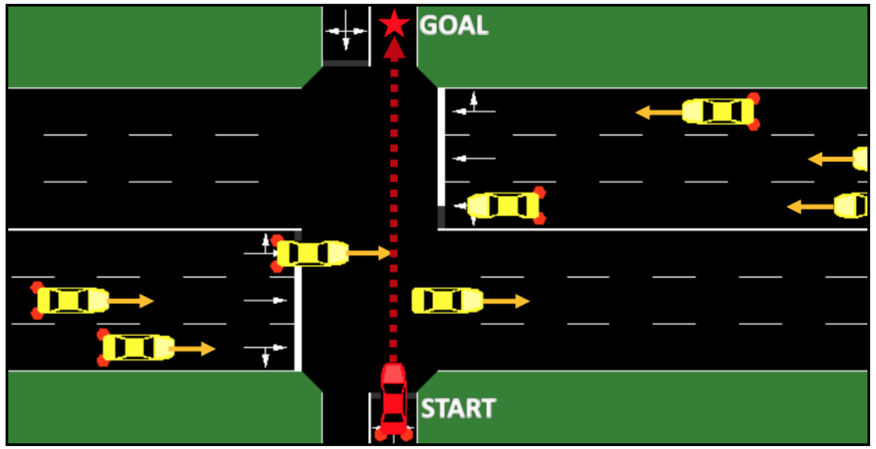
[Pommerman]

[Go]



[Robotics]





[Autonomous driving]

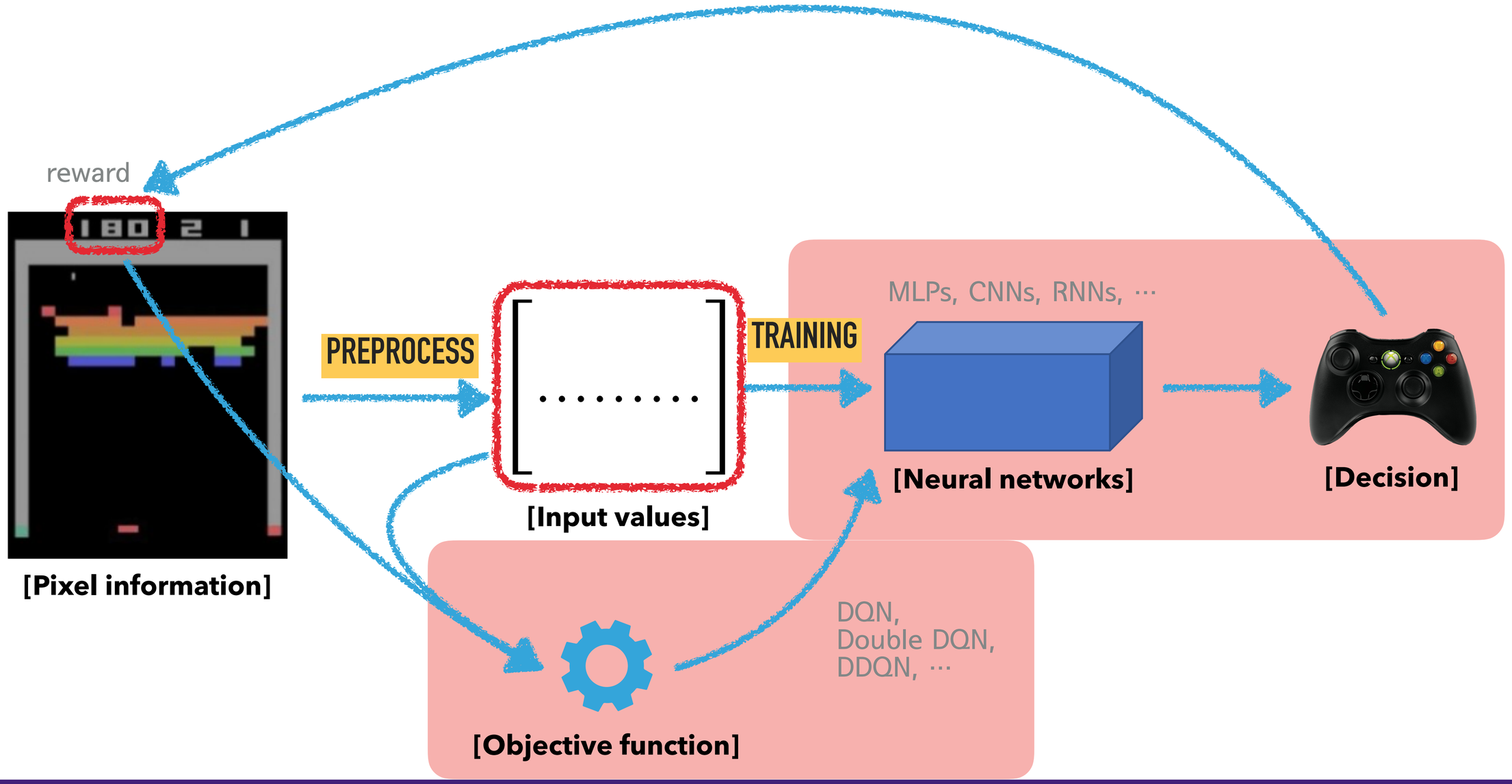
HIGH-LEVEL PROCESS

reward

[Pixel information]

[Decision]

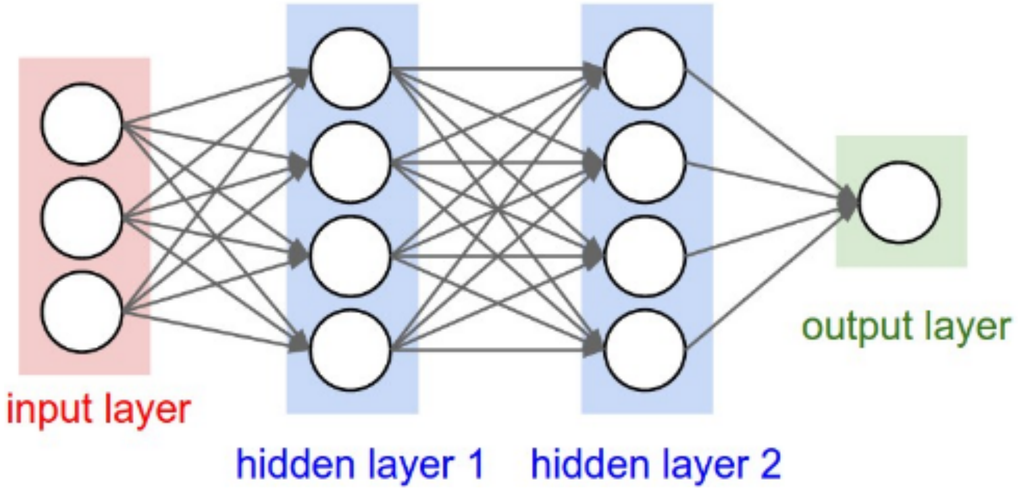
HIGH-LEVEL PROCESS



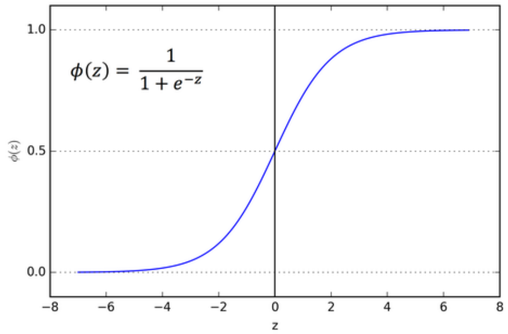
DEEP Q-NETWORK (DQN)

NEURAL NETWORKS IN ONE SLIDE

Weight 연산



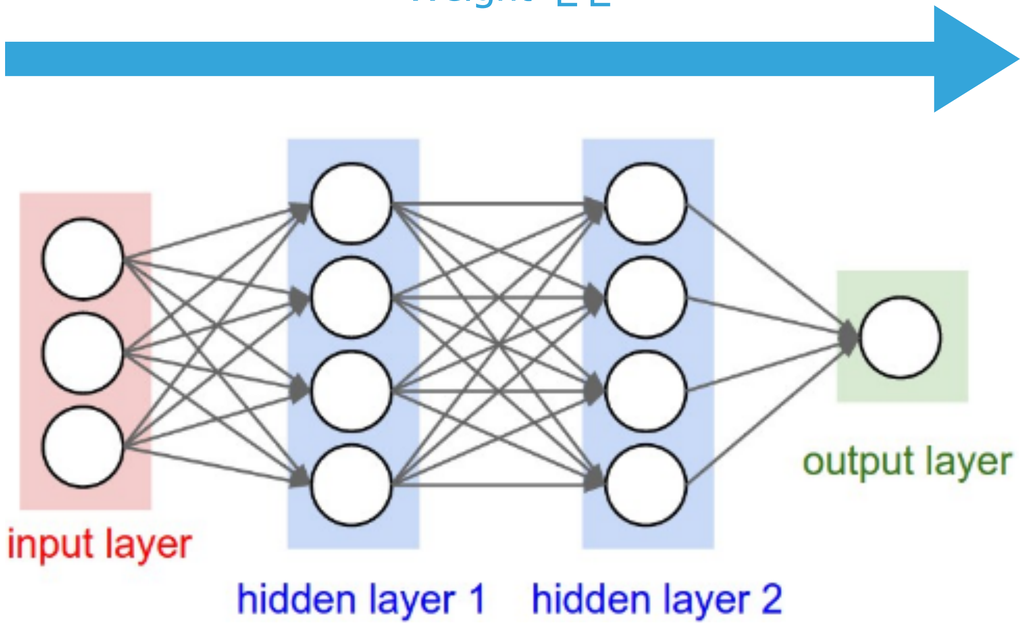
Backpropagation



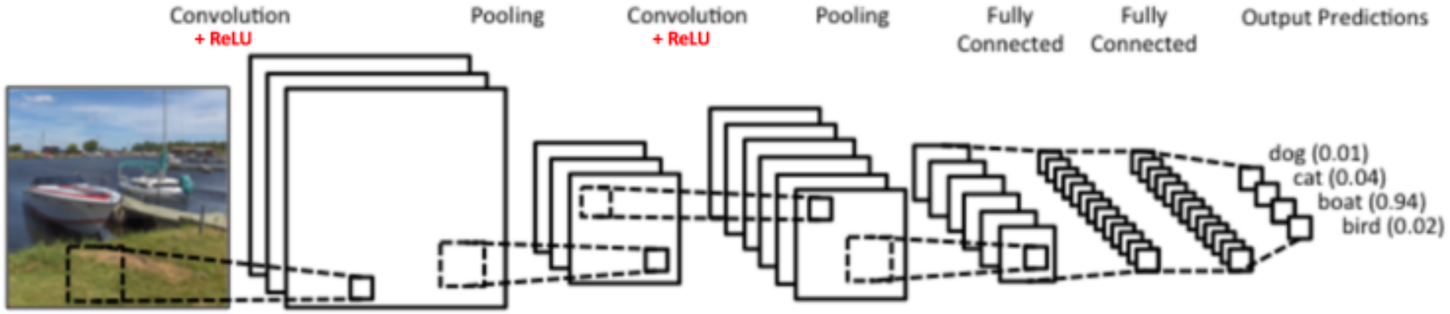
Non-linear function

NEURAL NETWORKS IN ONE SLIDE

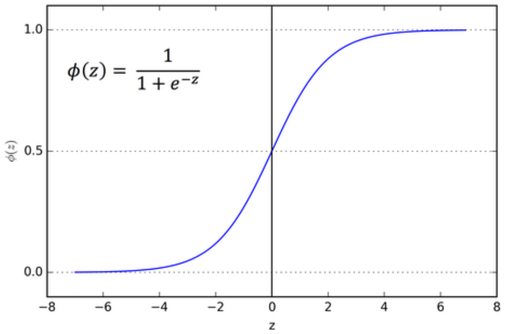
Weight 연산



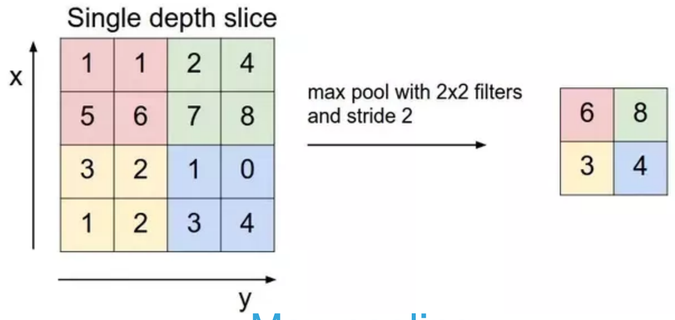
Backpropagation



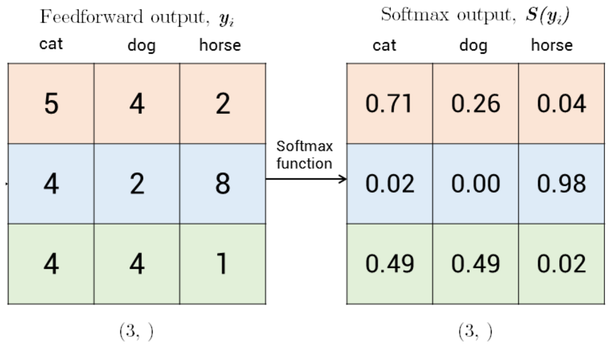
Convolutional neural network



Non-linear function



Max-pooling



Softmax

Q-LEARNING

- 목적: 현재의 상황에서 어떤 행동을 하는 것이 가장 좋은지

$$Q_\pi(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)], \gamma \in (0,1)$$

[Expected rewards]

현재의 reward값 다음 state의 reward값

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a_t))$$

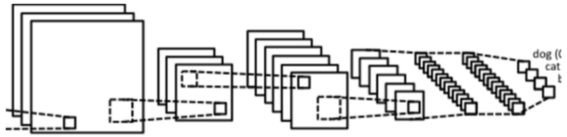
[Value iteration update]

C. J. C. H. Watkins, P. Dayan. Q-learning. 1992.

V. Minh, et al. Playing Atari with Deep Reinforcement Learning. NIPS, 2013

MOTIVATION

뉴럴 네트워크로 Q함수를 근사화

$$Q(s,a) \rightarrow \text{  } \rightarrow Q(s,a;\theta)$$

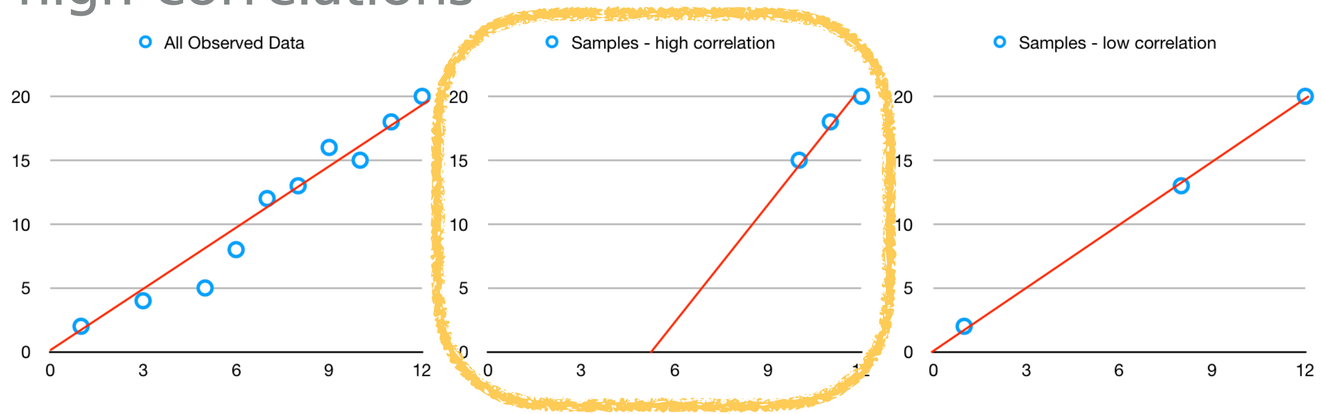
$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(\overbrace{r + \gamma max_{a'} Q(s',a';\theta_i) - Q(s,a;\theta_i)}^{\text{TD error}}\right)^2\right]$$

목표값 예측값

V. Minh, et al. Playing Atari with Deep Reinforcement Learning. NIPS, 2013

PROBLEM

- Unstable update
  - 입력 데이터간의 high correlations



  - Non-stationary targets (같은 네트워크 파라미터)

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

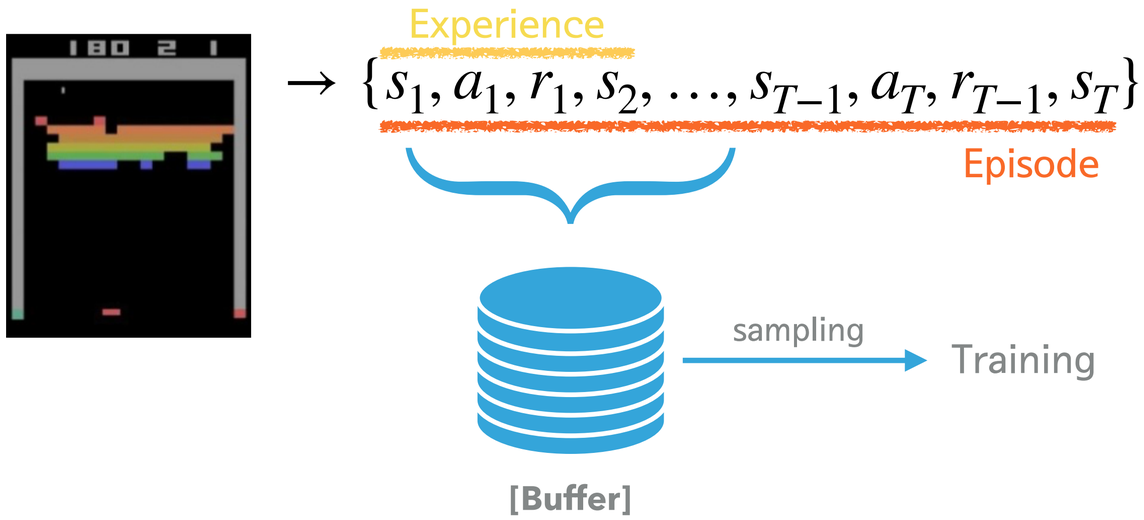
[Objective function]

https://curt-park.github.io/2018-05-17/dqn/

V. Minh, et al. Playing Atari with Deep Reinforcement Learning. NIPS, 2013

SOLUTION

- Experience replay

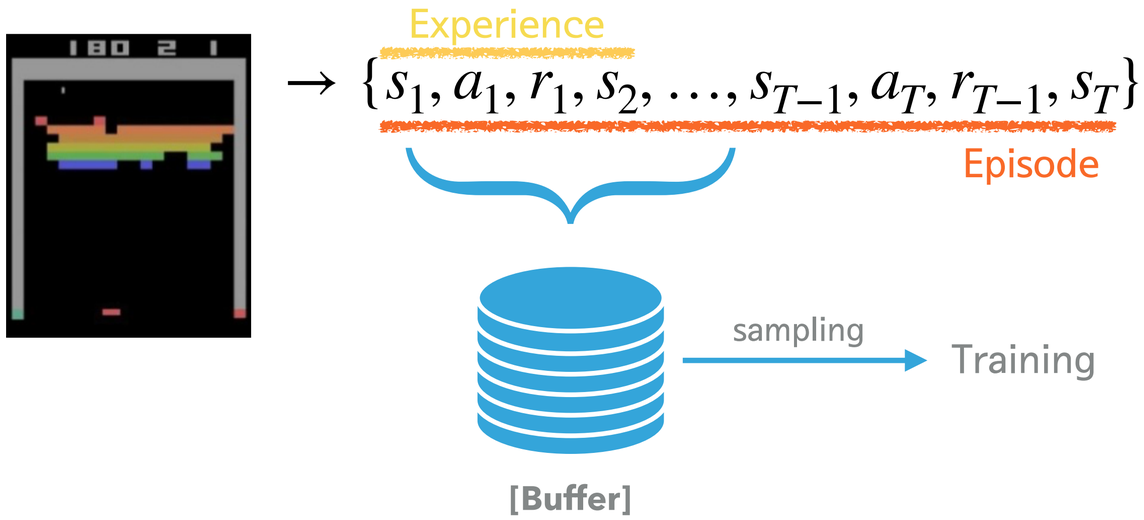


Matiisen, Tambet Demystifying Deep Reinforcement Learning. *Computational Neuroscience LAB*. 2015.

Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature,* 2015.

SOLUTION

- Experience replay



- Fixed Q-targets

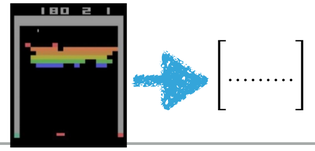
$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma max_{a'} Q(s', a'; \theta_i) - Q(s, a; \theta_i)\right)^2\right]$$

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

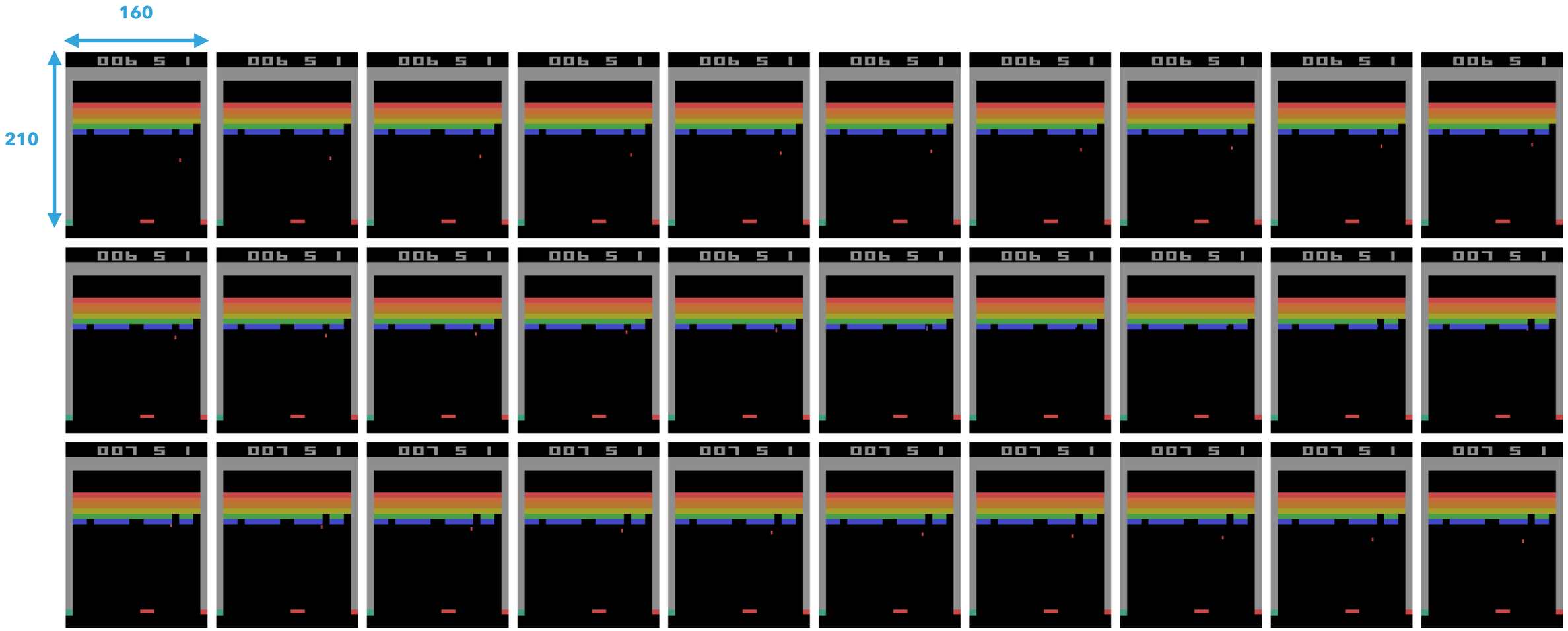
[Objective function]

Matiisen, Tambet Demystifying Deep Reinforcement Learning. *Computational Neuroscience LAB*. 2015.

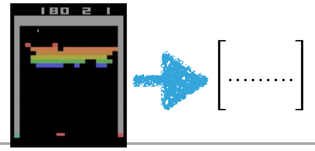
Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature,* 2015.



DQN – PREPROCESSING



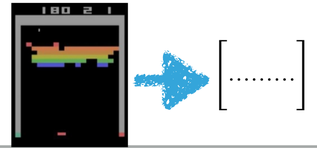
https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/



DQN - PREPROCESSING

사이즈를 줄이고 grayscale로 바꿈으로써 입력데이터 사이즈를 줄이자

https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/



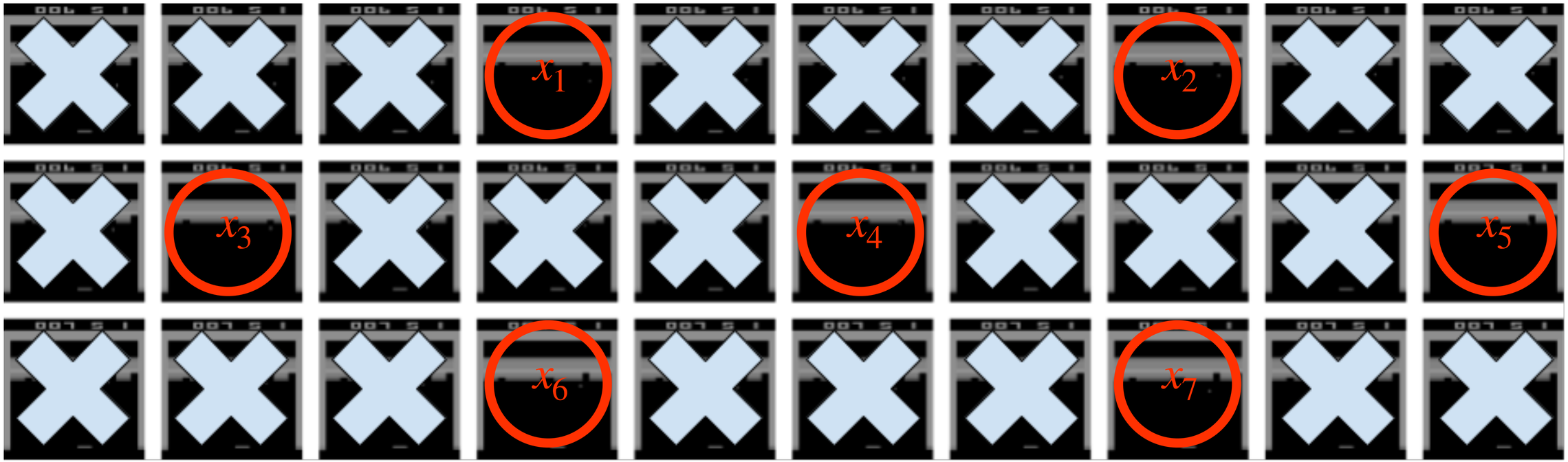
DQN – PREPROCESSING

[Input]: (84 x 84 x 4)

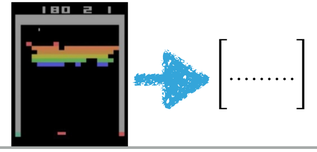
$$s_1 = (x_1, x_2, x_3, x_4)$$
$$s_2 = (x_2, x_3, x_4, x_5)$$
$$\vdots$$

Frame skipping

Parameter: 4



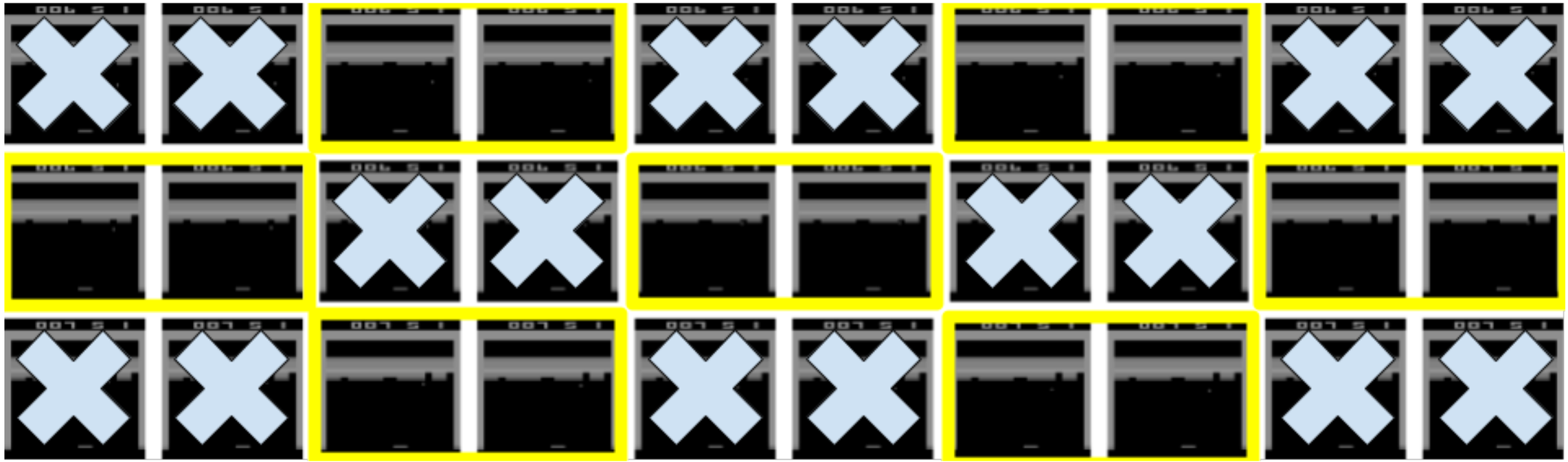
https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/



DQN – PREPROCESSING

Frame skipping

DeepMind took the component-wise maximum over two consecutive frames (Atari setting)



https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/

IMPLEMENTATION

Use Cuda

```
USE_CUDA = torch.cuda.is_available()
Variable = lambda *args, **kwargs: autograd.Variable(*args, **kwargs).cuda() if USE_CUDA else auto
grad.Variable(*args, **kwargs)
```

Atari Environment

```
from common.wrappers import make_atari, wrap_deepmind, wrap_pytorch
```

```
env_id = "PongNoFrameskip-v4"
env    = make_atari(env_id)
env    = wrap_deepmind(env)
env    = wrap_pytorch(env)
```

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

IMPLEMENTATION

Replay Buffer

```
from collections import deque

class ReplayBuffer(object):
    def __init__(self, capacity):
        self.buffer = deque(maxlen=capacity)

    def push(self, state, action, reward, next_state, done):
        state      = np.expand_dims(state, 0)
        next_state = np.expand_dims(next_state, 0)

        self.buffer.append((state, action, reward, next_state, done))

    def sample(self, batch_size):
        state, action, reward, next_state, done = zip(*random.sample(self.buffer, batch_size))
        return np.concatenate(state), action, reward, np.concatenate(next_state), done

    def __len__(self):
        return len(self.buffer)
```

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

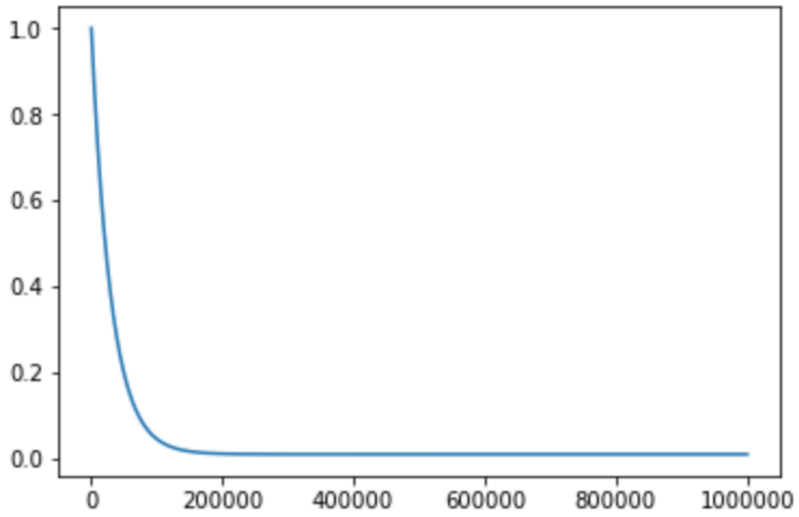
IMPLEMENTATION

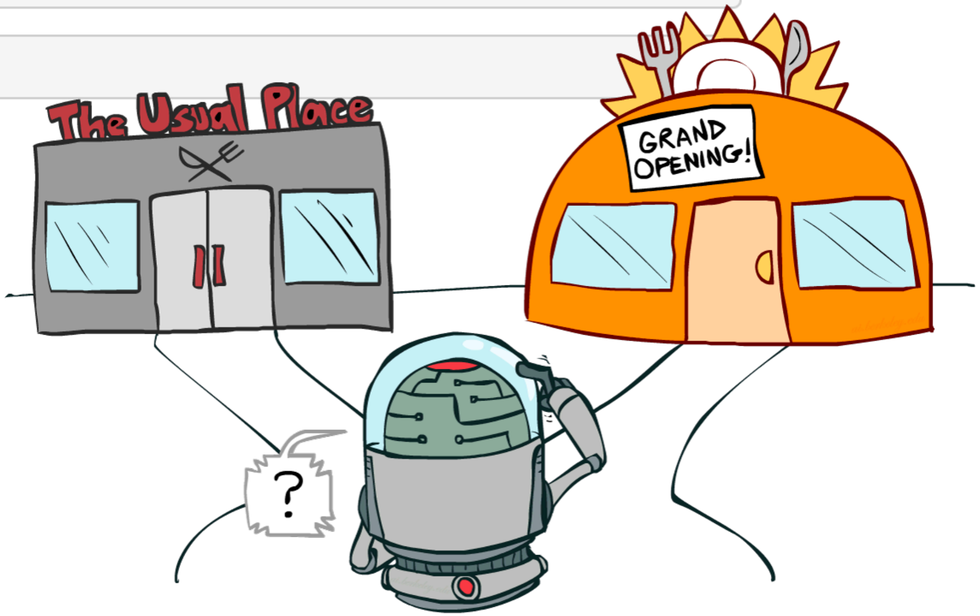
Epsilon greedy exploration

```
epsilon_start = 1.0
epsilon_final = 0.01
epsilon_decay = 30000

epsilon_by_frame = lambda frame_idx: epsilon_final + (epsilon_start - epsilon_final) * math.exp(-
1. * frame_idx / epsilon_decay)
```

```
plt.plot([epsilon_by_frame(i) for i in range(1000000)])
```

```
[<matplotlib.lines.Line2D at 0x7f73393b64d0>]
```





https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

IMPLEMENTATION

```
class CnnDQN(nn.Module):
    def __init__(self, input_shape, num_actions):
        super(CnnDQN, self).__init__()

        self.input_shape = input_shape
        self.num_actions = num_actions

        self.features = nn.Sequential(
            nn.Conv2d(input_shape[0], 32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU()
        )

        self.fc = nn.Sequential(
            nn.Linear(self.feature_size(), 512),
            nn.ReLU(),
            nn.Linear(512, self.num_actions)
        )
```

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

IMPLEMENTATION

```
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

    def feature_size(self):
        return self.features(autograd.Variable(torch.zeros(1, *self.input_shape))).view(1, -1).siz
e(1)

    def act(self, state, epsilon):
        if random.random() > epsilon:
            state   = Variable(torch.FloatTensor(np.float32(state)).unsqueeze(0), volatile=True)
            q_value = self.forward(state)
            action  = q_value.max(1)[1].data[0]
        else:
            action = random.randrange(env.action_space.n)
        return action
```

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

IMPLEMENTATION

```
model = CnnDQN(env.observation_space.shape, env.action_space.n)

if USE_CUDA:
    model = model.cuda()

optimizer = optim.Adam(model.parameters(), lr=0.00001)

replay_initial = 10000
replay_buffer = ReplayBuffer(100000)
```

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

IMPLEMENTATION

Computing Temporal Difference Loss

```
def compute_td_loss(batch_size):
    state, action, reward, next_state, done = replay_buffer.sample(batch_size)

    state      = Variable(torch.FloatTensor(np.float32(state)))
    next_state = Variable(torch.FloatTensor(np.float32(next_state)), volatile=True)
    action     = Variable(torch.LongTensor(action))
    reward     = Variable(torch.FloatTensor(reward))
    done       = Variable(torch.FloatTensor(done))

    q_values      = model(state)
    next_q_values = model(next_state)

    q_value          = q_values.gather(1, action.unsqueeze(1)).squeeze(1)
    next_q_value     = next_q_values.max(1)[0]
    expected_q_value = reward + gamma * next_q_value * (1 - done)

    loss = (q_value - Variable(expected_q_value.data)).pow(2).mean()

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    return loss
```

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

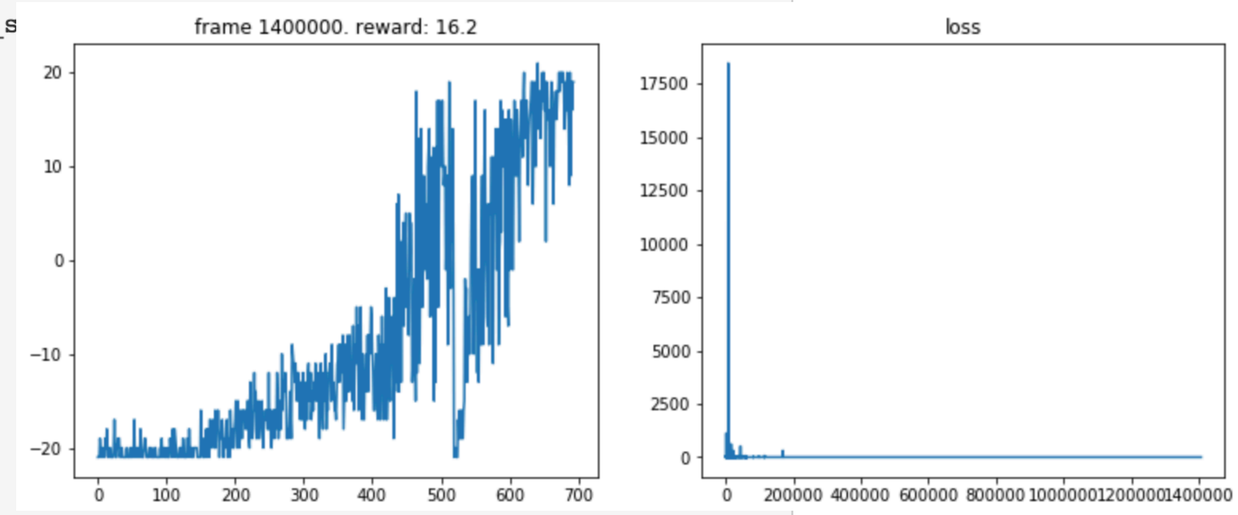
IMPLEMENTATION Training

```
num_frames = 1400000
batch_size = 32
gamma      = 0.99

losses = []
all_rewards = []
episode_reward = 0

state = env.reset()
for frame_idx in range(1, num_frames + 1):
    epsilon = epsilon_by_frame(frame_idx)
    action = model.act(state, epsilon)

    next_state, reward, done, _ = env.step(action)
    replay_buffer.push(state, action, reward, next_state, done)

    state = next_state
    episode_reward += reward

    if done:
        state = env.reset()
        all_rewards.append(episode_reward)
        episode_reward = 0

    if len(replay_buffer) > replay_initial:
        loss = compute_td_loss(batch_size)
        losses.append(loss.data[0])

    if frame_idx % 10000 == 0:
        plot(frame_idx, all_rewards, losses)
```

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

IMPLEMENTATION Training

```
num_frames = 1400000
batch_size = 32
gamma      = 0.99

losses = []
all_rewards = []
episode_reward = 0

state = env.reset()
for frame_idx in range(1, num_frames + 1):
    epsilon = epsilon_by_frame(frame_idx)
    action = model.act(state, epsilon)

    next_state, reward, done, _ = env.step(action)
    replay_buffer.push(state, action, reward, next_s

    state = next_state
    episode_reward += reward

    if done:
        state = env.reset()
        all_rewards.append(episode_reward)
        episode_reward = 0

    if len(replay_buffer) > replay_initial:
        loss = compute_td_loss(batch_size)
        losses.append(loss.data[0])

    if frame_idx % 10000 == 0:
        plot(frame_idx, all_rewards, losses)
```



https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb

DOUBLE DQN

MOTIVATION

- DQN의 문제:

$$Q(s,a) = r(s,a) + \gamma max_a Q(s',a)$$

Q-target Accumulated rewards Maximum Q-value of next state

Overestimating the action values. What if the environment is noisy?

Consider the classic problem in decision theory of having to choose between an envelope A which contains \$90.00 and envelope B which contains \$200.00 or \$0.00 with equal probability. Although $Var[A] \ll Var[B]$, our agent's ignorance of the bimodality of B would lead it to act in an over-optimistic fashion. Due to the $\max$ operator it would make a decision solely based on the fact that $\mathbb{E}[B] > \mathbb{E}[A]$.

van Hasselt H., Double Q-learning, NIPS, 2011

van Hasselt H., Guez A. and Silver D. Deep reinforcement learning with double Q-learning, AAAI, 2015

MOTIVATION

- DQN의 문제:

$$Q(s,a) = r(s,a) + \gamma max_a Q(s',a)$$

Q-target Accumulated rewards Maximum Q-value of next state

Overestimating the action values. What if the environment is noisy?

- 해결:

$$Q(s,a) = r(s,a) + \gamma Q(s', argmax_a Q(s',a))$$

DQN Network choose action for next state

van Hasselt H., Double Q-learning, NIPS, 2011

van Hasselt H., Guez A. and Silver D. Deep reinforcement learning with double Q-learning, AAAI, 2015

IMPLEMENTATION

```
current_model = CnnDQN(env.observation_space.shape, env.action_space.n)
target_model  = CnnDQN(env.observation_space.shape, env.action_space.n)

if USE_CUDA:
    current_model = current_model.cuda()
    target_model  = target_model.cuda()

optimizer = optim.Adam(current_model.parameters(), lr=0.0001)

replay_initial = 10000
replay_buffer = ReplayBuffer(100000)
```

Synchronize current policy net and target net

```
def update_target(current_model, target_model):
    target_model.load_state_dict(current_model.state_dict())

update_target(current_model, target_model)
```

https://github.com/higgsfield/RL-Adventure/blob/master/2.double%20dqn.ipynb

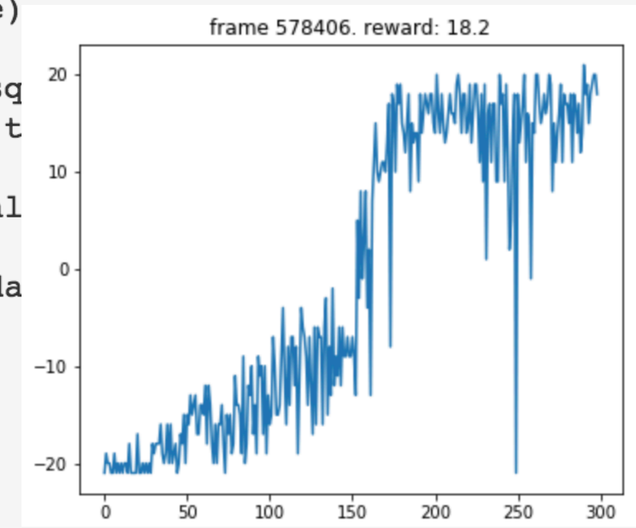
IMPLEMENTATION Computing Temporal Difference Loss

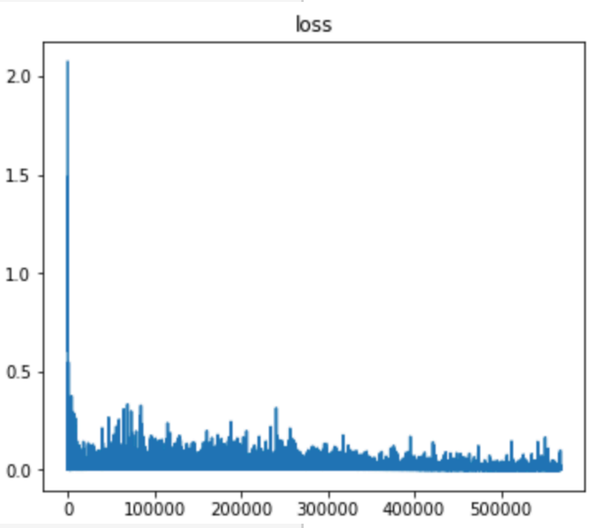
```
def compute_td_loss(batch_size):
    state, action, reward, next_state, done = replay_buffer.sample(batch_size)

    state      = Variable(torch.FloatTensor(np.float32(state)))
    next_state = Variable(torch.FloatTensor(np.float32(next_state)))
    action     = Variable(torch.LongTensor(action))
    reward     = Variable(torch.FloatTensor(reward))
    done       = Variable(torch.FloatTensor(done))

    q_values      = current_model(state)
    next_q_values = current_model(next_state)
    next_q_state_values = target_model(next_state)

    q_value       = q_values.gather(1, action.unsqueeze(1)).squeeze(1)
    next_q_value = next_q_state_values.gather(1, torch.max(next_q_values, 1)[1].unsqueeze(1)).squeeze(1)
    expected_q_value = reward + gamma * next_q_value * (1 - done)

    loss = (q_value - Variable(expected_q_value.data)).pow(2).mean()

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    return loss
```

https://github.com/higgsfield/RL-Adventure/blob/master/2.double%20dqn.ipynb

IMPLEMENTATION Computing Temporal Difference Loss

```
def compute_td_loss(batch_size):
    state, action, reward, next_state, done = replay_buffer.sample(batch_size)

    state      = Variable(torch.FloatTensor(np.float32(state)))
    next_state = Variable(torch.FloatTensor(np.float32(next_state)))
    action     = Variable(torch.LongTensor(action))
    reward     = Variable(torch.FloatTensor(reward))
    done       = Variable(torch.FloatTensor(done))

    q_values      = current_model(state)
    next_q_values = current_model(next_state)
    next_q_state_values = target_model(next_state)

    q_value       = q_values.gather(1, action.unsq
    next_q_value = next_q_state_values.gather(1, t
eze(1)
    expected_q_value = reward + gamma * next_q_val

    loss = (q_value - Variable(expected_q_value.da

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    return loss
```





https://github.com/higgsfield/RL-Adventure/blob/master/2.double%20dqn.ipynb

DUELING DQN

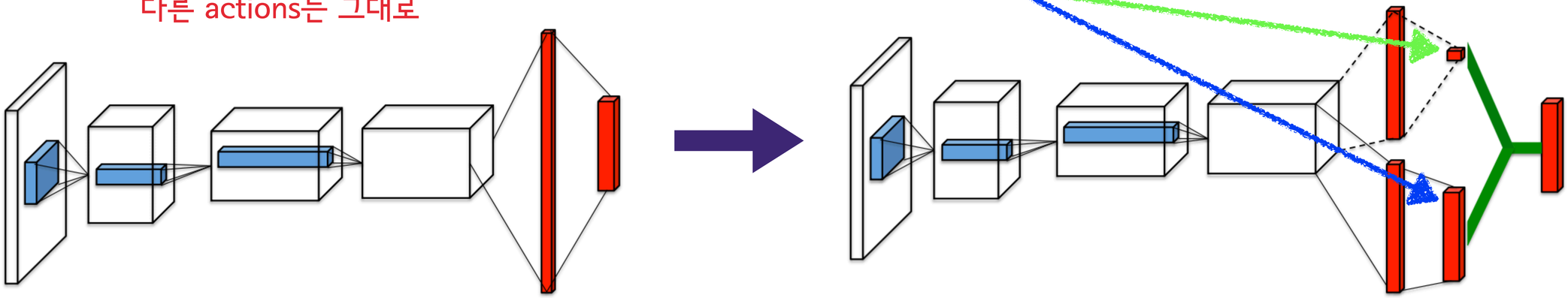
MOTIVATION

[Q-value decomposition] State value Advantage value

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

선택한 하나의 action보다
얼마나 더 좋은지(비교)를 나타낸다

하나의 action 값만 반영
다른 actions는 그대로



- 현재 state의 가치에 비교가치로 정보를 추가한다
- 가치의 차이(advantage value) —> 더 빠른 학습속도

IMPLEMENTATION

```
class DuelingCnnDQN(nn.Module):
    def __init__(self, input_shape, num_outputs):
        super(DuelingCnnDQN, self).__init__()


        self.input_shape = input_shape
        self.num_actions = num_outputs

        self.features = nn.Sequential(
            nn.Conv2d(input_shape[0], 32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU()
        )

        self.advantage = nn.Sequential(
            nn.Linear(self.feature_size(), 512),
            nn.ReLU(),
            nn.Linear(512, num_outputs)
        )

        self.value = nn.Sequential(
            nn.Linear(self.feature_size(), 512),
            nn.ReLU(),
            nn.Linear(512, 1)
        )
```

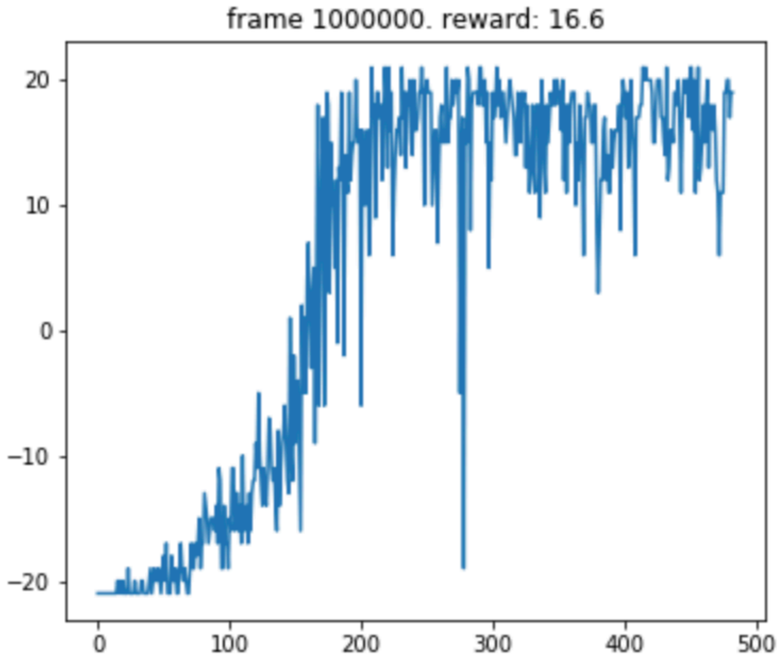
https://github.com/higgsfield/RL-Adventure/blob/master/3.dueling%20dqn.ipynb

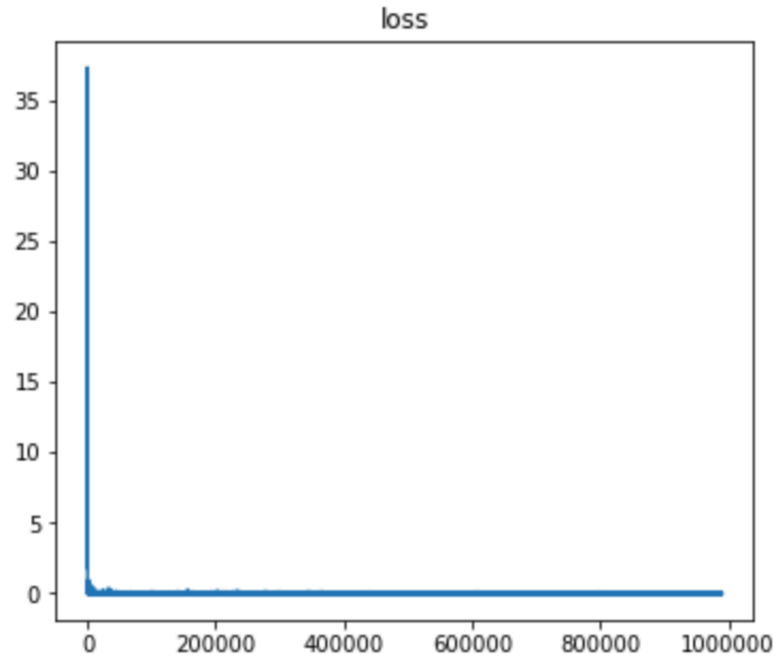
IMPLEMENTATION

```
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        advantage = self.advantage(x)
        value     = self.value(x)
        return value + advantage  - advantage.mean()

    def feature_size(self):
        return self.features(autograd.Variable(torch.zeros(1, *self.input_shape))).view(1, -1).siz
e(1)

    def act(self, state, epsilon):
        if random.random() > epsilon:
            state   = Variable(torch.FloatTensor(np.float32(state)).unsqueeze(0), volatile=True)
            q_value = self.forward(state)
            action  = q_value.max(1)[1].data[0]
        else:
            action = random.randrange(env.action_space.n)
        return action
```

https://github.com/higgsfield/RL-Adventure/blob/master/3.dueling%20dqn.ipynb

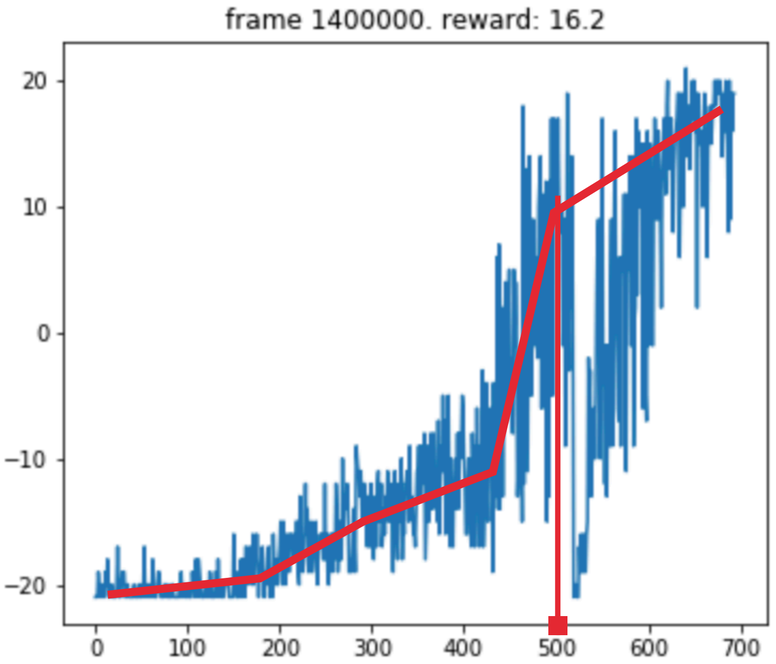
IMPLEMENTATION



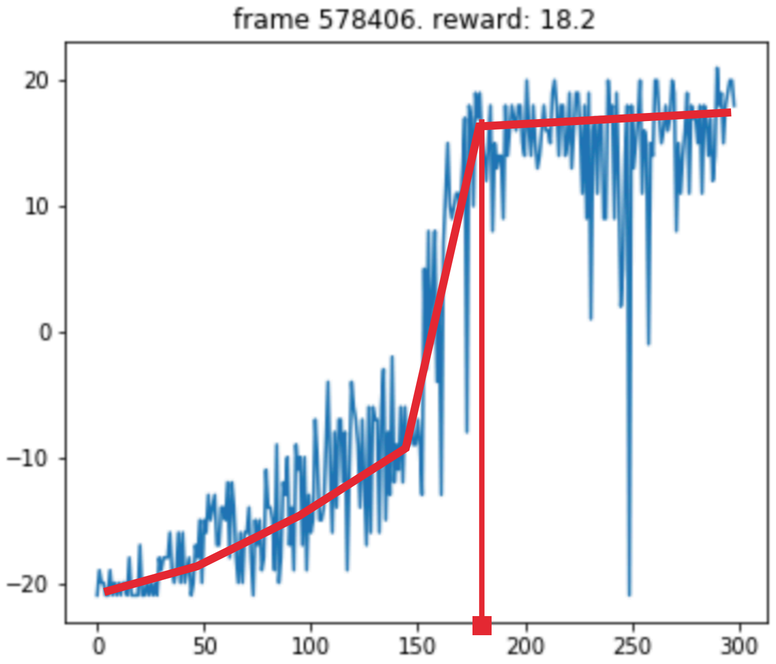


https://github.com/higgsfield/RL-Adventure/blob/master/3.dueling%20dqn.ipynb

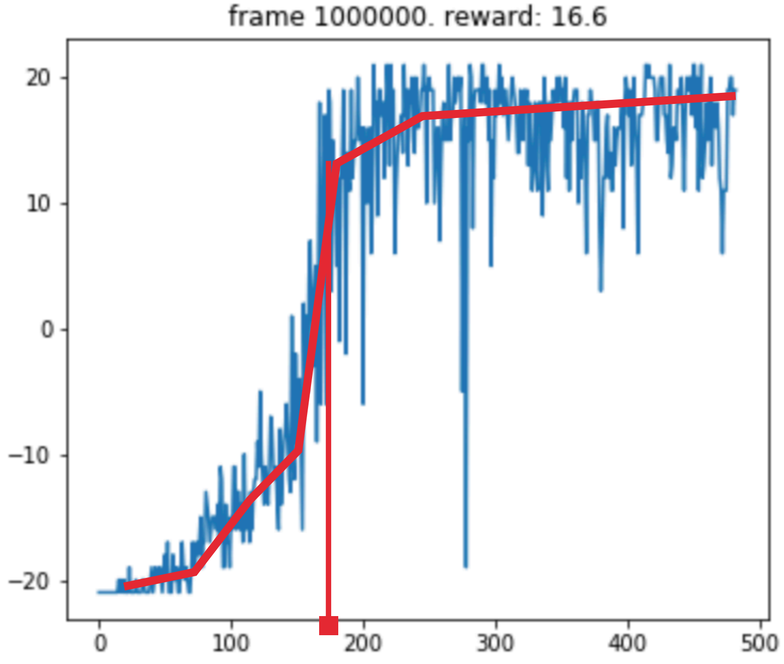
EXPERIMENTAL COMPARISON



[DQN]

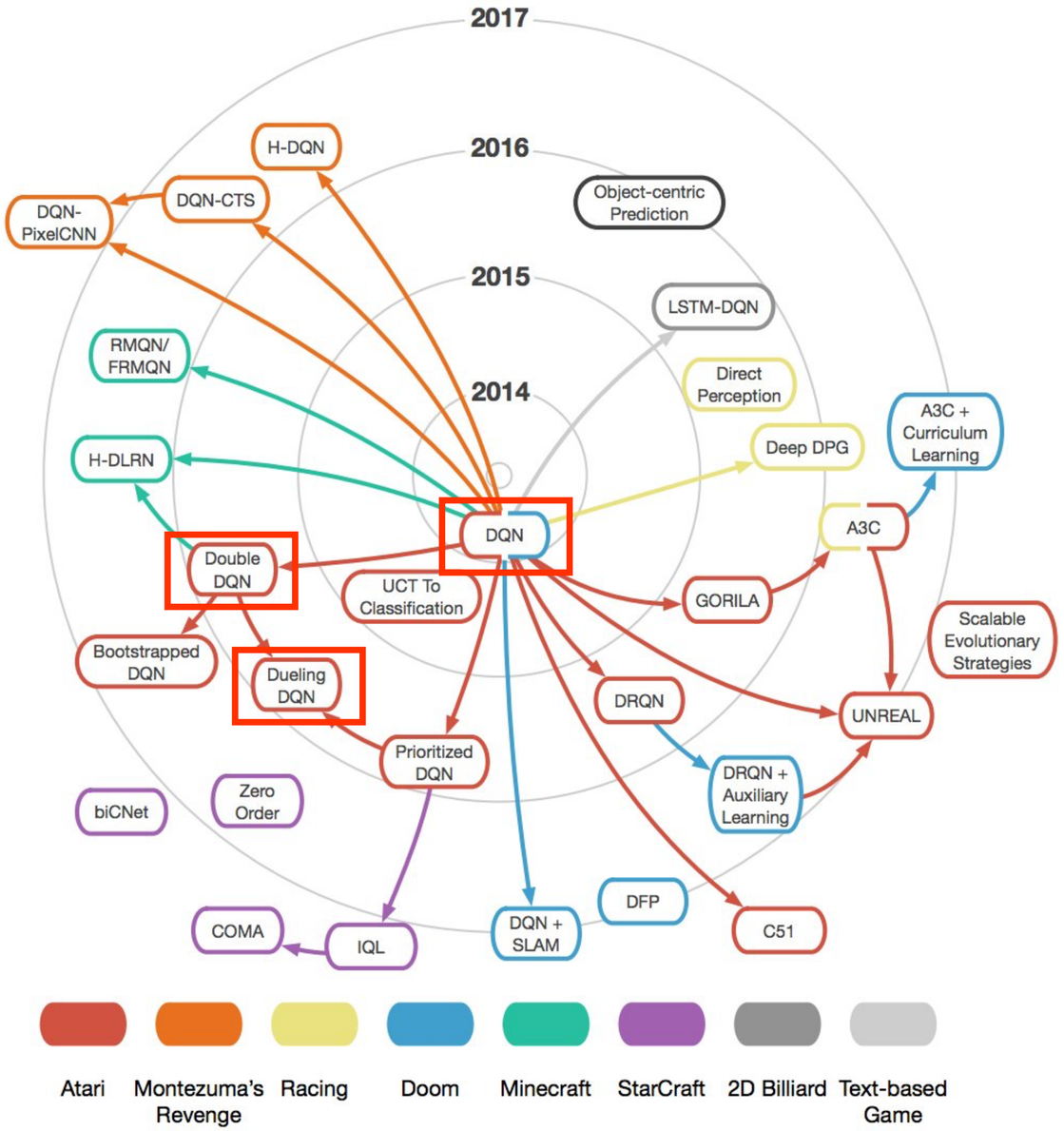


[Double DQN]



[Dueling DQN]

DQN LINEAGE



Niels Justesen, Philip Bontrager, Julian Togelius, Sebastian Risi. Deep Learning for Video Game Playing. 2017.

감사합니다