디자인 프로젝트 (2019년 9월 시작 과제)

강화학습을 이용한 빔트래킹

Beam Tracking using Deep Reinforcement Learning

-2019 Design Project-

December 5th, 2019

Digital Transmission Lab
Sogang University

Providing innovative solutions for the next-generation transmission technology.™

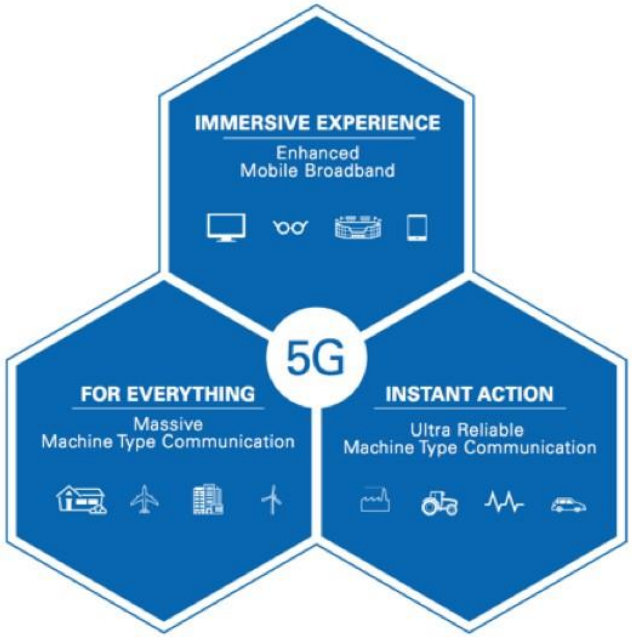
CONTENTS

- Introduction
- Concept
- Modeling
- Coding
- Conclusion
- Discussion

Theories & Concepts

1. 5G Technology

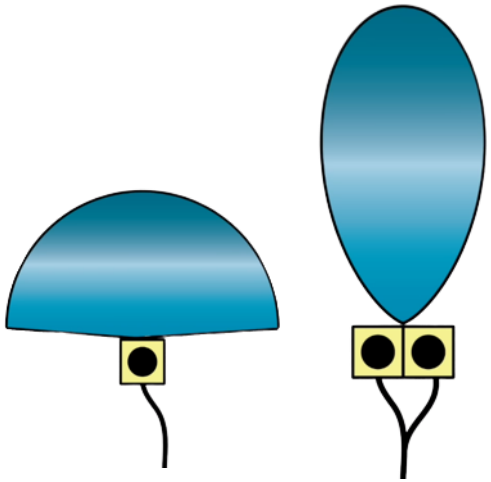
- 강화 무선 광대역(eMBB)은 20 Gbps의 peak data rate
- 6 GHz를 초과하는 넓은 가용 스펙트럼은 eMBB 사용 사례를 해결 가능한 대안
  - 6GHz 이후 mmWave 대역부터 매우 커지는 Propagation Loss
  - 파장이 짧아 송신 신호 포착량 감소
  - 넓은 대역폭으로 인한 thermal noise

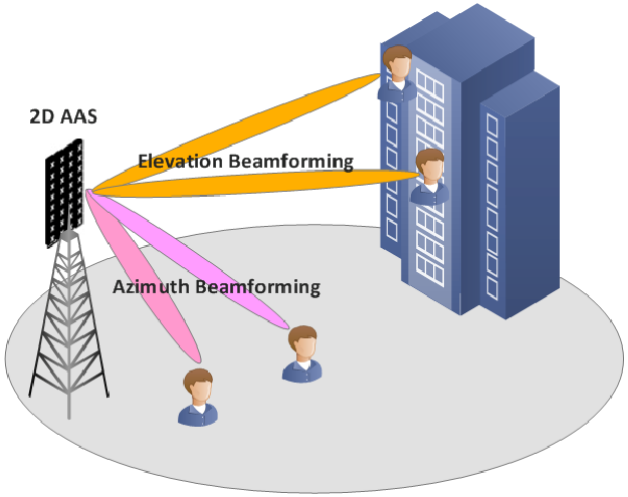


Theories & Concepts

2. Beamforming

- 기존에는 안테나와 같은 single radiating element에 input signal을 보내 통신
- mmWave를 사용하는 5G에서는 loss가 더 크기 때문에 multiple radiating element를 사용
- 높은 에너지를 이용해 더 멀리까지 통신하지만 에너지가 커지면서 형성되는 빔은 좁아짐
- 각 안테나에 같은 신호를 다른 phase를 입력하여 방향 조절





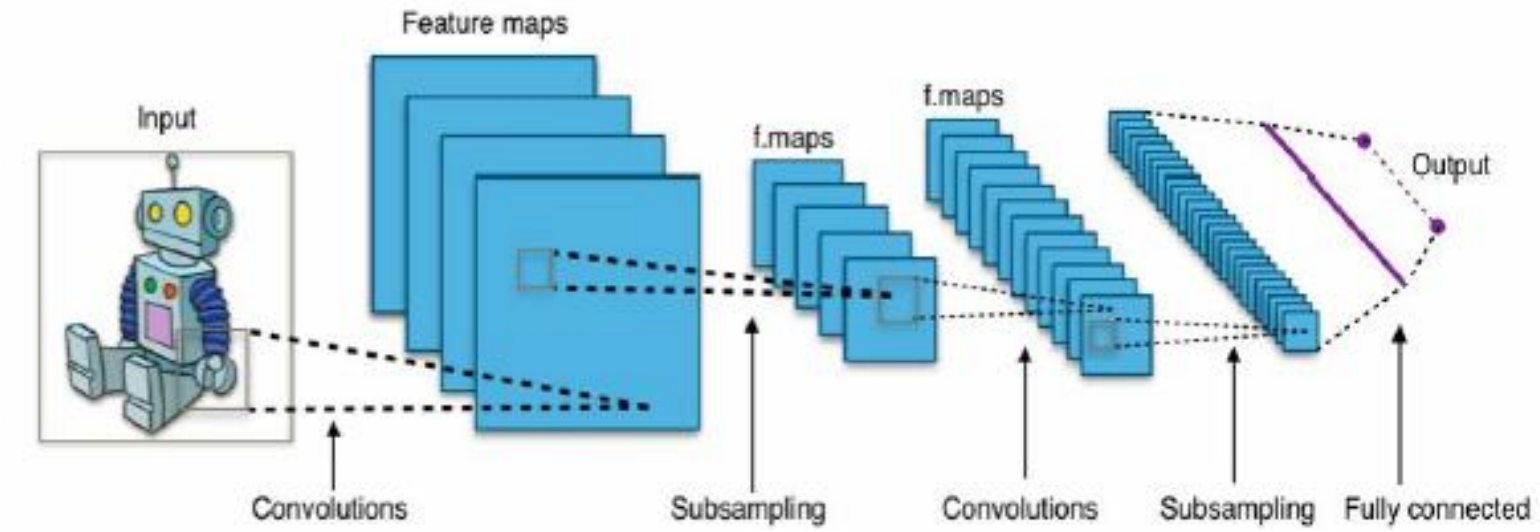
Theories & Concepts

3. Machine Learning – Convolutional Neural Networks

☐ Dense net (fully-connected neural network) 의 한계 :

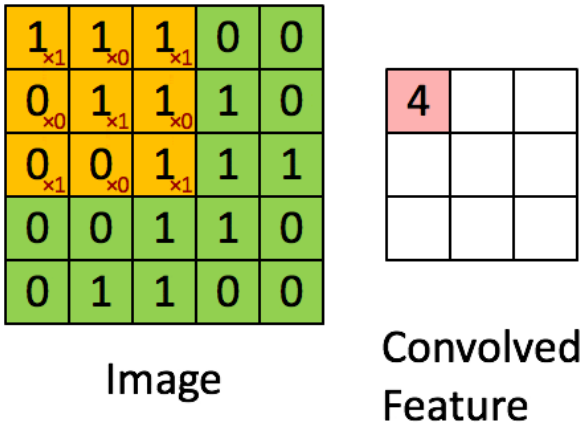
Spatial locality 를 활용하지 못함 -> 모든 지점을 같은 중요도로 생각

1) Parameter 가 커지는 문제
2) Overfitting 의 문제

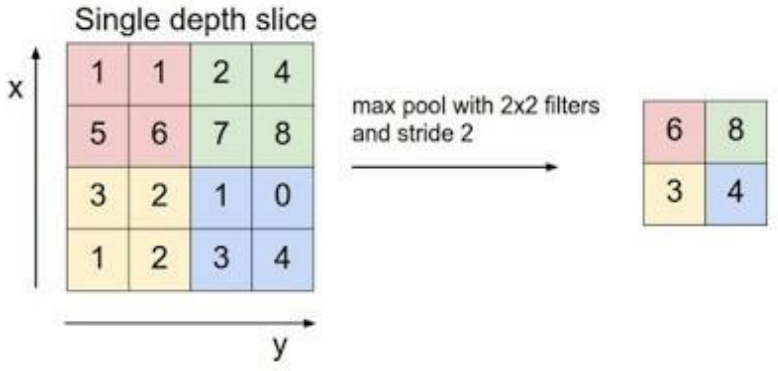


Theories & Concepts

- Convolution
  - 윈도우의 종류를 다르게 하여 여러 feature maps 생성
- Subsampling (MaxPooling)
  - 노이즈 감소, 영상 분별력 향상
  - 최종적으로는 신경망의 입력 단에 하나씩 mapping



Convolution



Subsampling

Theories & Concepts

4. Machine Learning – Reinforcement Learning

- 지도학습 : 각각의 트레이닝 예시를 위한 목표 라벨이 있음
- 비지도학습 : 목표 라벨이 아예 없음
- 강화학습 : 드문드문 시간이 지연되는 <Reward label> 가짐
  ⇒ Reward를 바탕으로 각각의 상황에서 어떻게 행동해야 할지 학습

Theories & Concepts

- Q-learning

Q-function을 최대화하는 것

Q-function: 상태 s에서 a라는 액션을 취할 때 maximum discounted future reward를 표현

주어진 상태에서 특정 행동의 quality를 표현

즉, 게임이 끝날 때 가장 높은 점수를 얻을 수 있는 행동을 선택하는 것

- Bellman equation

즉각적인 리워드 + 다음 상태에서의 미래의 리워드 최댓값

$$Q(s_t, a_t) = r + \gamma \, max_{a_{t+1}} \, Q(s_{t+1}, a_{t+1})$$

- approximation

: 이전의 Q value와 새로 제시된

α value의 차이를 컨트롤하는 학습률

```
initialize Q-table Q
observe initial state s
repeat
   select and carry out action a
   observe reward r and move to new state s'
   Q(s, a) = Q(s, a) + α(r + γ max_a' Q(s', a') - Q(s, a))
   s = s'
until game over
```

Theories & Concepts

5. RL example – Deepmind's Atari games

- 구글 딥마인드는 심층 인공지능 기술인 '심층 큐 네트워크'(Deep Q-network)를 독자적으로 개발

- 규칙을 알지 못하는 상태에서 점수와 픽셀 디스플레이를 정보로 활용하여 최고점을 만들기 위해 이전 게임 세션으로부터 학습하는 능력만을 갖춤

- 이를 통해 Atari 2600 비디오 게임을 플레이하는 법을 스스로 터득, 그 실력이 전문적인 게임 테스터의 실력과도 맞먹는 수준

Issues between RL and DL

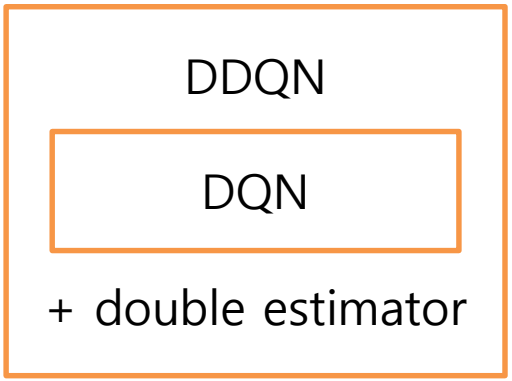
- 현실 세계에서 ML을 적용하기 위해선 반드시 high dimensional data를 다뤄야만 함.
- DL 분야에서 high dimensional data를 다루는 다양한 방법론들이 등장.
- 이 방법들을 RL에도 적용하려 하지만, issue들이 발생.

Issue 1. 성공적인 Deep Learning applications는 hand-labelled training data set을 요하는데, Reinforcement Learning에서는 오로지 reward를 통해 학습이 이루어지고, 그 reward도 드문드문하고 noisy 심지어는 delay되어 주어진다.

Issue 2. Deep Learning에서는 data sample이 독립 항등 분포를 가정하지만, Reinforcement Learning에서는 현재 state가 어디인지에 따라 갈 수 있는 다음 state가 결정되기때문에 state간의 correlation이 크다. 즉, data간의 correlation이 크다.

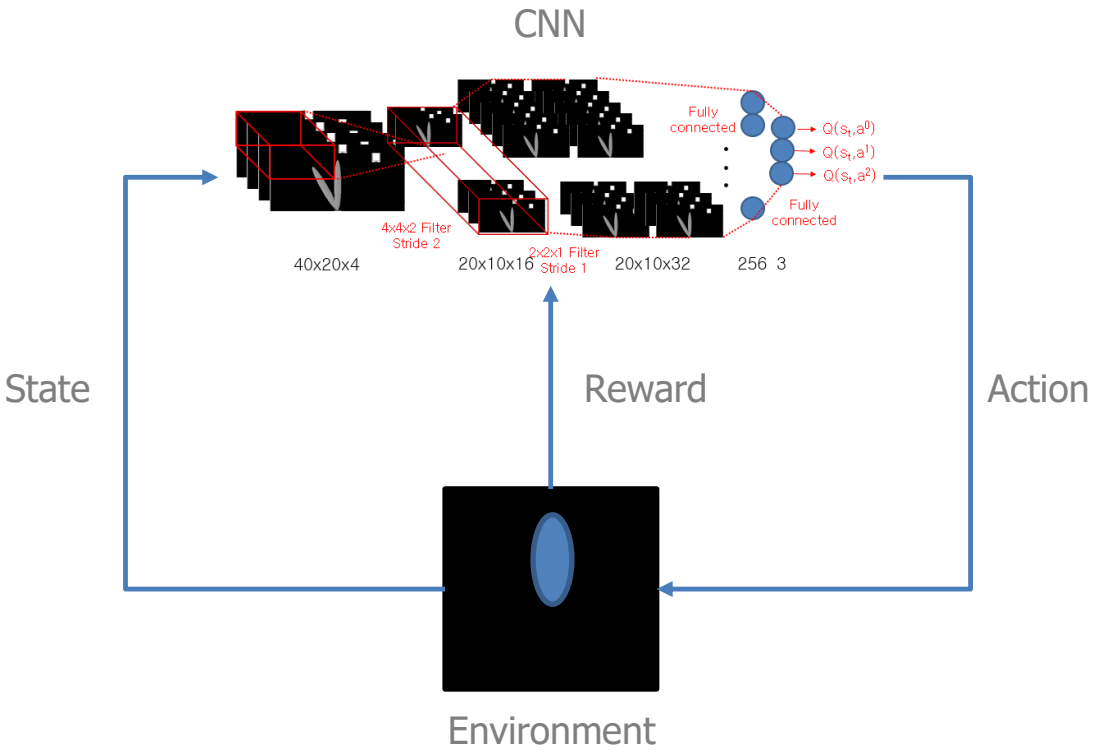
Solution: DDQN

- Issue들을 해결하기 위해 DDQN 사용
- DDQN = DQN 기능 + 특정 알고리즘



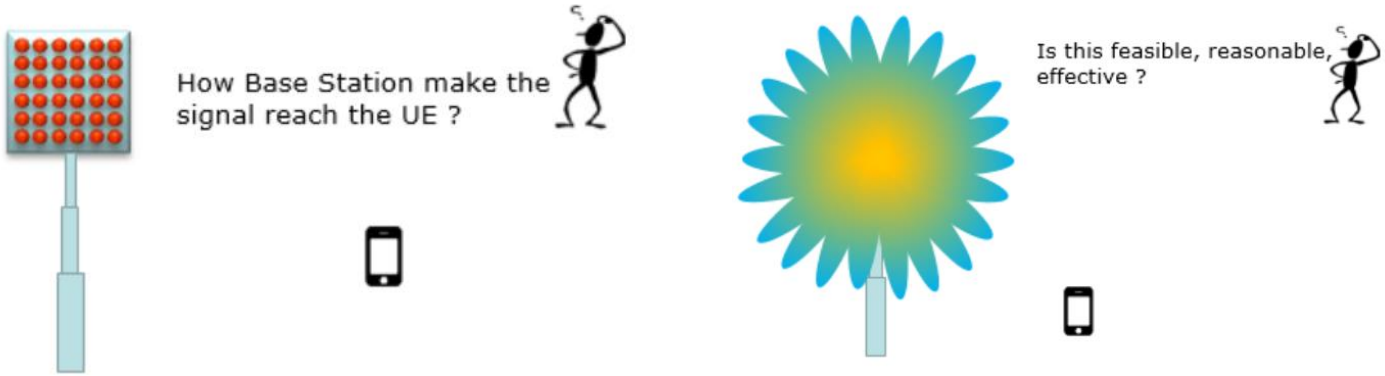
What is DQN?

- We refer to convolutional networks trained with our approach as Deep Q-Networks (DQN).
- The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards.

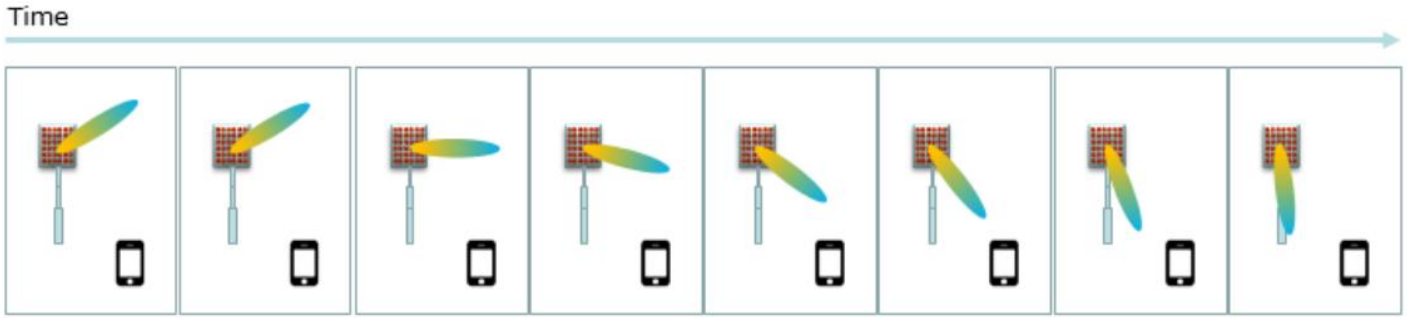


Issue

- The signal beam can point to a very narrow area and it cannot cover a very wide area at the same time.



- The base station transmit the beam to a specific direction at a specific time
  and then change the direction a little bit in a next time frame
  and so on until it can scan all the area it should cover.



Simplification – step by step

- Step 1: 게임 시간

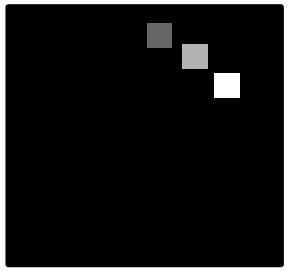
일정 시간 (ex. 32 slots)마다 시나리오 종료

  - 한 epoch은 32프레임, 첫 프레임에 유저는 랜덤위치에 생성
  - Beam은 90°에 생성

- Step 2: 유저의 움직임

유저는 이동하는 상황이며 일정 시간마다 영상데이터로 위치 파악

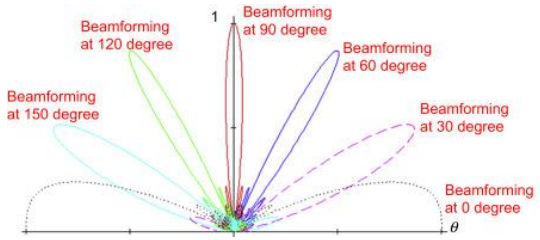
  - 1 action / 4 frames
  - 세 가지 중 하나의 action (Left, Pause, Right) 을 행함.
  - 32번째 프레임에서는 Pause.



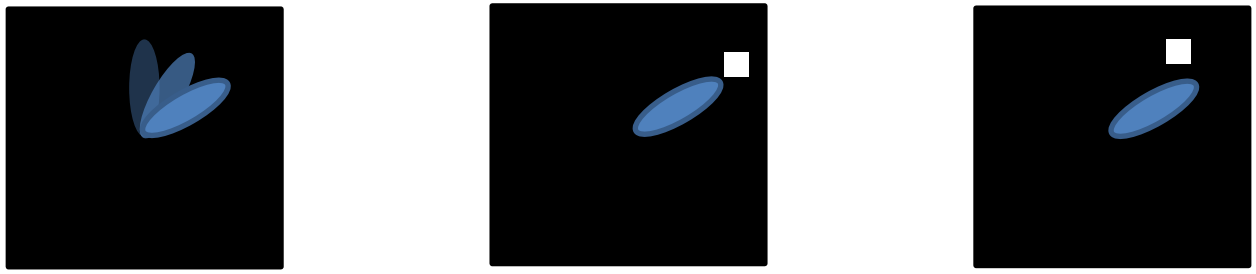
<게임 화면 _ 유저의 움직임>

Simplification – step by step

- Step 3: 빔의 움직임
  - 30°~150° 내에서 Beam과 유저의 움직임 구현
  - 빔이 유저를 트래킹
  - 1 action / 1 frame
  - 세 가지 중 하나의 action (Left, Pause, Right) 을 행함.



- Step 4: 리워드
  - 마지막 32 frame에 리워드 지급 (트래킹 성공 : +1, 실패 : -1)
  - positive feedback(+1): 점수화
  - Negative feedback(-1): 학습 피드백으로 사용



<게임 화면 _ 빔의 움직임, positive feedback, negative feedback>

Settings

Easy move mode　Normal move mode　Hard move mode

- Others
  - Beam과 유저가 한번에 움직이는 angle을 변경해가면서 난이도를 조절.

  (Easy : 30°, Normal : 15°, Hard : 10°, Very Hard : 5°)
- Scoring

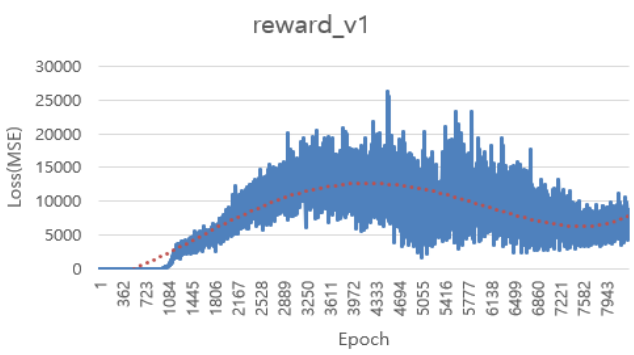
Positive reward를 받을 때 마다 win count가 올라가고 epoch에 대한 win count의 비율로 성능을 평가.

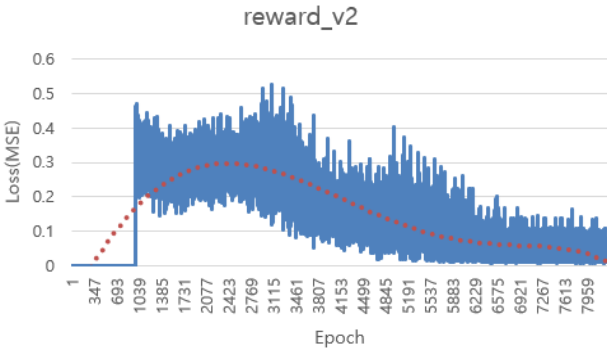
- Reward function

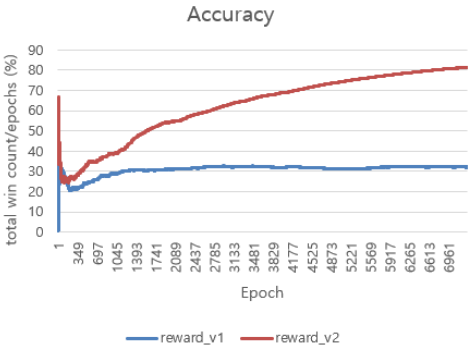
맨 마지막에만 positive/negative reward를 부여했을 때 가장 좋은 성능.

미래의 reward가 매 frame마다 고려되기 때문.

- reward_v1 : 매 frame마다 reward 부여, reward_v2 : 마지막 frame에만 reward 부여







Model

- CNN (Convolution Neural Network) in code

Layer	Input	Num filters	Filter size	Stride	Activation	Output	Num Parameter
Conv1	40x20x4	16	4x4	2	ReLU	20x10x16	1,040
Conv2	20x10x16	32	2x2	1	ReLU	20x10x32	2,080
Conv3	20x10x32	32	2x2	1	ReLU	20x10x32	4,128
Flatten	20x10x32					6,400	
Fc4	12800				ReLU	256	1,638,656
Fc5	256				Linear	3	771

```
# build the model
model = Sequential()
model.add(Conv2D(16, kernel_size=4, strides=2,
                 kernel_initializer="normal",
                 padding="same",
                 input_shape=(40, 20, 4)))
model.add(Activation("relu"))
model.add(Conv2D(32, kernel_size=2, strides=1,
                 kernel_initializer="normal",
                 padding="same"))
model.add(Activation("relu"))
model.add(Conv2D(32, kernel_size=2, strides=1,
                 kernel_initializer="normal",
                 padding="same"))
model.add(Activation("relu"))
model.add(Flatten())
model.add(Dense(256, kernel_initializer="normal"))
model.add(Activation("relu"))
model.add(Dense(3, kernel_initializer="normal"))
#model.load_weights('')
model.compile(optimizer=Adam(lr=1e-6), loss="mse")
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 20, 10, 16)	1040
activation_1 (Activation)	(None, 20, 10, 16)	0
conv2d_2 (Conv2D)	(None, 20, 10, 32)	2080
activation_2 (Activation)	(None, 20, 10, 32)	0
conv2d_3 (Conv2D)	(None, 20, 10, 32)	4128
activation_3 (Activation)	(None, 20, 10, 32)	0
flatten_1 (Flatten)	(None, 6400)	0
dense_1 (Dense)	(None, 256)	1638656
activation_4 (Activation)	(None, 256)	0
dense_2 (Dense)	(None, 3)	771

Total params: 1,646,675
Trainable params: 1,646,675
Non-trainable params: 0

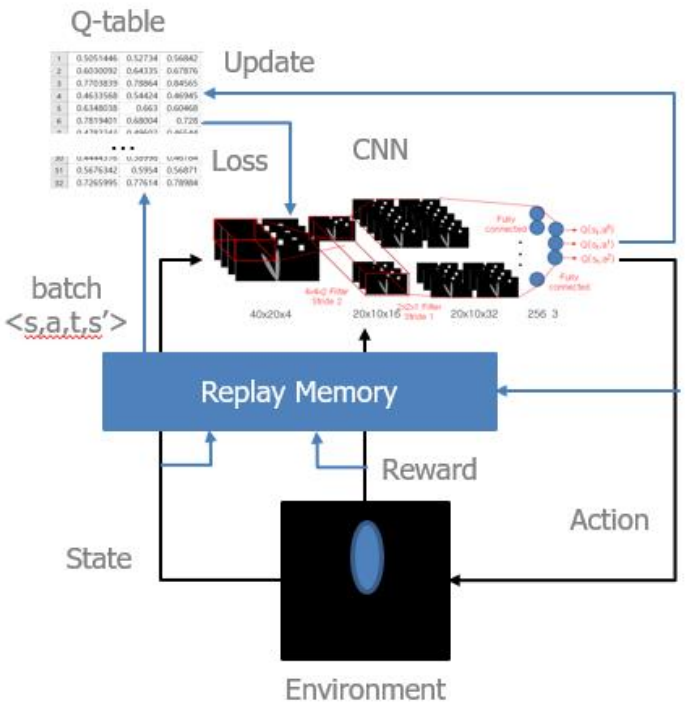
Skills of DQN

1. Experience Replay (Replay Memory)
2. Preprocessing (Resize, gray scale, extraction)
3. Exploration & Exploitation (ε -greedy)
4. Fixed-Q target

Skills of DQN

1. Experience Replay (Replay Memory)

- Experience (st,at,rt,st+1)를 step마다 D라는 memory에 저장한다.
- 이 data set D로부터 mini batch를 구성하여 학습을 진행한다.
- Mini batch는 순차적인 데이터로 구성되지 않으므로 입력 데이터 간의 correlation을 줄일 수 있다.




```
# store experience
experience.append((s_tm1, a_t, r_t, s_t, game_over))
# finished observing, now start training
# get next batch
X, Y = get_next_batch(experience, target_model, model, NUM_ACTIONS,
                      GAMMA, BATCH_SIZE)

loss = model.train_on_batch(X, Y)
```

```
def get_next_batch(experience, model, num_actions, gamma, batch_size):
    batch_indices = np.random.randint(low=0, high=len(experience),
                                      size=batch_size)
    batch = [experience[i] for i in batch_indices]
    X = np.zeros((batch_size, 40, 20, 4))
    Y = np.zeros((batch_size, num_actions))
    for i in range(len(batch)):
        s_t, a_t, r_t, s_tp1, game_over = batch[i]
        X[i] = s_t
        Y[i] = model.predict(s_t, batch_size = batch_size)[0]
        Q_sa = np.max(model.predict(s_tp1)[0])
        if game_over:
            Y[i, a_t] = r_t
        else:
            Y[i, a_t] = r_t + gamma * Q_sa
    return X, Y, Q_sa
```

Terminologies

Q-value function (Action-value function)

Definition

The *return* G_t is the total discounted reward from time-step t.

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

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

$$\Rightarrow Q^\pi(s,a) = r + \gamma Q^\pi(s', \pi(s'))$$

Loss function

$$Loss = \frac{1}{3}[\underbrace{r + \gamma max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s,a)}_{\text{prediction}}]^2$$

r = 현재 reward
γ = 차감 변수(0.9)
s = 현재 state
a = 현재 action
s' = 다음 state
a' = 다음 action

Concept

Loss function in code

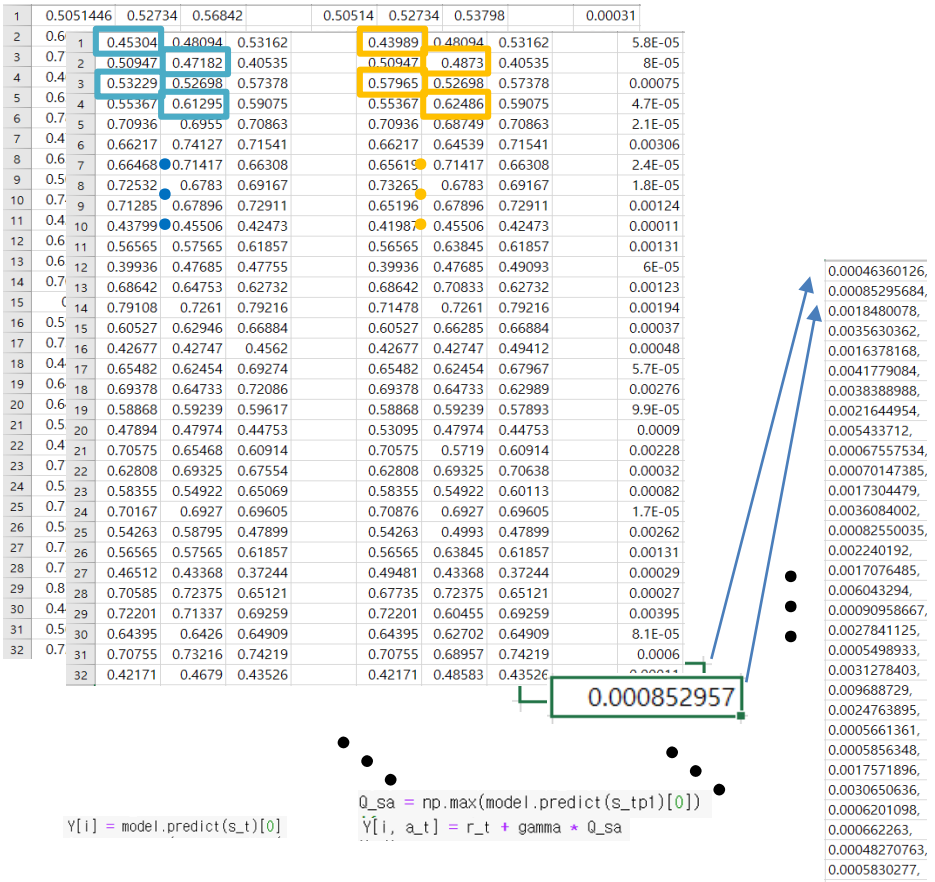
```
q = model.predict(s_t)[0]
a_t = np.argmax(q)
```

```
X, Y = get_next_batch(experience, target_model, model, NUM_ACTIONS,
                      GAMMA, BATCH_SIZE)

loss = model.train_on_batch(X, Y)                                    #
```

```
def get_next_batch(experience, model, num_actions, gamma, batch_size):
    batch_indices = np.random.randint(low=0, high=len(experience),
                                      size=batch_size)
    batch = [experience[i] for i in batch_indices]
    X = np.zeros((batch_size, 40, 20, 4))
    Y = np.zeros((batch_size, num_actions))
    for i in range(len(batch)):
        s_t, a_t, r_t, s_tp1, game_over = batch[i]
        X[i] = s_t
        Y[i] = model.predict(s_t, batch_size = batch_size)[0]
        Q_sa = np.max(model.predict(s_tp1)[0])
        if game_over:
            Y[i, a_t] = r_t
        else:
            Y[i, a_t] = r_t + gamma * Q_sa
    return X, Y, Q_sa
```

Experience(Batch_size) [MSE]



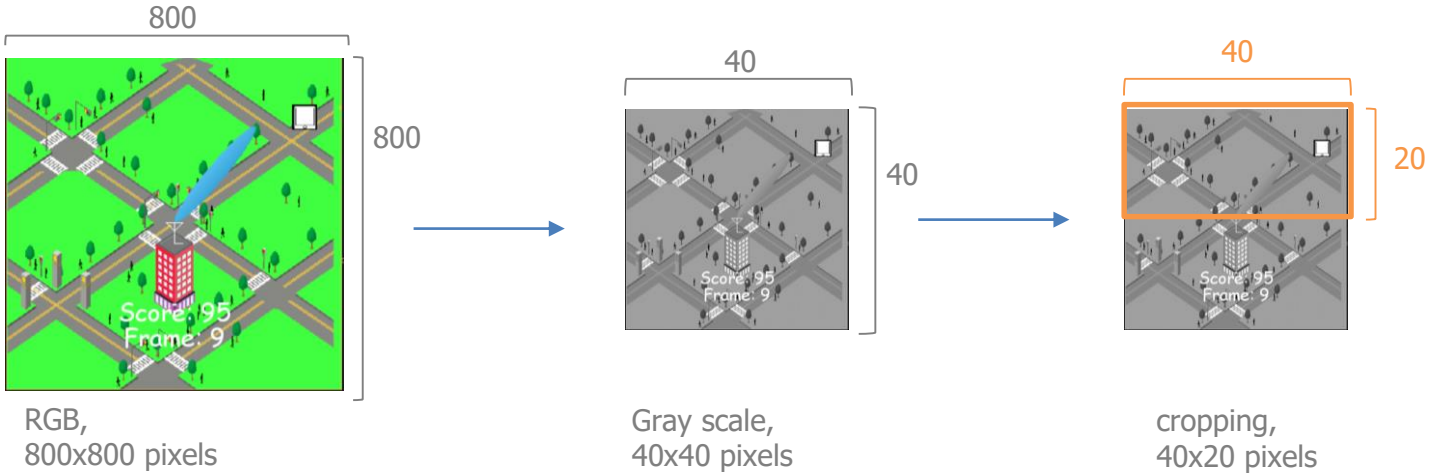
$$Loss = \frac{1}{3}[\underbrace{r + \gamma max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

Loss of 2nd Epoch : 0.06937164976261556

Skills of DQN

2. Preprocessing

1) Resize



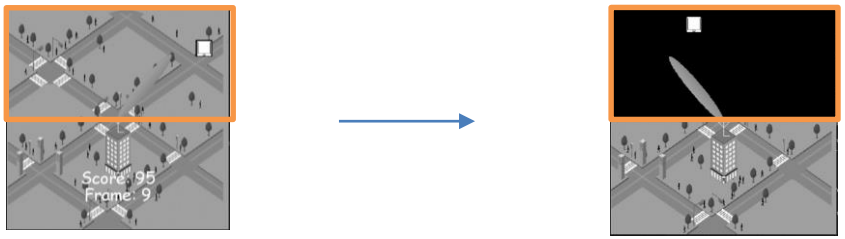
RGB,
800x800 pixels

Gray scale,
40x40 pixels

cropping,
40x20 pixels

```
x_t = np.array(PIL.Image.fromarray(x_t).resize((40, 40)).convert("L"))
x_t = x_t[0:40,0:20].copy()
```

2) Extraction



Skills of DQN

3. Exploration & Exploitation

☐ Exploration-exploitation

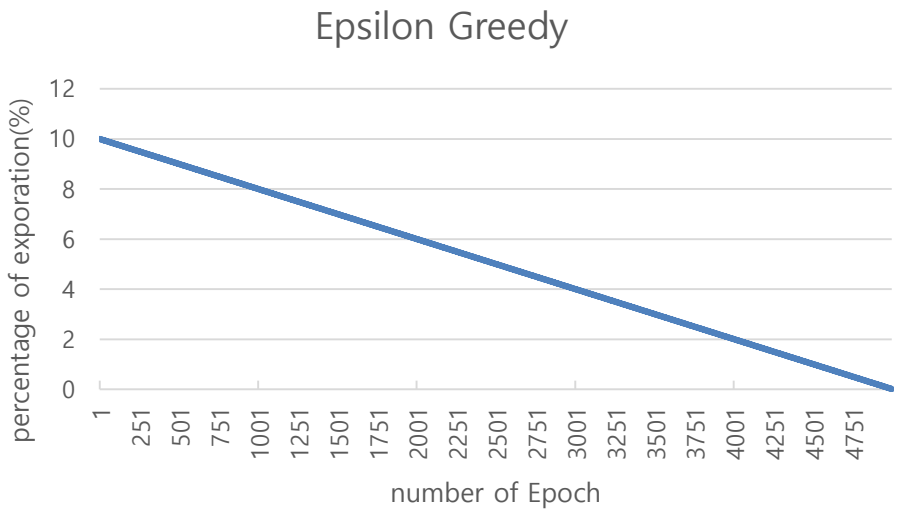
이미 알고 있는 방법을 고수할 것인지(exploit)

가능한 다른 방법들을 찾아 볼 것인지(explore)

```
INITIAL_EPSILON = 0.1 # starting value of epsilon
FINAL_EPSILON = 0.0001 # final value of epsilon
```

```
if np.random.rand() <= epsilon:
    a_t = np.random.randint(low=0, high=NUM_ACTIONS, size=1)[0]
else:
    q = model.predict(s_t)[0]
    a_t = np.argmax(q)
```

```
if epsilon > FINAL_EPSILON:
    epsilon -= (INITIAL_EPSILON - FINAL_EPSILON) / NUM_EPOCHS
```



Skills of DQN

4. Fixed Q-targets

- 문제점 : Q 함수의 타겟과 추정치가 동일한 네트워크에 있기 때문에 안정적인 학습이 어려움.

당근의 위치 당나귀의 움직임

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)$$

r=현재 reward
γ =차감 변수(0.9)
s =현재 state
a =현재 action
s'=다음 state
a'=다음 action
θ_t = *local net.*
θ_t^- = non updated net.

- Q 함수의 타겟을 설정하는 네트워크(Target Network)와 Q 함수를 추정하는 네트워크(Local Network)를 분리한다..
- Target network는 tau 마다 update (tau는 hyper parameter, 우리는 50epoch으로 설정.)

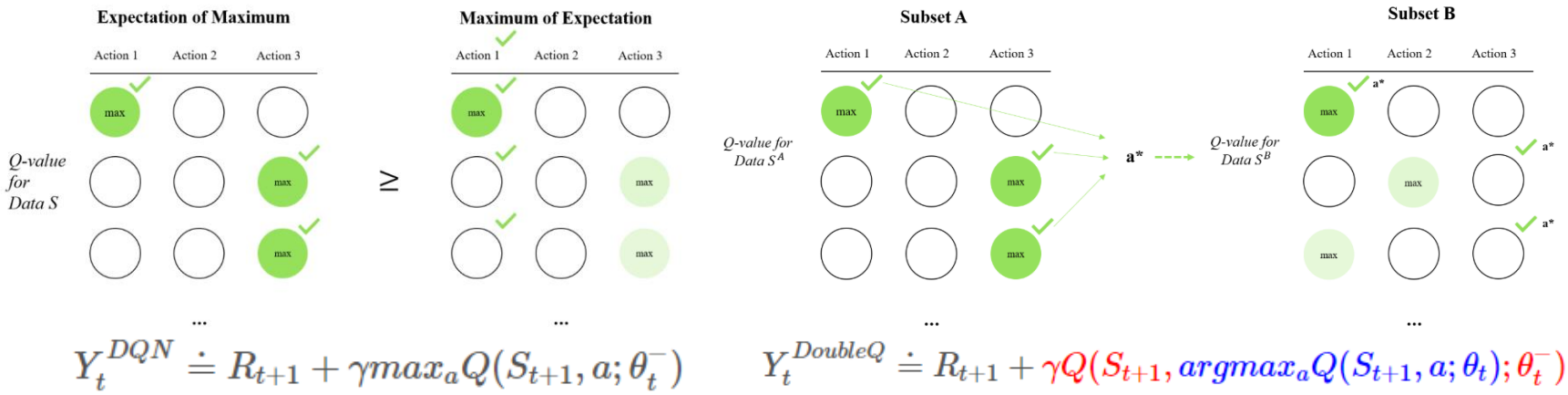
$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-)$$

- BUT, DQN의 추정법은 overestimation이 되기 쉽다.

Skill of DDQN(Double DQN)

5. Double Estimator

- Q-learning 의 목표 : $max_{a'}\{E(Q(s',a')\}$, 대신에 : $\mathrm{E}\{max_{a'}Q(s',a')\}$ 를 사용(DQN)
- 여기서 overestimation 발생 : ($\mathrm{E}\{max_{a'}Q(s',a')\} \geq max_{a'}\{E(Q(s',a')\}$ 이기 때문에.)
- 따라서 double estimator를 사용(DDQN)
- Separate selection and estimation



$$Y_t^{DQN} \doteq R_{t+1} + \gamma max_a Q(S_{t+1}, a; \theta_t^-)$$

$$Y_t^{DoubleQ} \doteq R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta_t^-)$$

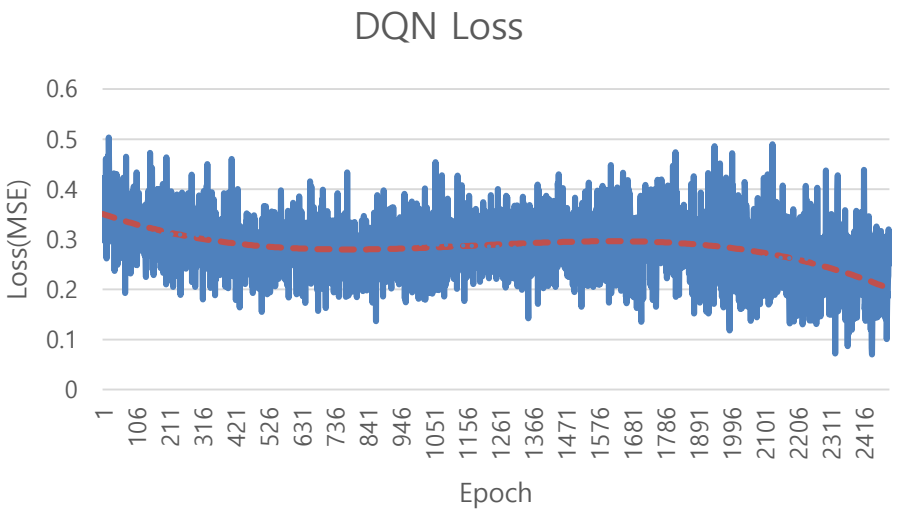
```
Q_sa = np.max(model.predict(s_tp1)[0])
```

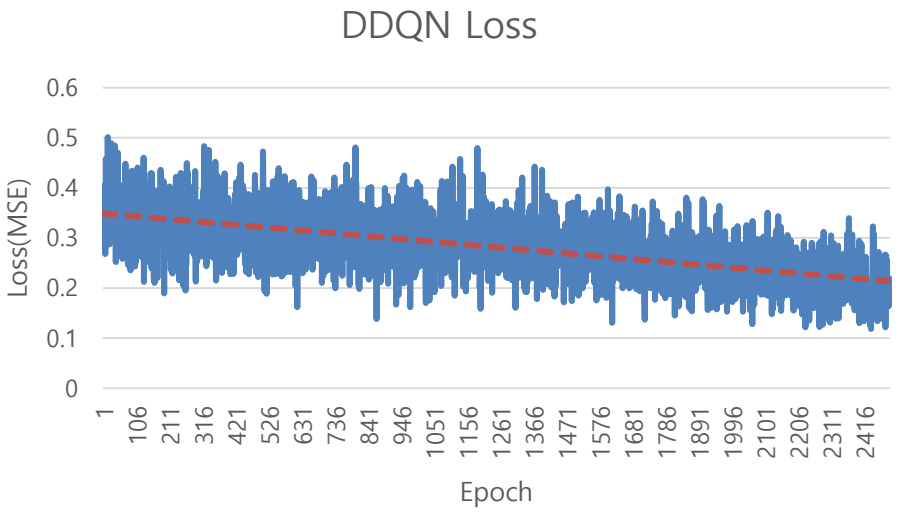
```
Y[i, a_t] = r_t + gamma * Q_sa
```

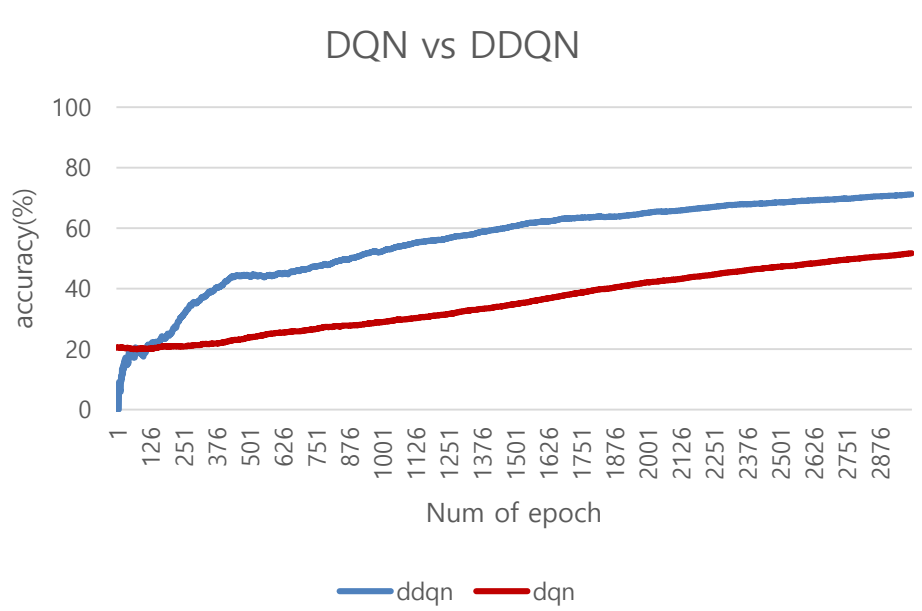
```
f_a_t = np.argmax(model.predict(s_tp1)[0])
Q_sa = target_model.predict(s_tp1)[0][f_a_t]
```

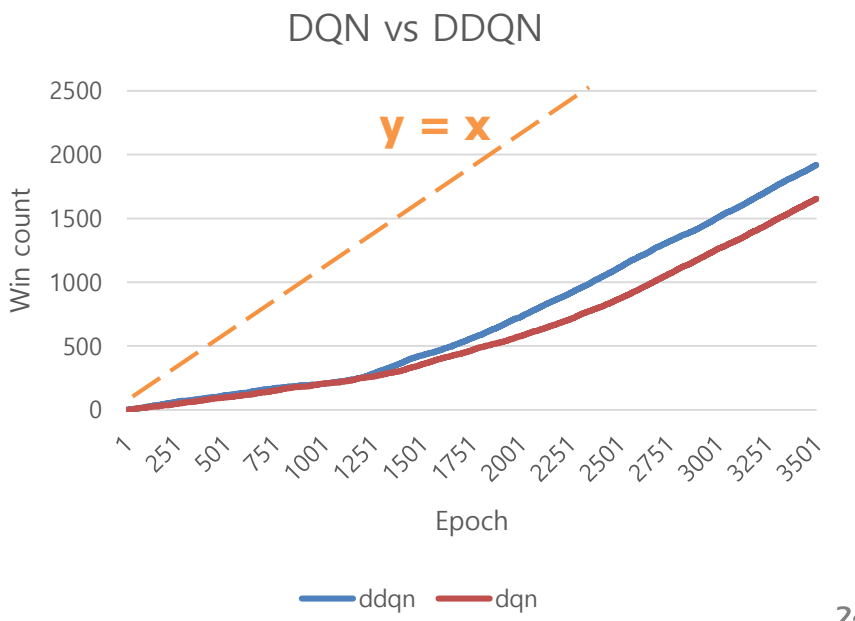
```
Y[i, a_t] = r_t + gamma * Q_sa
```

DDQN(Double Deep Q Net.)





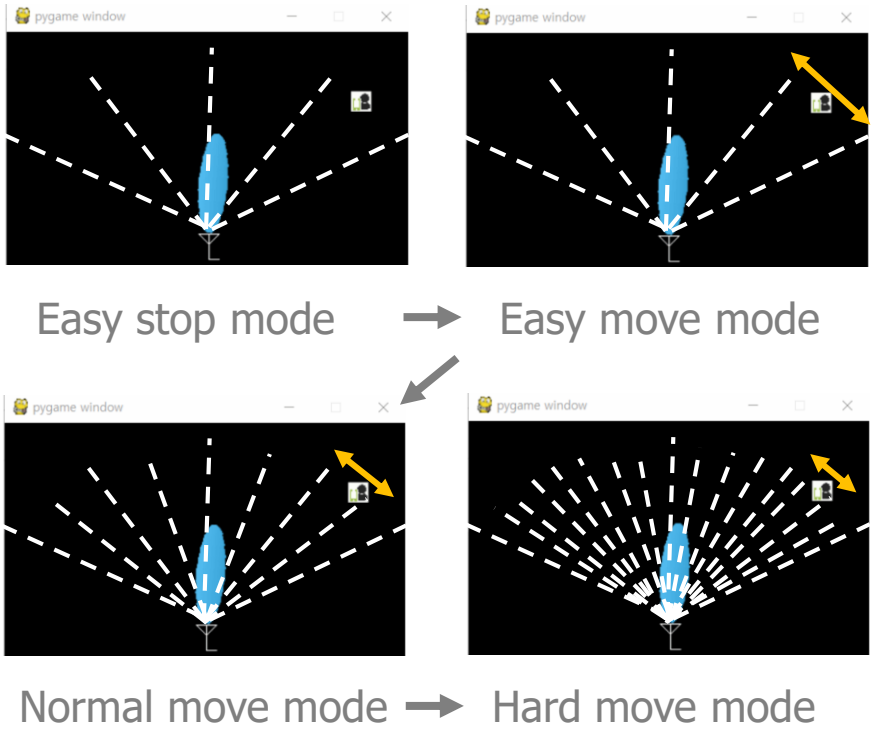


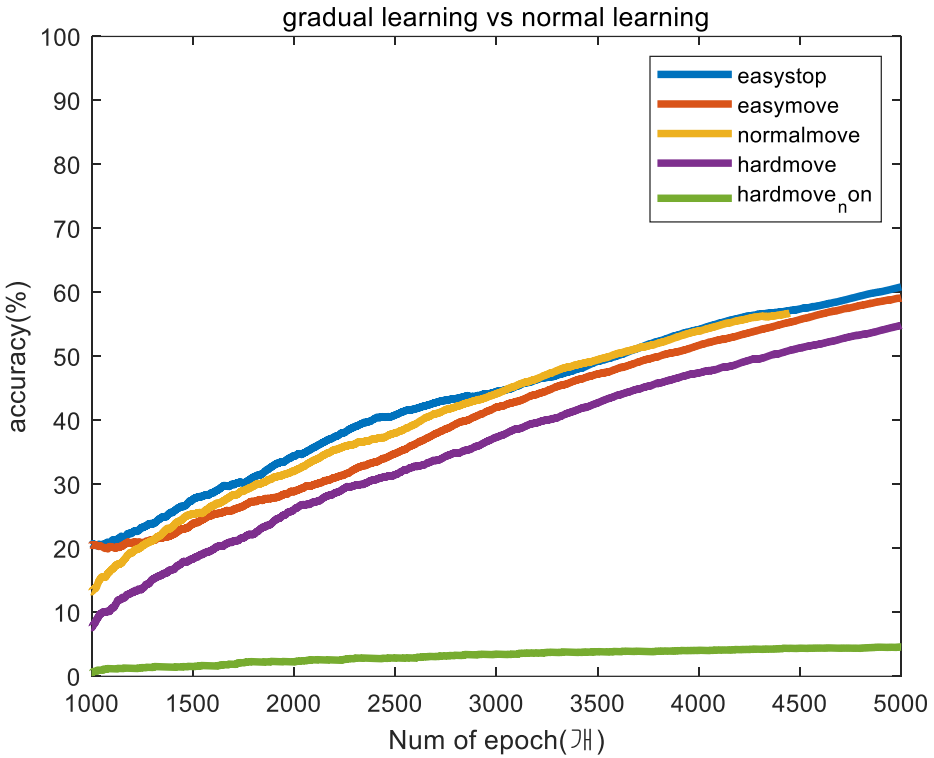


Our Differentiation

Gradually increasing the difficulty

- As game is getting difficult, learning is too late and unstable
- Load the weights of easier model
- x5 faster than the normal training, more stable learning



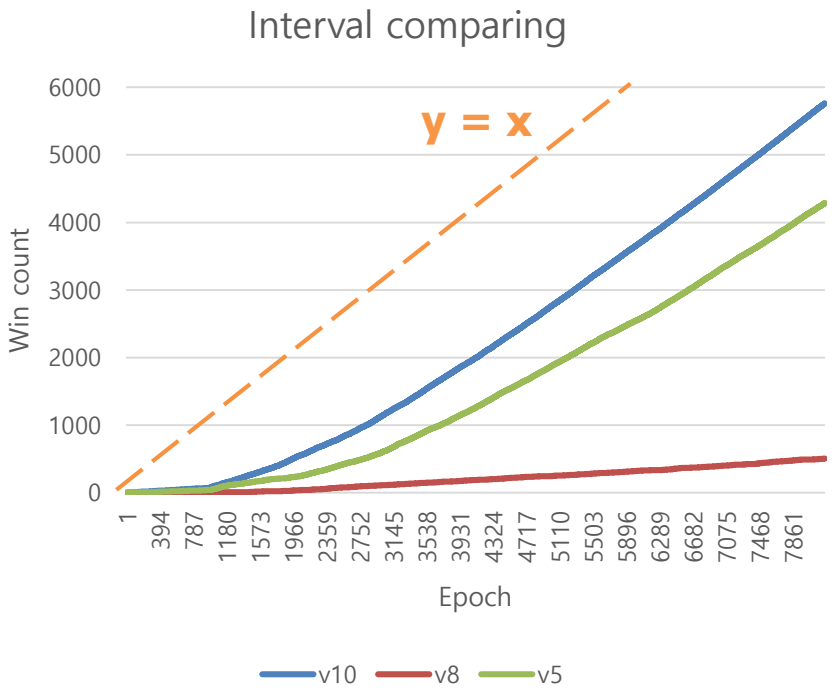


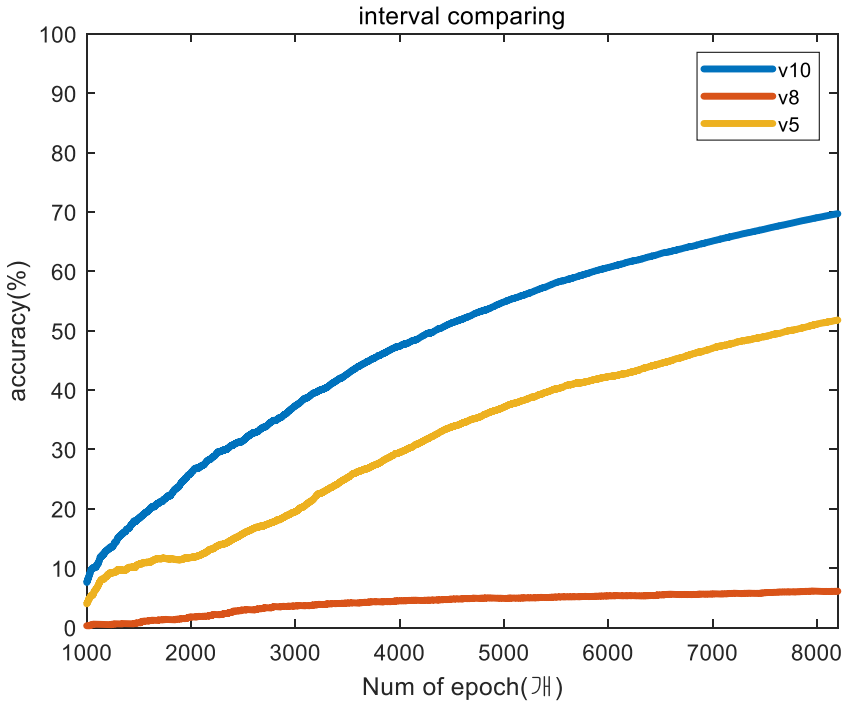
Our Differentiation

Gradually increasing the difficulty

- 주의할 점 : overlap되는 state들이 있어야 성능 개선이 됨
- 세 가지 model 모두 15도 간격으로 움직이는 normal mode model의 weights를 load. But,

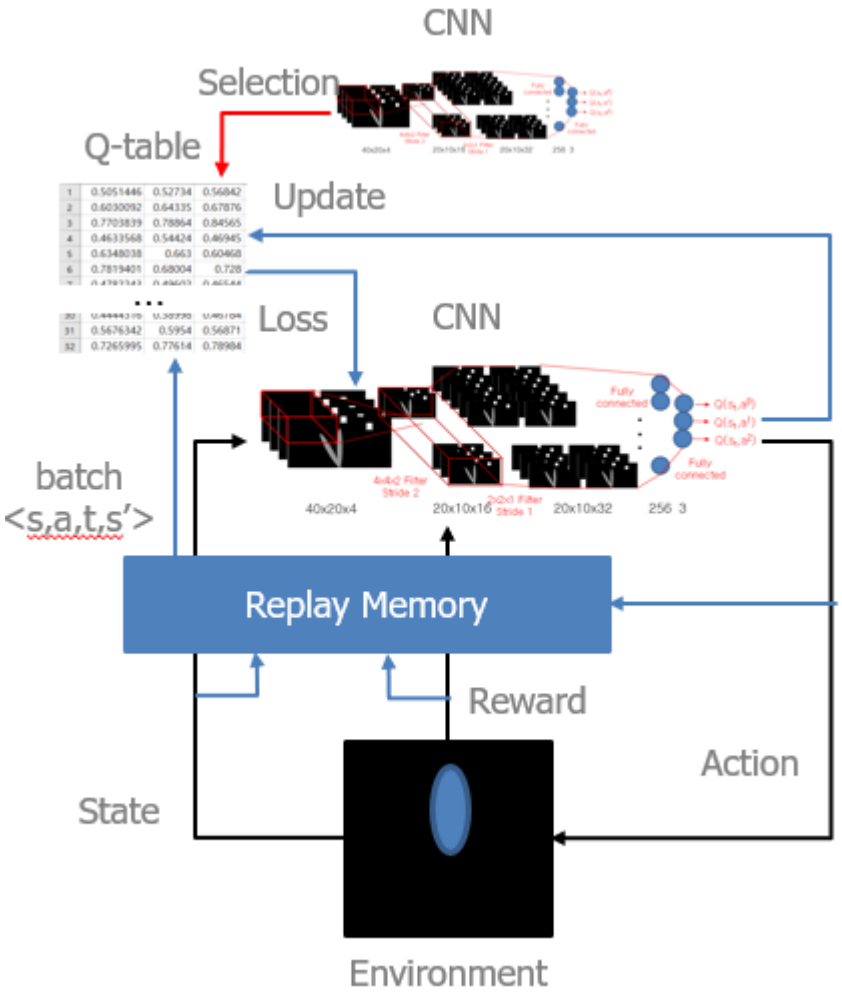
성능 : 10도 간격 > 5도 간격 >> 8도 간격

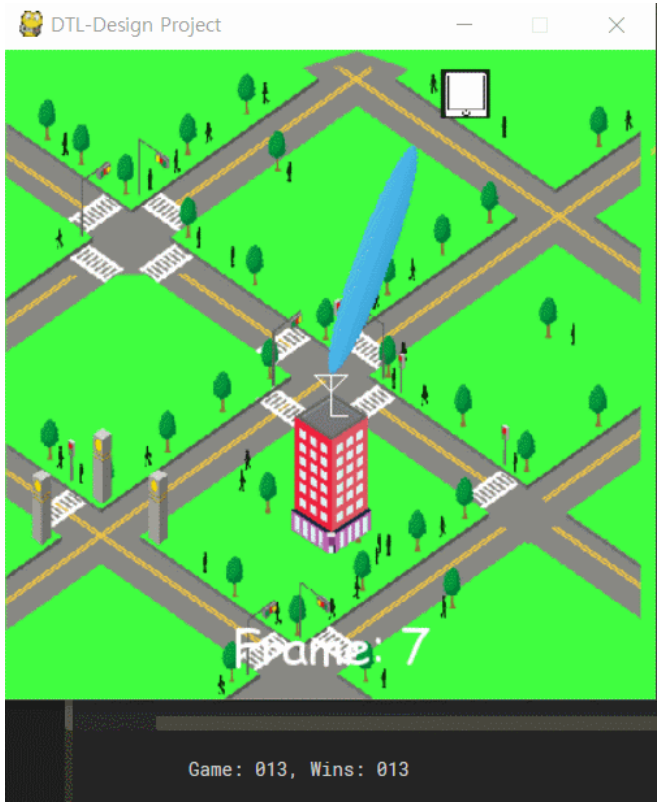




Structure

- 최종 코드 구조 설명
- 코드 실행 시 작동





Summary

- DDQN : 영상 데이터 입력, 유저의 확률적 이동에 따른 Overestimation 방지
- Training Model : CNN
  - 40x20x4 입력 이미지를 3개의 Conv. Layers를 사용
  - Activation function : Relu. / Optimizer : Adam.
- 점진적인 학습
  - 이전 학습 모델 로드
  - Resolution 5 -> 12 -> 24 (이동 각도는 30도 -> 10도 -> 5도) 로 증가하면서 학습
  - ➢ 더 많아진 위치 관계 중 이전학습에서와 겹치는 경우가 있어야 효과적

Above projects

- 점진적 학습을 통한 MU

: 유저와 빔의 수를 늘리는 방식으로 학습