

메디치소프트 기술연구소

딥러닝10일차

강화 학습(Reinforcement Learning)

- . 1950년대부터 시작 게임과 기계 제어 분야에서 관심을 끄는 애플리케이션 많이 나옴
- . 2013년 영국의 한 스타트업인 딥마인드의 연구원들이 아타리(Atari) 게임을 아무 정보 없이 플레이하면서 학습하는 시스템을 시연하면서 혁명이 일어남
- . 화면 픽셀에 대한 데이터만 입력, 게임 규칙에 대한 어떤 사전정보 없이 대부분 사람을 능가
- . 2017년 5월 알파고가 바둑 세계 챔피언 커제에게 승리
- . 구글은 2014년에 딥마인드를 5억달러 인수
- . 정책 그래디언트, 심층 Q-네트워크(DQN), 마르코프 결정 과정(MDP) 소개

What is Positive Reinforcement Dog Training?

- Teaching dogs desirable behaviors using SCIENCE-based & REWARD-based methods.
- Helping dogs learn and succeed step by step.
- Motivating dogs with fun exercises and games. No force! No pain!
- Encouraging dogs to think more for themselves.
- Valuing dogs' voluntary behaviors.
- Understanding dogs' feelings from their body language.
- Understanding how dogs learn, their needs and wants.
- Using methods that work humanely with ANY dog. Big dogs, small dogs, puppies, senior dogs, disabled dogs, fearful dogs, reactive dogs... can all learn and have fun!



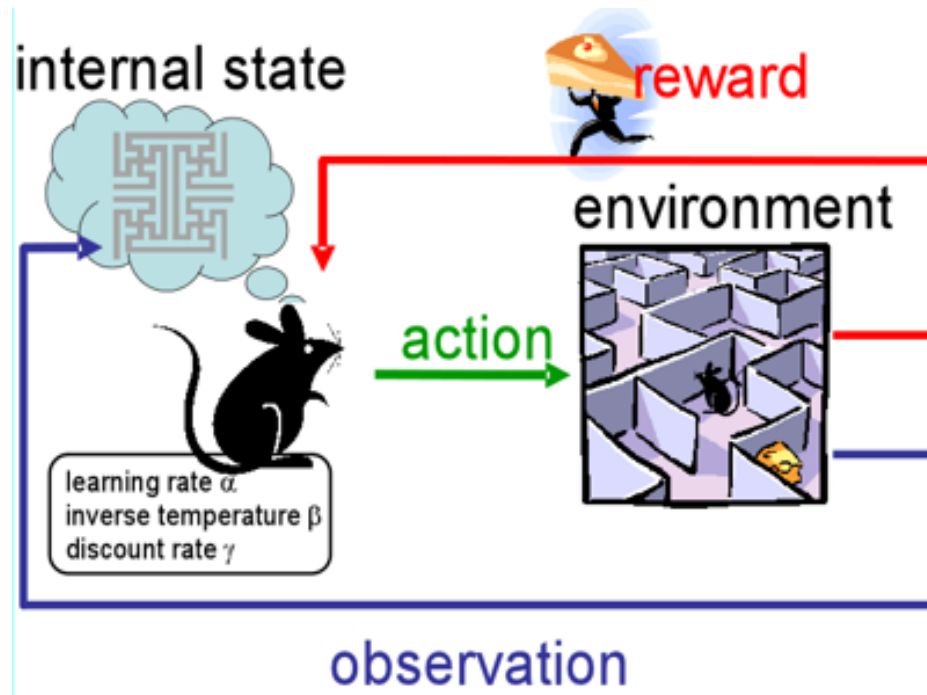
1. develop
dog's self-control

2. develop
a trust relationship

3. develop
dog's self-confidence

<http://angelpawstherapy.org/positive-reinforcement-dog-training.html>

강화 학습(Reinforcement Learning)



- . 에이전트(agent)는 관측(observation)을 하고 주어진 환경(environment)에서 행동(action)을 합니다.
=> 결과로 보상(reward)을 받습니다.
- . 에이전트는 환경 안에서 행동하고 시행착오를 통해 기쁨이 최대가 되고 아픔이 최소가 되도록 학습

<https://www.cs.utexas.edu/~eladlieb/RLRG.html>

강화 학습(Reinforcement Learning)

- . 머신러닝의 한분야로 주어진 환경에서 시간이 지남에 따라 보상이 최대화되는 행동을 할 수 있는 에이전트를 만드는 것을 목적으로 한다.
- . 지도학습과 비지도 학습의 목적은 데이터에 있는 패턴을 찾아 이를 사용해 예측을 만드는 것
- . 강화 학습의 목적은 좋은 정책을 찾는 것
- . 지도학습과 달리 에이전트에 올바른 정답이 명시적으로 주어지지 않는다.
- . 에이전트는 시행착오를 통해 학습
- . 비지도 학습과 달리 보상을 통한 감독의 형태가 존재
- . 에이전트에 어떻게 작업을 수행하라고 알려주지 않지만 일을 잘했는지 또는 실패 여부 알려줌
- . 강화 학습 에이전트는 보상을 얻기 위해 새로운 방식을 찾는 환경의 탐험과 이미 알고 있는 보상 방법 활용하는 것 사이에 적절한 균형을 가져야 함.
- . 이와 반대로 지도 학습과 비지도 학습 시스템은 탐험에 대해 신경을 쓸 필요가 없음.
즉 주어진 훈련데이터만 주입
- . 지도학습과 비지도 학습에서 훈련 샘플은 일반적으로 독립적
- 강화 학습에서는 연속된 관측이 보통 독립적이지 않다
- 에이전트가 잠시 동안 움직이지 않고 환경의 같은 영역에 머물러 있을 수 있다.
- 연속된 관측은 매우 상호 연관이 되어 있다.
- 어떤 경우에는 훈련 알고리즘이 독립적인 관측을 얻을 수 있도록 재현 메모리 사용

<https://www.cs.utexas.edu/~eladlieb/RLRG.html>

강화 학습(Reinforcement Learning)

. 소프트웨어 에이전트가 행동을 결정하기 위해 사용하는 알고리즘: 정책(Policy)

. Ex: 관측을 입력으로 받고 수행할 행동을 출력하는 신경망이 정책



. 정책: 매 초마다 어떤 확률 p 만큼 전진하는 것도 있고,
또는 $(1-p)$ 의 확률로 왼쪽 또는 오른쪽으로 랜덤하게 회전

. 회전 각도: $-r$ 과 $+r$ 사이의 랜덤한 각도:

확률적 정책(stochastic policy)

.=> 2개의 정책 파라미터 policy parameter:

확률 p 와 각도의 범위 r 알고리즘 선택?

. Open AI 짐(<https://gym.openai.com>)

- 에이전트가 훈련시키기 위해 먼저 작업 환경을 마련
- 훈련을 위한 최소한의 시뮬레이션 환경 필요
- 다양한 종류의 시뮬레이션 환경(아타리 게임, 보드 게임, 2D와 3D 물리 시뮬레이션 등)을 제공하는 툴
- 에이전트를 훈련시키고 이들을 비교 또는 새로운 RL 알고리즘을 개발

강화 학습(Reinforcement Learning) -OpenAI GYM Games

Frozen Lake



Environment

Action
(right, left,up,down)

State, reward

Agent

. GYM:환경을 만들어 주는 Open AI

강화 학습(Reinforcement Learning) -OpenAI GYM Games

.설치 방법

- Python
- TensorFlow
 - `sudo apt-get install python-pip python-dev`
 - `pip install tensorflow (or pip install tensorflow-gpu)`
- OpenAI Gym
 - `sudo apt install cmake`
 - `apt-get install zlib1g-dev`
 - `sudo -H pip install gym`
 - `sudo -H pip install gym[atari]`

강화 학습(Reinforcement Learning) -OpenAI GYM Games 환경

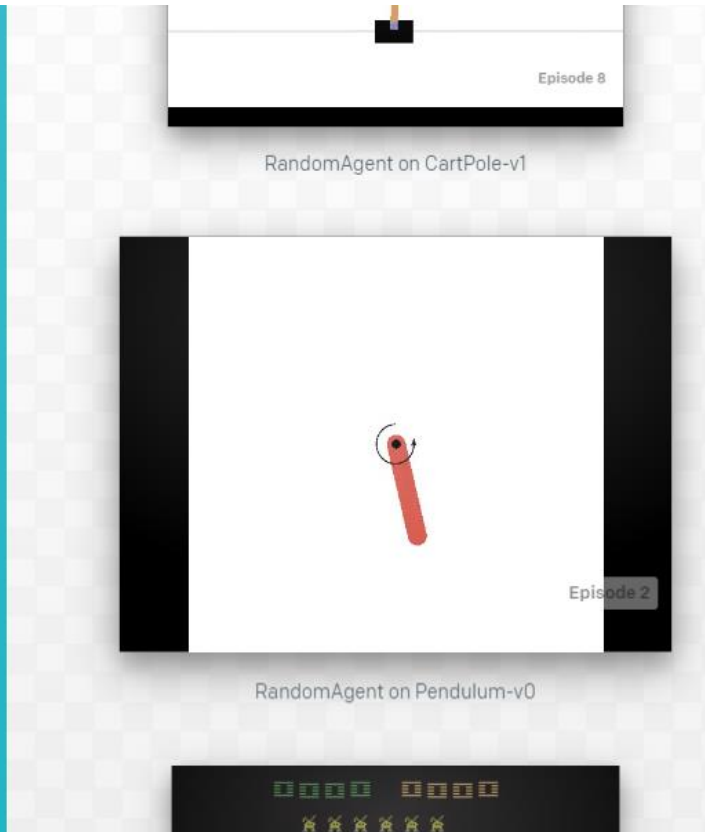
<https://gym.openai.com/>



Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

[View documentation >](#)

[View on GitHub >](#)



강화 학습(Reinforcement Learning) -OpenAI GYM Games 환경

```
import gym
from gym.envs.registration import register
# http://stackoverflow.com/questions/510357/python-read-a-single-character-from-the-user
import readchar # pip3 install readchar
env = gym.make("Taxi-v2") #환경 생성(정해진 이름)
observation = env.reset() #환경 초기화
for _ in range(1000): #action을 1000번을 취한다
    env.render() #출력
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
```

```
import ...
env = gym.make("FrozenLake-v0")
observation = env.reset()
for _ in range(1000):
    env.render() #출력
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
```



강화 학습(Reinforcement Learning) -OpenAI GYM Games

. 키보드를 이용한 게임

```
import readchar # pip3 install readchar
# MACROS
LEFT = 0
DOWN = 1
RIGHT = 2
UP = 3
# Key mapping
arrow_keys = {
    '\x1b[A': UP,
    '\x1b[B': DOWN,
    '\x1b[C': RIGHT,
    '\x1b[D': LEFT}
# Register FrozenLake with is_slippery False
register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name': '4x4', 'is_slippery': False}
)
env = gym.make('FrozenLake-v3')
env.render() # Show the initial board
```

```
while True:
    # Choose an action from keyboard
    key = readchar.readkey()
    if key not in arrow_keys.keys():
        print("Game aborted!") #화살표키가 아니면 끝남
        break
    action = arrow_keys[key]
    state, reward, done, info = env.step(action)
    env.render() # Show the board after action
    print("State: ", state, "Action: ", action,
          "Reward: ", reward, "Info: ", info)

    if done:
        print("Finished with reward", reward)
        break #게임이 끝나면 출력
```

```
SFFF
FHFF
F+L41mF+L0mFH
HFFG
State: 9 Action: 2 Reward: 0.0 Info: {'prob': 1.0}
(Down)
SFFF
FHFF
FHFF
H+L41mF+L0mFG
State: 13 Action: 1 Reward: 0.0 Info: {'prob': 1.0}
(Right)
SFFF
FHFF
FHFF
HF+L41mF+L0mG
State: 14 Action: 2 Reward: 0.0 Info: {'prob': 1.0}
(Right)
SFFF
FHFF
FHFF
HFF+L41mG+L0m
State: 15 Action: 2 Reward: 1.0 Info: {'prob': 1.0}
Finished with reward 1.0
```

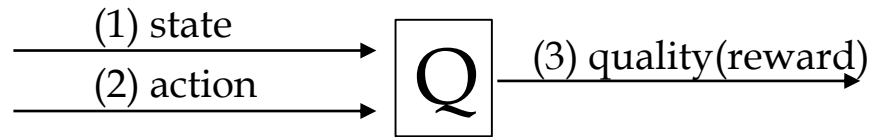
강화 학습(Reinforcement Learning) -Q-learning(Table)

S			
Action을 해야 정보를 get			
			Reward가 1이 되면 시행착오를 통해 전체적인 행동이 끝날때 알수 있음

- . 현재 $S \rightarrow S'$
- . Action \rightarrow get reward r
- . $Q(s', a')$ 이미 존재
- . $Q(s, a) \Rightarrow Q(s', a')$

강화 학습(Reinforcement Learning) -Q-learning(Table)

Q-function(state-action value function)



. Agent상태 이런 action을 하면 보상을 받을 수 있다.

$Q(\text{state}, \text{action})$

$Q(\text{state}, \text{action})$

$Q(s|, \text{LEFT}): 0$
 $Q(s|, \text{RIGHT}): 0.5$
 $Q(s|, \text{UP}): 0$
 $Q(s|, \text{DOWN}): 0.3$

Policy:
어떤 방향으로 갈까?

$$\text{Max } Q = \max_{a'} Q(s, a')$$

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

. $\operatorname{argmax} Q(s1,a) \rightarrow \text{right}$

Optimal Policy

State, action, reward

- My condition

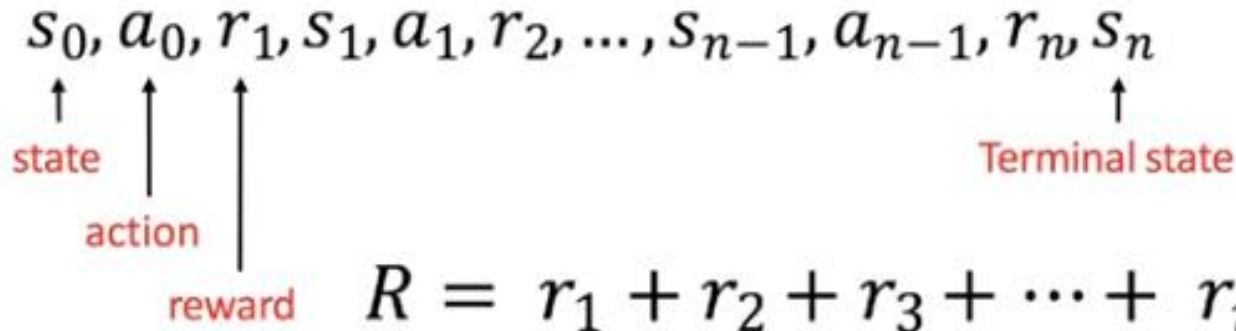
- I am in s
- when I do action a , I'll go to s'
- when I do action a , I'll get reward r
- Q in s' , $Q(s', a')$ exist!

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

$s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$



Future reward



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

Reward의 총합

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

Learning $Q(s, a)$

Learning $Q(s, a)$: Table

initial Q values are 0

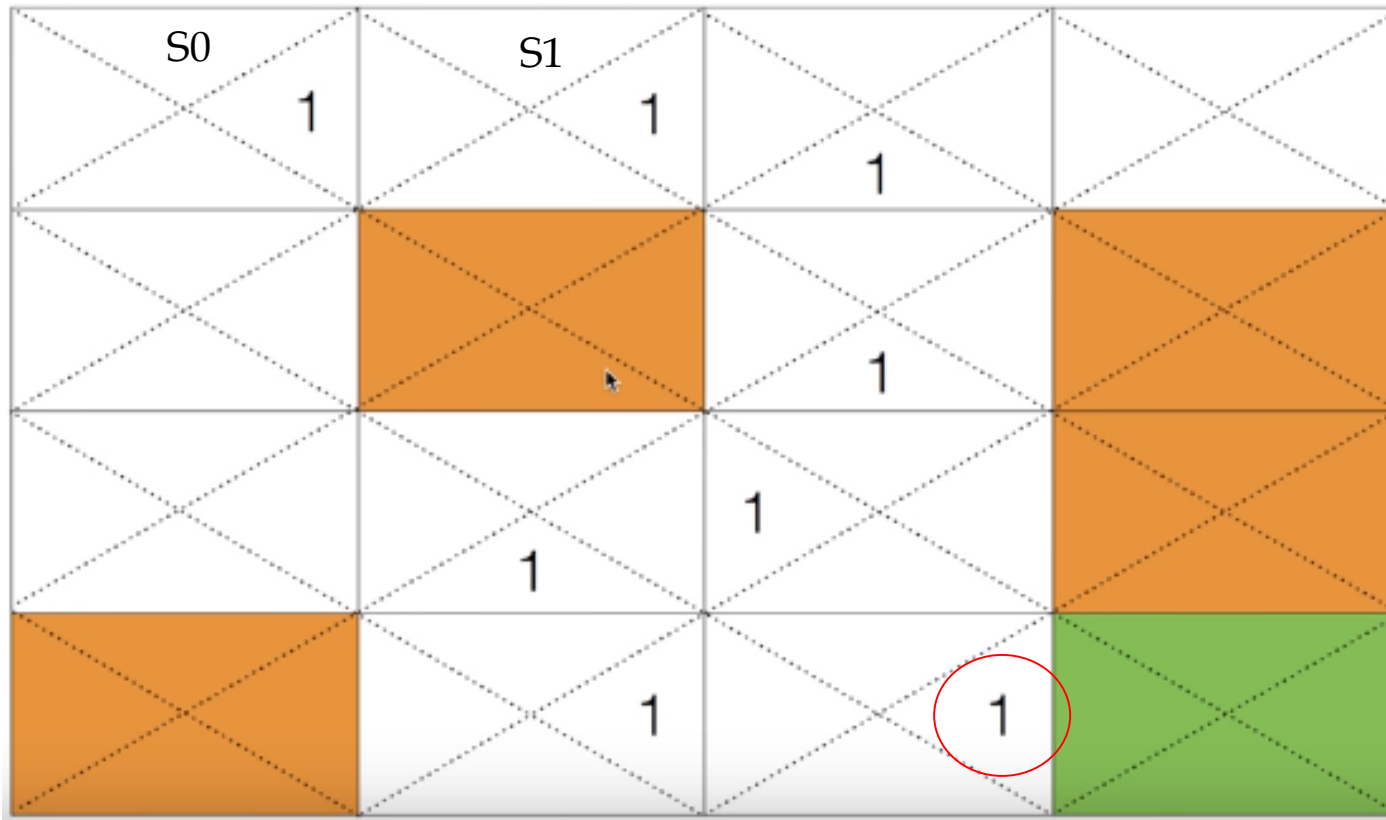
16 X 4 table

<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>
<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>
<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>
<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>	<div><div>0</div><div>0</div></div>

강화 학습(Reinforcement Learning)- Q-learning (table)

. Learning Q(s, a) Table : one success! (16 X 4 Table)

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$



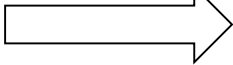
Final Q-Table Values
LEFT DOWN RIGHT UP

[0. 0. 1. 0.]
[0. 0. 1. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 1. 0. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 1. 0.]
[0. 0. 1. 0.]
[0. 0. 0. 0.]

$$Q(s_{13}, a_{\text{right}}) = r + \max(Q(s_{14}, a)) = 0 + \max(0, 0, 1, 0) = 1$$

강화 학습(Reinforcement Learning)- Dummy Q-learning (table)

- . s, a 초기 table의 값은 $Q(s, a) \leftarrow 0$
 - . Current state s 관찰
 - . Action a 선택 및 실행
 - . Reward r
 - . New state s' 관찰
 - . Update the table $Q(s, a)$
- 무한 반복


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

강화 학습(Reinforcement Learning)- Dummy Q-learning (table) (전체1)

```
import ...  
# https://gist.github.com/stober/1943451  
def rargmax(vector):  
    """ Argmax that chooses randomly among eligible maximum indices. """  
    m = np.amax(vector)  
    indices = np.nonzero(vector == m)[0]  
    return pr.choice(indices)  
  
register(  
    id='FrozenLake-v3',  
    entry_point='gym.envs.toy_text:FrozenLakeEnv',  
    kwargs={'map_name': '4x4',  
           'is_slippery': False}  
)  
env = gym.make('FrozenLake-v3')  
  
# Initialize table with all zeros  
Q = np.zeros([env.observation_space.n, env.action_space.n])  
# Set learning parameters  
num_episodes = 2000  
  
# create lists to contain total rewards and steps per episode  
rList = []
```

강화 학습(Reinforcement Learning)- Dummy Q-learning (table) (전 체2)

```
for i in range(num_episodes):
    # Reset environment and get first new observation
    state = env.reset()    #첫번째 state
    rAll = 0
    done = False

    # The Q-Table learning algorithm
    while not done:
        action = rargmax(Q[state, :])
        #랜덤한 방향으로 간다

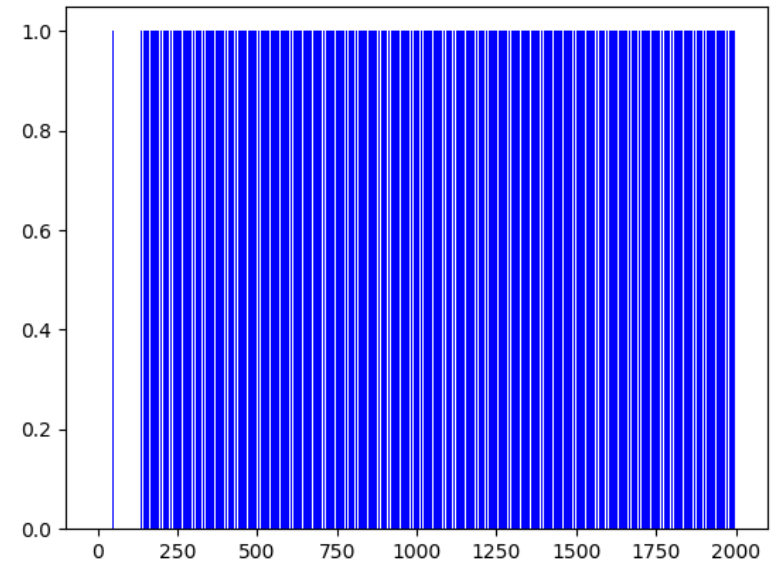
        # Get new state and reward from environment
        new_state, reward, done, _ = env.step(action)

        # Update Q-Table with new knowledge using learning rate
        Q[state, action] = reward + np.max(Q[new_state, :])

        rAll += reward
        state = new_state
    rList.append(rAll)

print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)
plt.bar(range(len(rList)), rList, color="blue")
plt.show()
```

. s, a initialize table
. current state a
. For a문
- action a 선택 실행
- reward r 즉시 보상
- 새로운 s'에 대해 관찰
- update table



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

강화 학습(Reinforcement Learning)- Dummy Q-learning algorithm

For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state s

Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

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

<http://computingkoreanlab.com/app/jAI/jQLearning/>

- $s \leftarrow s'$

Machine Learning, T. Mitchell, McGraw Hill, 1997

강화 학습(Reinforcement Learning) Exploit VS Exploration

Exploit : 내가 아는 길

Exploration : 모험을 해야하는 길

Exploit VS Exploration: E-greedy

```
e = 0.1  
if rand < e:  
    a = random  
else:  
    a = argmax(Q(s, a))
```

10% 랜덤하게 가고

Else 내가 아는 길 중 좋은길 선택

Exploit VS Exploration: decaying E-greedy

```
for i in range (1000)  
    e = 0.1 / (i+1)  
    if random(1) < e:  
        a = random  
    else:  
        a = argmax(Q(s, a))
```

랜덤하게 가는 길이 점점 적어진다.

[강화학습모델]

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

↓
↓
 현재의 받는reward 미래의 받는reward

- Future reward $R = r_1 + r_2 + r_3 + \dots + r_n$
 $R_t = r_t + r_{t+1} + r_{t+2} + \dots + r_n$

- Discounted future reward (environment is stochastic)

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{n-t} r_n \\ &= r_t + \gamma (r_{t+1} + \gamma (r_{t+2} + \dots)) \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

- A good strategy for an agent would be to always choose an action that **maximizes the (discounted) future reward**

Exploit VS Exploration: decaying E-greedy

```
for i in range (1000)
    e = 0.1 / (i+1)
    if random(1) < e:
        a = random
    else:
        a = argmax(Q(s, a))
```

```
for i in range(num_episodes):
    e = 1. / ((i / 100)+1) # Python2
    # The Q-Table learning algorithm
    while not done:
        # Choose an action by e greedy
        if np.random.rand(1) < e:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state, :])
```

Exploit VS Exploration: add random noise

```
for i in range (1000)  
    a = argmax(Q(s, a) + random_values / (i+1))
```

```
# Choose an action by greedily (with noise) picking from Q table  
action = np.argmax(Q[state, :] + np.random.randn(1, env.action_space.n) / (i + 1))
```

Decay될 수 있게

Q-learning algorithm

For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state s

Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

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

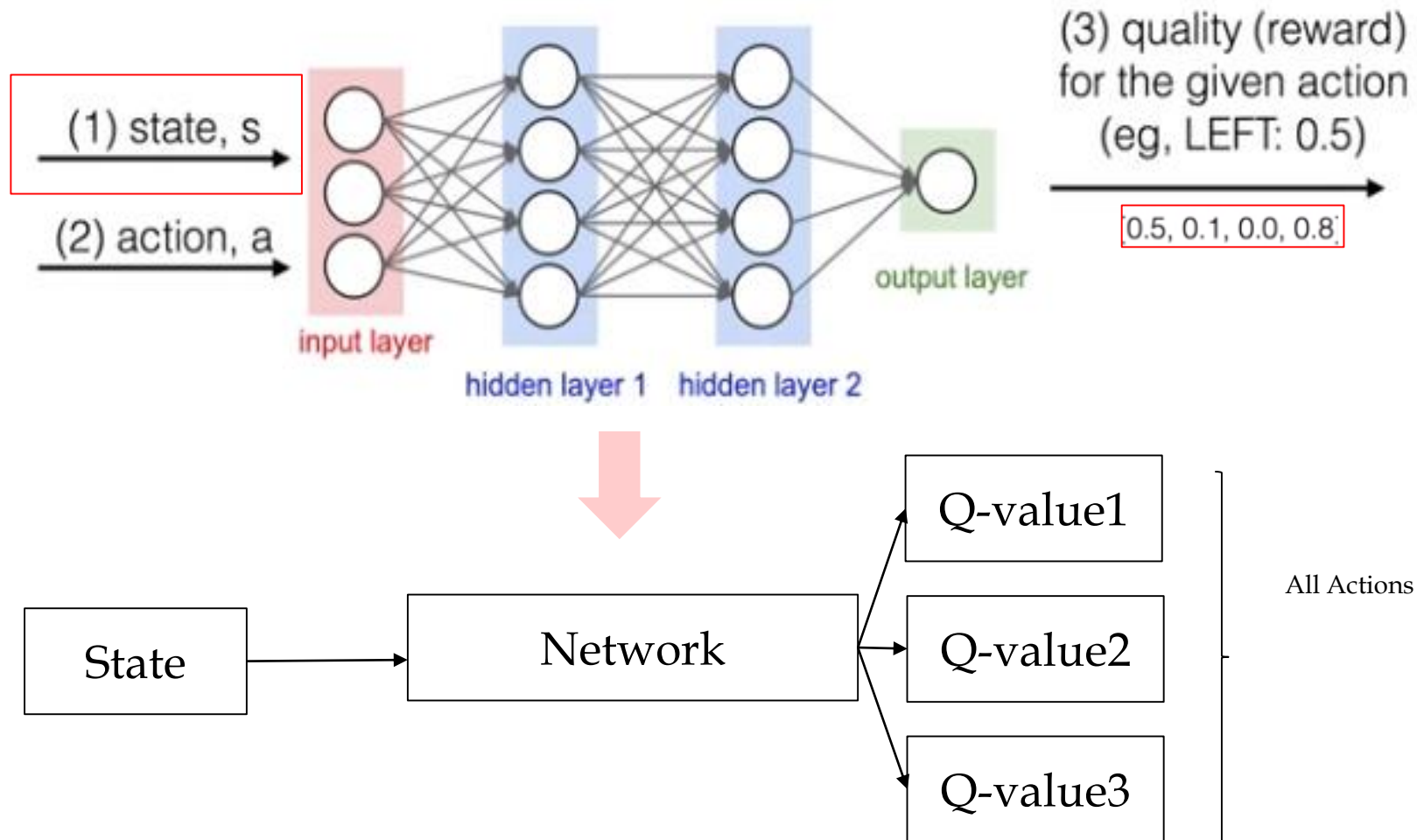
- $s \leftarrow s'$

```
# Discount factor  
dis = .99
```

```
# Update Q-Table with new knowledge using decay rate  
Q[state,action] = reward + dis * np.max(Q[new_state,:])
```

<http://computingkoreanlab.com/app/jAI/jQLearning/>

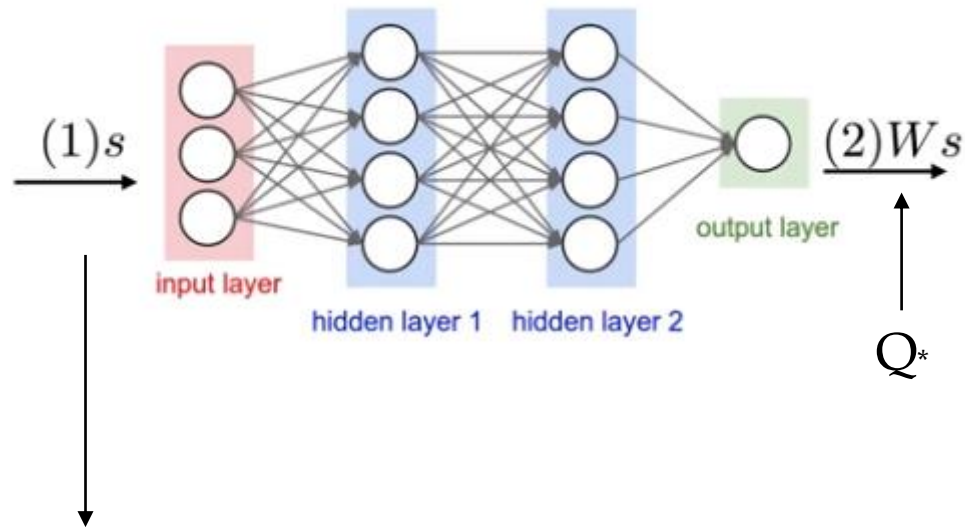
강화 학습(Reinforcement Learning)- Q-Network



강화 학습(Reinforcement Learning)- Q-Network traing

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$



$$cost(W) = (Ws - y)^2$$

$$y = r + \gamma \max Q(s')$$

Q^*

Q^*

State 7

- Approximate Q^* function using θ

$$\hat{Q}(s, a|\theta) \sim Q^*(s, a)$$

weight

우리가 가고자 하는
Q같아지게 하

- Choose θ to minimize

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

강화 학습(Reinforcement Learning)- Algorithm

Algorithm 1 Deep Q-learning

Initialize action-value function Q with random weights

for episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

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

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

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

```
def one_hot(x):  
    return np.identity(16)[x:x + 1]
```

전처리

Choose an action by greedily (with e chance of random action) from the Q-network

```
Qs = sess.run(Qpred, feed_dict={X: one_hot(s)})
```

```
if np.random.rand(1) < e:
```

```
    a = env.action_space.sample()
```

```
else:
```

```
    a = np.argmax(Qs)
```

Table => network

학습부분

University of Toronto by V Mnih et al.

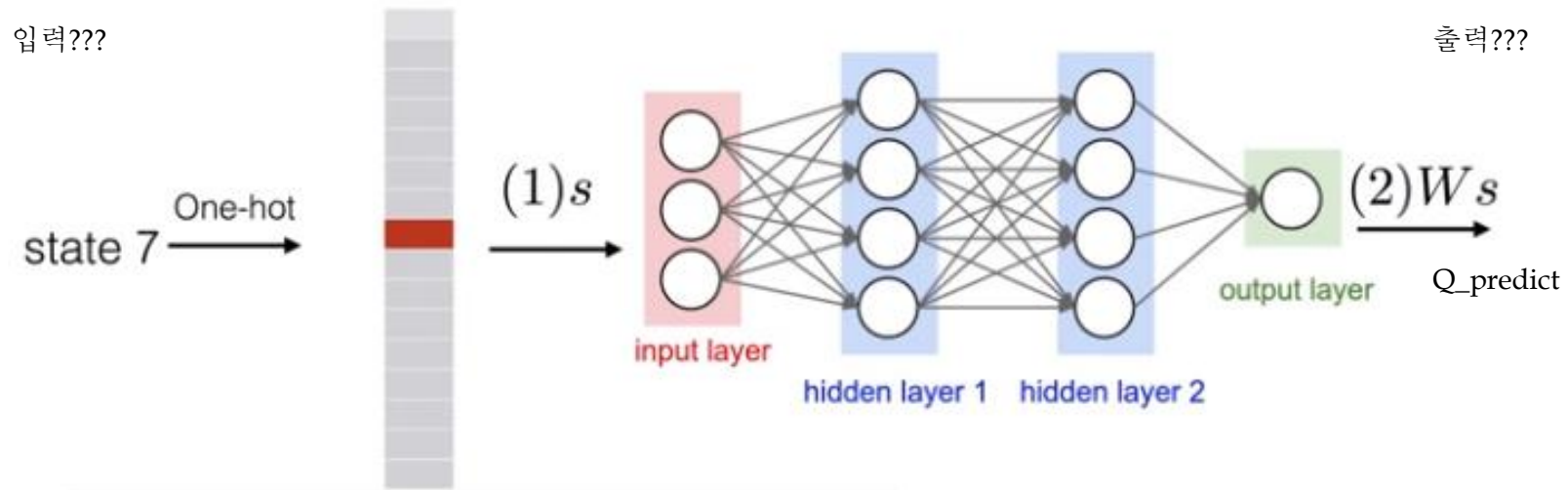
강화 학습(Reinforcement Learning)- Y label and loss function

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

```
if done:
    # Update Q, and no Qs+1, since it's a terminal state
    Qs[0, a] = reward
else:
    # Obtain the Q_s1 values by feeding the new state through our network
    Qs1 = sess.run(Qpred, feed_dict={X: one_hot(s1)})
    # Update Q
    Qs[0, a] = reward + dis * np.max(Qs1)
```

강화 학습(Reinforcement Learning)- Q-Network training



Def on_hot(x) :

```
return np.identity(16)[x:x+1]
```

```
Input_size = env.observation_space.n # 16
```

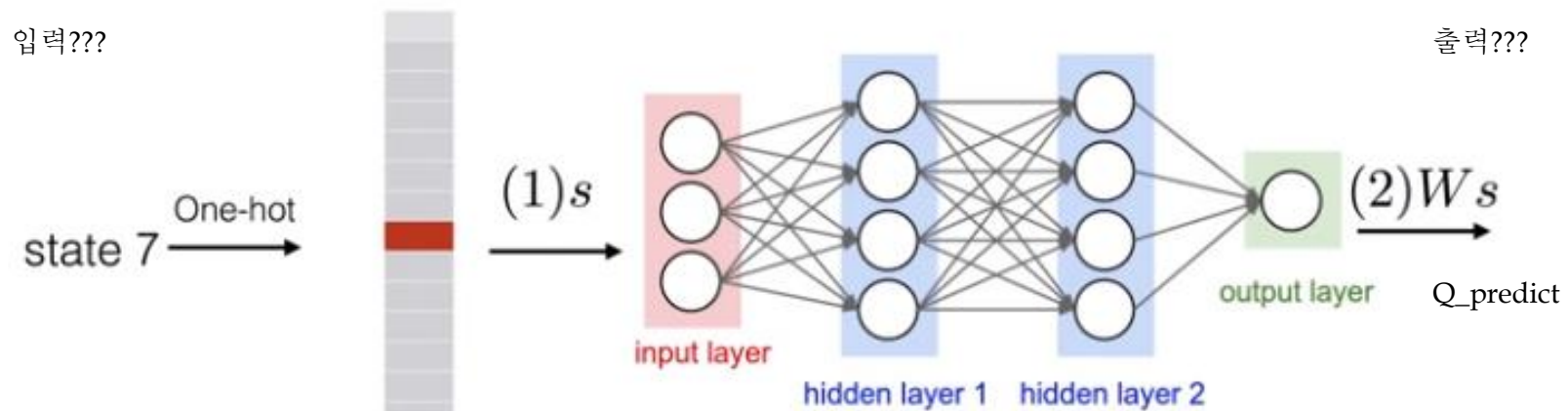
```
Output_size = env.action_space.n      #4
```

```
X=tf.placeholder(shape=[1,input_size],dtype=tf.float32) #(1,16)
```

```
W=tf.Variable(tf.random_uniform([input_size, output_size],0,0.01)) #(16,4)
```

```
Qpred =tf.matmul(X, W)
```

강화 학습(Reinforcement Learning)- Q-Network training(linear regression)



```
Qpred= tf.matmul(X, W)
```

```
Y = tf.placeholder(shape=[1,output_size],dtype=tf.float32) # Y label (4)
```

```
Loss = tf.reduce_sum(tf.square(Y-Qpred))
```

$$cost(W) = (Ws - y)^2$$

```
Train =
```

```
tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(loss)
```

```
Qs[0,a] = reward + dis * np.max(Qs1)  $y = r + \gamma \max Q(s')$ 
```

```
Sess.run(train, feed_dict={X:one_hot(s), Y:Qs})
```


강화 학습(Reinforcement Learning)- Q-Network training

```
import numpy as np
print(np.identity(16)[0:1])
print(np.eye(16)[10:11])
print(np.identity(16))
```

State : `np.identity(16)[s1:s1+1]`

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

강화 학습(Reinforcement Learning)- Code:Network and setup

```
import gym
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
env = gym.make('FrozenLake-v0')

# Input and output size based on the Env
input_size = env.observation_space.n
output_size = env.action_space.n
learning_rate = 0.1

# These lines establish the feed-forward part of the network used to
# choose actions
X = tf.placeholder(shape=[1, input_size], dtype=tf.float32) # state input 원핫코딩
W = tf.Variable(tf.random_uniform(
    [input_size, output_size], 0, 0.01)) # weight

Qpred = tf.matmul(X, W) # Out Q prediction
Y = tf.placeholder(shape=[1, output_size], dtype=tf.float32) # Y label

loss = tf.reduce_sum(tf.square(Y - Qpred))
train = tf.train.GradientDescentOptimizer(
    learning_rate=learning_rate).minimize(loss)

# Set Q-learning related parameters
dis = .99
num_episodes = 2000

# Create lists to contain total rewards and steps per episode
rList = [] #결과 지정 list
```

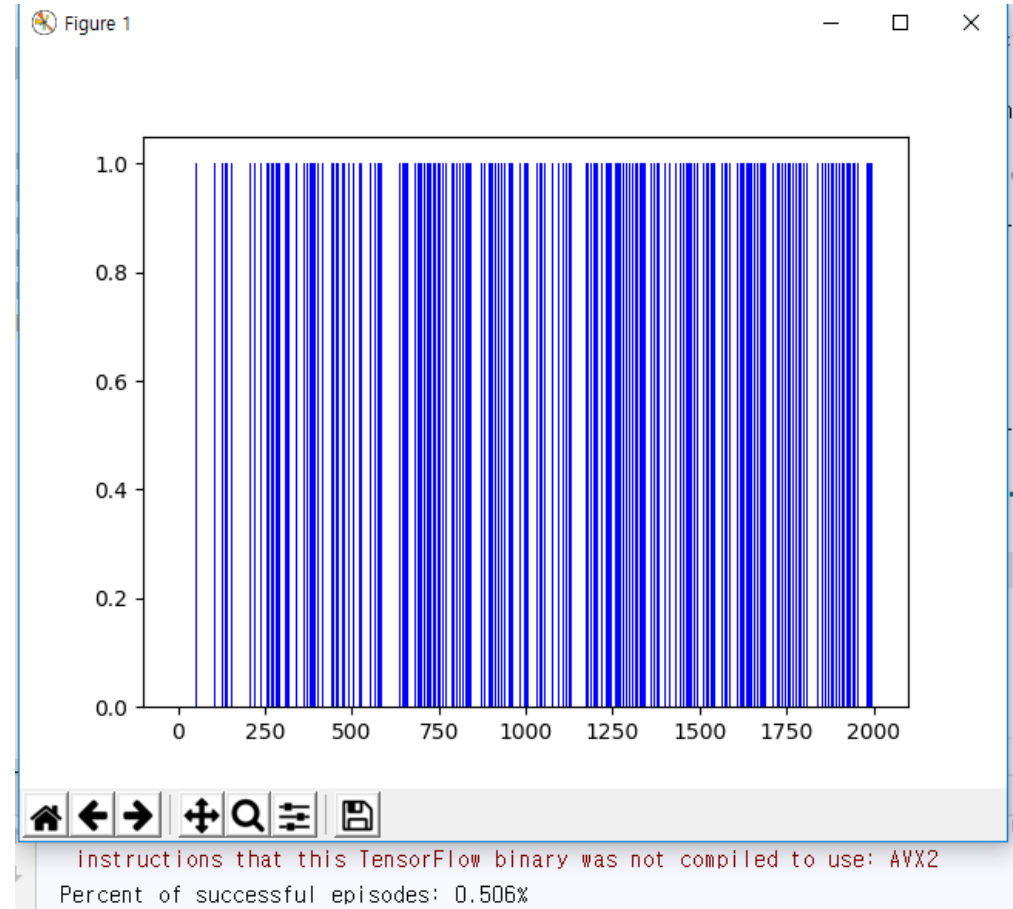
강화 학습(Reinforcement Learning)- Code:Network and setup

```
def one_hot(x):  
    return np.identity(16)[x:x + 1]  
  
init = tf.global_variables_initializer() #초기화 세션  
with tf.Session() as sess:  
    sess.run(init)  
    for i in range(num_episodes):  
        # Reset environment and get first new observation  
        s = env.reset()  
        e = 1. / ((i / 50) + 10)  
        rAll = 0  
        done = False  
        local_loss = []
```

```
# The Q-Network training  
while not done:  
    # Choose an action by greedily (with e chance of random action)  
    # from the Q-network  
    Qs = sess.run(Qpred, feed_dict={X: one_hot(s)})  
    if np.random.rand(1) < e:  
        a = env.action_space.sample()  
    else:  
        a = np.argmax(Qs)  
  
    # Get new state and reward from environment  
    s1, reward, done, _ = env.step(a)  
    if done:  
        # Update Q, and no Qs+1, since it's a terminal state  
        Qs[0, a] = reward  
    else:  
        # Obtain the Q_s1 values by feeding the new state through our  
        # network  
        Qs1 = sess.run(Qpred, feed_dict={X: one_hot(s1)})  
        # Update Q  
        Qs[0, a] = reward + dis * np.max(Qs1)  
  
    # Train our network using target (Y) and predicted Q (Qpred) values  
    sess.run(train, feed_dict={X: one_hot(s), Y: Qs})  
  
    rAll += reward  
    s = s1  
    rList.append(rAll)
```

강화 학습(Reinforcement Learning)- Code:Network and setup

```
print("Percent of successful episodes: " +  
      str(sum(rList) / num_episodes) + "%")  
plt.bar(range(len(rList)), rList, color="blue")  
plt.show()
```



Cart Pole

```
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
```

It should look something like this:



강화 학습(Reinforcement Learning)- Q Network for Cart Pole(Random trials)

```
import gym

env = gym.make('CartPole-v0')
env.reset()
random_episodes = 0
reward_sum = 0

while random_episodes < 10:
    env.render()
    action = env.action_space.sample()
    observation, reward, done, _ = env.step(action)
    print(observation, reward, done)
    reward_sum += reward
    if done:
        random_episodes += 1
        print("Reward for this episode was:", reward_sum)
        reward_sum = 0
        env.reset()
```

```
[-0.05429501 -0.18910736 -0.02901918  0.2469003 ] 1.0 False
[-0.05807716  0.00641677 -0.02408117 -0.05479284] 1.0 False
[-0.05794882 -0.18835177 -0.02517703  0.23019607] 1.0 False
[-0.06171586 -0.38310507 -0.02057311  0.51483217] 1.0 False
[-0.06937796 -0.57793134 -0.01027647  0.80096167] 1.0 False
[-0.08093659 -0.38266996  0.00574277  0.50506383] 1.0 False
[-0.08858999 -0.57787237  0.01584404  0.79955098] 1.0 False
[-0.10014743 -0.3829713  0.03183506  0.51189404] 1.0 False
[-0.10780686 -0.57852687  0.04207294  0.81443669] 1.0 False
[-0.1193774  -0.38400556  0.05836168  0.53527872] 1.0 False
[-0.12705751 -0.57989753  0.06906725  0.845765  ] 1.0 False
[-0.13865546 -0.7758904  0.08598255  1.159343  ] 1.0 False
[-0.15417327 -0.97202095  0.10916941  1.47769974] 1.0 False
[-0.17361369 -1.16829356  0.13872341  1.80238811] 1.0 False
[-0.19697956 -1.36466658  0.17477117  2.13477189] 1.0 False
[-0.22427289 -1.17165428  0.21746661  1.90078541] 1.0 True
Reward for this episode was: 20.0
```

강화 학습(Reinforcement Learning)- Q Network for Cart Pole

```
import numpy as np
import tensorflow as tf
from collections import deque

import gym
env = gym.make('CartPole-v0')

# Constants defining our neural network
learning_rate = 1e-1
input_size = env.observation_space.shape[0]
output_size = env.action_space.n

X = tf.placeholder(tf.float32, [None, input_size], name="input_x")

# First layer of weights
W1 = tf.get_variable("W1", shape=[input_size, output_size],
                    initializer=tf.contrib.layers.xavier_initializer())
Qpred = tf.matmul(X, W1)

# We need to define the parts of the network needed for learning a policy
Y = tf.placeholder(shape=[None, output_size], dtype=tf.float32)

# Loss function
loss = tf.reduce_sum(tf.square(Y - Qpred))

# Learning
train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

# Values for q learning
max_episodes = 5000
dis = 0.9
step_history = []
```

강화 학습(Reinforcement Learning)- Q Network for Cart Pole

```
# Setting up our environment
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for episode in range(max_episodes):
    e = 1. / ((episode / 10) + 1)
    step_count = 0
    state = env.reset()
    done = False

    # The Q-Network training
    while not done:
        step_count += 1
        x = np.reshape(state, [1, input_size])
        # Choose an action by greedily (with e chance of random action) from
        # the Q-network
        Q = sess.run(Qpred, feed_dict={X: x})
        if np.random.rand(1) < e:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q)

        # Get new state and reward from environment
        next_state, reward, done, _ = env.step(action)
        if done:
            Q[0, action] = -100
        else:
            x_next = np.reshape(next_state, [1, input_size])
            # Obtain the Q' values by feeding the new state through our network
            Q_next = sess.run(Qpred, feed_dict={X: x_next})
            Q[0, action] = reward + dis * np.max(Q_next)

        # Train our network using target and predicted Q values on each episode
        sess.run(train, feed_dict={X: x, Y: Q})
        state = next_state
```


강화 학습(Reinforcement Learning)- Q Network for Cart Pole

```
step_history.append(step_count)
print("Episode: {} steps: {}".format(episode, step_count))
# If last 10's avg steps are 500, it's good enough
if len(step_history) > 10 and np.mean(step_history[-10:]) > 500:
    break
```

```
# See our trained network in action
```

```
observation = env.reset()
```

```
reward_sum = 0
```

```
while True:
```

```
    env.render()
```

```
    x = np.reshape(observation, [1, input_size])
```

```
    Q = sess.run(Qpred, feed_dict={X: x})
```

```
    action = np.argmax(Q)
```

```
    observation, reward, done, _ = env.step(action)
```

```
    reward_sum += reward
```

```
    if done:
```

```
        print("Total score: {}".format(reward_sum))
```

```
        break
```

```
Episode: 4984 steps: 9
Episode: 4985 steps: 9
Episode: 4986 steps: 9
Episode: 4987 steps: 34
Episode: 4988 steps: 45
Episode: 4989 steps: 31
Episode: 4990 steps: 38
Episode: 4991 steps: 31
Episode: 4992 steps: 36
Episode: 4993 steps: 21
Episode: 4994 steps: 29
Episode: 4995 steps: 28
Episode: 4996 steps: 15
Episode: 4997 steps: 14
Episode: 4998 steps: 29
Episode: 4999 steps: 37
Total score: 21.0
```

Implementing Nature Paper

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

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 $\hat{Q} = Q$

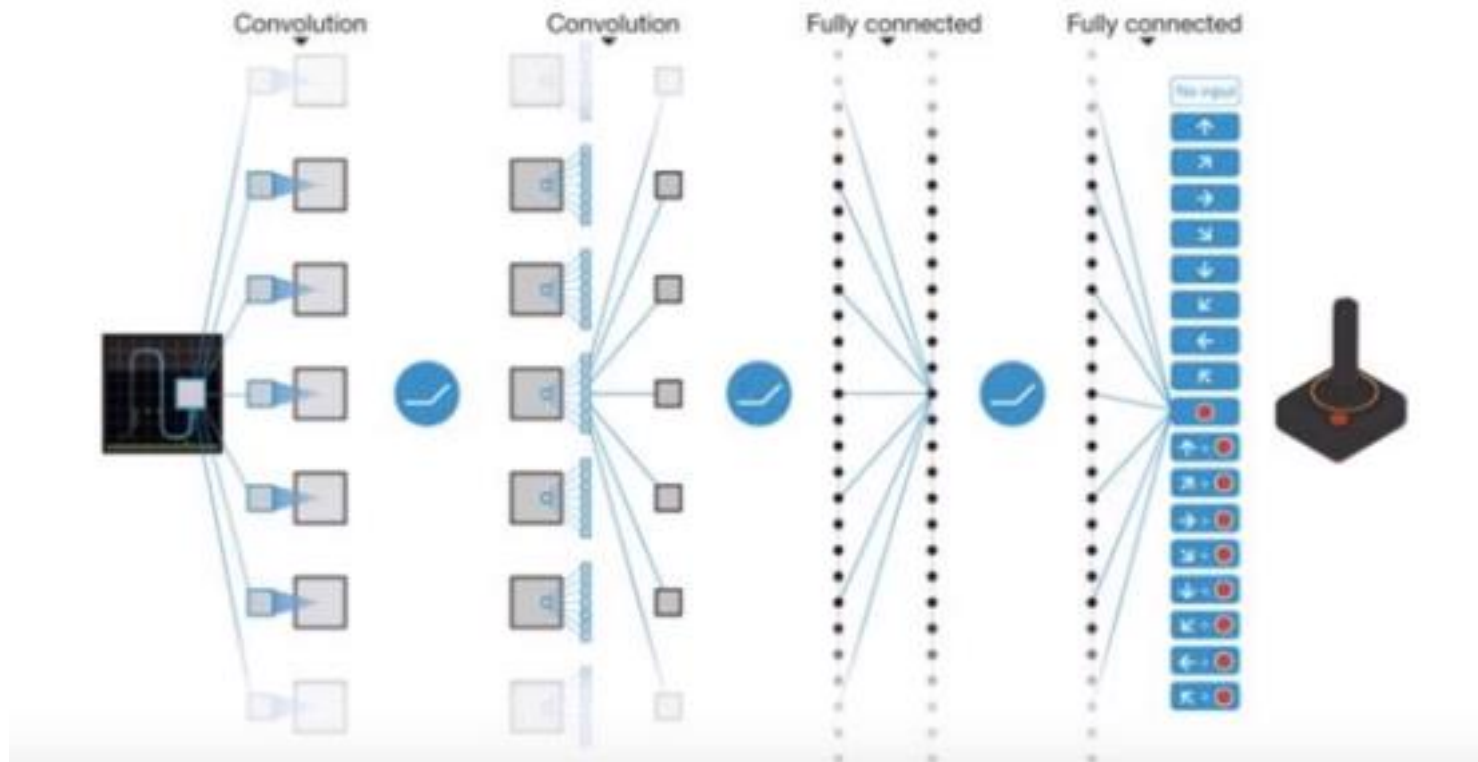
End For

```
x = np.reshape(s, [1, input_size])  
return sess.run(self._Qpred, feed_dict={self._X: x})
```

Main 네트워크 업데이트하면서 target은 실행

강화 학습(Reinforcement Learning)- DQN(Q-network issues)

1. Go deep : 깊은 네트워크 사용

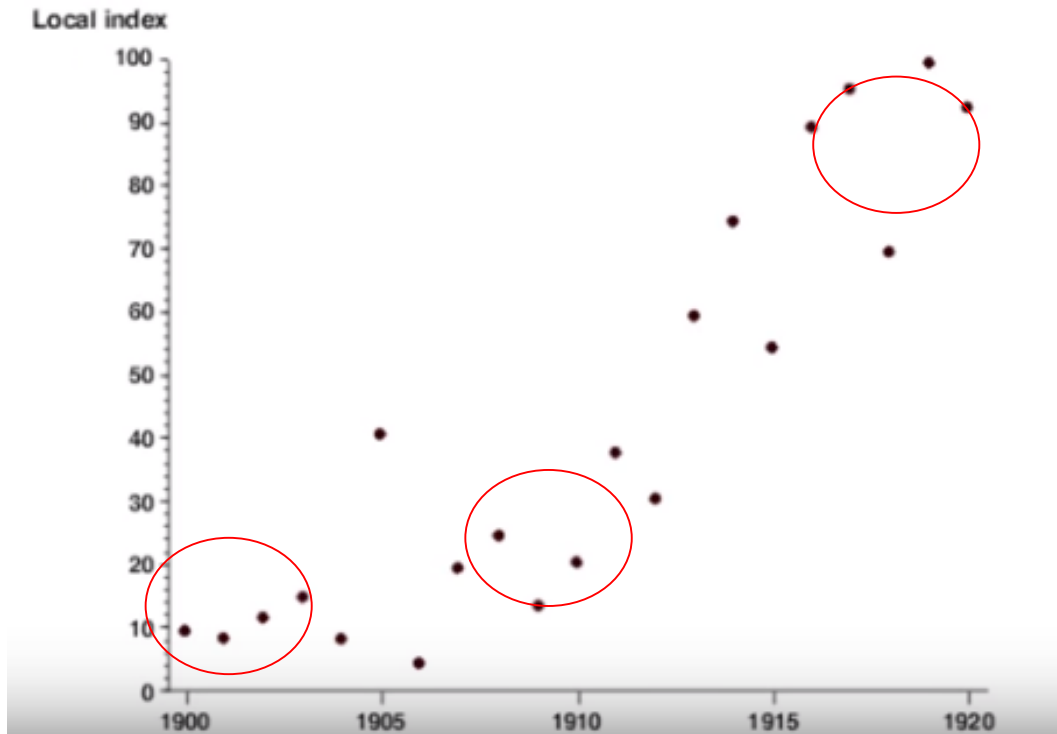


Human-level control through deep reinforcement learning, Nature

<http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html>

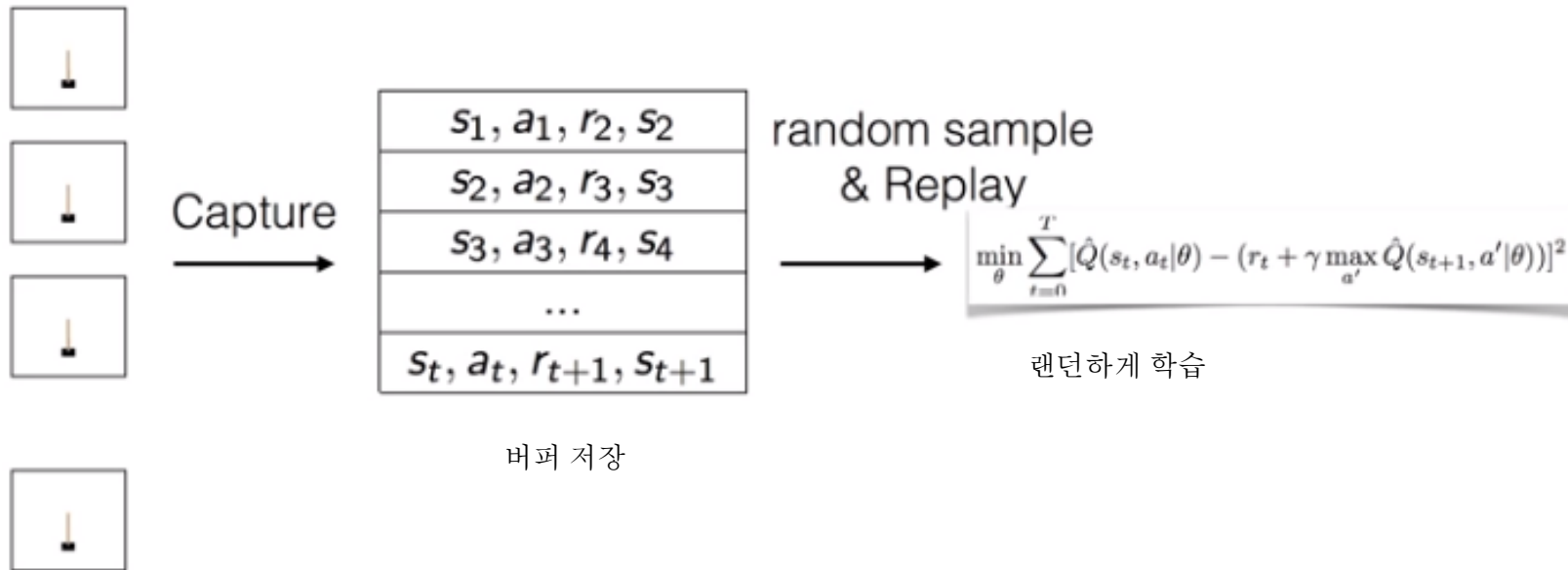
강화 학습(Reinforcement Learning)- DQN(Q-network issues)

2. Correlations between samples



강화 학습(Reinforcement Learning)- DQN(Q-network issues)

2. Experience replay



일정 공간에 버퍼를 각각의 action들을 저장한 후에 랜덤하게 가지고 와서 학습 추진

3. non-stationary targets

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

Target

$$\hat{Y} = \hat{Q}(s_t, a_t | \theta)$$

Update

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

Update

Y의 예측값을 업데이트 하면 타겟도 **update**되는 것이 문제이다.

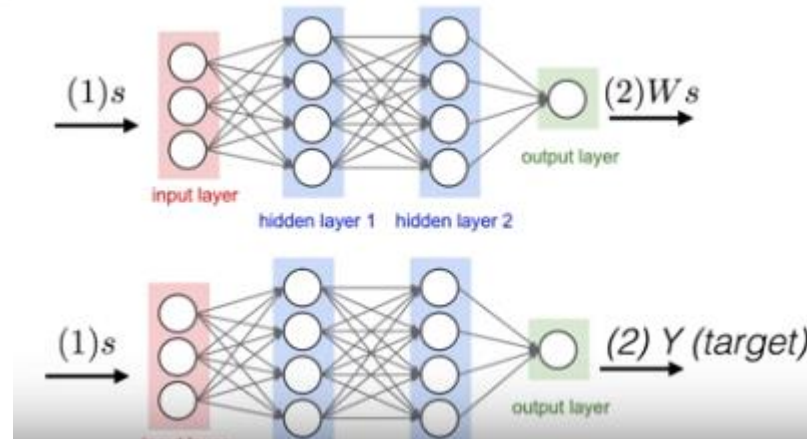
강화 학습(Reinforcement Learning)- DQN(Q-network issues)

3. Copy network

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

한쪽만 업데이트

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



Human-level control through deep reinforcement learning, Nature

<http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html>

강화 학습(Reinforcement Learning)- DQN(Q-network issues)

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N 버퍼 생성 초기화

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$ 2개의 네트워크 생성

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 Action 선택

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

 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 랜덤한 샘플을 가지고 와서 네트워크 분리

 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 update

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

End For

Understanding Nature Paper (2015)

Human-level control through deep reinforcement learning, Nature

<http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html>

강화 학습(Reinforcement Learning)- DQN(Q-network issues)- 설치

머신 러닝 분야에서 가장 핫한 기업인 Deepmind에서 배포한 DQN Agent을 설치하고 실행



강화 학습(Reinforcement Learning)- DQN(Q-network issues)- 설치

Step1 :먼저, DQN Agent를 설치하고 실행해보기 위해선 Linux(Ubuntu)나 Mac OS X 환경이 필요

Ubuntu 환경을 기본으로 설명을 진행

첫째로, 아래의 링크로 가서 DQN 소스 코드를 다운 받고 압축을 푼다.

<https://sites.google.com/a/deepmind.com/dqn/>

Step 2 : 압축을 푼 뒤 실행에 필요한 라이브러리들을 설치하기 위해서 터미널 창에서 다음을 명령어 실행
설치에 약 10분 정도가 소요

`./install_dependencies.sh`

Step 3: 설치가 완료되면, roms 폴더에 rom file을 집어넣고 아래의 명령어로 DQN Agent를 실행
실행하려 게임이 breakout 일 경우 cpu로 실행 할 경우 `./run_cpu breakout`
gpu로 실행 할 경우 `./run_gpu breakout`

이때, gpu로 실행하려고 할 경우, 다음과 같은 에러 메시지가 뜬다.

`../torch/bin/luajit: ./convnet.lua:22: attempt to call local 'convLayer' (a nil value)`

dqn 파일의 convnet.lua의 SpatialConvolutionCUDA 함수가 deprecated 되서 생기는 문제

dqn폴더의 convnet.lua 파일을 아래 첨부 파일로 바꾸면 간단히

해결된다.(<https://github.com/soumith/deepmind-atari> 을 참조)

강화 학습(Reinforcement Learning)- DQN(Q-network issues)- 설치

Step 4 :

이제 Deepmind에서 배포한 DQN Agent를 실행해 볼 수 있다. 하지만 제공되는 코드는 터미널 창에서만 학습이 진행된다. Agent가 게임을 플레이하면서 학습하는 모습을 display 창으로 보고 싶다면 아래의 과정을 추가로 해주어야 한다. (<http://superintelligence.ch/deepmind/> 참조)

1. 아래의 명령어로 qtorch를 install한다.

```
torch/bin/luarocks install qtorch
```

2. run_cpu 혹은 run_gpu 파일의 46번째 줄을

```
../torch/bin/luajit train_agent.lua $args
```

에서

```
../torch/bin/qlua train_agent.lua $args
```

로 바꿔준다.

3. torch/share/lua/5.1/alewrap/AleEnv.lua 파일의 52번째 줄을

```
display=false,
```

에서

```
display=true,
```

로 바꿔준다.

이제 모든 과정이 다 끝났다. ./run_cpu breakout 명령어를 실행해 보면 아래와 같이 Agent가 Breakout을 플레이하면서 학습하는 모습을 지켜볼 수 있다.

머신러닝 -자전거 수요 예측

<https://www.kaggle.com/fkstepz/step-by-step-predict-bike-sharing-demand/data>

공공 자전거 데이터를 바탕으로 수요를 예측 (2년 전에 경진대회 끝남)

Data


Data Sources

▼ Bike Sharing Demand

sampleSubmission.c... 6494 x 2

test.csv 6494 x 9

train.csv 10.9k x 12



Bike Sharing Demand

Forecast use of a city bikeshare system

Last Updated: 4 years ago

About this Competition

[See, fork, and run a random forest benchmark model through Kaggle Scripts](#)

You are provided hourly rental data spanning two years. For this competition, the training set is comprised of the first 19 days of each month, while the test set is the 20th to the end of the month. You must predict the total count of bikes rented during each hour covered by the test set, using only information available prior to the rental period.

Data Fields

datetime - hourly date + timestamp

season - 1 = spring, 2 = summer, 3 = fall, 4 = winter

holiday - whether the day is considered a holiday

workingday - whether the day is neither a weekend nor holiday

weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp - temperature in Celsius

atemp - "feels like" temperature in Celsius

humidity - relative humidity

windspeed - wind speed

casual - number of non-registered user rentals initiated

registered - number of registered user rentals initiated

count - number of total rentals

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy import stats

#그래프에서 격자로 숫자 범위가 눈에 잘 띄도록 ggplot 스타일 이용
plt.style.use('ggplot')

#그래프에서 마이너스 폰트 깨지는 문제에 대한 대처
mpl.rcParams['axes.unicode_minus'] = False

train = pd.read_csv('./datasets/train.csv', parse_dates=["datetime"])
test = pd.read_csv('./datasets/test.csv')
```

```
# train, columns
# train, dtypes
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
datetime      10886 non-null  datetime64[ns]
season        10886 non-null    int64
holiday       10886 non-null    int64
workingday    10886 non-null    int64
weather       10886 non-null    int64
temp          10886 non-null    float64
atemp         10886 non-null    float64
humidity      10886 non-null    int64
windspeed     10886 non-null    float64
casual        10886 non-null    int64
registered    10886 non-null    int64
count         10886 non-null    int64
dtypes: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.6 KB
```

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
print(train.head(20)) # 상위 20개  
print(train.temp.describe()) #기온에 대해서  
print(train.isnull().sum()) #null인 데이터 확인
```

```
[20 rows x 12 columns]  
count      10886.00000  
mean        20.23086  
std         7.79159  
min         0.82000  
25%        13.94000  
50%        20.50000  
75%        26.24000  
max         41.00000  
Name: temp, dtype: float64  
datetime    0  
season      0  
holiday     0  
workingday  0  
weather     0  
temp        0  
atemp       0  
humidity    0  
windspeed   0  
casual      0  
registered  0  
count       0  
dtype: int64
```

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
import missingno as msno
msno.matrix(train, figsize=(12,5)) #null plot
train["year"] = train["datetime"].dt.year
train["month"] = train["datetime"].dt.month
train["day"] = train["datetime"].dt.day
train["hour"] = train["datetime"].dt.hour
train["minute"] = train["datetime"].dt.minute
train["second"] = train["datetime"].dt.second
```

```
#print(train.shape) #12 -> 18 늘어남
```

```
#train.head()
```

```
figure, ((ax1,ax2,ax3),(ax4,ax5,ax6)) = plt.subplots(nrows=2, ncols=3)
```

```
figure.set_size_inches(18,8)
```

```
sns.barplot(data=train, x="year", y="count", ax=ax1)
sns.barplot(data=train, x="month", y="count", ax=ax2)
sns.barplot(data=train, x="day", y="count", ax=ax3)
sns.barplot(data=train, x="hour", y="count", ax=ax4)
sns.barplot(data=train, x="minute", y="count", ax=ax5)
sns.barplot(data=train, x="second", y="count", ax=ax6)
```

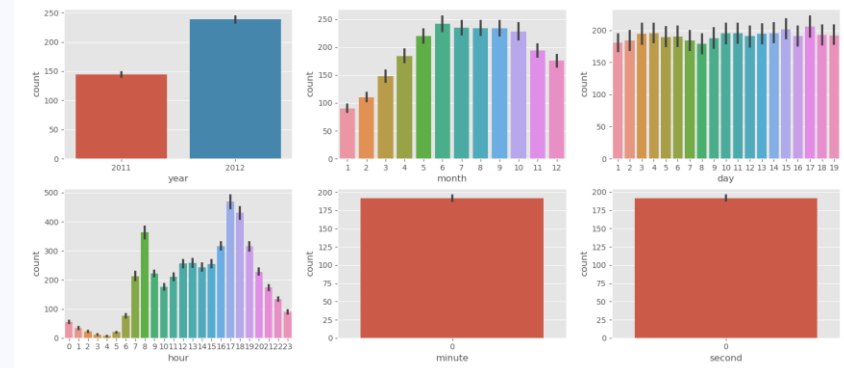
```
plt.show()
```

```
#ax1.set(ylabel="Count",title="연도별 대여량")
```

```
#ax2.set(xlabel="month",title="월별 대여량")
```

```
#ax3.set(xlabel="day",title="일별 대여량")
```

```
#ax4.set(xlabel="hour",title="시간별 대여량")
```



연도별 대여량은 2011년보다 2012년이 더 많다
월별대여량은 6월에 가장 많고 7~10월 대여량이 많다 1월이 적다
일별대여량은 1일부터 19일까지만 나머지 날짜는 test.csv 이 데이터 피쳐로 사용하면 안된다.
시간 대 대여량을 보면 출퇴근 시간에 대여량이 많은거 같다. 주말과 나누어 볼필요가 있을 것 같다.
분초, 0이기 의미가 없다

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
import missingno as msno
msno.matrix(train, figsize=(12,5)) #null plot
train["year"] = train["datetime"].dt.year
train["month"] = train["datetime"].dt.month
train["day"] = train["datetime"].dt.day
train["hour"] = train["datetime"].dt.hour
train["minute"] = train["datetime"].dt.minute
train["second"] = train["datetime"].dt.second
#print(train.shape) #12 -> 18 늘어남
#train.head()
figure, ((ax1,ax2,ax3),(ax4,ax5,ax6)) = plt.subplots(nrows=2, ncols=3)
figure.set_size_inches(18,8)

sns.barplot(data=train, x="year", y="count", ax=ax1)
sns.barplot(data=train, x="month", y="count", ax=ax2)
sns.barplot(data=train, x="day", y="count", ax=ax3)
sns.barplot(data=train, x="hour", y="count", ax=ax4)
sns.barplot(data=train, x="minute", y="count", ax=ax5)
sns.barplot(data=train, x="second", y="count", ax=ax6)
plt.show()

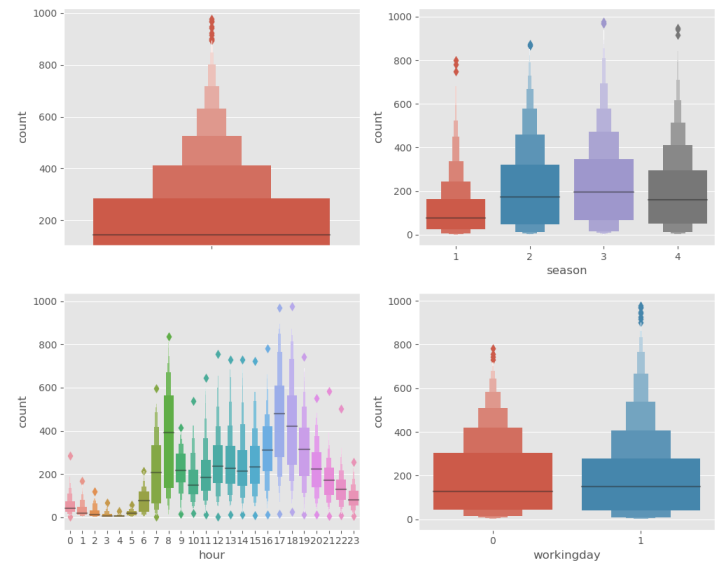
#ax1.set(ylabel="Count",title="연도별 대여량")
#ax2.set(xlabel="month",title="월별 대여량")
#ax3.set(xlabel="day",title="일별 대여량")
#ax4.set(xlabel="hour",title="시간별 대여량")
```

#연도별 대여량은 2011년보다 2012년이 더 많다
#월별대여량은 6월에 가장 많고 7~10월 대여량이
 많다 1월이 적다
#일별대여량은 1일부터 19일까지만 나머지 날짜는
 test.csv 이 데이터 피쳐로 사용하면 안된다.
#시간 대 대여량을 보면 출퇴근 시간에 대여량이
 많은거 같다. 주말과 나누어 볼필요가 있을 것 같다.
#분초, 0이기 의미가 없다

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_size_inches(12, 10)
sns.boxenplot(data=train, y="count", orient="v", ax=axes[0][0])
sns.boxenplot(data=train, y="count", x="season", orient="v", ax=axes[0][1])
sns.boxenplot(data=train, y="count", x="hour", orient="v", ax=axes[1][0])
sns.boxenplot(data=train, y="count", x="workingday", orient="v", ax=axes[1][1])

# axes[0][0].set(ylabel='Count', title="대여량")
# axes[0][1].set(xlabel='Season', ylabel="Count", title="계절별 대여량")
# axes[1][0].set(xlabel='Hour of The Day', ylabel="Count", title="시간별 대여량")
# axes[1][1].set(xlabel='Working Day', ylabel="Count", title="근무일 여부에 따른 대여량")
plt.show()
```

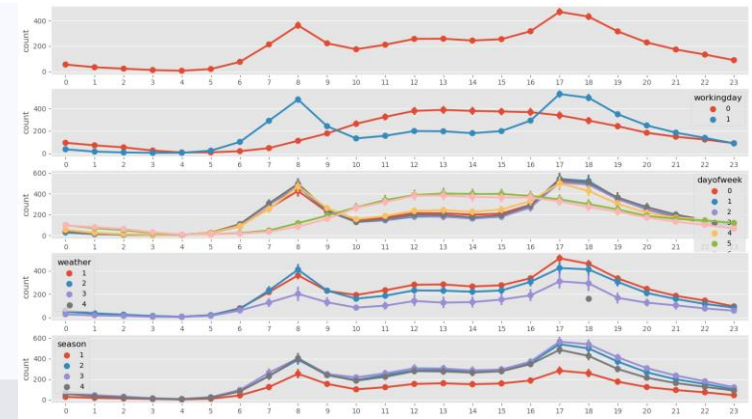


#대여량: 특정구간
#계절별: 봄이 적고 여름과 가을이 가장 크다
#시간별: 흡사
#휴일에 대여량이 많다.

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
train["dayofweek"] = train["datetime"].dt.dayofweek  
#print(train.shape)
```

```
train["dayofweek"].value_counts()  
fig,(ax1,ax2,ax3,ax4,ax5) = plt.subplots(nrows=5)  
fig.set_size_inches(18,25)  
sns.pointplot(data=train, x="hour",y="count", ax=ax1)  
sns.pointplot(data=train, x="hour",y="count", hue="workingday",ax=ax2)  
sns.pointplot(data=train, x="hour",y="count", hue="dayofweek",ax=ax3)  
sns.pointplot(data=train, x="hour",y="count", hue="weather",ax=ax4)  
sns.pointplot(data=train, x="hour",y="count", hue="season",ax=ax5)  
plt.show()
```

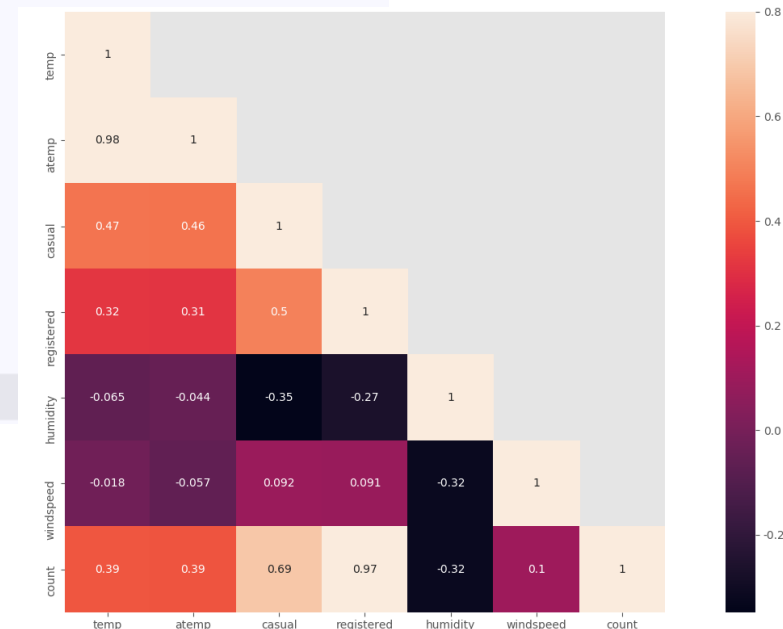


#시간대별 대여량
#출퇴근 시간에 사용
#워킹데이 휴일 점심시간(11~17)
#토,일요일 워킹데이 흡사
#계절 날씨가 좋을 때 많이 빌림

머신러닝 -처음부터 끝까지-자전거 대여량 예측

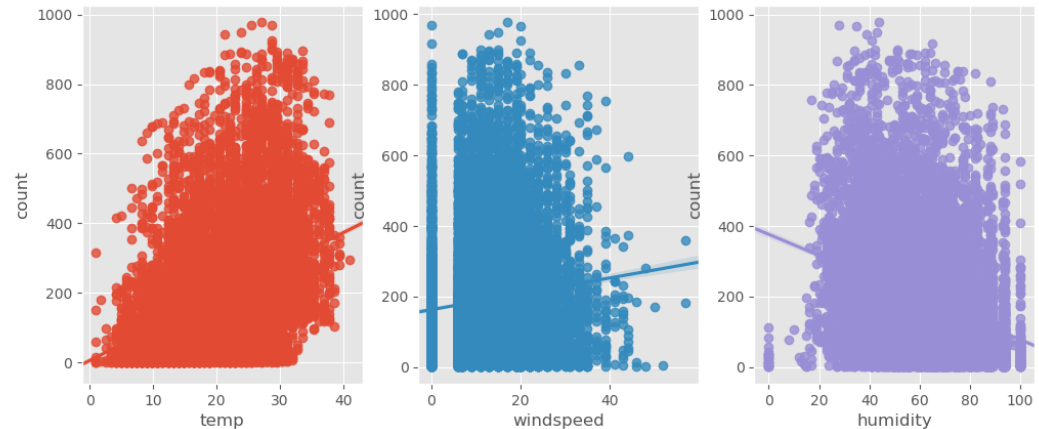
```
corrMatt =train[["temp","atemp","casual","registered","humidity","windspeed","count"]]  
corrMatt =corrMatt.corr()  
print(corrMatt)  
#  
mask = np.array(corrMatt)  
mask[np.tril_indices_from(mask)] = False  
  
fig, ax =plt.subplots()  
fig.set_size_inches(20,10)  
sns.heatmap(corrMatt, mask=mask, vmax=.8, square=True, annot=True)  
plt.show()
```

온도, 습도, 풍속, 연관관계 없다.
대여량과 가장 연관이 높은 건 registered로 등록된
대여자가 많지만, test데이터에는 이 값이 없다.
atemp와 temp는 0.98로 상관관계가 높지만 온도와
체감온도로 피처로 사용하기에 적합하지 않을 수 있다.



머신러닝 -처음부터 끝까지-자전거 대여량 예측

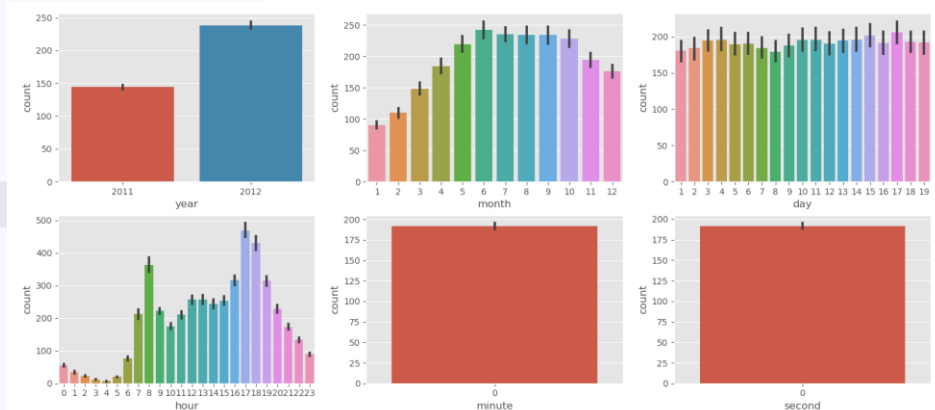
```
fig, (ax1,ax2,ax3) = plt.subplots(ncols=3)
fig.set_size_inches(12,5)
sns.regplot(x="temp", y="count", data=train, ax=ax1)
sns.regplot(x="windspeed", y="count", data=train, ax=ax2)
sns.regplot(x="humidity", y="count", data=train, ax=ax3)
plt.show()
```



#풍속의 경우 0에 숫자가 몰려 있는 것으로 보인다.
#아마도 관측되지 않은 수치에 대해 0으로 기록된 것이 아닐까 추측해본다.

머신러닝 -처음부터 끝까지-자전거 대여량 예측

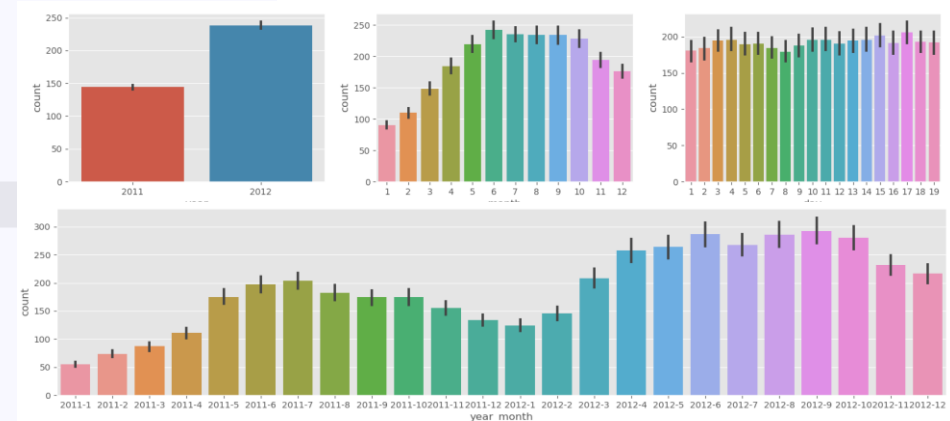
```
def concatenate_year_month(datetime):  
    return "{0}-{1}".format(datetime.year, datetime.month)  
train["year_month"] = train["datetime"].apply(concatenate_year_month)  
print(train.shape)  
train[["datetime", "year_month"]].head()  
  
fig,(ax1,ax2) =plt.subplots(nrows=1, ncols=2)  
fig.set_size_inches(18,4)  
|  
sns.barplot(data=train, x="year",y="count", ax=ax1)  
sns.barplot(data=train, x="month",y="count", ax=ax2)  
  
fig, ax3 = plt.subplots(nrows=1, ncols=1)  
fig.set_size_inches(18,4)  
sns.barplot(data=train, x="year_month",y="count",ax=ax3)  
plt.show()
```



#2011년보다 2012의 대여량이 더 많다
#겨울보다는 여름에 대여량이 많다.
#2011년과 2012년의 월별 데이터를 이어 보면 전체적으로 증가하는 추세다.

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
def concatenate_year_month(datetime):  
    return "{0}-{1}".format(datetime.year, datetime.month)  
  
train["year_month"] = train["datetime"].apply(concatenate_year_month)  
print(train.shape)  
train[["datetime", "year_month"]].head()  
  
fig,(ax1,ax2) =plt.subplots(nrows=1, ncols=2)  
fig.set_size_inches(18,4)  
  
sns.barplot(data=train, x="year",y="count", ax=ax1)  
sns.barplot(data=train, x="month",y="count", ax=ax2)  
  
fig, ax3 = plt.subplots(nrows=1, ncols=1)  
fig.set_size_inches(18,4)  
sns.barplot(data=train, x="year_month",y="count",ax=ax3)  
plt.show()
```



#2011년보다 2012의 대여량이 더 많다
#겨울보다는 여름에 대여량이 많다.
#2011년과 2012년의 월별 데이터를 이어 보면 전체적으로 증가하는 추세다.

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
# trainWithoutOutliers
trainWithoutOutliers = train[np.abs(train["count"] - train["count"].mean()) <= (3*train["count"].std())]

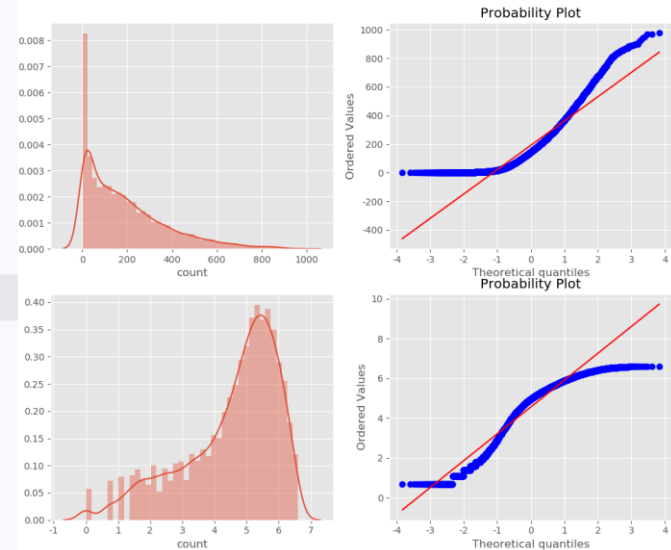
print(train.shape)
print(trainWithoutOutliers.shape)
```

→ (10886, 18)
(10739, 18)

count값의 데이터 분포도를 파악

```
figure, axes = plt.subplots(ncols=2, nrows=2)
figure.set_size_inches(12, 10)

sns.distplot(train["count"], ax=axes[0][0])
stats.probplot(train["count"], dist='norm', fit=True, plot=axes[0][1])
sns.distplot(np.log(trainWithoutOutliers["count"]), ax=axes[1][0])
stats.probplot(np.log1p(trainWithoutOutliers["count"]), dist='norm', fit=True, plot=axes[1][1])
plt.show()
```



#count변수가 한쪽에 치우쳐져 있다. 대부분의 기계학습은 종속변수가 normal 이어야 하기에 정규분포를 갖는 것이 바람직하다.
대안으로 outlier data를 제거하고 "count" 변수에 로그를 씌워 변경

머신러닝 -처음부터 끝까지-자전거 대여량 예측

```
# trainWithoutOutliers
trainWithoutOutliers = train[np.abs(train["count"] - train["count"].mean()) <= (3*train["count"].std())]

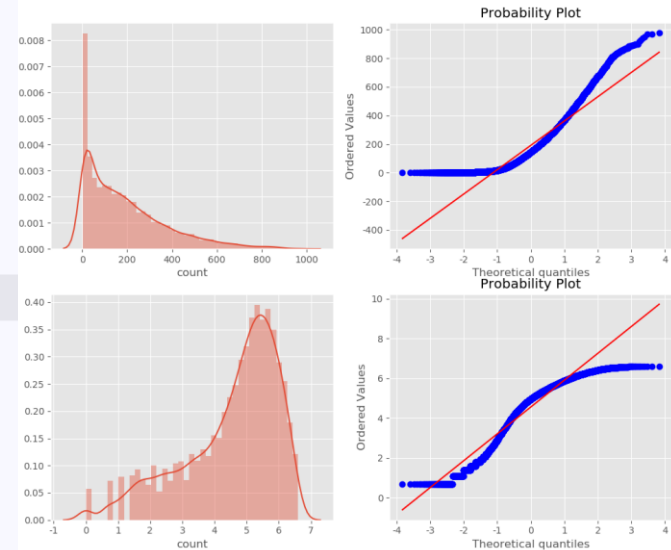
print(train.shape)
print(trainWithoutOutliers.shape)
```

→ (10886, 18)
(10739, 18)

count값의 데이터 분포도를 파악

```
figure, axes = plt.subplots(ncols=2, nrows=2)
figure.set_size_inches(12, 10)

sns.distplot(train["count"], ax=axes[0][0])
stats.probplot(train["count"], dist='norm', fit=True, plot=axes[0][1])
sns.distplot(np.log(trainWithoutOutliers["count"]), ax=axes[1][0])
stats.probplot(np.log1p(trainWithoutOutliers["count"]), dist='norm', fit=True, plot=axes[1][1])
plt.show()
```



#count변수가 한쪽에 치우쳐져 있다. 대부분의 기계학습은 종속변수가 normal 이어야 하기에 정규분포를 갖는 것이 바람직하다.
대안으로 outlier data를 제거하고 "count" 변수에 로그를 씌워 변경

머신러닝 -자전거 수요 예측(평가방법)

RMSLE

과대평가 된 항목보다는 과소평가 된 항목에 패널티를 준다.

오차(Error)를 제곱(Square)해서 평균(Mean)한 값의 제곱근(Root) 으로 값이 작을 수록 정밀도가 높다.

0에 가까운 값이 나올 수록 정밀도가 높은 값이다.

Submissions are evaluated one the Root Mean Squared Logarithmic Error (RMSLE)

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2}$$

```
def rmsle(y, y_, convertExp=True):
    if convertExp:
        y = np.exp(y),
        y_ = np.exp(y_)
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
```

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2}$$

```
from sklearn.metrics import make_scorer

def rmsle(predicted_values, actual_values):
    # 넘파이로 배열 형태로 바꿔준다.
    predicted_values = np.array(predicted_values)
    actual_values = np.array(actual_values)

    # 예측값과 실제 값에 1을 더하고 로그를 씌워준다.
    log_predict = np.log(predicted_values + 1)
    log_actual = np.log(actual_values + 1)

    # 위에서 계산한 예측값에서 실제값을 빼주고 제곱을 해준다.
    difference = log_predict - log_actual
    # difference = (log_predict - log_actual) ** 2
    difference = np.square(difference)

    # 평균을 낸다.
    mean_difference = difference.mean()

    # 다시 루트를 씌운다.
    score = np.sqrt(mean_difference)

    return score
```

머신러닝 -자전거 수요 예측(평가방법)

RMSLE

과대평가 된 항목보다는 과소평가 된 항목에 패널티를 준다.

오차(Error)를 제곱(Square)해서 평균(Mean)한 값의 제곱근(Root) 으로 값이 작을 수록 정밀도가 높다.

0에 가까운 값이 나올 수록 정밀도가 높은 값이다.

Submissions are evaluated one the Root Mean Squared Logarithmic Error (RMSLE)

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2}$$

```
def rmsle(y, y_, convertExp=True):
    if convertExp:
        y = np.exp(y),
        y_ = np.exp(y_)
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
```

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2}$$

```
from sklearn.metrics import make_scorer

def rmsle(predicted_values, actual_values):
    # 넘파이로 배열 형태로 바꿔준다.
    predicted_values = np.array(predicted_values)
    actual_values = np.array(actual_values)

    # 예측값과 실제 값에 1을 더하고 로그를 씌워준다.
    log_predict = np.log(predicted_values + 1)
    log_actual = np.log(actual_values + 1)

    # 위에서 계산한 예측값에서 실제값을 빼주고 제곱을 해준다.
    difference = log_predict - log_actual
    # difference = (log_predict - log_actual) ** 2
    difference = np.square(difference)

    # 평균을 낸다.
    mean_difference = difference.mean()

    # 다시 루트를 씌운다.
    score = np.sqrt(mean_difference)

    return score
```

머신러닝 -자전거 수요 예측(평가방법)

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

train = pd.read_csv("datasets/train.csv", parse_dates=["datetime"])
print(train.shape)
test = pd.read_csv("datasets/test.csv", parse_dates=["datetime"])
print(test.shape)
train["year"] = train["datetime"].dt.year
train["month"] = train["datetime"].dt.month
train["day"] = train["datetime"].dt.day
train["hour"] = train["datetime"].dt.hour
train["minute"] = train["datetime"].dt.minute
train["second"] = train["datetime"].dt.second
train["dayofweek"] = train["datetime"].dt.dayofweek
print(train.shape)

test["year"] = test["datetime"].dt.year
test["month"] = test["datetime"].dt.month
test["day"] = test["datetime"].dt.day
test["hour"] = test["datetime"].dt.hour
test["minute"] = test["datetime"].dt.minute
test["second"] = test["datetime"].dt.second
test["dayofweek"] = test["datetime"].dt.dayofweek
print(test.shape)
```

(10886, 12)

(6493, 9)

(10886, 19)

(6493, 16)

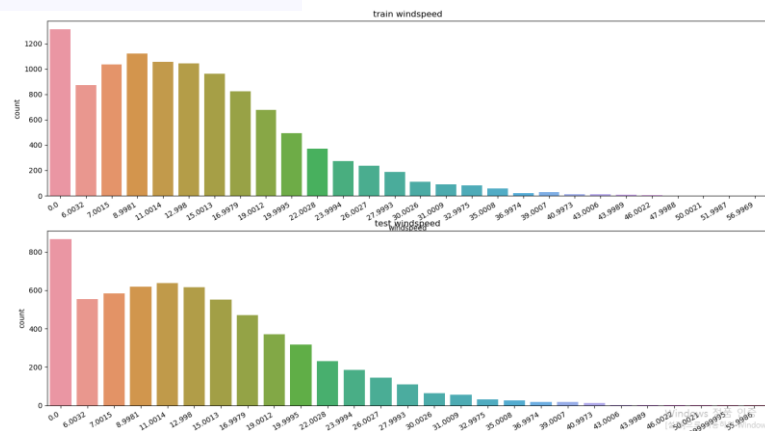
머신러닝 -자전거 수요 예측(평가방법)

```
# windspeed 풍속에 0 값이 가장 많다. => 잘못 기록된 데이터를 고쳐 줄 필요가 있음
fig, axes = plt.subplots(nrows=2)
fig.set_size_inches(18,10)

plt.sca(axes[0])
plt.xticks(rotation=30, ha='right')
axes[0].set(ylabel='Count', title='train windspeed')
sns.countplot(data=train, x='windspeed', ax=axes[0])

plt.sca(axes[1])
plt.xticks(rotation=30, ha='right')
axes[1].set(ylabel='Count', title='test windspeed')
sns.countplot(data=test, x='windspeed', ax=axes[1])
plt.show()
```

#0에 가장 많은 값에 몰려있다(측정이 되지 않은값=> 예측)



머신러닝 -자전거 수요 예측(평가방법)

```
# 풍속의 0값에 특정 값을 넣어준다.  
# 평균을 구해 일괄적으로 넣어줄 수도 있지만, 예측의 정확도를 높이는 데 도움이 될것 같진 않다.  
# train.loc[train["windspeed"] == 0, "windspeed"] = train["windspeed"].mean()  
# test.loc[train["windspeed"] == 0, "windspeed"] = train["windspeed"].mean()  
# 풍속이 0인것과 아닌 것의 세트를 나누어 준다.
```

```
trainWind0 = train.loc[train['windspeed'] == 0]  
trainWindNot0 = train.loc[train['windspeed'] != 0]  
print(trainWind0.shape)  
print(trainWindNot0.shape)
```


머신러닝 -자전거 수요 예측(평가방법)

```
from sklearn.ensemble import RandomForestClassifier

def predict_windspeed(data):
    # 풍속이 0인것과 아닌 것을 나누어 준다.
    dataWind0 = data.loc[data['windspeed'] == 0]
    dataWindNot0 = data.loc[data['windspeed'] != 0]
    # 풍속을 예측할 피처를 선택한다.
    wCol = ["season", "weather", "humidity", "month", "temp", "year", "atemp"]

    # 풍속이 0이 아닌 데이터들의 타입을 스트링으로 바꿔준다.
    dataWindNot0['windspeed'] = dataWindNot0['windspeed'].astype('str')

    # 랜덤포레스트 분류기를 사용한다.
    rfModel_wind = RandomForestClassifier()

    # wCol에 있는 피처의 값을 바탕으로 풍속을 학습시킨다.
    rfModel_wind.fit(dataWindNot0[wCol], dataWindNot0['windspeed'])

    # 학습한 값을 바탕으로 풍속이 0으로 기록 된 데이터의 풍속을 예측한다.
    wind0Values = rfModel_wind.predict(X=dataWind0[wCol])
```


머신러닝 -자전거 수요 예측(평가방법)

```
# 값을 다 예측 후 비교해 보기 위해
# 예측한 값을 넣어 줄 데이터 프레임을 새로 만든다.
predictWind0 = dataWind0
predictWindNot0 = dataWindNot0

# 값이 0으로 기록 된 풍속에 대해 예측한 값을 넣어준다.
predictWind0["windspeed"] = wind0Values

# dataWindNot0 0이 아닌 풍속이 있는 데이터프레임에 예측한 값이 있는 데이터프레임을 합쳐준다.
data = predictWindNot0.append(predictWind0)
```

```
# 풍속의 데이터타입을 float으로 지정해 준다.
data["windspeed"] = data["windspeed"].astype("float")

data.reset_index(inplace=True)
data.drop('index', inplace=True, axis=1)
return data
```

0값을 조정한다.

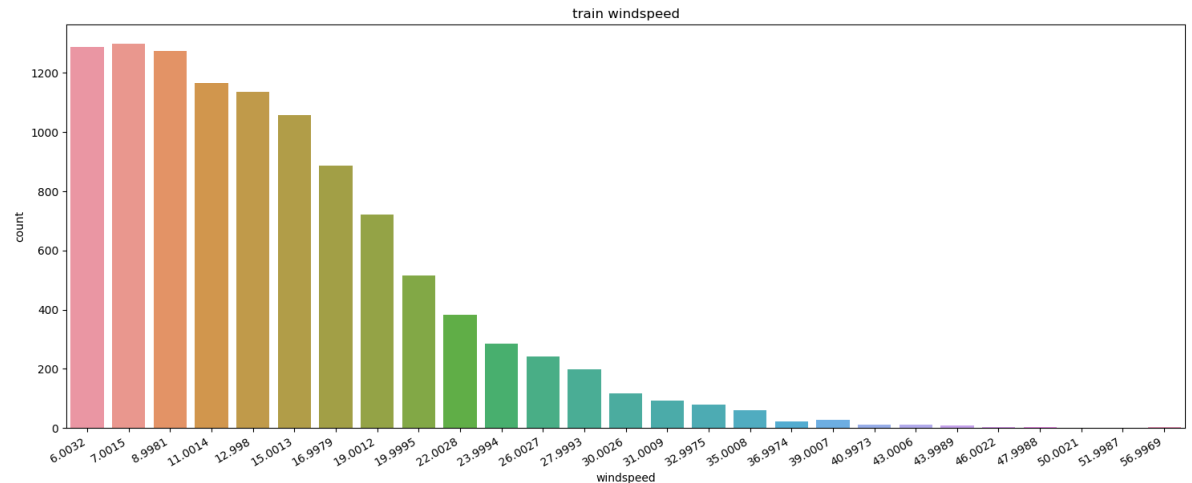
머신러닝 -자전거 수요 예측(평가방법)

```
# #####feature engineering#####33333
train = predict_windspeed(train)
# test = predict_windspeed(test)

# widspeed 의 0값을 조정한 데이터를 시각화

fig, ax1 = plt.subplots()
fig.set_size_inches(18,6)

plt.sca(ax1)
plt.xticks(rotation=30, ha='right') #글씨가 30도로 겹쳐보이지 않음
ax1.set(ylabel='Count',title='train windspeed')
sns.countplot(data=train, x="windspeed", ax=ax1)
plt.show() #0인데이터 사라짐
```



머신러닝 -자전거 수요 예측(평가방법)

```
categorical_feature_names = ["season", "holiday", "workingday", "weather",  
                             "dayofweek", "month", "year", "hour"]  
  
for var in categorical_feature_names:  
    train[var] = train[var].astype("category")  
    test[var] = test[var].astype("category")  
  
feature_names = ["season", "weather", "temp", "atemp", "humidity", "windspeed",  
                 "year", "hour", "dayofweek", "holiday", "workingday"]  
  
print(feature_names)  
X_train = train[feature_names] #새로운 행렬  
  
print(X_train.shape)  
X_train.head()  
X_test = test[feature_names]  
  
print(X_test.shape)  
X_test.head()  
  
label_name = "count"  
  
y_train = train[label_name]  
  
print(y_train.shape)  
y_train.head()
```

#신호와 잡음을 구분
#피처가 많다고 해서 무조건 좋은 성능을 내지 않는다
#피처를 하나씩 추가 및 변경해 가면서 성능이 좋지 않은 피처가 제거

연속형 feature와 범주형 feature
연속형 feature = ["temp", "humidity", "windspeed", "atemp"]
범주형 feature의 type을 category로 변경 해 준다.

머신러닝 -자전거 수요 예측(평가방법)

```
from sklearn.metrics import make_scorer

def rmsle(predicted_values, actual_values): #정밀도 0에 가까울수록 예측이 좋다
    # 넘파이로 배열 형태로 바꿔준다.
    predicted_values = np.array(predicted_values)
    actual_values = np.array(actual_values)

    # 예측값과 실제 값에 1을 더하고 로그를 씌워준다.
    log_predict = np.log(predicted_values + 1)
    log_actual = np.log(actual_values + 1)

    # 위에서 계산한 예측값에서 실제값을 빼주고 제곱을 해준다.
    difference = log_predict - log_actual
    # difference = (log_predict - log_actual) ** 2
    difference = np.square(difference)

    # 평균을 낸다.
    mean_difference = difference.mean()

    # 다시 루트를 씌운다.
    score = np.sqrt(mean_difference)

    return score

rmsle_scorer = make_scorer(rmsle)
```

머신러닝 -자전거 수요 예측(평가방법)

```
#####교차검증#####
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
max_depth_list = []
```

```
model = RandomForestRegressor(n_estimators=100, # 높일수록 더 높은 값 시간이 오래 걸림
                              n_jobs=-1,
                              random_state=0)
```

```
print(model)
```

oob_score

Score= 0.33116

```
score = cross_val_score(model, X_train, y_train, cv=k_fold, scoring=rmsle_scorer)
```

```
score = score.mean()
```

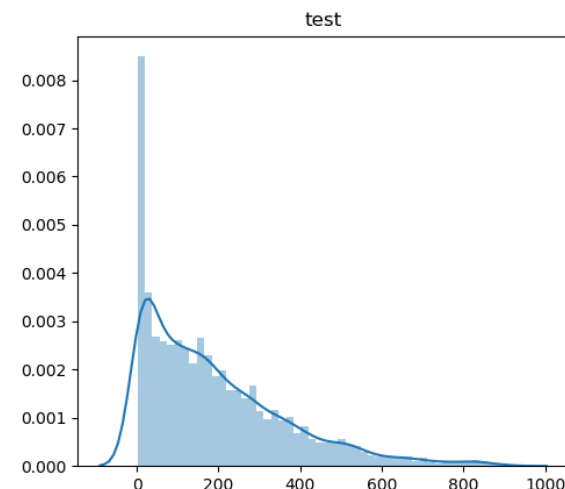
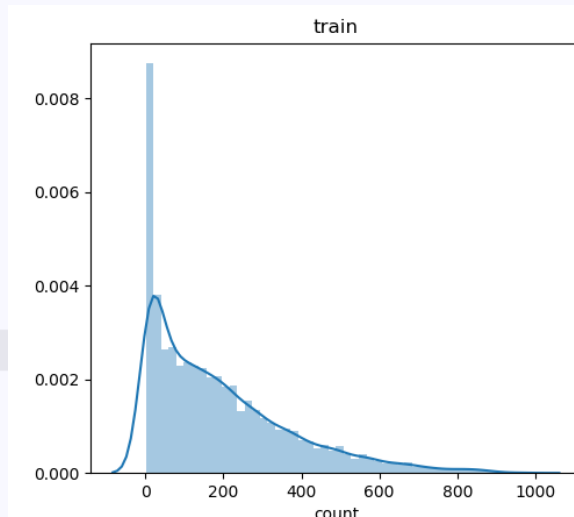
```
# 0에 근접할수록 좋은 데이터
```

```
print("Score= {0:.5f}".format(score))
```

```
# #####train#####
```

머신러닝 -자전거 수요 예측(평가방법)

```
# #####train#####  
# 학습시킴, 피팅(옷을 맞출 때 사용하는 피팅을 생각함) - 피쳐와 레이블을 넣어주면 알아서 학습을 함  
model.fit(X_train, y_train)  
# 예측  
predictions = model.predict(X_test)  
  
print(predictions.shape)  
print(predictions[0:10])  
# 예측한 데이터를 시각화 해본다.  
fig,(ax1,ax2)= plt.subplots(ncols=2)  
fig.set_size_inches(12,5)  
sns.distplot(y_train,ax=ax1,bins=50)  
ax1.set(title="train")  
sns.distplot(predictions,ax=ax2,bins=50)  
ax2.set(title="test")  
  
plt.show()  
  
submission = pd.read_csv("datasets/sampleSubmission.csv")  
print(submission)  
  
submission["count"] = predictions  
print(submission.shape)  
submission.head()  
submission.to_csv("datasets/Score_{0:.5f}_submission.csv".format(score), index=False)
```




머신러닝 -자전거 수요 예측(평가방법)

10

datasets

- sampleSubmission.csv
- Score_0.00489_submission.csv
- Score_0.33098_submission.csv
- Score_0.33103_submission.csv
- Score_0.33116_submission.csv
- Score_0.33177_submission.csv
- test.csv
- train.csv

<https://www.kaggle.com/c/bike-sharing-demand/submit>



Bike Sharing Demand

Forecast use of a city bikeshare system
3,251 teams · 3 years ago

[Overview](#) [Data](#) [Kernels](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#) [My Submissions](#) [Late Submission](#)

Your most recent submission


Name	Submitted	Wait time	Execution time	Score
Score_0.00489_submission.csv	an hour ago	0 seconds	0 seconds	0.49941

Complete

[Jump to your position on the leaderboard](#)

Make a submission for [Miran](#)

Step 1
Upload submission file



Upload Submission File