강화학습으로 풀어보는 슈퍼마리오 part 1.

강화학습(reinforcement learning)은 머신러닝의 꽃이라고도 불린다. 강화학습의 매력에 깊이 빠진 필자는 더 많은 분들에게 강화학습의 '복음'을 전하기 위해 튜토리얼을 작성하고자 한다. 지금 IT업계에서 일하고 계시는 분들 중 많은 분이 경험하셨을법한 유명한 게임, '슈퍼마리오'를 강화학습 튜토리얼의 테마로 잡았다. 딥마인드(Deepmind)가 공개한 DQN(Deep Q Network) 알고리듬으로부터 최신 트렌드인 A3C까지 여러 가지 알고리듬을 하나씩 소개하면서 슈퍼마리오라는 게임 세상을 강화학습으로 풀어 보도록 하자. 슈퍼마리오에 강화학습 적용하기는 총 세 번에 걸쳐 연재된다.

튜토리얼을 통해 여러분은 이론이 아닌 현실에서 돌아가는 프로그램을 볼 수 있게 될 것이다. 우선 트레이닝이 되는 프로그램을 따라서 돌려 본 후에 이론을 설명하는 방식으로 진행해서, 조금 더 쉽고 흥미롭게 강화학습을 공부하실 수 있도록 진행할 예정이다.

글 | 송호연 chris.song@kakaocorp.com

Kakao R&D Center Data Engineer.
강화학습과 썸을 타다가 최근에 연애를 시작했습니다. 자기 전에도, 아침에 일어났을 때도 강화학습 생각이 납니다. 앞으로 세상을 파괴적으로 혁신시킬 범용인공지능(Artificial General Intelligence)에 완전히 빠져있습니다. 카카오에서는 대규모 데이터 유저 프로파일링, 머신러닝 기술을 활용해 현실의 문제를 해결하는 데이터 엔지니어로 일하고 있습니다.

가장 먼저, 강화학습의 환경을 먼저 세팅해 보도록 하자.

준비물
- Anaconda3 (Python3.6)
- Homebrew (MacOS 유저)
- GIT
- pip

Pip 라이브러리
- https://github.com/chris-chris/gym-super-mario (forked from ppaquette/gym-super-mario)
- https://github.com/openai/baselines
- https://github.com/openai/gym

게임 에뮬레이터
- FCEUX (http://www.fceux.com)

깃허브(Github) 프로젝트
- https://github.com/chris-chris/mario-rl-tutorial

슈퍼마리오 강화학습 실행 환경 설정

강화학습 스크립트가 실행이 가능하도록 환경설정을 성공하는 것이 첫 번째 목표다. 게임 실행이 되어야 강화학습 코드도 작성하고 하이퍼파라미터도 튜닝할 수 있을 테니 말이다. 오픈AI(OpenAI) 짐(gym)[1]에 설치된 기본 아타리(Atari) 게임 환경들이 있지만, 좀 더 재미있는 슈퍼마리오 강화학습 환경을 만들기 위해선 조금 더 수고로운 환경설정이 필요하다. 이 튜토리얼은 맥OS(MacOS) 환경 기준으로 진행한다(윈도우의 경우엔 cygwin과 fceux의 조합을 활용해서 설정이 가능하다). (※지면에 다 담기 어려운 점을 고려해 설치가 쉬운 아나콘다3(Anaconda3) 설치와 기본적인 GIT 사용법은 설명하지 않겠다.)

가장 먼저 fceux 를 설치해야 한다. fceux는 PC 환경에서 오래된 고전 게임들을 실행할 수 있는 에뮬레이터다. 가장 간단하게 설치하는 방법은 바로 홈브류(Homebrew)를 활용한 설치법이다. 홈브류가 깔려 있지 않다면, 터미널(Terminal)을 열고 간단하게 아래 명령어를 실행해서 설치가 가능하다. (홈브류의 설치 가이드는 https://brew.sh 에서 확인할 수 있다)

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/master/install)"
```

홈브류 설치가 완료되면, 다시 터미널을 열고 아래 명령어로 fceux를 설치하자.

```
brew install fceux
```

위 명령어를 실행하면 다음과 같은 결과가 표시된다. 홈브류를 활용하면 다양한 패키지를 편하게 설치할 수 있다.

[그림 1] fceux 설치

Fceux 프로그램을 설치했다면, 터미널을 열고 아래 명령어들을 실행해서 필요한 pip 라이브러리들을 설치한다.

첫 번째로 필요한 pip 패키지는 OpenAI의 gym이다.

```
pip install gym
```

그리고 이미지 처리를 위한 opencv-python 패키지를 설치한다.

```
pip install opencv-python
```

그다음 설치할 pip 패키지는 OpenAI의 baselines[2]다. 주의할 점은, pip install baselines 명령으로 baselines 를 설치하면 최신 버전을 받을 수 없다는 것이다. 깃허브에 올라와 있는 최신 빌드를 설치하기 위해선 아래 명령으로 설치하기 바란다.

```
pip install git+https://github.com/openai/baselines
```

그리고 설치할 pip 패키지는 슈퍼마리오의 다양한 난이도가 들어 있는 gym-super-mario 패키지다. 원래 이 환경은 필립 파퀘트(Phillip Paquette, @ppaquette)[3]가 제작했으며, 파퀘트의 gym-super-mario에서 멀티 에이전트(multi agent) 처리 등 몇 가지 이슈를 수정하기 위해 포크(fork)한 후 코드를 수정했다. 아래 명령어를 실행해서 필자가 수정한 코드로 환경을 설치하자.

```
pip install git+https://github.com/chris-chris/gym-super-mario
```

그리고 슈퍼마리오 강화학습 예제 프로젝트를 클론(clone)하자.

```
git clone https://github.com/chris-chris/mario-rl-tutorial
```

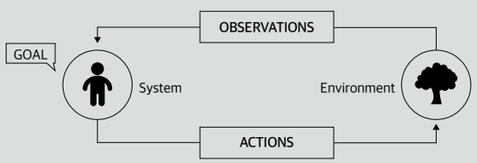
이제 클론을 받은 프로젝트 폴더로 들어간 후 아래 명령어를 실행하면 슈퍼마리오 에뮬레이터가 화면에 뜨면서 학습이 시작된다.

```
python train.py
```

강화학습 Observation Space 설정

이제 슈퍼마리오의 학습 환경에 대해 설명한다. 딥마인드에서 발표한 슬라이드에서 잘 나타나 있듯이, 강화학습의 학습환경은 아래와 같이 Observations, Actions 두 가지 정보로 이루어진다.

[그림 2] 강화학습 프레임워크[*4]



이제 observation space와 action space 각각이 어떻게 처리되는지를 알아보자.

첫 번째로, observation space는 다른 Atari 게임들과 마찬가지로 화면의 RGB픽셀을 그대로 입력 받는다. 슈퍼마리오의 observation space의 구체적인 설정은 아래와 같다.

```
spaces.Box(low=0, high=255, shape=(224, 256, 3))
```

화면의 세로 사이즈(height)는 224, 가로 사이즈(width)는 256, 색상의 종류(RGB)는 3, 그리고 각 색상의 수치 값의 범위는 0부터 255까지다. 실제로, 화면의 픽셀데이터를 받아서 출력해 보면 아래와 같이 나온다.

```
[[[200  76  12]
  [200  76  12]
  [200  76  12]
  ...,
  [252 188 176]
  [200  76  12]
  [200  76  12]]]
```

하나의 픽셀에 RGB 값 3가지가 들어 있다. 224×255 2차원 행렬 안에 각 픽셀이 3가지 RGB 값을 가지고 있으니 3차원 행렬이 된다.

강화학습에서는 이렇게 들어오는 데이터를 CNN(Convolution Neural Network)에 넣어서 강화학습 모델이 화면의 여러 피처들을 인식할 수 있도록 한다. 그런데 observation space데이터를 CNN에 넣기 전에 중요한 과정이 있다. 바로, 데이터의 사이즈를 줄이는 것이다. OpenAI gym상에서 구동되는 아타리(Atari) 게임 환경들과 예시들을 보면 RGB를 그레이스케일로 변환하고 이미지 사이즈를 줄이는 코드를 확인할 수 있다.

일반적으로 이렇게 RGB값이 들어오면, 연산을 단순화하기 위해 RGB값을 그레이스케일(회색)로 변환한다. 그리고 이미지 사이즈를 84 × 84 사이즈로 줄인다.

```
wrapper.py : 화면에 출력된 RGB 이미지 데이터를 그레이스케일로 변환하고 크기를 줄이는 Wrapper

import cv2
import gym
import numpy as np
from gym import spaces

class ProcessFrame84(gym.ObservationWrapper):
 def __init__(self, env=None):
  super(ProcessFrame84, self).__init__(env)
  self.observation_space = spaces.Box(low=0, high=255, shape=(84, 84, 1))

 def _observation(self, obs):
  return ProcessFrame84.process(obs)

 @staticmethod
 def process(img):
  img = img[:, :, 0] * 0.299 + img[:, :, 1] * 0.587 + img[:, :, 2] * 0.114
  x_t = cv2.resize(img, (84, 84), interpolation=cv2.INTER_AREA)
  x_t = np.reshape(x_t, (84, 84, 1))
  x_t = np.nan_to_num(x_t)
  return x_t.astype(np.uint8)
```

이렇게 화면의 RGB값을 전처리하여 작은 데이터 사이즈로 만들어 보면, 아까 예시로 들었던 행렬 데이터에서 RGB데이터는 그레이스케일로 변하면서 3배수로 줄어들었고(1/3로 감소), 공간 사이즈(가로×세로)는 약 8.12배수로 줄어들었다(약 12.3% 감소). 이렇게 피처를 인식할 데이터의 크기를 상당히 줄여서 우리는 강화학습 모델을 좀 더 효율적으로 학습시킬 수 있게 된다.

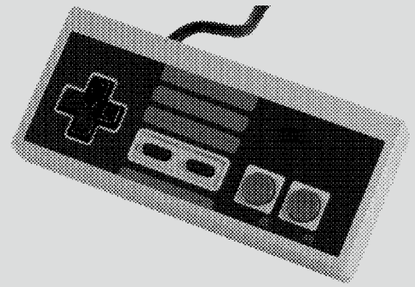
강화학습 Action Space 설정

이제 action space를 살펴보자. Action space는 강화학습 모델이 에이전트에 명령을 전달할 때 명령의 가짓수다. Atari 게임 중 Pong의 경우엔 action space가 정말로 작다. 위로 가거나 아래로 가면 된다. 하지만, 슈퍼마리오는 그보다 좀 더 복잡한 action space를 가지고 있다. 바로 방향키와 A, B 버튼이다.

· 방향키: 상, 하, 좌, 우 4가지
· 버튼: 버튼은 A, B 2가지

그렇다면 action space의 크기는 6일까? 아니다. 우선, 슈퍼마리오에서 우리가 흔히 내리는 명령들을 한번 생각해 보자. 우리는 오른쪽 방향키를 눌러서 앞으로 가면서 동시에 A 버튼을 눌러서 점프를 할 수 있다. 우리는 6가지 명령을 동시에 내릴 수 있다. 그러면 갑자기 space가 엄청나게 커진다. 2의 6제곱이 된다. 최대 경우의 수는 64다. 하지만, 64가지의 경우의 수가 모두 쓸모 있지는 않다. 예를 들어, 위 방향키와 아래 방향키를 굳이 같이 누를 필요가 없으니 말이다.

[그림 3] 게임용 콘솔(console)



흔히 쓰일 만한 action space를 14가지로 추려 보았다. 14가지 각각의 명령 조합은 아래와 같다.

```
0 : 아무것도 안 함
1 : 위
2 : 아래
3 : 왼쪽
4 : 왼쪽 + A
5 : 왼쪽 + B
6 : 왼쪽 + A + B
7 : 오른쪽
8 : 오른쪽 + A
9 : 오른쪽 + B
10 :  오른쪽 + A + B
11 : A
12 : B
13 : A + B
```

이렇게 14개의 조합을 만들어 놓고 게임 환경에 명령을 보낼 때는 각각의 명령을 동시에 입력 가능한 형식으로 변환시켜서 전달하게 만들 것이다.

예를 들어 오른쪽 키를 누르는 명령과 A 키를 동시에 누르는 8번 명령은 [0, 0, 0, 1, 1, 0] 이런 형식의 명령으로 변환되어 게임 에이전트에게 전달된다. 그리고 왼쪽으로 움직이는 3번 명령은 [0, 1, 0, 0, 0, 0] 데이터로 변환되어 에이전트에 전달된다. 이렇게 14가지 명령을 게임 에이전트가 알아듣기 좋게 변환시키는 래퍼(Wrapper)가 바로 MarioActionSpaceWrapper다.

```
MarioActionSpaceWrapper : 슈퍼마리오 게임의 Action Space를 재정의해주는 gym Wrapper

class MarioActionSpaceWrapper(gym.ActionWrapper):

 mapping = {
  0: [0, 0, 0, 0, 0, 0],  # NOOP
  1: [1, 0, 0, 0, 0, 0],  # Up
  2: [0, 0, 1, 0, 0, 0],  # Down
  3: [0, 1, 0, 0, 0, 0],  # Left
  4: [0, 1, 0, 0, 1, 0],  # Left + A
  5: [0, 1, 0, 0, 0, 1],  # Left + B
  6: [0, 1, 0, 0, 1, 1],  # Left + A + B
  7: [0, 0, 0, 1, 0, 0],  # Right
  8: [0, 0, 0, 1, 1, 0],  # Right + A
  9: [0, 0, 0, 1, 0, 1],  # Right + B
  10: [0, 0, 0, 1, 1, 1],  # Right + A + B
  11: [0, 0, 0, 0, 1, 0],  # A
  12: [0, 0, 0, 0, 0, 1],  # B
  13: [0, 0, 0, 0, 1, 1],  # A + B
 }

 def __init__(self, env):
  super(MarioActionSpaceWrapper, self).__init__(env)
  self.action_space = spaces.Discrete(14)

 def _action(self, action):
  return self.mapping.get(action)

 def _reverse_action(self, action):
  for k in self.mapping.keys():
   if(self.mapping[k] == action):
    return self.mapping[k]
  return 0
```

우리가 만들어 본 observation space wrapper와 action space wrapper를 환경에 적용하는 법은 간단하다. 아래 코드와 같이 실행하면 된다.

```
- train_dqn

환경을 생성하고
Action Space Wrapper를 적용하고
Observation Space Wrapper를 적용한 코드

def train_dqn(env_id, num_timesteps):
 """Train a dqn model.

  Parameters
  -------
  env_id: environment to train on
  num_timesteps: int
      number of env steps to optimizer for

 """

 # 1. Create gym environment
 env = gym.make(FLAGS.env)
 # 2. Apply action space wrapper
 env = MarioActionSpaceWrapper(env)
 # 3. Apply observation space wrapper to reduce input size
 env = ProcessFrame84(env)
```

MarioActionSpaceWrapper는 action space를 커스터마이징하는 wrapper이고, ProcessFrame84는 observation space의 사이즈를 줄이는 wrapper이다. 이제 학습을 위한 action space와 observation space 설정을 완료했다.

강화학습 Q-Function 모델 구조 설정

이제 Q-Function 모델에 대해서 살펴보자.

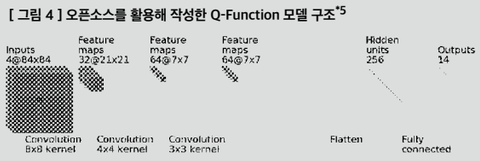
```
- train_dqn

환경을 생성하고
Action Space Wrapper를 적용하고
Observation Space Wrapper를 적용한 후
```

CNN 모델을 적용한 Q 함수를 생성한 코드

```
def train_dqn(env_id, num_timesteps):
 """Train a dqn model.

  Parameters
  -------
  env_id: environment to train on
  num_timesteps: int
     number of env steps to optimizer for

  """

 # 1. Create gym environment
 env = gym.make(FLAGS.env)
 # 2. Apply action space wrapper
 env = MarioActionSpaceWrapper(env)
 # 3. Apply observation space wrapper to reduce input size
 env = ProcessFrame84(env)
 # 4. Create a CNN model for Q-Function
 model = cnn_to_mlp(
 convs=[(32, 8, 4), (64, 4, 2), (64, 3, 1)],
 hiddens=[256],
 dueling=FLAGS.dueling
 )
```

우리는 총 3개의 convolution layer를 사용할 것이다. 그리고 마지막에 CNN Layer를 Flatten 방식으로 1자로 펼친 후 fully connected 방식으로 14개의 action space에 연결할 것이다. 그림으로 그려서 설명하는 게 가장 이해하기 쉬울 것 같아 오픈소스로 작성된 시각화 예제를 활용하여 Q-function을 그려 보았다.

[그림 4] 오픈소스를 활용해 작성한 Q-Function 모델 구조*5



우리는 3개의 CNN 레이어를 활용해 화면상의 물체를 식별한 후, 식별한 결과를 256개 유닛으로 이루어진 fully connected로 연결한 후 최종적으로 14개의 결과를 받아 오도록 하였다.

여기서 84×84 흑백 이미지가 4겹으로 된 것을 확인할 수 있는데, 바로 최근 4프레임의 화면을 쌓아서 입력값으로 넣은 것이다. 4개의 84×84 사이즈 흑백 이미지를 14개의 명령 신호와 연결시키는 모델을 완성한 것이다.

랜덤 에이전트(Random Agent) 만들기

실제로 움직이는 에이전트(agent)를 만들어서 돌려 보자. 랜덤 에이전트는 각 단계(step)마다 취할 수 있는 모든 행동(action)을 랜덤으로 골라서 명령을 보내는 에이전트이다. 이것을 강화학습이라 할 수는 없지만 게임 환경(enviroment)에서 action Space에 대한 프로세스를 이해하기 위해 돌려 보도록 한다.

random_agent.py : 랜덤으로 명령을 보내는 agent 예제

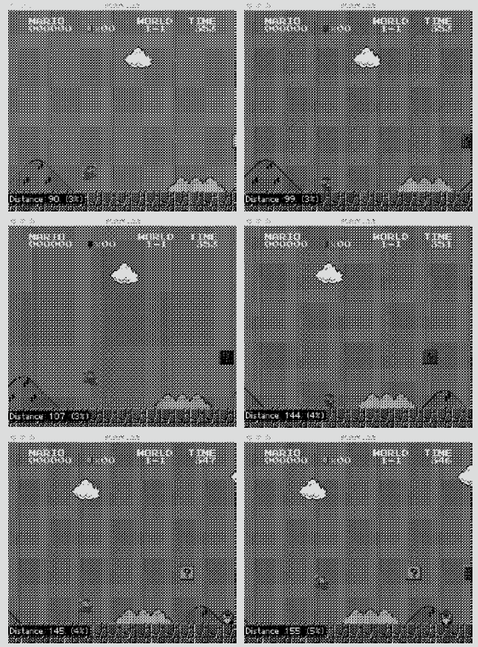
```
import gflags as flags
import sys
import gym

import ppaquette_gym_super_mario

from wrappers import MarioActionSpaceWrapper, ProcessFrame84

FLAGS = flags.FLAGS
flags.DEFINE_string("env", "ppaquette/SuperMarioBros-1-1-v0", "RL
environment to train.")

class RandomAgent(object):
 """The world's simplest agent!"""
 def __init__(self, action_space):
  self.action_space = action_space

 def act(self, observation, reward, done):
  return self.action_space.sample()

def main():
 FLAGS(sys.argv)
 # Choose which RL algorithm to train.

 print("env : %s" % FLAGS.env)

 # 1. Create gym environment
 env = gym.make(FLAGS.env)
 # 2. Apply action space wrapper
 env = MarioActionSpaceWrapper(env)
 # 3. Apply observation space wrapper to reduce input size
 env = ProcessFrame84(env)

 agent = RandomAgent(env.action_space)

 episode_count = 100
 reward = 0
 done = False

 for i in range(episode_count):
  ob = env.reset()
  while True:
   action = agent.act(ob, reward, done)
   ob, reward, done, _ = env.step(action)
   if done:
    break

if __name__ == '__main__':
 main()
```

랜덤 에이전트의 알고리듬은 단순하다. 매 스텝마다 슈퍼마리오 게임 환경에서 취할 수 있는 14개의 action space 중 하나를 랜덤으로 골라서 명령을 보내는 것이다. 이 random_agent.py를 실행하기 위해서는 아래 명령어를 실행하면 된다.

```
python random_agent.py
```

그러면, 화면*6에서 랜덤으로 움직이는 슈퍼마리오를 확인할 수 있다.

[그림 5] 랜덤으로 움직이는 슈퍼마리오*7



Agent 학습시키기

학습을 시킬 땐 간단히 아래 명령어를 실행하면 된다. 프로그램이 실행되면 화면에 슈퍼마리오가 실행되면서 학습이 진행된다.

```
python train.py --log=stdout
```

위 명령으로 실행하면, 학습 결과가 콘솔에 찍힌다. 하지만 기록이 길어지다 보면 보기가 불편할 것이다. 그래서 텐서보드(tensorboard)로 확인할 수 있는 방법을 소개한다.

```
python train.py --log=tensorboard
```

위 명령어를 실행시키면, 프로젝트 폴더 내에 텐서보드 폴더가 생성될 것이다. 여기에는 텐서보드에 맞는 로그(log)가 쌓이기 시작한다. 이 학습로그를 텐서보드를 띄워서 확인해 보자.

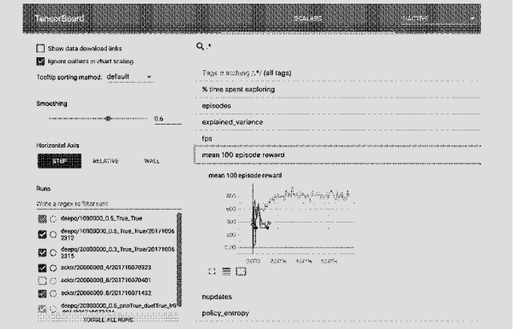
```
tensorboard logdir=tensorboard
```

이렇게 명령어를 실행하면, 아래와 같이 콘솔에 찍힐 것이다.

```
$tensorboard --logdir=tensorboard
TensorBoard 0.1.6 at http://Chrisui-MacBook-Pro.local:6006 (Press CTRL+C
to quit)
```

그러면 브라우저를 열고 이렇게 이 주소(http://localhost:6006)로 이동해 보자. 그러면 텐서보드에서 학습결과를 확인할 수 있을 것이다.

[그림 6] 텐서보드를 통해 확인한 학습 결과



학습된 Agent 실행시키기

학습을 진행하면서 최근 100건의 평균 보상(reward)이 최고치를 경신하면, 현재 학습된 모델을 파일로 저장하도록 만들어 두었다. 프로젝트 폴더 내에 models/deepq/mario_reward_930.6.pkl 파일이 들어 있는데, 이 모델로 실행시켜 보도록 하자.

```
python enjoy.py --algorithm=deepq --file=mario_reward_930.6.pkl
```

위 명령을 실행하면 학습된 모델을 실행시켜 볼 수 있다. --file= 파라미터에 본인이 만들어 낸 파일명 매개변수로 입력하면 된다.

이렇게, 슈퍼마리오 게임에 강화학습 에이전트를 구현하기 위한 세부 작업을 설명하였다. 다음 편에서는 DQN 강화학습의 이론에 대해서 설명한 후, Prioritized Replay Memory, Dueling DQN 등 점진적으로 개선된 방법론을 이론과 같이 예제를 하나씩 소개하고자 한다.

*1 참고 | https://github.com/openai/gym *2 참고 | https://github.com/openai/baselines *3 참고 | https://github.com/ppaquette/gym-super-mario *4 참고 | Demis Hassabis, CEO, DeepMind Technologies - The Theory of Everything, Youtube, https://www.youtube.com/watch?v=rbsqaJwpu6A *5 참고 | https://github.com/gwding/draw_convnet *6 편집자 주 | [그림 5]라고 표기했지만, 실제 필자가 집필진에서 넘겨 주신 파일은 움직이는 영상입니다. 물리적 한계 상 필자가 의도하는 바를 간행물에는 온전히 담지 못했습니다. 해당 영상은 온라인에는 온전히 담았습니다. 다음 링크를 참고해 주세요(https://brunch.co.kr/@kakao-it/124/).
*7 참고 | https://youtu.be/bicOms7rkcA