# 2020 OSS 개발자 포럼 겨울 캠프 2 일차 과제

## ~ AlphaZero 학습시키기 ~

## 주의사항

대전은 **2020 년 1 월 12 일 오전 10 시 00 분**부터입니다.

판의 크기는 7x7 이며 **돌을 놓는데 걸리는 시간은 최대 10 초로 제한**합니다. **10 초를 초과하면 자동 패배 처리합니다.**

과제와 궁금한 점이 있다면 카카오톡 채팅방에 문의해주시기 바랍니다.

## 과제 안내

오늘 수업 시간 때 배웠던 AlphaZero 를 기반으로 여러분 만의 AlphaZero 를 학습시켜봅시다.

### 1. 학습시키기

AlphaZero 폴더에서 다음의 명령어를 통해 AlphaZero 를 학습시킬 수 있습니다.

```
python3 train.py
```

학습된 결과는 models 폴더에 checkpoint-<숫자>.bin 파일로 저장되며, 학습 과정에서 loss 의 변화 추이는 tensorboard 를 통해 확인할 수 있으며, tensorboard 는 다음과 같이 켤 수 있습니다.

```
python3 -m tensorboard.main --logdir runs --port 52525 --bind_all
```

(만약 52525 가 아닌 다른 포트를 열었다면 52525 자리에 해당 포트를 입력해야 합니다.)

### 2. Agent 의 플레이 구경하는 방법

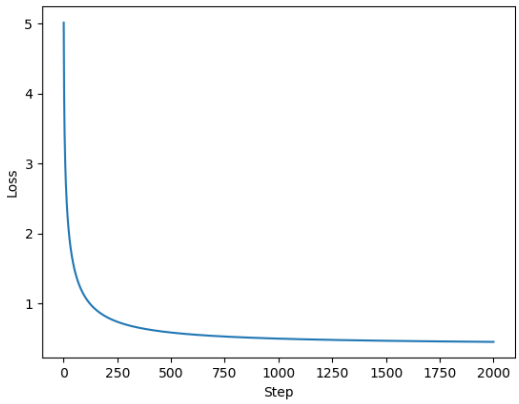
Agent 의 플레이를 구경하기 위해선 AlphaZero 폴더의 bot_v_bot.py 파일의 다음 부분을 수정하면 됩니다. RandomBot 과 붙어도 되고, 어제 만든 Agent 와 대결해도 되며, AlphaZero 봇끼리 붙여도 됩니다. 실행 방법은 어제와 같습니다.

```
bots = {
    types.Player.black: ~
    types.Player.white: ~
}
```

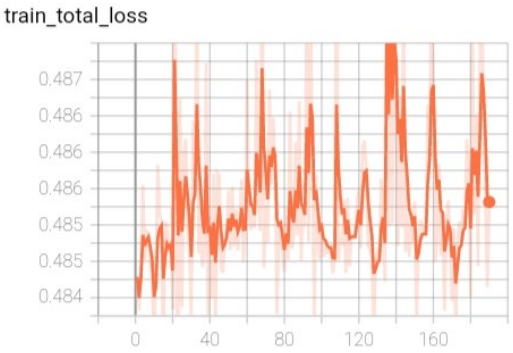
### 3. AlphaZero 의 게임별 noise alpha 값

| Chess | Shogi | Go |
|---|---|---|
| 0.3 | 0.15 | 0.03 |

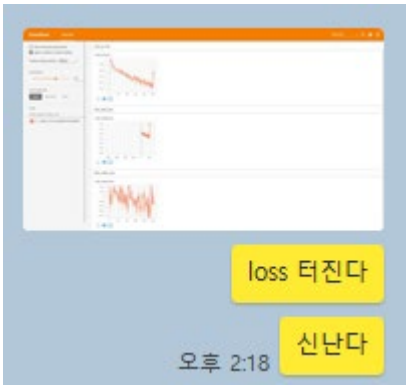
### 4. Loss 그래프의 이상과 현실

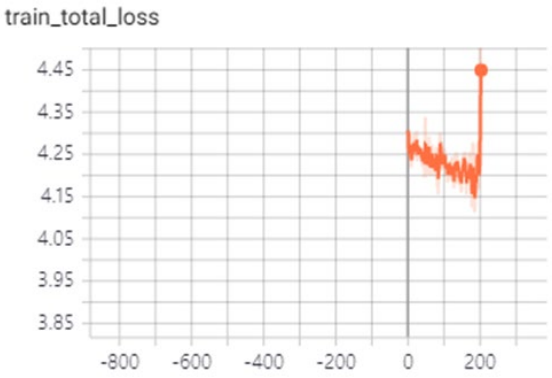



<희망편>




<현실편>




<파멸편>

## 5. 설정값

train.py 파일엔 다음과 같은 설정값이 있습니다.

```
TRAINING_CONFIG = {
    'BOARD_SIZE': 7,

    'LEARNING_RATE': 1e-2,
    'WEIGHT_DECAY': 1e-4,

    'ROUNDS_PER_MOVE': 800,
    'PUCT': 1.25,
    'PUCT_INIT': 1.25,
    'PUCT_BASE': 19652,

    'MCTS_NOISE': True,
    'MCTS_ALPHA': 0.03,
    'MCTS_EPS': 0.25,

    'SELFPLAY_WORKERS': 12,
    'START_TRAINING': 1280,
    'EPOCH': 1,
    'BATCH_SIZE': 128,
    'CAPACITY': 10000,

    'LOAD_CHECKPOINT': 0
}
```

위 설정 중 수정해도 되는 부분을 정리하면 다음과 같습니다. (다른 부분은 수정하지 않는 걸 권합니다)

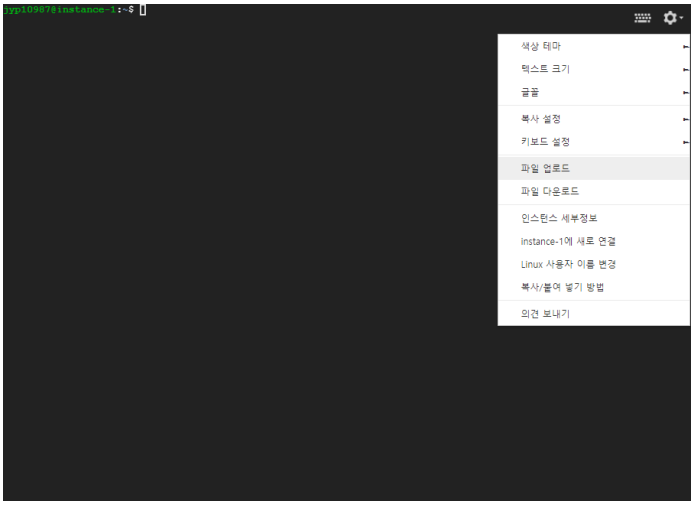
- LEARNING_RATE
  학습률로, 0 에서 1 사이의 값을 사용합니다. 이 값에 따라 학습 결과가 달라질 수 있으니 적절한 값을 사용해야 합니다.
- ROUNDS_PER_MOVE
  학습 데이터를 만들 때 한 수에 수행할 MCTS simulation 수를 정합니다. 이 값이 높을수록 좋은 데이터가 만들어지겠지만 느려집니다.
- PUCT
  이 값이 None 이면 밑의 PUCT_INIT 와 PUCT_BASE 가 쓰입니다. 그게 아니라면 상수인 PUCT 가 사용됩니다. 역시 적절한 값을 찾아야 합니다. (알파고는 5, 어떤 오픈소스 프로그램은 0.5 를 사용하기도 합니다)
- PUCT_INIT, PUCT_BASE
  $C(s_t)$에 쓰이는 상수입니다. PUCT_INIT 만 바꾸는 걸 추천합니다.

- MCTS_ALPHA
  노이즈의 퍼지는 정도를 결정합니다. 자세한 내용은 강의자료를 참고해주세요.
- SELFPLAY_WORKERS
  데이터를 만드는 작업자의 수를 결정합니다. 코어의 개수 - 1 개를 추천합니다.
- START_TRAINING
  학습을 시작할 버퍼의 크기를 결정합니다. 크기가 클수록 많은 데이터를 사용해 학습할 수 있습니다만, 학습 속도가 느려집니다.
- EPOCH
  한 번 학습할 때 몇 번 학습할지 결정합니다. 많을수록 수렴은 빨리 되는 것처럼 보이겠지만 그리 좋은 현상은 아닐 수 있습니다.
- BATCH_SIZE
  한 번에 학습할 데이터의 개수를 결정합니다. 클수록 학습 속도는 느리지만 정확도는 높아질 수 있습니다.
- CAPACITY
  값이 높으면 메모리가 아파합니다. 적절한 값을 주는 것이 좋습니다.
- LOAD_CHECKPOINT
  이전에 학습했던 것에 이어서 학습할 때 불러올 모델의 숫자를 적으면 이어서 학습합니다.

그 이외에 네트워크 구조를 바꾸는 것도 도움이 될 수 있고, buffer 를 비우는 방법(안 비움/절반 비움/전부 비움)에 따라서도 학습 결과가 달라질 수 있습니다. 또한 착수 위치를 결정하는 방법을 바꾸는 것도 도움이 될 수 있습니다.

## 6. 파일 업로드 방법

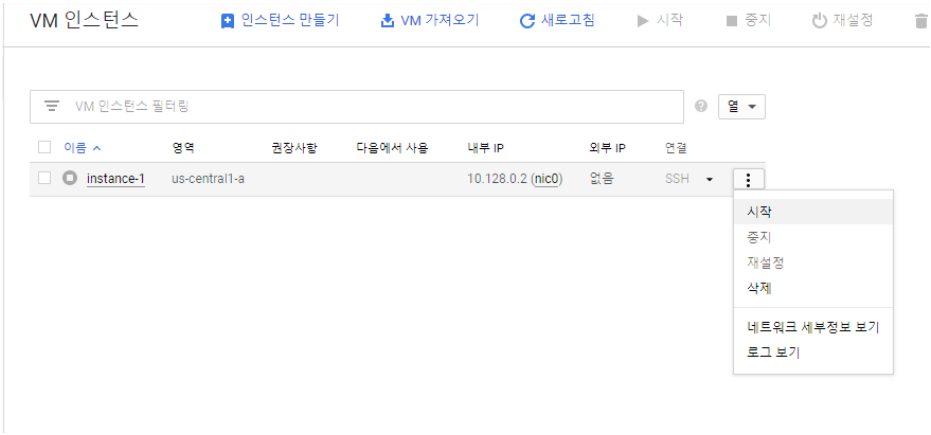
ssh 화면에서 우측 상단에 톱니바퀴 모양을 누르면 **파일 업로드**가 있습니다. 해당 기능을 이용해 작업한 소스코드를 gcp 인스턴스에 업로드해 사용하면 됩니다.




업로드 된 파일은 ~에 저장됩니다. 폴더 단위 업로드는 폴더를 압축한 뒤 unzip 명령어를 이용해 압축 해제하는 것을 추천합니다.

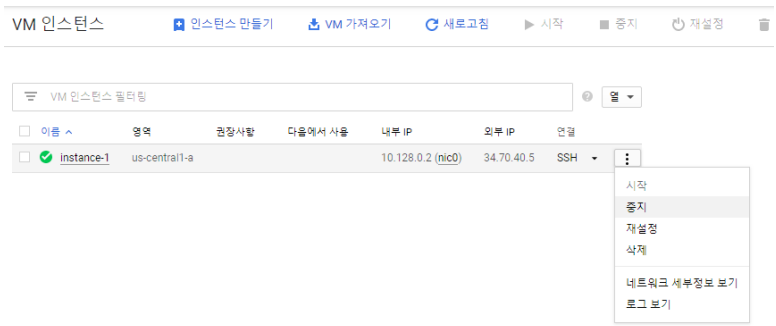
## 7. GCP 인스턴스 시작/종료 방법

인스턴스를 시작하려면 : 를 누른 뒤 **시작**을 누릅니다.




시작이 완료되면 초록색 체크 표시가 나옵니다. 그때 SSH 버튼을 눌러 접속하면 됩니다.

인스턴스를 종료하려면 이번엔 : 를 누른 뒤 **종료**를 누릅니다.




---

GCP 사용이 끝나면 반드시 인스턴스를 종료해야 합니다.

---

## 8. 대전 준비 방법

내일 진행할 대전을 위해서는 몇 가지 사전 준비가 필요합니다.

먼저 아래 명령어로 requests 를 설치합니다.  서버와의 통신을 도와줄 패키지입니다.

```
pip3 install requests
```

다음으로 connect5_connecter.py 를 AlphaZero 에 만들고, 아래 코드를 복사해 줍니다.

```
import requests
import time

class Connect5Connecter:
    def __init__(self, url: str, name: str):
        self.url = url
        self.name = name
        self.sess = requests.Session()
        self.login(self.name)

    def login(self, name: str):
        assert Connect5Connecter.is_success(self.sess.post(self.url +
'login', {'player_name': name}))

    @classmethod
    def is_success(cls, rep):
        return rep.status_code // 100 == 2

    def get_state(self, delay=1):
        while True:
            rep = self.sess.get(self.url + 'state')
            if Connect5Connecter.is_success(rep):
                state = rep.json()
                if state['opponent_action'] != None:
                    state['opponent_action']['x'] += 1
                    state['opponent_action']['y'] += 1
                return state
            time.sleep(delay)

    def send_action(self, column: int, row: int):
        assert Connect5Connecter.is_success(self.sess.post(self.url +
'action', {'x': column - 1, 'y': row - 1}))

    def notice_winner(self, winner):
     rep = self.sess.post(self.url + 'notice_winner', {'player_name': self.name, 'winner':
winner})
      print(rep.text)
      assert Connect5Connecter.is_success(rep)
```

아래는 위 코드의 간단한 설명입니다. 중요하지는 않습니다.

대전용 코드는 10. 대전 코드 부분을 참고하세요.

```
connecter = Connect5Connecter(HOST, name)
```
HOST에는 'http://18.189.17.31:80/'을 그대로 넣어주면 됩니다. 오타 나면 안됩니다.

name 에는 bot 의 이름을 넣어주면 됩니다. 다른 팀과 중복되지 않게 'c301-botname'과 같이 앞에 방 번호를 붙여주세요.

get_state(delay) → dict
delay초에 한번 서버와 통신해 현재 state를 가져옵니다. (delay 수정하지 마세요)
자기 차례에만 state를 가져올 수 있기 때문에,

```
state = connecter.get_state()
```
이렇게 쓰면 자기 차례까지 계속 대기하다가, 자기 차례일 때 state를 가져옵니다.

get_state 는 python dictionary 를 반환합니다.

state['grid'] : 빈 칸은 0, 검은 돌이 놓여있는 칸은 1, 흰 돌이 놓여있는 칸은 2 가 적힌 2 차원 배열입니다.

state['color'] : 현재 차례인 bot 이 검은색이면 'black', 흰색이면 'white'가 들어있습니다.

state['opponent_action'] : opponent가 바로 직전 차례에 보낸 action이 들어있습니다.
state['opponent_action']['x']로 col을, state['opponent_action']['y']로 row를 얻을 수 있습니다.

```
send_action(column, row)
```
(column, row) 위치에 돌을 놓도록 서버에 요청을 보냅니다.

자기 차례가 아닐 때 놓으려고 시도하거나 시간 제한을 어기거나 좌표가 유효하지 않으면 강제로 패배 처리되고, 프로그램이 종료됩니다.

```
notice_winner(winner)
```
게임이 끝났을 때 game.winner 를 인수로 넣어주며 반드시 호출해야 합니다.

## 9. 대전 규칙

각 bot 은 돌아가며 착수하고, 매 차례마다 10 초의 제한시간을 가집니다.

각 차례에 처음으로 get_state 로 state 를 받은 순간부터 타이머가 작동합니다.

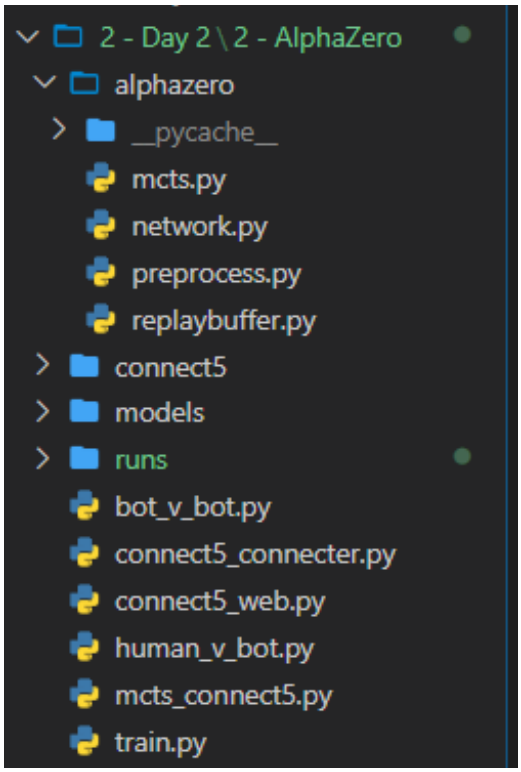
State 를 받은 이후 send_action 으로 서버에 요청을 보내기까지 10 초 이상이 경과했으면 시간 제한 초과로 인해 강제 패배 처리됩니다.

## 10. 대전 코드

아래 부분 그대로 복사하고, bot만 바꿔서 실행해도 됩니다.
connect5_web.py

```
from __future__ import print_function
from six.moves import input

from connect5 import board as connect5_board
from connect5 import types
from connect5 import agent
from connect5 import mcts
from connect5.utils import print_board, print_move, point_from_coords

import requests

from connect5_connecter import Connect5Connecter

from alphazero.mcts import AZAgent

import torch
import time
import sys

BOARD_SIZE = 7

# 서버 주소입니다.
HOST = 'http://18.189.17.31:80/'

def main():
    game = connect5_board.GameState.new_game(BOARD_SIZE)
    # 사용하고 싶은 bot을 넣어주세요.
    bot = AZAgent(BOARD_SIZE, torch.load(sys.argv[1]), rounds_per_move=400)

    # bot의 name을 정합니다. 다른 팀과 중복되면 안됩니다.
    name = 'c301-bot'

    # 서버와의 통신에 사용되는 Connect5Connecter 클래스 객체를 만듭니다.
    # connecter 관련 부분은 수정하지 말아주세요. 서버랑 통신할 때 문제가 생깁니다.
    connecter = Connect5Connecter(HOST, name)
```

```
    # 서버에서 자기 차례에 state를 가져옵니다.
    state = connecter.get_state()
    # 서버에서 정해준 bot의 color로 is_my_turn을 결정합니다.
    if state['color'] == 'black':
        print("I'm black.")
        is_my_turn = True
    elif state['color'] == 'white':
        print("I'm white.")
        is_my_turn = False
    else:
        print('invalid color')

    while not game.is_over():
        # 서버에서 자기 차례에 state를 가져옵니다.
        state = connecter.get_state()
        # bot의 차례라면 send_action으로 서버에 bot의 action을 전송합니다.
        # 아니면 서버에서 oppnent의 action을 가져와 game에 적용합니다.
        if is_my_turn:
            move = bot.select_move(game)
            game = game.apply_move(move)
            connecter.send_action(move.point.col, move.point.row)
        else:
            move = connect5_board.Move.play(types.Point(row=int(state['opponent_act
ion']['y']), col=int(state['opponent_action']['x'])))
            game = game.apply_move(move)
        print_board(game.board)
        # is_my_turn을 뒤집습니다.
        is_my_turn = not is_my_turn

    # 누가 이겼는지 서버에 알려줍니다.
    connecter.notice_winner(game.winner)

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




폴더 구조가 위 스크린샷처럼 되게 connect5_connecter.py 와 connect5_web.py 를 위치시켜주세요. 그 다음 아래처럼 학습이 끝난 connect5_web.py 를 실행시키면 됩니다

```
python3 connect5_web.py (torch.save 로 저장한 파일 이름)
```

## 11. 대전 연습

대전용 코드에 문제가 없는지 확인하고 싶거나, 다른 팀과 모의 대전을 해 보고 싶거나, bot 의 성능을 확인해 보고 싶으면, 대전 코드를 실행해 로그인한 다음 bot 의 name 을 단체 카톡방에  올려주시면 됩니다.

그러면 조교가 대전 결과를 아래처럼 스크린샷으로 찍어서 보내줄 겁니다.