DNN을 이용한 LOL
무작위 총력전 챔피언 추천

목차

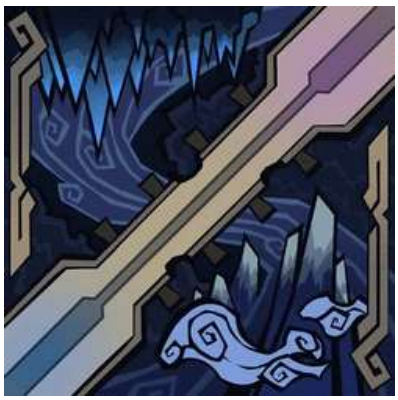
1. 개요 및 목적

1. 프로젝트 개요

현재 가장 많은 인기를 끌고 있는 게임 중 하나인 리그 오브 레전드는 5명이 한 팀을 이루어 각각 상대팀의 주요 건물을 무너뜨리는 것을 목표로 하는 전략 게임이다. 플레이어는 약 150여명이 넘는 챔피언(캐릭터)을 사용할 수 있을 뿐만 아니라 룬과 아이템, 캐릭터의 배치 등등 상황에 따라 고려할 요소가 많기 때문에, 게임을 플레이했을 때 승률을 조금이라도 높이기 위해, 통계 사이트나 논문 등에서 활발히 데이터 분석이나, 연구가 이루어지고 있다.

리그 오브 레전드의 맵은 두 가지 형태로 나뉘는데, 하나는 맵이 넓고, 다른 모드에 비해 전투가 잘 일어나지 않으며, 그로 인해 변수가 많고, 다양한 전략이 요구되는 소환사의 협곡이 있으며, 다른 하나는 무작위 총력전으로 불리는 칼바람 나락이다. 무작위 총력전은 무작위 챔피언으로 한 공격로에서 싸우는 게임 모드이다. 소환사의 협곡과는 다르게 5명으로 구성된 두 팀이 한 공격로에서 끝까지 전투를 펼치는 게임 모드이다.각 플레이어에게 무작위 챔피언이 배정되며, 빠른 속도로 끊임없이 전투가 벌어진다는 특징이 있다.

소환사의 협곡



칼바람 나락

2. 개발 동기

소환사의 협곡은 리그 오브 레전드 플레이어들이 자주 하는 맵인 만큼, kaggle 같은 데이터 분석 사이트에서 연구가 자주 이루어진다. 하지만 칼바람 나락(무작위 총력전)에 대한 연구는 잘 이루어지지 않는다. 칼바람 나락도 소환사의 협곡만큼이나 자주 플레이 하는 맵이기 때문에 마찬가지로 다양한 연구나분석이 필수적이다. 또한 칼바람 나락은 소환사의 협곡이나 달리 챔피언들이 한정된 풀 안에서 랜덤으로 선택된다. 안 그래도 변수가 많은 리그 오브 레전드 게임에서, 무작위성이 부여되다 보니, 챔피언에 대한 숙련도가 부족하거나, 무작위 총력전에 익숙하지 않은 사람들은 무슨 챔피언을 선택해야 할지 혼란이 온다. 따라서 선택할 수 있는 챔피언들 중, 각 챔피언들의 데이터, 특성, 통계 사이트의 승률, 조합 시너지등의 자료를 통해 인공 신경망을 통해 학습, 최종적으로 예상 승률이 가장 높게 나오는 챔피언을 선택하는 것이 이번 프로젝트의 목표이다.

2. 배경

1. 관련 기존 연구 조사 및 분석

1-1. RNN을 사용한 LOL 30분 게임 승률 예측

칼바람이 아닌 소환사의 협곡에서의 대회 30분 길이의 시리얼 데이터를 RNN으로 학습 시켜서 새로운 경기의 예상 승률을 퍼센트 형태로 제시한다.

1. 롤 30분 경기의 데이터셋을 불러와 데이터그램 형태로 만든다.

Dataset

```
df = pd.read_csv('../input/leagueoflegends/LeagueofLegends.csv', sep=',')
df = df[df['gamelength'] >= MAX_TIME_STEP]
df.reset_index(drop = True, inplace = True)
matches = len(df)
print(f'# of matches: {matches}')

from ast import literal_eval
df['golddiff'] = df['golddiff'].apply(literal_eval)
df[['golddiff']].head()
```

League	Year	Season	Type	blueTeam	bResult	rResult	redTeamT	gameleng	golddiff	goldblue
NALCS	2015	Spring	Season	TSM	1	0	C9	40	[0, 0, -14,	[2415, 241
NALCS	2015	Spring	Season	CST	0	1	DIG	38	[0, 0, -26,	[2415, 241
NALCS	2015	Spring	Season	WFX	1	0	GV	40	[0, 0, 10, -	[2415, 241
NALCS	2015	Spring	Season	TIP	0	1	TL	41	[0, 0, -15,	[2415, 241
NALCS	2015	Spring	Season	CLG	1	0	T8	35	[40, 40, 44	[2415, 241
NALCS	2015	Spring	Season	DIG	0	1	TIP	24	[0, 0, 20, -	[2415, 241
NALCS	2015	Spring	Season	CST	1	0	WFX	39	[0, 13, -7,	[2415, 243
NALCS	2015	Spring	Season	TL	1	0	CLG	43	[0, 0, 26, 5	[2415, 241
NALCS	2015	Spring	Season	C9	0	1	GV	41	[0, -10, 0,	[2415, 241
NALCS	2015	Spring	Season	T8	1	0	TSM	32	[0, 0, 0, 68	[2415, 241
NALCS	2015	Spring	Season	GV	1	0	DIG	52	[0, -10, -1	[2415, 241
NALCS	2015	Spring	Season	T8	1	0	CST	46	[0, 0, 15, -	[2415, 241
NALCS	2015	Spring	Season	C9	1	0	TL	38	[40, 40, 56	[2415, 241
NALCS	2015	Spring	Season	TIP	0	1	CLG	31	[0, 0, 6, -5	[2415, 241

2. 데이터그램을 학습하기 용이하도록 불필요한 데이터는 삭제하고 feature를 전처리한다

```
stats = ['golddiff','dragondiff', 'barondiff', 'heralddiff', 'towerdiff', 'inhibitordiff',
'killdiff']
x = df[stats]
y = df['bResult']

x.tail()
```

	golddiff	dragondiff	barondiff	heralddiff	towerdiff	inhibitordiff	killdiff
6379	[0, -8, -187, -37, -92, -164, -229, -424, -256...	[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, 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, -1, -1...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, -1, -1, -1, -3, -3, -...
6380	[0, 0, -18, -95, 45, -87, -117, 199, 126, 92, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -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, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 2, 2, 2, 0, 0, ...
6381	[0, 0, -86, -39, -207, -349, -60, -140, 187, -...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -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, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, -2,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, -1, -1, -1, -1, -1, -2, -2, -2, -4, ...
6382	[0, 0, -97, 33, 351, 284, 299, 263, 403, 623, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 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, 0, 0, 0, 0, ...	[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
6383	[0, 0, -8, -225, -36, 73, 464, 184, 1171, 1409...	[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, 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, 0, 0, 0, 1, 1, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, -1, -1, 0...

3. StandardScaler를 통해 실제로 데이터를 전처리하고 RNN모델을 만든다.

```
from sklearn.preprocessing import StandardScaler

data = {}
scalers = {}
for stat in stats:
    scalers[stat] = StandardScaler()
    for row in df[stat]:
        scalers[stat].partial_fit(np.asanyarray(row).reshape(-1, 1))
    data[stat] = [scalers[stat].transform(np.asanyarray(row).reshape(-1, 1)).reshape(-1) f
or row in df[stat]]

num_features = len(data)
print(f'# of features per timestep: {num_features}')
```

```
class RNN(nn.Module):
    def __init__(self):
        super(RNN,self).__init__()
        self.hidden_size = 256

        self.rnn= nn.RNN(
            nonlinearity = 'relu',
            input_size = num_features,
            hidden_size = self.hidden_size,
            num_layers = 1,
            batch_first = True
        )

        self.out = nn.Linear(self.hidden_size, 2)

    def forward(self,x):
        r_out, hn = self.rnn(x, torch.zeros(1, len(x), self.hidden_size))
        out = self.out(r_out[:, -1, :])
        return out
```

4. 만든 RNN 모델을 통해 실제로 데이터를 학습시킨다

Training RNN

```
MUTE = False
EPOCH = 100
LR = 0.0001

torch.manual_seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)

model = RNN()
print(model)

optimizer = torch.optim.Adam(model.parameters(), lr = LR)
loss_func = nn.CrossEntropyLoss()

best_acc = 0
early_stopping = 0
early_stopping_threshold = 5

for epoch in range(1, EPOCH + 1):
    print(f"--------- Epoch #{epoch} ---------")
    train(trainLoader, model, loss_func, optimizer, mute = MUTE)
    valid_acc = test(validLoader, model, loss_func, validation = True)
    if valid_acc > best_acc :
        early_stopping = 0
        best_acc = valid_acc
        torch.save(model.state_dict(), f"./{MAX_TIME_STEP}.pt")
    else :
        early_stopping += 1
        if early_stopping == early_stopping_threshold :
            print(f"Early stopped at epoch #{epoch} with best validation accuracy {best_acc*100:.2
f}%.")
            break
```

```
RNN(
  (rnn): RNN(7, 256, batch_first=True)
  (out): Linear(in_features=256, out_features=2, bias=True)
)
--------- Epoch #1 ---------
loss: 0.662169  [    0/ 3832]
loss: 0.442477  [  960/ 3832]
loss: 0.354426  [ 1920/ 3832]
loss: 0.329362  [ 2880/ 3832]
Valid Acc:0.829154, Avg Loss: 0.011723
--------- Epoch #2 ---------
```

Table of

Datase

Neural

Trainin

Test

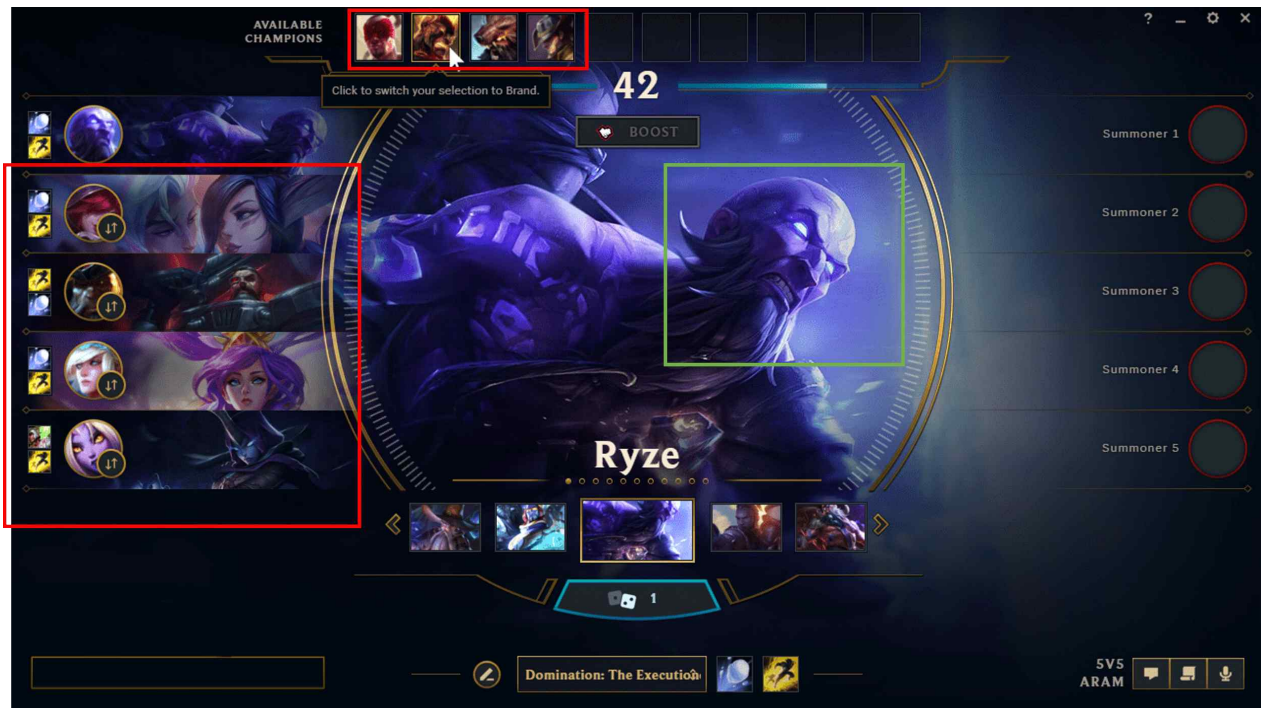
5. 학습 완료된 모델을 이용하여 새로운 경기 데이터에 대해서 예상 승률을 prediction 한다.

```
model.eval()
with torch.no_grad():
    x = torch.from_numpy(x)
    predict = model(x)
    winner = ['red', 'blue'][predict.argmax(1)]
    prob_red = math.exp(predict[0][0].item()) / (math.exp(predict[0][0].item()) + math.exp(predict
[0][1].item()))
    prob_blue = math.exp(predict[0][1].item()) / (math.exp(predict[0][0].item()) + math.exp(predict
[0][1].item()))
    print(f"model predicted winner: { winner }")
    print(f"red wins: {prob_red * 100 :.1f}% | blue wins: {prob_blue * 100:.1f}%")
```

```
model predicted winner: red
red wins: 68.9% | blue wins: 31.1%
```

3. 요구, 명세

1. 기능적 요구사항



1. 자신을 제외한 팀원들의 챔피언을 입력

- 주어진 환경에 (python, 과 데이터 학습 모델 이용) 자신이 선택한 챔피언 (1명)과 다른 플레이어 4명이 픽한 챔피언을 입력한다. (좌측 빨간 상자)

2. 자신이 현재 선택한 챔피언을 입력

- 중앙에 초록색 상자로 표시된 자신이 선택된 챔피언을 입력한다.

3. 현재 선택되지 않은 후보군 챔피언을 입력

- 위 쪽 빨간 사작형으로 둘러 싸인 4명의 챔피언을 차례대로 입력한다.

4. 선택된 챔피언과 후보군 챔피언의 승률을 각각 출력

- 중앙에 초록색 상자로 표시된 자신이 선택된 챔피언과 위 쪽 빨간 사작형으로 둘러 싸인 4명의 챔피언을 선택했을 경우 예상 승률을 각각 퍼센트 형태로 출력한다.
- ex) 라이즈 : 50.1% 리 신 : 49.9% 브랜드 : 52.3% 우디르 : 47.7%
  트페 : 49.2%

2. 비기능적 요구사항

- 데이터를 학습하기 위한 Python 환경과 , IDE, 그들을 학습시킬수 있는 라이브러리가 필요하다

- 다량의 데이터를 병렬 처리 할 수 있는 일정 성능 이상의 GPU, 또는 이에 준하는 가상 환경 또는 Docker가 필요하다.

- 실제 게임 환경에 적용할수 있으므로 게임 리그 오브 레전드를 실행 시킬 수 있는 최소한의 PC 시스템 사양을 갖추어야 한다.

시스템 권장 사양과 최소 사양

	최소 사양	권장 사양
CPU	Intel: Core i3-530	Intel: Core i5-3300
	AMD: A6-3650	AMD: Ryzen 3 1200
	ARM: 미지원	ARM: 미지원
CPU 기능	SSE3	SSE4
GPU	NVidia: GeForce 9600GT	NVidia: GeForce 560
	AMD: HD 6570	AMD: Radeon HD 6950
	Intel: Intel HD 4600 내장 그래픽	Intel: Intel UHD 630 내장 그래픽
GPU 기능	DX10 급 하드웨어	DX11 급 하드웨어

- 데이터 : 학습에 사용되는 데이터들은 최소 일정 수준 이상의 표본이 있어야 하며, 학습에 이용될 취지에 맞는 feature들을 보유해야 한다.
- 게임 리그 오브 레전드는 2주 간격의 패치에 따라 게임의 흐름과 양상이 다소 바뀌므로 학습에 사용될 데이터는 현 패치 버전과 맞아야 한다.

ex) 패치 버전에 따른 챔피언의 승률

4. 설계 , 구현

1. 프로젝트 일정

일정	계획
3월	롤 api 이용방법 학습, json을 통한 데이터 추출 방법 학습 후 데이터 추출
4월	추출한 데이터를 학습에 맞게 전처리
5월	기계학습, 및 인공신경망, DNN 개념 학습, Keras 및 Pytorch 사용 방법 익힌 후 더 적합한 라이브러리 선택
6월	추가 학습 및 코드 분석, 코드 구현 및 실행, 코드 개선

2. 개발 환경

- Python + Google colab

- 라이브러리 : Pytorch

- Riot Api : MATCH - V5

3. 설계

1. 학습에 사용될 데이터셋 생성

- Riot api 홈페이지 접속, 후 롤 api를 이용, 60000개의 칼바람 매칭 데이터와 승리 유무를 뽑아낸다.

2. 롤 조합 승률에 영향을 미치는 지표 조사

- 리그 오브 레전드 공식 홈페이지와 , 여러 게임 공략 사이트나 유튜브 등을 통해 승률에 영향을 미치는 지표를 조사하여 feature를 뽑아낼 준비를 한다.

3. 조사한 feature에 맞게 데이터셋 수정

- 조사한 챔피언 데이터로부터 10개의 feature를 추출한다. 이후 뽑아낸 매칭 데이터를 학습시킬 feature애 맞게 변환한다.

4. 인공 신경망 모델 구현 및 학습

- 학습에 사용될 인공 신경망 모델을 정한 후 데이터를 모델 안에 넣어 학습시킨다.

5. 실제 환경에서의 적용, 모델 개선

- 이후 실제 게임에서 사용할 데이터를 학습된 모델에 넣어 봐서 성능을 측정한다. 얼마나 잘 동작하는 지 보고, 모델을 수정, accuracy가 높은 모델을 선정한다.

4. 구현

1. 롤 api 이용

- 라이엇 개발자 페이지에 접속하여, API key를 발급, 이후 puuid를 추출한다. 추출한 puuid를 통해, MatchID를 발급할 수 있고, MatchID를 통해 인게임 데이터를 추출 할 수 있다.

2. 데이터셋 생성

- 앞에 설명한 일련의 과정을 Python과 api 키를 이용해 자동화 할 수 있다. Colab 환경에서의 코드를 이용해 매치 데이터를 추출한다.

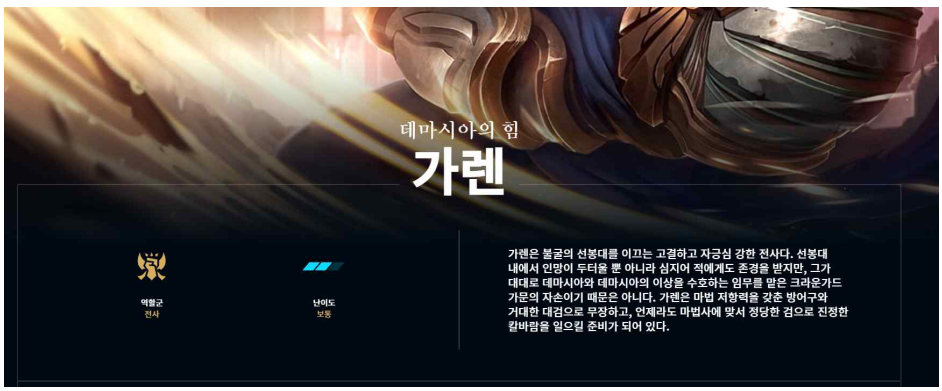
```
api_key = "----------------------"   #발급 받은 api 값을 넣어 줍니다.

csv = pd.read_csv('Game_id.csv', encoding="UTF-8")   #Game_id.csv 파일을 pandas를 이용하여 읽어옵니다.
temp = csv['Game_id']   #Game_id.csv 파일에서 'Game_id'column을 'temp'이라는 list에 넣어줍니다
```

```
match = []      #데이터를 담을 빈 list 'match'를 만들어 줍니다.


for i in range(len(temp)):      #for문을 통해 데이터들을 'match'안에 넣어줍니다
    try:
        URL = "https://asia.api.riotgames.com/lol/match/v5/matches/" + game_id_final[i]
        res = requests.get(URL, headers={"X-Riot-Token": api_key})
        resobj = json.loads(res.text)
        win_experience = 0
        lose_experience = 0
        win_goldspent = 0
        lose_goldspent = 0

        for j in range(10):
            if resobj["info"]["participants"][j]["win"] == False:
                if resobj["info"]["participants"][j]["teamPosition"] == "TOP" :
                    top_l = resobj["info"]["participants"][j]["championName"]
                elif resobj["info"]["participants"][j]["teamPosition"] == "MIDDLE" :
                    middle_l = resobj["info"]["participants"][j]
```

```
df = pd.DataFrame(match, columns=['TOP','MIDDLE','JUNGLE','BOTTOM','UTILITY','OUTCOME','experience','goldspent','baron','champion_kills','dragon','inhibitor','riftHerald','tower','teamId'])
df.to_csv('match.csv', index=False, encoding='utf-8')
```

3. 이용할 feature 조사

- 라이엇 공식 홈페이지나 여러 사이트들을 이용한다.



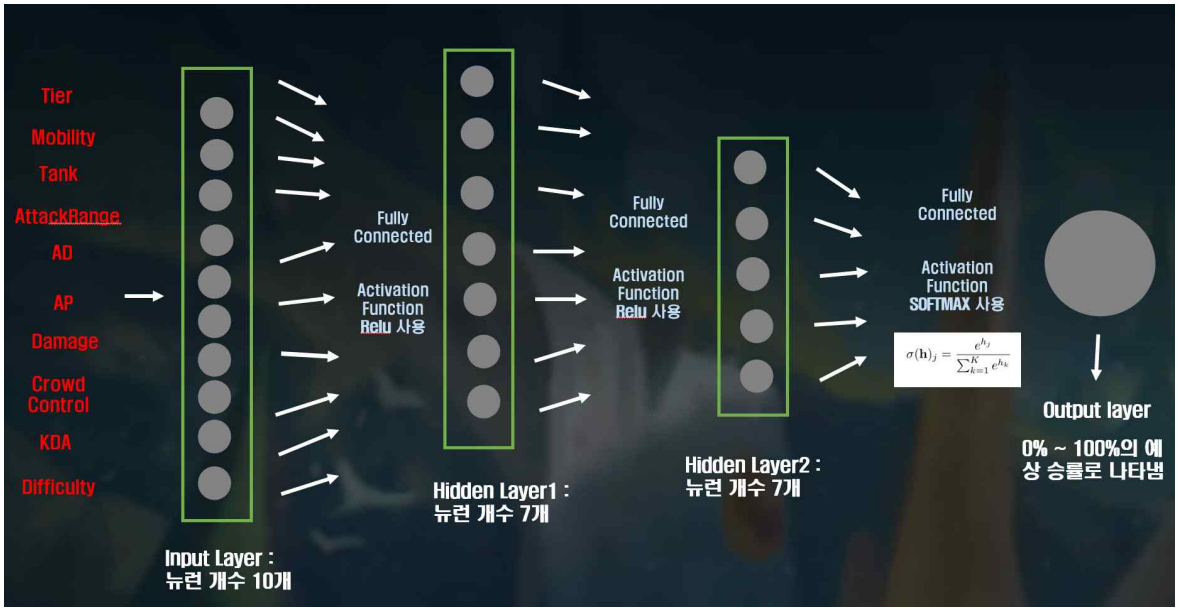
- 이후 이를 이용하여 10개의 feature를 추출하였다.
- Name, Tier, Mobility, Attack range, AD, AP, Tank, Damage /100 , Crowd control, KDA

4. 데이터셋 변환

- Python의 Pandas 라이브러리를 이용하여, 데이터그램으로 변환시키고, feature에 맞게 수정한다.

5. 모델 학습

- Pytorch 라이브러리를 이용해 모델을 구현한다. 구조는 다음과 같다.



- Layer가 10 - 7 - 5 = 1(softmax) 로 되어있고 Output Layer에 100을 곱해 예상 승률을 퍼센트로 나타낸다.

6. Prediction

- 학습한 환경에 이어 붙여 새 데이터에 대해서 prediction하는 코드를 구현한다.

ex) Console

NoN player : a, b, c, d (입력)
Player : E (입력)
Win rate Expectiation -> a : 50% b : 75% ... (출력)

5. 실행 결과

1. 실행 결과

- 매치 데이터 추출

	id	Champ1	Champ2	Champ3	Champ4	Champ5	Result
1	4.52E+09	Kayle	Zeri	Sett	Varus	Ezreal	1
2	4.52E+09	Katarina	Ashe	Janna	Twitch	Leblanc	0
3	4.52E+09	Gnar	Sivir	Brand	Fizz	Yuumi	1
4	4.52E+09	Amumu	Orianna	Veigar	Miss Fortu	Varus	0
5	4.44E+09	K Sante	Master Yi	Ahri	Zyra	Miss Fortu	1
6	4.48E+09	Milio	Vladimir	Graves	Talon	Lillia	0
7	4.49E+09	Katarina	Kalista	Ahri	Fiddlestick	Udyr	1
8	4.5E+09	Zilean	Neeko	Karma	Ashe	Pyke	0
9	4.44E+09	Kalista	Nilah	Sejuani	Braum	Lissandra	1
10	4.51E+09	Ashe	Elise	Camille	Braum	Irelia	0
11	4.45E+09	Ashe	Yorick	Varus	Sejuani	Vex	1
12	4.45E+09	Amumu	Nocturne	Warwick	Gwen	Trundle	0
13	4.52E+09	Syndra	Lux	Kha Zix	Yone	Sivir	1
14	4.52E+09	Ashe	Garen	Yasuo	Thresh	Morgana	0
15	4.52E+09	Riven	Malzahar	Pyke	Akshan	Shen	1
16	4.48E+09	Master Yi	Nidalee	Volibear	Ivern	Akshan	0
17	4.52E+09	Nasus	Ryze	Akali	Nilah	Viego	1
18	4.5E+09	Kassadin	Yasuo	Nasus	Caitlyn	Taliyah	0
19	4.49E+09	Vayne	Shen	Vladimir	Maokai	Swain	1
20	4.46E+09	Ornn	Braum	Shaco	Lucian	Qiyana	0
21	4.47E+09	Kennen	Bard	Pantheon	Fiora	Sona	1
22	4.45E+09	Taliyah	Nami	Rell	Tahm Ken	Taliyah	0

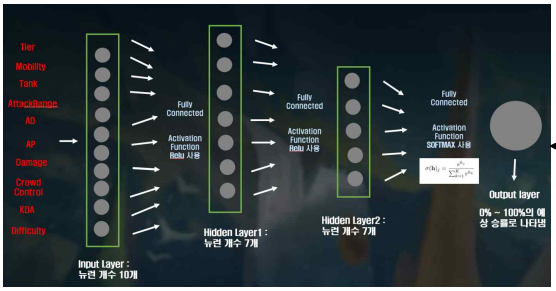
- 데이터셋 변환

Id	Name	Tier	Mobility	Attack ran	AD	AP	Tank	Damage /	Crowd cor	KDA * 100	Difficulty
1	Aatrox	3	2	175	1	0	4924	264	2	281	2
2	Ahri	2	3	550	0	1	3922	276	2	340	2
3	Akali	3	3	125	0	1	4859	298	1	285	2
4	Akshan	4	3	500	1	0	4191	270	1	284	3
5	Alistar	3	1	125	0	1	5365	147	3	355	1
6	Amumu	3	1	125	0	1	4165	197	3	270	1
7	Anivia	5	1	600	0	1	3767	236	3	363	3
8	Annie	2	1	625	0	1	4026	273	3	331	1
9	Aphelios	2	1	550	1	0	4044	295	2	312	3
10	Ashe	3	1	600	1	0	4199	220	3	343	1
11	Aurelion S	3	2	550	0	1	3678	313	2	353	3
12	Azir	2	2	525	0	1	4620	315	2	298	3
13	Bard	2	2	500	0	1	4458	254	3	433	3
14	Bel Veth	4	3	175	1	0	4342	227	2	247	2
15	Blitzcrank	5	1	125	0	1	4758	185	3	315	1
16	Brand	1	1	550	0	1	4113	335	2	323	2
17	Braum	1	1	125	0	1	5117	131	3	414	2
18	Caitlyn	3	2	650	1	0	4306	281	2	335	1
19	Camille	2	3	125	1	0	4505	260	2	256	3
20	Cassiopeia	3	1	550	0	1	4220	271	3	336	3
21	Cho Gath	4	1	125	0	1	4370	255	2	334	1
22	Corki	2	2	550	0	1	4271	315	1	335	2
23	Darius	4	1	175	1	0	5106	225	2	254	2
24	Diana	4	2	150	0	1	4624	293	2	282	1
25	Dr.Mundo	3	1	125	1	0	4291	245	1	285	1
26	Draven	4	2	550	1	0	4368	310	1	307	3
27	Ekko	3	3	125	0	1	4328	286	2	285	3

2. 예상 실행 결과

- 모델 구현

```
def __init__(self, channels, action_size, seed=42):
    """Initialize parameters and build model.
    Params
    ======
        state_size (int): Dimension of each state
        action_size (int): Dimension of each action
    """
    super(QNetworkDuellingCNN, self).__init__()
    self.fc1 = nn.Linear(10, 7)
    self.fc2 = nn.Linear(7, 5)
    self.fc3 = nn.Linear(5, 1)

def forward(self, x):
    """Build a network that maps state -> action values."""
    h1 = F.relu(self.fc1(x.view(-1, in_feature)))
    h2 = F.relu(self.fc2(h1))
    h3 = F.relu(self.fc3(h2))
    return F.log_softmax(h3, dim=1)
```

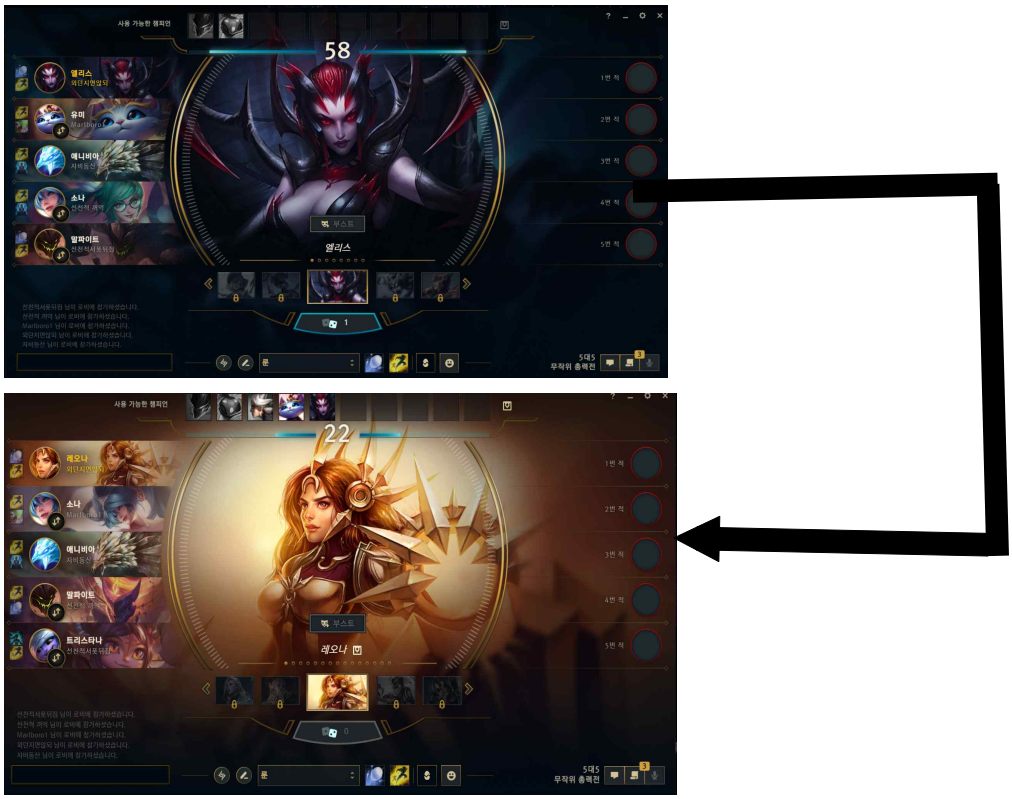


- Prediction

1. Console

NoN player : a, b, c, d (입력)
Player : E (입력)
Win rate Expectiation -> a : 50% b : 75% ... (출력)

2. 롤 클라이언트 내에서 챔프 변경(가장 예상 승률이 높은 챔피언)



6. 개선 사항 및 향후 계획

1. 개선 사항

아직 개발이 미진행 된 부분이 많고 위 본몬처럼 데이터셋 생성 부분과 데이터셋 변환 부분이 파이썬에 대한 지식 부분으로 자동화하여 하는 부분에서 계속 실패해, 현 시점에는 수동으로 데이터 편집기를 사용하여 직접 타이핑하는 방식으로 구현하였다. 그로 인해 학습에 대한 데이터의 양과 품질이 계획 한 것에 비해 낮다. 따라서 파이썬과 이와 관련된 라이브러리에 대한 충분한 학습이 요구되며, 그 이후 자동화한 코드를 통해 계획한 대로 데이터 수집과 가공을 온전히 구현해야 한다. 또한 기존 연구 분석한 것과 같이 데이터를 잘 학습시키면 Scaler같은 데이터 전처리 과정이 필요하나 현 시점에는 구현되어 있지 않다. 따라서 Scaling 기법을 포함한 다양한 데이터 전처리 기법을 학습 후 적용해야 한다.

2. 향후 계획

1. 위 과정 중 미구현한 내용 구현

2. 구현 예정인 DNN 모델 제외한 여러 가지 모델을 사용 및 분석, 적용 후 개선점 찾기

3. 문제점이나 결함 있는 부분을 개선하고 발전 시킬 부분이 있으면 update하기
   (ex classification을 이용히여챔피언 이미지를 학습 후 게임 화면 내애서 챔피언 추천하기)

7. 참고 문헌

1. https://www.leagueoflegends.com/ko-kr/
2. https://developer.riotgames.com/
3. https://tutorials.pytorch.kr/beginner/basics/buildmodel_tutorial.html.
4 .
https://www.kaggle.com/code/nailian23/lol-winner-prediction-from-30min-84-acc-rnn