Assignment 3 A.I final report

컴퓨터과학과
2015147533 유현석

1. Introduction

이번 과제를 진행하기 위해서 저는 Colab을 사용하였습니다. 먼저 코드를 시행시키기 위해서 구글 드라이브와 연동 후 드라이브에 해당 data를 이동시켜주었습니다.

<드라이브 연동>

```
from  google.colab import  drive
drive.mount ('/content/gdrive')
```

또한, 아래와 같이 결과 파일이 저장될 위치와 사용될 파일들의 위치 역시 같이 지정해주었습니다.

<결과 파일 저장 위치>

```
cd /content/gdrive/My Drive/Colab Notebooks/data
```

<사용할 파일 위치 지정>

```
train = pd.read_csv ('/content/gdrive/My Drive/Colab Notebooks/data/train_covid.csv')
test = pd.read_csv ('/content/gdrive/My Drive/Colab Notebooks/data/test_covid.csv')
```

2. How did you preprocess the data for given task?

```
# Fill missing value by average
train["new_cases"]=train["new_cases"].fillna(train["new_cases"].mean())
train["new_deaths"]=train["new_deaths"].fillna(train["new_deaths"].mean())
train["stringency_index"]=train["stringency_index"].fillna(train["stringency_index"].mean())
train["population"]=train["population"].fillna(train["population"].mean())
train["population_density"]=train["population_density"].fillna(train["population_density"].mean())
train["median_age"]=train["median_age"].fillna(train["median_age"].mean())
train["aged_65_older"]=train["aged_65_older"].fillna(train["aged_65_older"].mean())
train["aged_70_older"]=train["aged_70_older"].fillna(train["aged_70_older"].mean())
train["female_smokers"]=train["female_smokers"].fillna(train["female_smokers"].mean())
train["male_smokers"]=train["male_smokers"].fillna(train["male_smokers"].mean())
train["gdp_per_capita"]=train["gdp_per_capita"].fillna(train["gdp_per_capita"].mean())
train["handwashing_facilities"]=train["handwashing_facilities"].fillna(train["handwashing_facilities"].mean())
train["hospital_beds_per_thousand"]=train["hospital_beds_per_thousand"].fillna(train["hospital_beds_per_thousand"].mean())
train = train.fillna(train.mean())
test = test.fillna(train.mean())
```

-> 주어진 파일에는 데이터가 없는 자료가 존재할 것입니다. 그러한 데이터에 대해서는 같은 행의 데이터의 평균 값으로 넣어줌으로써 preprocess를 하였습니다. 국가를 제외한 총 13개의 data에 대해서만 preprocess 해주었습니다. 그 후 마지막으로 전체에 대해서 데이터가 없는 것이 있는지 다시 한번 확인 후, 만약 존재하면 평균값으로 다시 한번 넣어주었습니다.

3. Which columns are used for input/output?

최종적으로 사용된 column은 'new_cases', 'new_deaths', 'median_age' 이렇게 3가지 data를 사용하였습니다. 여러 가지 경우를 시도하기 위해 위에 3가지 정보 외에 'stringency_index' , 'population_density'를 사용하였지만, 결과가 좋지 못하여 최종적으로 'median_age'만을 사용하였습니다. 또한, 그 외에 data에 대해서는 확실한 연관성을 확신할 수 없었고 fc 하는 과정에서 정확도가 떨어질 가능성이 발생하여 최종적으론 3가지 column만을 input으로 사용하였습니다. output으로 기존과 동일하게 'ID', 'new_cases', 'new_deaths'만을 사용하였습니다.

<median_age가 들어갈 공간을 만드는 과정>

```
# initialize the dataset
train_case = [0.] * 10
train_death = [0.] * 10
train_age = [0.] * 10
train_input = train_case + train_death + train_age
train_inputs = []
train_outputs = []
locations = []


test_case = [0.] * 10
test_death = [0.] * 10
test_age = [0.] * 10
test_input = test_case + test_death + test_age
test_inputs = []
```

<median_age의 데이터를 추가하는 과정>

```
train_case = train_case[1:] + [float(row["new_cases"])]
train_death = train_death[1:] + [float(row["new_deaths"])]
train_age = train_age[1:] + [float(row["median_age"])]
train_input = train_case + train_death + train_age
#train_inputs.append(train_input)
```

4. How did you design your base deep learning model?

제가 사용한 deep learning model은 2가지 fc layer을 사용하는 fc model입니다. 뒤에서도 설명을 할 예정이지만 여러 가지 fc layer을 추가하고 hidden layer을 추가하고 dropout 역시 해보았지만, 결과가 좋지 못하여 2개의 layer만을 사용하였습니다.

```
class  myModel (torch .nn .Module ):
    def  __init__(self ):
        super (myModel , self ).__init__()
        self .model = torch.nn.Sequential (
                        torch.nn.Linear (30 , 10 ),
                        torch.nn.GELU (),
                        torch.nn.Linear (10 , 2 ),
                        torch.nn.ReLU ()
        )
```

5. How did you improve your base deep learning model?

5-1 Batch_size 증가

“가장 극적인 변화를 보여준 변수가 아닐까”라고 생각합니다. Batch_size에서 좋은 성능을 이끌어 내기 위해서는 처리 가능한 사이즈 내에서 크면서 또 작지도 않아야 하는 조건이 있습니다. 이를 위해서 다양한 Batch_size를 넣어주면서 test를 해보았습니다.

```
batch_size = 100
## Dataset Load
```

5-2 예측에 사용하는 data 숫자 변경

기존의 7일 전의 데이터를 이용하여 결과를 예측했던 것을 7일에서 -> 10일로 변경하였습니다. 이렇게 되면 모든 data 값의 크기가 달라지기 때문에 fc 값과 해당 숫자들 모두 변경해주었습니다. 10일로 한 이유로는 총 3가지 경우인 7일, 10일, 30일을 시도했을 때 가장 결과가 좋았던 것이 10일이기 때문입니다.

<test_data에서 사용될 data를 10일로 변경, train_data에 10일치 정보 저장>

```
# To predict 5/1
temp_df = train[train["location"] == key]
temp_df = temp_df[(temp_df['date'] >= '2020-04-21') & (temp_df['date'] <= '2020-04-30')]
```

```
# Make dataset for training
# [c*7, d*7] --> [c, d]
# (ex: given: 4/1~4/7 case & death, predict: 4/8 case & death)
for key in tqdm(sorted(set(train["location"]))):
    locations.append(key)
    temp_df = train[train["location"] == key]
    train_case = [0.] * 10
    train_death = [0.] * 10
    train_age = [0.] * 10
```

5-3 예측에 사용되는 column 추가

위에서 설명했듯이 'median_age'의 column을 추가해주었습니다. 이 외에도 다양한 column을 사용하였지만, 결과가 좋지 않았던 점을 생각하면 정확한 연관성이 있는 column data만을 사용하는 것이 올바른 예측이 도움이 되는거 같습니다.

5-4 Preprocess 하기

이 역시, 제일 처음 설명했다시피 각 행에 없는 data 값에 전체 평균을 넣어주었습니다. 이를 개선할 수 있는 방법으론 해당 국가별 평균값을 넣는 방법이 존재하는거 같습니다.

5-5 total loss 식 변경

기존의 total loss를 구하는 식을 변경하여 loss_c 즉 new_cases에 더 큰 가중치를 주었습니다. total loss를 변경하면 예측되는 값이 실제와 얼마나 차이가 있는 알게 되는 값이 달라져 예측되는 방향이 달라질 수 있습니다.

<변경 전>

```
total_loss = (loss_c + loss_d ) / 2
```

<변경 후>

```
total_loss = (loss_c )*0.9   + (loss_d ) * 0.1
```

5-6 Epoch 사이즈 변경

Epoch 사이즈는 충분히 학습될 수 있을 정도의 수치만을 넣어주었습니다. 이 이유로는 만약 너무 큰 epoch 사이즈를 넣어주면 train_data에 대해서는 높은 학습률을 보여줄 수 있지만, 그 외의 data들에 대해서는 학습된 것과의 차이점에 대해 대처를 하지 못하게 될 수 있습니다.

5-6 FC layer 변경

FC layer는 결국 제가 사용한 모델의 시작과 끝이라고도 볼 수 있습니다. 이에 따라 FC layer을 얼마나 잘 만들었는지가 얼마나 잘 예측할 수 있는지 여부를 알려줄 수 있을 것입니다. 아래의 그림들은 제가 시도한 다양한 fc layer들과 그 layer들에 대한 결과값 입니다.

<시도한 다양한 FC layer들과 결과>

```
torch.nn.Linear (60 , 120 ),
torch.nn.GELU (),
torch.nn.Linear (120 , 300 ),
torch.nn.ReLU (),
torch.nn.Linear (300 , 180 ),
torch.nn.GELU (),
torch.nn.Linear (180 , 90 ),
torch.nn.ReLU (),
torch.nn.Linear (90 , 30 ),
torch.nn.GELU (),
torch.nn.Linear (30 , 2 ),
torch.nn.ReLU ()
```

1.69934

```
torch.nn.Linear (60 , 120 ),
torch.nn.GELU (),
torch.nn.Dropout (0.2 ),
torch.nn.Linear (120 , 300 ),
torch.nn.ReLU (),
torch.nn.Dropout (0.2 ),
torch.nn.Linear (300 , 180 ),
torch.nn.GELU (),
torch.nn.Dropout (0.2 ),
torch.nn.Linear (180 , 90 ),
torch.nn.ReLU (),
torch.nn.Dropout (0.2 ),
torch.nn.Linear (90 , 30 ),
torch.nn.GELU (),
torch.nn.Dropout (0.2 ),
torch.nn.Linear (30 , 2 ),
torch.nn.ReLU ()
```

3.56395

```
torch.nn.Linear (60 , 120 ),
torch.nn.GELU (),
torch.nn.Linear (120 , 90 ),
torch.nn.ReLU (),
torch.nn.Dropout (0.2 ),
torch.nn.Linear (90 , 30 ),
torch.nn.GELU (),
torch.nn.Dropout (0.2 ),
torch.nn.Linear (30 , 2 ),
torch.nn.ReLU ()
```

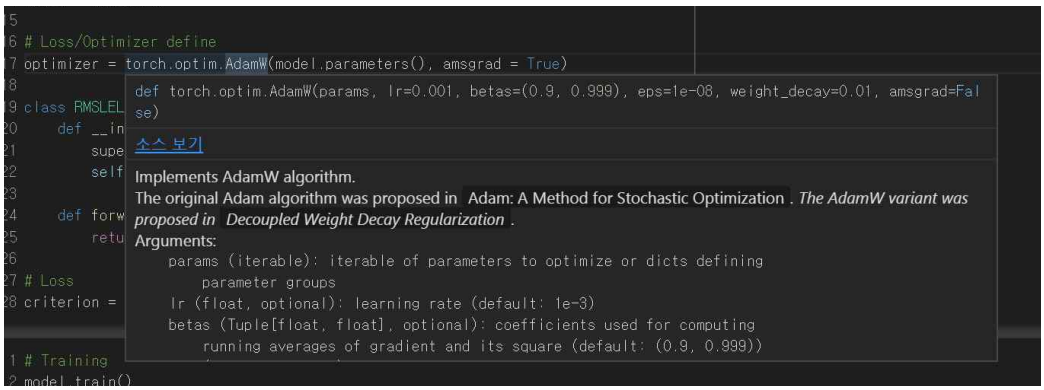
1.75361

6. Learning parameters (learning rate, epslion, etc)

Learning parameters는 아래와 같이 AdamW에서 사용된 기본 값과 똑같이 설정해 주었습니다.

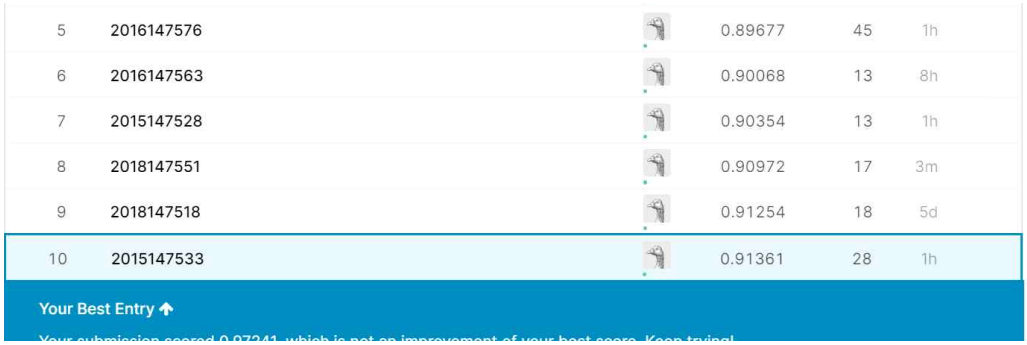
```
optimizer = torch.optim.AdamW (model.parameters (), amsgrad = True )
```

<learning rate : 0.001 , betas =(0.9, 0.999), epslion = (1 e-08)>



7. Result and analysis for your agent

7-1 <Best Result>



7-2 Analysis for my agent

앞에서 언급했던 성능을 높이는 방법인 batch_size 변경, 사용되는 데이터의 날짜 변경, 사용되는 column 변화, Preprocessing, Total loss식 변경, Epoch 사이즈 변화, FC layer 변경에 대하여 시도했던 것 중 가장 잘 예측이 될 가능성이 높은 경우들을 모아 만든 것이 제 agent입니다.

시도한 batch_size들은 10, 100, 115, 128 등이었고 그 중 가장 좋은 결과가 나왔던 100을 사용하였습니다. 하지만 100, 115, 128에서의 정확도는 눈에 뜨는 변화를 보여주지 못하였습니다. 그러나 10에서 100의 증가량은 누가 봐도 변화를 알아볼 수 있는 수치가 나왔습니다.

날짜로는 기본 7일에서 10일, 30일을 시도하였고 그 중 가장 좋은 결과는 10일이었습니다. 제가 생각한 그 이유로는 코로나가 발생하고 나서 발생자 수 혹은 사망자 수는 한 달 정도의 긴 기간 동안 비슷한 수치가 나오는 것이 아니라 급격한 변화 및 경향성을 따라갔습니다. 그렇기 때문에 오히려 30일이라는 과도한 수치를 사용한 것이 역효과를 일으켰던 거 같습니다.

사용되는 column은 구하자고 하는 것과 비슷한 경향성을 지니는 것을 찾는 것이 중요하다고 생각됩니다. 이 방식은 preprocess의 방식과 함께 더 개선될 여지가 있다고 생각합니다.

Total loss에서 사망자보다 발병자에 초점을 맞춘 이유는 사망하는 숫자에 비해 발병자가 더 큰 수치로 변화되고 더 큰 값을 갖을 수 밖에 없다고 생각해 큰 가중치를 주어 정확도를 높이고자 하였습니다.