Kaggle Competition –

Mechanisms of Action(MoA) Prediction

한현수

## 1. Problem

In this competition, you will have access to a unique dataset that combines gene expression and cell viability data. The data is based on a new technology that measures simultaneously (within the same samples) human cells' responses to drugs in a pool of 100 different cell types (thus solving the problem of identifying ex-ante, which cell types are better suited for a given drug). In addition, you will have access to MoA annotations for more than 5,000 drugs in this dataset.

As is customary, the dataset has been split into testing and training subsets. Hence, your task is to use the training dataset to develop an algorithm that automatically labels each case in the test set as one or more MoA classes. Note that since drugs can have multiple MoA annotations, the task is formally a multi-label classification problem.

### 1) Data Fields

- g-feature: 유전자 발현 데이터
- c-feature: 세포 생존 가능성(능력) 데이터
- cp_type: 화합물로 처리된 표본 (cp_type 이 ctl_vehicle 일 때 MoAs 는 항상 0 (어떻게 처리할지, 행을 삭제할지 생각해야함))
- cp_time: 치료기간 (24 시간, 48 시간, 72 시간)
- cp_dose: 복용량

## 2. MoA Prediction

### 2-1) Base line

Import package

```
import numpy as np
import pandas as pd
import tensorflow as tf
import random
import os
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

Read csv with pandas

```
train_features = pd.read_csv("/content/train_features.csv")
train_targets = pd.read_csv("/content/train_targets_scored.csv")
test_features = pd.read_csv("/content/test_features.csv")
submission = pd.read_csv("/content/sample_submission.csv")
```

One-hot Encoding

```
COLS = ['cp_type','cp_dose']
FE = []
for col in COLS:
    for mod in train_features[col].unique():
        train_features[mod] = (train_features[col] == mod).astype(int)
del train_features['sig_id']
del train_features['cp_type']
del train_features['cp_dose']
FE+=list(train_features.columns)
del train_targets['sig_id']
```

one-hot encoding과 함께 FE에 학습에 이용하는 Feature를 저장하였다.

Model

```
def model():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(len(FE)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(4096, activation="relu"),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(4096, activation="relu"),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(206, activation="sigmoid")
        ])

    model.compile(optimizer=tf.keras.optimizers.Adam(lr=2.75e-5),
                  loss='binary_crossentropy', metrics=["accuracy", "AUC"])
    return model
```

이 모델은 간단한 2-layer 모델인데, 여기서의 특징은 각 layer 사이마다 배치정규화를 진행하여 학습의 효율을 올리려고 하였다.

## Model Summary

Model: "sequential_16"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| batch_normalization_48 (Batc | (None, 877) | 3508 |
| dense_48 (Dense) | (None, 4096) | 3596288 |
| batch_normalization_49 (Batc | (None, 4096) | 16384 |
| dropout_32 (Dropout) | (None, 4096) | 0 |
| dense_49 (Dense) | (None, 4096) | 16781312 |
| batch_normalization_50 (Batc | (None, 4096) | 16384 |
| dropout_33 (Dropout) | (None, 4096) | 0 |
| dense_50 (Dense) | (None, 206) | 843982 |

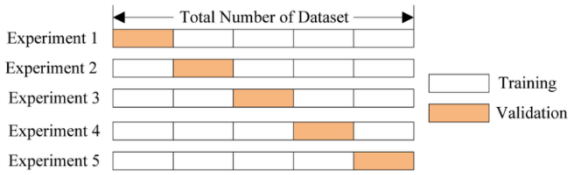
Total params: 21,257,858
Trainable params: 21,239,720
Non-trainable params: 18,138

## Learning with KFold

```
from sklearn.model_selection import KFold
NFOLD = 5
kf = KFold(n_splits=NFOLD)

BATCH_SIZE=128
EPOCHS=35

for col in COLS:
    for mod in test_features[col].unique():
        test_features[mod] = (test_features[col] == mod).astype(int)
sig_id = pd.DataFrame()
sig_id = test_features.pop('sig_id')
del test_features['cp_type']
del test_features['cp_dose']

pe = np.zeros((test_features.shape[0], 206))

train_features = train_features.values
train_targets = train_targets.values
pred = np.zeros((train_features.shape[0], 206))
cnt=0
for tr_idx, val_idx in kf.split(train_features):
    reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                                       patience=3, verbose=1, epsilon=1e-4, mode='min')
    cnt += 1
    print(f"FOLD {cnt}")
    net = model()
    net.fit(train_features[tr_idx], train_targets[tr_idx], batch_size=BATCH_SIZE,
            epochs=EPOCHS,
            validation_data=(train_features[val_idx], train_targets[val_idx]),
            verbose=0, callbacks=[reduce_lr_loss])
    print("train", net.evaluate(train_features[tr_idx], train_targets[tr_idx]
                                , verbose=0, batch_size=BATCH_SIZE))
    print("val", net.evaluate(train_features[val_idx], train_targets[val_idx],
                              verbose=0, batch_size=BATCH_SIZE))
    print("predict val...")
    pred[val_idx] = net.predict(train_features[val_idx], batch_size=BATCH_SIZE,
                                verbose=0)
    print("predict test...")
    pe += net.predict(test_features, batch_size=BATCH_SIZE, verbose=0) / NFOLD
```

여기선 5개의 fold를 만들어 진행하였고, 사용한 callback 함수로 Learning rate의 개선이 없을 경우 Learning rate에 0.1을 곱하여 var_loss를 줄이려고 하였다.

## K-Folds Cross Validation




이는 training data를 K개의 fold로 나누고, 그 중 한 개를 validation data로 지정하고 남은 K-1개를 training data로 지정하여 학습한다. 그 다음 Fold에서 validation data를 바꿔서 지정하여 총 K번 학습하는 방법이다. 위의 코드에서는 각 fold 마다 prediction값을 구하여 평균을 냄으로 최종 prediction값을 결정하였다.

이것의 단점은 일반적인 학습법에 비해 시간소요가 많이 되며, 레이블데이터의 분포가 균일하지 않아 정확도가 떨어질 수 있다는 점이다. 그래서 이후, stratifiedkFold를 이용해 개선하려고 합니다.