1. (1%)請比較有無normalize(rating)的差別。並說明如何normalize.

❖ 作法:

- 將rating減掉平均再除以標準差

```
std = np.std(train_out)
mean = np.mean(train_out)
train_out = (train_out - mean) / std;
```

- predict完的答案乘以標準差再加上平均

```
out = model.predict([test_user, test_movie])
out = out * std + mean
```

❖ 結果:

- public score: 0.85811

  origin public score: 0.85007

- 標準化對於我的model會降低準確度
- 標準化使training的速度變快

2. (1%)比較不同的latent dimension的結果。

❖ 作法:

- 使用random seed: 7122, validation split: 0.1, earlystop: 5 進行測試

❖ 結果:

- Dimension - Min Validation loss (Mean Square Error)
  - 160 - 0.7278
  - 128 - 0.7278
  - 96 - 0.7291
  - 64 - 0.7310
  - 32 - 0.7373
- 大於一定的dimension後，已經可以足夠的表示出一個user & movie的性質，所以再繼續變大並沒有幫助
- 選擇validation結果最佳的128

3. (1%)比較有無bias的結果。

❖ 作法:

- 將user和movie做dimension = 1的embedding當作bias，並加到答案上

```
user_input = keras.layers.Input(shape=[1])
user_vec = keras.layers.Flatten()(keras.layers.Embedding(user_num, 128)(user_input))

movie_input = keras.layers.Input(shape=[1])
movie_vec = keras.layers.Flatten()(keras.layers.Embedding(movie_num, 128)(movie_input))

dot_vecs = keras.layers.dot([movie_vec, user_vec], axes=-1)
user_bias = keras.layers.Flatten()(keras.layers.Embedding(user_num, 1)(user_input))
movie_bias = keras.layers.Flatten()(keras.layers.Embedding(movie_num, 1)(movie_input))

input_vecs = keras.layers.merge([dot_vecs, user_bias, movie_bias], 'sum')
```

❖ 結果:

- public score: 0.84809
  origin public score: 0.85007
- 比原本的結果上升了一點，看來某些使用者會有偏見，因此透過bias可以提升準確度

4. (1%)請試著用DNN來解決這個問題，並且說明實做的方法(方法不限)。並比較MF和NN的結果，討論結果的差異。

❖ 作法:

- 將user & movie改成用concate，然後flatten後連接兩層dense (64, 32)，最後linear到預測的答案進行training

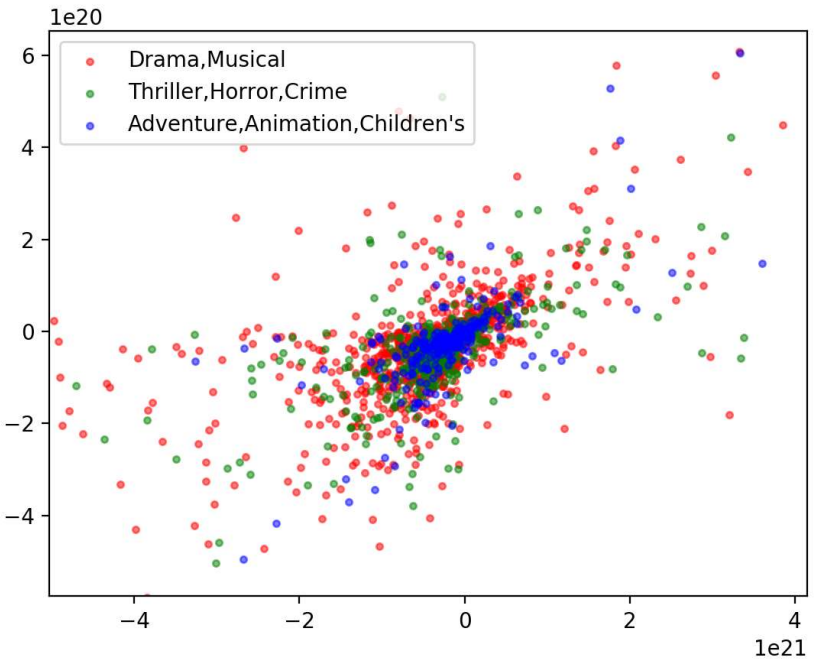
```
user_input = keras.layers.Input(shape=[1])
user_vec = keras.layers.Flatten()(keras.layers.Embedding(user_num, 128)(user_input))

movie_input = keras.layers.Input(shape=[1])
movie_vec = keras.layers.Flatten()(keras.layers.Embedding(movie_num, 128)(movie_input))

input_vecs = keras.layers.concatenate([movie_vec, user_vec], axis=-1)
dnn = keras.layers.Dropout(0.1)(keras.layers.Dense(64, activation='relu')(input_vecs))
dnn = keras.layers.Dropout(0.1)(keras.layers.Dense(32, activation='relu')(input_vecs))
dnn = keras.layers.Dense(1, activation='linear')(dnn)

model = keras.models.Model([user_input, movie_input], dnn)
model.compile(loss='mean_squared_error', optimizer='adam')
```

❖ 結果:

- public score: 0.87911
  origin public score: 0.85007
- 我的DNN model比MF的準確度差很多，且訓練的速度慢很多，由此可見複雜的model不見得比較好。也有可能是資料量不夠大，使NN沒辦法有效地找到準確的model。

5. (1%)請試著將movie的embedding用tsne降維後，將movie category當作label來作圖。

❖ 使用sklearn.manifold的TSNE降維



6. **(BONUS)(1%)試著使用除了rating以外的feature, 並說明你的作法和結果，結果好壞不會影響評分。**

❖ 作法:

- user的Gender::Age::Occupation::Zip-code concat在user vector的最左邊
- movie的category轉成encode成01 concat在movie vector的最右邊
- 將user vector和movie vector dot起來進行training

```
# get user vector
user_id_input = keras.layers.Input(shape=[1])
user_data_input = keras.layers.Input(shape=[4])

user_vec_left = user_id_input
user_vec_right = keras.layers.normalization.BatchNormalization()(user_data_input)

user_vec_left = keras.layers.Flatten()(keras.layers.Embedding(user_num, 100 - 4)(user_vec_left))
user_vec_left = keras.layers.normalization.BatchNormalization()(user_vec_left)
user_vec = keras.layers.concatenate([user_vec_left, user_vec_right], axis=-1)


# get movie vector
movie_id_input = keras.layers.Input(shape=[1])
movie_data_input = keras.layers.Input(shape=[18])

movie_vec_left = keras.layers.normalization.BatchNormalization()(movie_data_input)
movie_vec_right = movie_id_input

movie_vec_right = keras.layers.Flatten()(keras.layers.Embedding(movie_num, 100 - 18)(movie_vec_right))
movie_vec_right = keras.layers.normalization.BatchNormalization()(movie_vec_right)
movie_vec = keras.layers.concatenate([movie_vec_left, movie_vec_right], axis=-1)

# dot
input_vecs = keras.layers.dot([movie_vec, user_vec], axes=-1)

model = keras.models.Model([user_id_input, user_data_input, movie_id_input, movie_data_input], input_vecs)
model.compile(loss='mean_squared_error', optimizer='adam')
```

❖ 結果:

- public score: 0.84690
  origin public score: 0.84425
- 結果和原本的方法差不多，我認為可能是在原本的embedding，model就學到了電影的分類，所以再加上原本的分類沒有比較好。