學號：R05921069 系級： 電機碩一 姓名：黃武昱

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

   有 normalize 的結果:

```
Epoch 12/1000
809472/809886 [============================>.] - ETA: 0s - loss: 0.2563 - root_mean_squared_error: 0.3400Epoch 00011: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.2563 - root_mean_squared_error: 0.3400 - val_loss: 0.7609 - val_root_mean_squar
ed_error: 0.6825
Epoch 13/1000
806400/809886 [============================>.] - ETA: 0s - loss: 0.2453 - root_mean_squared_error: 0.3295Epoch 00012: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.2455 - root_mean_squared_error: 0.3296 - val_loss: 0.7766 - val_root_mean_squar
ed_error: 0.6898
Epoch 14/1000
806656/809886 [============================>.] - ETA: 0s - loss: 0.2357 - root_mean_squared_error: 0.3201Epoch 00013: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.2358 - root_mean_squared_error: 0.3202 - val_loss: 0.7915 - val_root_mean_squar
ed_error: 0.6960
Epoch 15/1000
806144/809886 [============================>.] - ETA: 0s - loss: 0.2274 - root_mean_squared_error: 0.3120Epoch 00014: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.2274 - root_mean_squared_error: 0.3120 - val_loss: 0.8060 - val_root_mean_squar
ed_error: 0.7023
```

   無 normalize 的結果:

```
Epoch 15/1000
808960/809886 [============================>.] - ETA: 0s - loss: 0.4325 - root_mean_squared_error: 0.4709Epoch 00014: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.4326 - root_mean_squared_error: 0.4710 - val_loss: 0.8476 - val_root_mean_squar
ed_error: 0.7190
Epoch 16/1000
808960/809886 [============================>.] - ETA: 0s - loss: 0.4119 - root_mean_squared_error: 0.4560Epoch 00015: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.4118 - root_mean_squared_error: 0.4560 - val_loss: 0.8625 - val_root_mean_squar
ed_error: 0.7250
Epoch 17/1000
808960/809886 [============================>.] - ETA: 0s - loss: 0.3930 - root_mean_squared_error: 0.4420Epoch 00016: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.3931 - root_mean_squared_error: 0.4421 - val_loss: 0.8789 - val_root_mean_squar
ed_error: 0.7322
Epoch 18/1000
806656/809886 [============================>.] - ETA: 0s - loss: 0.3759 - root_mean_squared_error: 0.4291Epoch 00017: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.3760 - root_mean_squared_error: 0.4292 - val_loss: 0.8943 - val_root_mean_squar
ed_error: 0.7385
Epoch 19/1000
807936/809886 [============================>.] - ETA: 0s - loss: 0.3603 - root_mean_squared_error: 0.4172Epoch 00018: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 10s - loss: 0.3605 - root_mean_squared_error: 0.4173 - val_loss: 0.9094 - val_root_mean_squar
ed_error: 0.7452
```

   從上兩表中，可以很明顯地觀察到，有 normalize 的 RMSE 好過於沒有 normalize 的。
   Normalize 的方法很容易，先取得 rating 的 mean 與 STD，並且用 rating 減去 mean 並除於 STD，就可以得到了。
   之後再把 predict 出來的答案還原就好。
   以下為 code 的實現:

```
##normalize ratings
rating_mean = np.mean(Y_data)
rating_SD = np.std(Y_data)
rationg_norm = (Y_data-rating_mean)/(rating_SD*1.0)
```

```
Y_test = Y_test*rating_SD+rating_mean
```

2. (1%)比較不同的 latent dimension 的結果。
   當 latent dimension 為 10 時:

```
Epoch 48/1000
808704/809886 [============================>.] - ETA: 0s - loss: 0.6826 - root_mean_squared_error: 0.6277Epoch 00047: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 6s - loss: 0.6826 - root_mean_squared_error: 0.6277 - val_loss: 0.7795 - val_root_mean_square
d_error: 0.6849
Epoch 49/1000
803584/809886 [============================>.] - ETA: 0s - loss: 0.6811 - root_mean_squared_error: 0.6267Epoch 00048: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 6s - loss: 0.6811 - root_mean_squared_error: 0.6267 - val_loss: 0.7804 - val_root_mean_square
d_error: 0.6855
Epoch 50/1000
806656/809886 [============================>.] - ETA: 0s - loss: 0.6794 - root_mean_squared_error: 0.6258Epoch 00049: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 6s - loss: 0.6795 - root_mean_squared_error: 0.6258 - val_loss: 0.7812 - val_root_mean_square
d_error: 0.6857
Epoch 51/1000
803072/809886 [============================>.] - ETA: 0s - loss: 0.6783 - root_mean_squared_error: 0.6250Epoch 00050: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 6s - loss: 0.6783 - root_mean_squared_error: 0.6250 - val_loss: 0.7809 - val_root_mean_square
d_error: 0.6856
```

當 latent dimension 為 50 時:

```
Epoch 19/1000
804608/809886 [============================>.] - ETA: 0s - loss: 0.5709 - root_mean_squared_error: 0.5607Epoch 00018: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 8s - loss: 0.5709 - root_mean_squared_error: 0.5607 - val_loss: 0.7915 - val_root_mean_square
d_error: 0.6928
Epoch 20/1000
805888/809886 [============================>.] - ETA: 0s - loss: 0.5583 - root_mean_squared_error: 0.5523Epoch 00019: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 8s - loss: 0.5583 - root_mean_squared_error: 0.5524 - val_loss: 0.7985 - val_root_mean_square
d_error: 0.6957
Epoch 21/1000
805888/809886 [============================>.] - ETA: 0s - loss: 0.5466 - root_mean_squared_error: 0.5444Epoch 00020: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 7s - loss: 0.5467 - root_mean_squared_error: 0.5444 - val_loss: 0.8048 - val_root_mean_square
d_error: 0.6987
Epoch 22/1000
808704/809886 [============================>.] - ETA: 0s - loss: 0.5353 - root_mean_squared_error: 0.5368Epoch 00021: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 7s - loss: 0.5353 - root_mean_squared_error: 0.5368 - val_loss: 0.8122 - val_root_mean_square
d_error: 0.7019
Epoch 23/1000
806144/809886 [============================>.] - ETA: 0s - loss: 0.5251 - root_mean_squared_error: 0.5298Epoch 00022: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 8s - loss: 0.5251 - root_mean_squared_error: 0.5298 - val_loss: 0.8199 - val_root_mean_square
d_error: 0.7052
```

由上兩圖可得知，當 latent dimension 越大時，RMSE 的下降比當 latent dimension 為 10 來的好很多。

3. (1%)比較有無 bias 的結果。

有 bias：

```
Epoch 15/1000
809728/809886 [============================>.] - ETA: 0s - loss: 0.1077 - root_mean_squared_error: 0.2143Epoch 00014: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 19s - loss: 0.1077 - root_mean_squared_error: 0.2143 - val_loss: 1.0138 - val_root_mean_squar
ed_error: 0.7913
Epoch 16/1000
809472/809886 [============================>.] - ETA: 0s - loss: 0.0939 - root_mean_squared_error: 0.2005Epoch 00015: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 19s - loss: 0.0939 - root_mean_squared_error: 0.2005 - val_loss: 1.0369 - val_root_mean_squar
ed_error: 0.8006
```

無 bias：

```
Epoch 15/1000
807936/809886 [============================>.] - ETA: 0s - loss: 0.1338 - root_mean_squared_error: 0.2419Epoch 00014: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 18s - loss: 0.1338 - root_mean_squared_error: 0.2419 - val_loss: 1.0088 - val_root_mean_squar
ed_error: 0.7883
Epoch 16/1000
808704/809886 [============================>.] - ETA: 0s - loss: 0.1166 - root_mean_squared_error: 0.2250Epoch 00015: val_root_mean_squared_er
ror did not improve
809886/809886 [==============================] - 18s - loss: 0.1167 - root_mean_squared_error: 0.2250 - val_loss: 1.0324 - val_root_mean_squar
ed_error: 0.7978
```

由上兩圖可觀察到，有 bias 的 RMSE 結果會比較好。

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。

MF model 架構:

```python
#MODEL-MF
user_input = Input(shape=[1])
user_vec = Embedding(n_users+1, latent_dim, embeddings_initializer="random_normal")(user_input)
user_vec = Flatten()(user_vec)
user_bias = Embedding(n_users+1, 1, embeddings_initializer="zeros")(user_input)
user_bias = Flatten()(user_bias)

movie_input = Input(shape=[1])
movie_vec = Embedding(n_movies+1, latent_dim, embeddings_initializer="random_normal")(movie_input)
movie_vec = Flatten()(movie_vec)
movie_bias = Embedding(n_movies+1, 1, embeddings_initializer="zeros")(movie_input)
movie_bias = Flatten()(movie_bias)

r_hat = Dot(axes=1)([user_vec,movie_vec])
r_hat = Add()([r_hat,user_bias,movie_bias])

model = Model([user_input, movie_input], r_hat)
```

結果:

Your Best Entry ↑

Your submission scored **0.88900**, which is not an improvement of your best score. Keep trying!

DNN model 架構:

```python
#MODEL-DNN
user_input = Input(shape=[1])
user_vec = Embedding(n_users+1, latent_dim, embeddings_initializer="random_normal")(user_input)
user_vec = Flatten()(user_vec)

movie_input = Input(shape=[1])
movie_vec = Embedding(n_movies+1, latent_dim, embeddings_initializer="random_normal")(movie_input)
movie_vec = Flatten()(movie_vec)

vec_inputs = Concatenate()([user_vec, movie_vec])
model = Dense(1024, activation='relu')(vec_inputs)
model = Dropout(0.4)(model)
model = Dense(512, activation='relu')(model)
model = Dropout(0.5)(model)
model = Dense(256, activation='relu')(model)
model = Dropout(0.6)(model)
model = Dense(128, activation='relu')(model)
model = Dropout(0.7)(model)
model = Dense(64, activation='relu')(model)
model = Dropout(0.8)(model)
model_out = Dense(1, activation='linear')(model)

model = Model([user_input, movie_input], model_out)
```

結果:

Your Best Entry ↑

Your submission scored **0.86789**, which is not an improvement of your best score. Keep trying!

由上圖的 MF 與 DNN 結果比較，

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

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：

首先讀取 user.csv 的檔案，並且把裡面性別的部分，區分為 0 和 1 ，男性為 0 女性為 1 ，因為 user 裡面的資料只有 3 千多筆，跟 train 的 88 萬筆有很大的

差距，所以我的作法是直接用 train data 裡面的 userID 去與 user 裡面的 userID 匹對，一樣的 ID 編號就覆蓋到 train data 裡面，在 feature 方面，我選擇 user 裡面的 Gender 和 Occupation 作為我的 feature。
在 test 方面也是同樣的作法。

Post-Deadline: Fri, 09 Jun 2017 07:05:06 Edit description	predict.csv	0.92281	0.92613