學號：B02705027 系級： 資管四 姓名：陳信豪

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

切同樣的 train/valid dataset，並運用同樣的 mf model 來跑。

差別只在於有無對 ratings 做 normalization。

patience = 10　　epochs = 1000

batch_size = 512　optimizer = 'adam'

normalization 的做法是對 y_train (要訓練的 ratings label)

算平均和標準差，之後將 y_train 和 y_valid 減掉平均，再除以標準差。

訓練好的 model 預測出來的結果 y_test，也是同樣處理。

mean = np.mean(y_train)

std = np.std(y_train)

y_train = y_train - mean

y_trian = y_train/std

y_valid = y_valid - mean

y_valid = y_valid/std

	Train RMSE	Valid RMSE	Kaggle Public RMSE	Kaggle Private RMSE
有 normalize	0.8780	0.8752	0.87344	0.87876
沒有 normalize	0.7792	0.8656	0.86642	0.86933

從結果來看，做 normalization 無益於增加 model performance。

2. (1%)比較不同的 latent dimension 的結果。

切同樣的 train/valid dataset，並運用同樣的 mf model 來跑。

（沒有 normalization）

差別只在於 latent dimension。

patience = 10　　epochs = 1000

batch_size = 512　optimizer = 'adam'

Epoch through: Ealry Stopping 前歷經的 epoch 數目

Epoch Take Per Time: 每個 Epoch 所需的時間

latent dimension	Epoch through	Epoch Take Per Time	Train RMSE	Valid RMSE
3	109	6s	0.8269	08709
5	84	6s	0.8038	0.8690
10	31	7s	0.8062	0.8649
15	27	8s	0.7867	0.8655
20	24	9s	0.7792	0.8656
30	20	10s	0.7800	0.8643

50	18	15s	0.7662	0.8643
100	15	31s	0.7310	0.8669
200	15	60s	0.6641	0.8699

從上表我們可以看到隨著 latent dimension 增大

Epoch through 會越小（越快找到最小點）

Epoch Take Per Time 會越大

Train RMSE 會越小

Valid RMSE 在 20.30 左右就可以有很好的結果

latent dimension 再增大也沒有顯著的下降了。

3. (1%)比較有無 bias 的結果。

切同樣的 train/valid dataset，並運用 mf model 來跑。

（沒有 normalization）

差別只在於有無 bias。

有 bias

```
# bias
user_bias = Embedding(n_users, 1, embeddings_initializer='zeros')(user_input)
user_bias = Flatten()(user_bias)
item_bias = Embedding(n_items, 1, embeddings_initializer='zeros')(item_input)
item_bias = Flatten()(item_bias)
r_hat = Dot(axes=1)([user_vec, item_vec])
r_hat = Add()([r_hat, user_bias, item_bias])
model = keras.models.Model([user_input, item_input], r_hat)
```

無 bias

```
# no bias
r_hat = Dot(axes=1)([user_vec, item_vec])
model = keras.models.Model([user_input, item_input], r_hat)

return model
```

	Train RMSE	Valid RMSE	Kaggle Public RMSE	Kaggle Private RMSE
有 bias	0.7792	0.8656	0.86642	0.86932
沒有 bias	0.7743	0.8670	0.86796	0.87185

從 Validation 和 Test 的結果來看，有加上 bias 會使約果好一些。

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 (param 數：209874)

Layer (type)	Output Shape	Param #	Connected to
input_115 (InputLayer)	(None, 1)	0	
input_116 (InputLayer)	(None, 1)	0	
embedding_201 (Embedding)	(None, 1, 20)	120820	input_115[0][0]
embedding_202 (Embedding)	(None, 1, 20)	79060	input_116[0][0]
flatten_201 (Flatten)	(None, 20)	0	embedding_201[0][0]
flatten_202 (Flatten)	(None, 20)	0	embedding_202[0][0]
embedding_203 (Embedding)	(None, 1, 1)	6041	input_115[0][0]
embedding_204 (Embedding)	(None, 1, 1)	3953	input_116[0][0]
dot_46 (Dot)	(None, 1)	0	flatten_201[0][0] flatten_202[0][0]
flatten_203 (Flatten)	(None, 1)	0	embedding_203[0][0]
flatten_204 (Flatten)	(None, 1)	0	embedding_204[0][0]
add_44 (Add)	(None, 1)	0	dot_46[0][0] flatten_203[0][0] flatten_204[0][0]

Total params: 209,874
Trainable params: 209,874
Non-trainable params: 0

DNN model (param 數：213631)

Layer (type)	Output Shape	Param #	Connected to
input_113 (InputLayer)	(None, 1)	0	
input_114 (InputLayer)	(None, 1)	0	
embedding_199 (Embedding)	(None, 1, 20)	120820	input_113[0][0]
embedding_200 (Embedding)	(None, 1, 20)	79060	input_114[0][0]
flatten_199 (Flatten)	(None, 20)	0	embedding_199[0][0]
flatten_200 (Flatten)	(None, 20)	0	embedding_200[0][0]
concatenate_12 (Concatenate)	(None, 40)	0	flatten_199[0][0] flatten_200[0][0]
dense_34 (Dense)	(None, 150)	6150	concatenate_12[0][0]
dense_35 (Dense)	(None, 50)	7550	dense_34[0][0]
dense_36 (Dense)	(None, 1)	51	dense_35[0][0]

Total params: 213,631
Trainable params: 213,631
Non-trainable params: 0

DNN model

```
def create_model(n_users, n_items, latent_dim=20):
    user_input = Input(shape=[1])
    item_input = Input(shape=[1])
    user_vec = Embedding(n_users, latent_dim, embeddings_initializer='random_normal')(user_input)
    user_vec = Flatten()(user_vec)
    item_vec = Embedding(n_items, latent_dim, embeddings_initializer='random_normal')(item_input)
    item_vec = Flatten()(item_vec)

    merge_vec = Concatenate()([user_vec,item_vec])
    hidden = Dense(150, activation='relu')(merge_vec)
    hidden = Dense(50, activation='relu')(hidden)
    output = Dense(1)(hidden)
    model = keras.models.Model([user_input, item_input], output)


    return model
```

DNN model 的做法是在 embedding 和 flatten 完 user 和 movie 後

透過 keras.layers.Concarenate 將兩者合併起來

然後直接餵給 Dense

在這邊疊了兩層 Dense(150)、Dense(50)

切同樣的 train/valid dataset，並分別運用 MF 和 DNN model 來跑。

	Train RMSE	Valid RMSE	Kaggle Public RMSE	Kaggle Private RMSE
MF	0.7792	0.8656	0.86642	0.86932
DNN	0.8101	0.8783	0.87934	0.88253

在約略相同的 param 數目下，MF 的表現比 DNN 好。

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。