學號：B03705006 系級： 資管三 姓名： 侯舜元

1. (1%)請比較有無 normalize(rating)的差別。並說明如何 normalize.
先取 training data rating 的 mean 和 standard deviation 並記錄下來。把 training data 的 rating normalize，再用 normalized 後的 training data 當作 target。
最後在 testing 的時候，$result_{test}$ = $result^*_{test}$ x $train_{std}$ + $train_{mean}$
沒有 normalize rating: dimension 64, learning rate 0.0003, 175 epoch → RMSE = 0.714410790961
有 normalize rating: dimension 64, learning rate 0.0003, 175 epoch → RMSE = 0.853027468075

有 normalize rating 的結果顯著較差，可能是 normalize 後數字都太小太接近了，缺分 rating 不易。

2. (1%)比較不同的 latent dimension 的結果。
調整 moive_input 和 user_input 在 keras.layers.Embedding 的 dimension（設為一樣）。
dimension 32, learning rate 0.0003, 175 epoch → RMSE = 0.73801263
dimension 64, learning rate 0.0003, 175 epoch → RMSE = 0.71441079
dimension 84, learning rate 0.0003, 175 epoch → RMSE = 0.7196014
dimension 128 , learning rate 0.0003, 175 epoch → RMSE = 0.715795

當 latent dimension 在大到一定程度時，其 performance 就已經很相近了
可以看出最好的 performance 時，userid 和 movieid 的 latent dimension 在 64 附近。

3. (1%)比較有無 bias 的結果。
無 movie bias：rmse 0.71492147052
無 user bias: rmse 0.757419093437
可以看出 movie bias 對於結果的影響並沒有很大，但是拿掉 user bias 後 performance 顯著下降，可以推論出每個使用者有自己在評分的 bias 的趨勢，會把每個電影評的叫較高或較低。

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。
DNN 的結構圖如下，將 user embedding 以及 movie embedding concatenate 在一起再過 DNN 得出 rating，output 直接為 rating 的值。

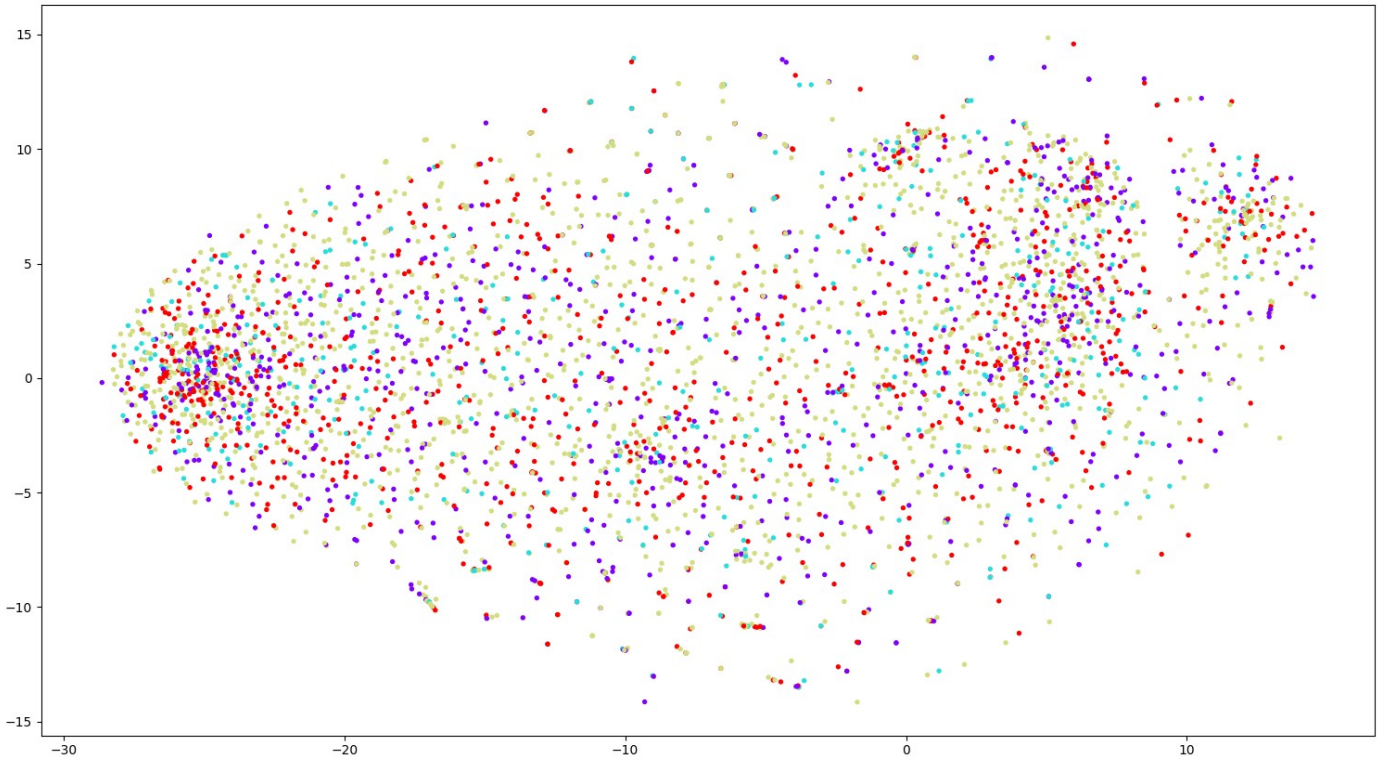
```
# movie input and embedding
movie_input = keras.layers.Input(shape=[1])
movie_vec = keras.layers.Flatten()(keras.layers.Embedding(n_movies + 1, 48)(movie_input))
movie_vec = keras.layers.Dropout(0.4)(movie_vec)

# users input and embedding
user_input = keras.layers.Input(shape=[1])
user_vec = keras.layers.Flatten()(keras.layers.Embedding(n_users + 1,48)(user_input))
user_vec = keras.layers.Dropout(0.4)(user_vec)

# dnn architecture
input_vecs = keras.layers.concatenate([movie_vec, user_vec])
nn = keras.layers.Dropout(0.5)(keras.layers.Dense(1024, activation='relu')(input_vecs))
nn = keras.layers.normalization.BatchNormalization()(nn)
nn = keras.layers.Dropout(0.5)(keras.layers.Dense(512, activation='relu')(nn))
nn = keras.layers.normalization.BatchNormalization()(nn)
nn = keras.layers.Dropout(0.5)(keras.layers.Dense(256, activation='relu')(nn))
nn = keras.layers.normalization.BatchNormalization()(nn)
nn = keras.layers.Dropout(0.5)(keras.layers.Dense(128, activation='relu')(nn))
nn = keras.layers.normalization.BatchNormalization()(nn)
result = keras.layers.Dense(1, activation='relu')(nn)
```

DNN 的 performance 為 rmse =0.86614（上 kaggle 有過 strong)，比使用 MF 的 performance 略差。可能是 network 不夠深，或是 embedding 的 dimension 不夠大。

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




紅：["Children's", "Musical", "Animation" , 'Documentary','Comedy']
綠：['War', 'Crime', 'Sci-Fi','Action', 'Adventure']
淺藍：[ 'Drama', 'Romance']
紫：[ 'Fantasy','Thriller', 'Horror' ]
紅色跟紫色類型的電影有比較多集中的狀況，藍色和綠色類別的電影比較分散。

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。