# ML HW6 Report

學號：B04611015 系級： 資工二 姓名：陳佳佑

1. (1%)請比較有無 normalize(rating)的差別。並說明如何 normalize.
   我將資料的 rating 減掉總共的平均再除以標準差，得到 normalize 的 rating。
   然後 predict 的結果再乘以標準差加上平均，得到還原後的結果。

```
std  = np.std(y)
mean  = np.mean(y)
y = (y-mean)/std
```

   Without Normalized : 0.84326 (Kaggle Best)
   Normalized : 0.86417

   Normalized 後的結果較差，有可能是因為標準化，造成最佳化時，極點附近是平面，gradient 太小造成無法逼近，而原先 unnormalized 的曲面，有可能相比之下較尖。

2. (1%)比較不同的 latent dimension 的結果。

| Latent Dimension | Best Validation Loss |
|---|---|
| 120 | 0.8513 |
| 110 | 0.8458 |
| 100 | 0.8512 |
| 90 | 0.8477 |
| 80 | 0.8485 |
| 70 | 0.8476 |
| 60 | 0.8506 |
| 50 | 0.8520 |
| 40 | 0.8536 |

   從上圖可以觀察到，不同 latent dimension 之間，在 70~110 時，其實並沒有差非常多，可以看成是，實際的 latent feature 只有到 70 維，而往上以後，只是多很多為 0 的 feature。而實際上在 Training 時，Best Validation Loss 出現的 epoch，隨著 Latent Dimension 增加而遞減，可以想成是因為在沒有任何 normalization 下，越多的維度 dot 所產生的 gradient 會越大，造成更快走向 local minimum。

3. (1%)比較有無 bias 的結果。

**Bias Model**

```
def generate_model(n_movies, n_users):
        movie_input = keras.layers.Input(shape=(1,))
        movie_vec = keras.layers.Flatten()(keras.layers.Embedding(n_movies + 1, 100, embeddings_initializer='random_uniform')(movie_input))
        movie_vec = keras.layers.Dropout(0.2)(movie_vec)

        user_input = keras.layers.Input(shape=(1,))
        user_vec = keras.layers.Flatten()(keras.layers.Embedding(n_users + 1, 100, embeddings_initializer='random_uniform')(user_input))
        user_vec = keras.layers.Dropout(0.2)(user_vec)

        input_vecs = keras.layers.merge([movie_vec,user_vec],'dot')

        mbv = keras.layers.Flatten()(keras.layers.Embedding(n_movies + 1,1,embeddings_initializer='random_uniform')(movie_input))
        ubv = keras.layers.Flatten()(keras.layers.Embedding(n_users + 1,1,embeddings_initializer='random_uniform')(user_input))
        output = keras.layers.merge([input_vecs,mbv,ubv],'sum')

        model = kmodel.Model([movie_input, user_input], output)
        model.compile(optimizer = 'adam',loss = 'mean_squared_error', metrics=[root_mean_squared_error])
        return model
```

**Bias Score (kaggle) : 0.84442 (2nd place)**

**Orginal Score (kaggle) : 0.84326 (2nd place)**

**基本上兩者的差距很小，可以算是統計上的誤差。**

**Bias term 最主要的目的是為了消除單一 user 或單一電影的偏見。而在此資料集上 bias 的不明顯，有可能是因為每個用戶的偏見都不大。**

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 model**

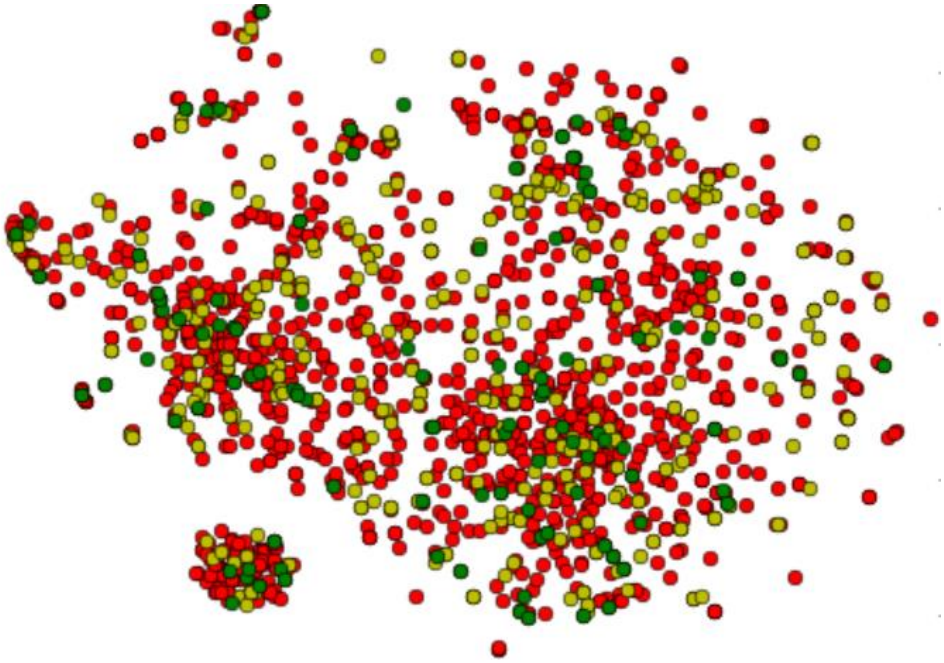
```
movie_input = keras.layers.Input(shape=[1])
movie_vec = keras.layers.Flatten()(keras.layers.Embedding(n_movies + 1, 120)(movie_input))
movie_vec = keras.layers.Dropout(0.1)(movie_vec)

user_input = keras.layers.Input(shape=[1])
user_vec = keras.layers.Flatten()(keras.layers.Embedding(n_users + 1, 120)(user_input))
user_vec = keras.layers.Dropout(0.1)(user_vec)

input_vecs = keras.layers.merge([movie_vec, user_vec], mode='concat')
nn = keras.layers.Dropout(0.2)(keras.layers.Dense(128, activation='relu')(input_vecs))
nn = keras.layers.normalization.BatchNormalization()(nn)
nn = keras.layers.Dropout(0.3)(keras.layers.Dense(128, activation='relu')(nn))
nn = keras.layers.normalization.BatchNormalization()(nn)
nn = keras.layers.Dense(128, activation='relu')(nn)

result = keras.layers.Dense(5, activation='softmax')(nn)

model = kmodels.Model([movie_input, user_input], result)
model.compile('adam', 'categorical_crossentropy')
```

Embedding 的做法與 MF 一樣，但是把 dot 改成 concat，然後接 DNN。

MF 的結果為 0.84, DNN 的結果為 1.2

DNN 的結果會比較差的原因有可能如下，首先，因為 DNN 的結果，我是用 softmax 所以出來結果只會是整數，在 evaluation 公式為 RMSE 的情況下，每個錯誤都是非常巨大的。再者，DNN 的 model 對於這個訓練集而言有點過於複雜，造成 Model 的 complexity 更高，更容易 overfitting，或許把 Dropout 調大，跑更多的 epoch 能有一樣的表現，但是就效率而言，已經慢 MF 很多了。

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



negative = [‘Horror’,’War’,’Crime’,’Thriller’,’Film-Noir’] : red
Child = [‘Animation’,’Children’s’,’Fantasy’] : yellow
Misc = [‘Mystery’,’Sci-Fi’] : green

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。

首先，我將 Genre 變成 19 維的 0,1 矩陣，然後接下來我直接讓他與 Embedding 過後的 movie_id concat，形成一個 120 維的 tensor，最後與 userid 的 embedding 內積。

```
def generate_model(n_movies, n_users):
        movie_input = keras.layers.Input(shape=(1,))
        movie_vec = keras.layers.Flatten()(keras.layers.Embedding(n_movies + 1, 120, embeddings_initializer='random_uniform')(movie_input))
        movie_vec = keras.layers.Dropout(0.2)(movie_vec)

        movie_cat = keras.layers.Input(shape=(19,))

        user_input = keras.layers.Input(shape=(1,))
        user_vec = keras.layers.Flatten()(keras.layers.Embedding(n_users + 1, 101, embeddings_initializer='random_uniform')(user_input))
        user_vec = keras.layers.merge([movie_cat,user_vec],"concat")
        user_vec = keras.layers.Dropout(0.2)(user_vec)

        input_vecs = keras.layers.merge([movie_vec,user_vec],'dot')
        model = kmodel.Model([movie_input, user_input,movie_cat], input_vecs)
        model.compile(optimizer = 'adam',loss = 'mean_squared_error')
        #model.summary()
        return model
```

會如此設計是因為剛好這學期軒田老師的技法課的 final project 也是做 recommender system，而我們也使用了類似的技巧，獲得不錯的 performance。除此之外，或許在 concat 前加一層 Dense 會有更好的結果，因為 genre 就不會都是 0,1。

Bonus (Kaggle) : 0.84978