學號：R05922130 系級： 資工所 姓名：王瀚磊

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

首先將 training data 讀進來後計算其 rating 的平均值以及標準差，然後依照以下公式計算 normalize（公式 1），在 predict 完後再將結果 inverse 回 1-5。我將這個結果套用至 Matrix Factorization(MF)以及 DNN 兩種 model 上，得到的結果如下表所示（表 1），可以發現在 MF 中有稍微明顯的進步（大約 0.003），然而在 DNN model 上並沒有顯著的進步（大約 0.001）。

$$\text{norm}(x) = \frac{x - \mu}{\sigma}, \mu\text{: mean}, \sigma\text{: } deviation$$

公式 1. Normalization

	MF Model	DNN Model
Without normalization	0.86040	0.85269
With normalization	0.85709	0.85173

表 1. Comparison of normalization on Kaggle Score

2. (1%)比較不同的 latent dimension 的結果。

我試著比較不同 latent dimension 在 MF 上的表現，但我發現並沒有什麼太大的差別，我的 latent factor 試過 50、150 以及 200，其結果如表 2 所示。由於在比較這部分的時候才剛過 simple baseline，每個 model 的 epoch 還沒調整到最佳，因此這邊表現的結果並沒有像上面那麼好。

	50	150	200
Validation RMSE	0.88217	0.88531	0.88002

表 2. Comparison of different latent dimension on validation RMSE

3. (1%)比較有無 bias 的結果。

在實作 bias 至 MF Model 中，利用新開的兩個 embedded layer（1 維）與原本內積 layer 再做一次 merge，如圖 1 所示。而比較的結果如表 3 所示，在這邊可以發現有沒有做 bias 其實效果差蠻多的，有了 bias 後 kaggle score 進步了 0.02 左右。

	Without bias	With bias
Kaggle Score	0.88079	0.86466

表 3. Comparison of bias on Kaggle Score

```
class CFModel(Sequential):

    def __init__(self, n_users, m_items, k_factors, **kwargs):
        P = Sequential()
        P.add(Embedding(n_users, k_factors, input_length=1))
        P.add(Reshape((k_factors,)))
        Q = Sequential()
        Q.add(Embedding(m_items, k_factors, input_length=1))
        Q.add(Reshape((k_factors,), name='movie_embedded'))
        P_bias = Sequential()
        P_bias.add(Embedding(n_users, 1, input_length=1))
        P_bias.add(Reshape((1,)))
        Q_bias = Sequential()
        Q_bias.add(Embedding(m_items, 1, input_length=1))
        Q_bias.add(Reshape((1,)))
        super(CFModel, self).__init__(**kwargs)
        self.add(Merge([Merge([P, Q], mode='dot', dot_axes=1), P_bias,
Q_bias], mode='sum'))
```

圖 1. MF Model implementation

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。

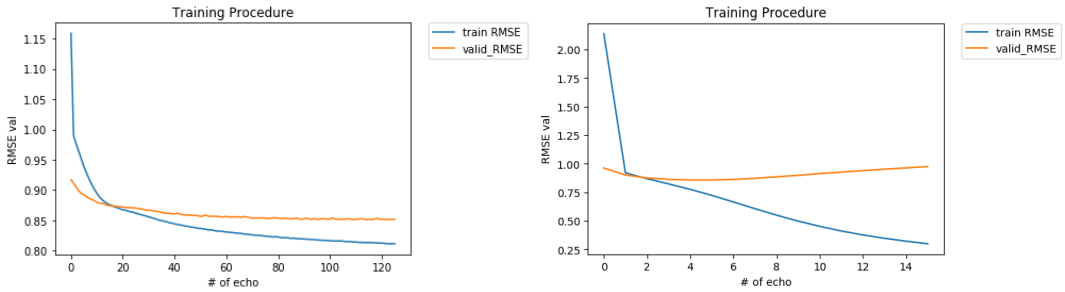
在實作中與 MF Model 並沒有太大的差別，只是在 merge layer 中，原本所指定的 mode 為 dot，為了實作 DNN 我將 mode 改為 concat（將 userId 以及 movieId 的 Embedding layer 串接在一起），並且用一層 Dense Layer 去接（Dropout 設為 0.5），如圖 2 所示。這邊原本想照 MF Model 一樣新增一個 bias layer，然而最後發現加錯位置了，應該將兩層 bias 與最後 output layer 做 merge 才能得到正確結果，照我原本實作的方法基本上就跟把 latent dimension 多增加 1 是一樣的意思，並沒有意義。然而 kaggle 上成績是利用 ensemble（四個 model），怕來不及更新，因此暫且不更動。

```
class DeepModel(Sequential):

    def __init__(self, n_users, m_items, k_factors, p_dropout=0.1, **kwargs):
        P = Sequential()
        P.add(Embedding(n_users, k_factors, input_length=1))
        P.add(Reshape((k_factors,)))
        Q = Sequential()
        Q.add(Embedding(m_items, k_factors, input_length=1))
        Q.add(Reshape((k_factors,)))
        P_bias = Sequential()
        P_bias.add(Embedding(n_users, 1, input_length=1))
        P_bias.add(Reshape((1,)))
        Q_bias = Sequential()
        Q_bias.add(Embedding(m_items, 1, input_length=1))
        Q_bias.add(Reshape((1,)))
        super(DeepModel, self).__init__(**kwargs)
        self.add(Merge([P, Q, P_bias, Q_bias], mode='concat'))
        self.add(Dropout(p_dropout))
        self.add(Dense(k_factors+2))
        self.add(Activation(PReLU()))
        self.add(Dropout(p_dropout))
        self.add(Dense(1, activation='linear'))
```

圖 2. DNN Model implementation

透過將 MF Model 與 DNN Model 兩個 training procedure 畫出來，可以發現在 training 的時候 MF Model 的 RMSE 下降一下後就穩定上升了，不像 DNN 下降很久才會上升，我認為是因為參數量的差別，MF Model 的參數量遠比 DNN 來得少，因此再訓練一下後即達到 overfitting，不像 DNN 需要訓練多一點的參數才達到 overfitting，如圖 3 所示。



(a) DNN Training Procedure (b) MF Training Procudure

圖 3. Training Procedure of DNN and MF Model

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

在這題中，我原本將所有 training data 丟進我 train 好的 model，並且將 movieId 的 embedding output 取出來，利用 sklearn 的 TSNE 降維至 2-dim，但是這邊有一個嚴重的問題是當所有 training data 的 embedding layer output 一起丟去 TSNE 做降維時，sklearn 會有 OOM 的問題，為了電腦的壽命以及程式的成功執行我將 training data 隨機 sample 5000 筆資料，並且利用助教的 slide 所提供的分類例子做分類，因此分成三項大類，結果如圖 4 所示。從圖中可以看見，這樣的分類是很失敗的，因為並沒有有效地把 movieId 透過降維分散開來。

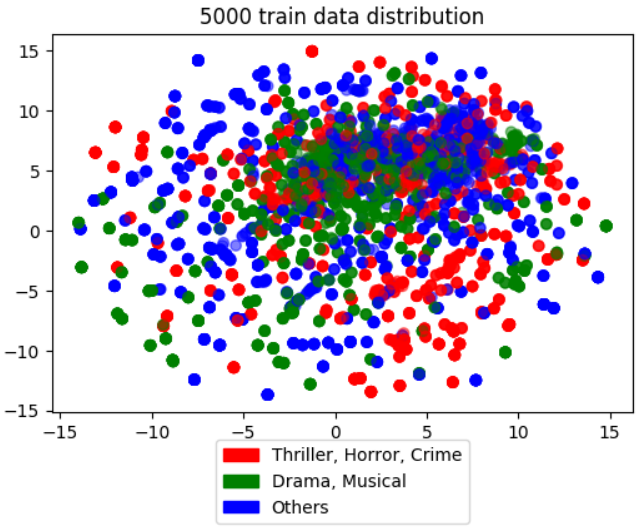


圖 4. 5000 training data distribution (By TSNE)

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。

由於在 problem 5. 中已經將所有 data 做降維並且分成新的類別（依照圖 4 中

的分類方式），因此這邊我將 movieId 全部降維並且分類好後丟進 model 重新 train 一次觀察，結果果不其然，由於分類並沒有成功分群，因此在 validation RMSE 上表現也是沒有明顯的改變，我將 TSNE 後的結果跑了兩次觀察，如表 4 所示。

	Without TSNE	With TSNE (1)	With TSNE (2)
Validation RMSE	0.85721	0.85684	0.85801

表 4. Comparison of TSNE with movieId on validation RMSE