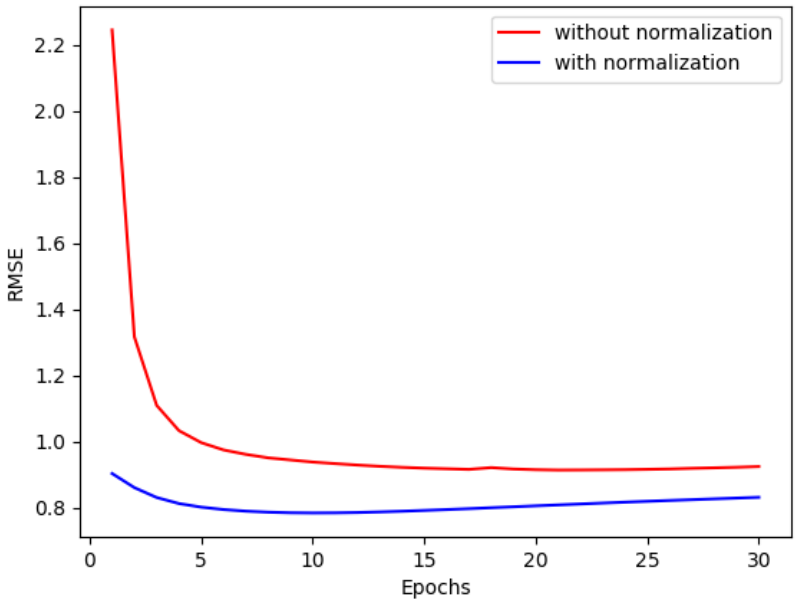
1. (1%)請比較有無 normalize(rating)的差別。並說明如何 normalize.

   本題使用的 model 為 MF，架構如下圖，將 UserID 及 MovieID 各自當成 input 並使用 keras 提供的 embedding 來建立此兩層 layer，latent dimension 設定為 50，並利用內積的方式來完成 matrix factorization，加上 UserID 的 bias 與 MovieID 的 bias 後完成此 model。Loss function 採用 MSE，optimizer 使用 Adam，取 10% training data 作為 validation data。

   而 normalize 則是針對 rating 的部分，計算出 rating 的平均與標準差後，將 training data 減去平均再除以標準差來完成 normalization，結果如下。可以觀察到 normalization 能有效的降低 validation data 的 RMSE。

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 1)	0	
input_4 (InputLayer)	(None, 1)	0	
embedding_3 (Embedding)	(None, 1, 50)	302000	
embedding_4 (Embedding)	(None, 1, 50)	194150	
flatten_3 (Flatten)	(None, 50)	0	
flatten_4 (Flatten)	(None, 50)	0	
embedding_5 (Embedding)	(None, 1, 1)	6040	
embedding_6 (Embedding)	(None, 1, 1)	3883	
dot_1 (Dot)	(None, 1)	0	
flatten_5 (Flatten)	(None, 1)	0	
flatten_6 (Flatten)	(None, 1)	0	
add_1 (Add)	(None, 1)	0	

Total params: 506,073.0
Trainable params: 506,073.0
Non-trainable params: 0.0



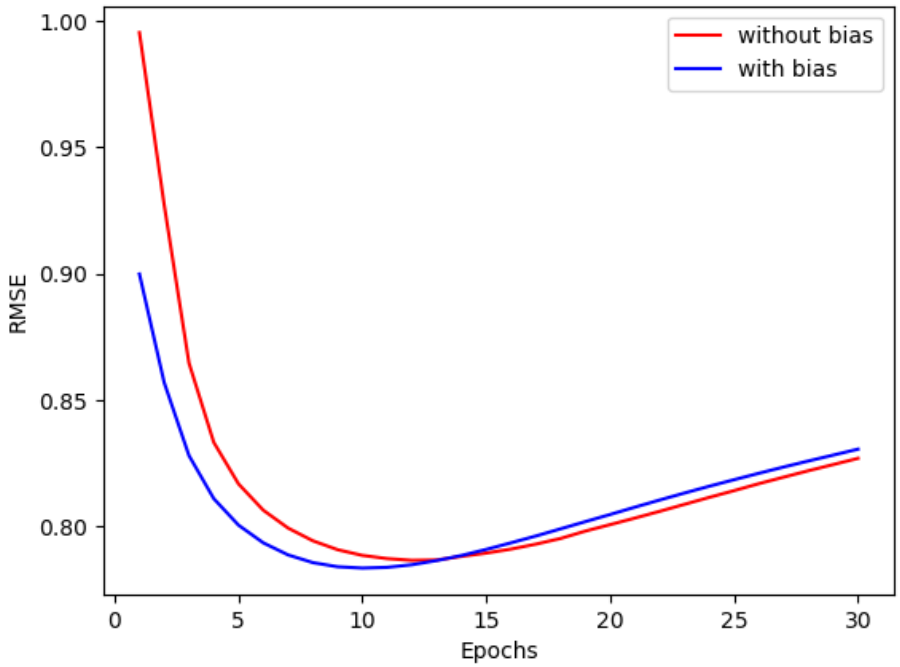
2. (1%)比較不同的 latent dimension 的結果。

本題使用上題所描述的 MF Model，但是並沒有做 normalization，針對 model 內 embedding layer 的 latent dimension 做比較，測試了不同的 latent dimension 如下表所示。經過實驗後，我的 MF model 在 latent dimension 為 50 時能於 kaggle 上得到最低的 RMSE，因此以下的題目皆設定 model 的 latent dimension 為 50。

latent dimension	5	10	15	20	30	50	60
Kaggle public RMSE	0.971	1.399	0.926	0.947	0.994	0.906	0.934

3. (1%)比較有無 bias 的結果。

使用第一題所描述的 MF model，對 rating 做完 normalization 後，將 UserID、MovieID 向量各自的 bias 加入 MF model 比較於 validation data 上的 RMSE 差異。如下圖所示，加入 bias 後能使 RMSE 進一步的下降。



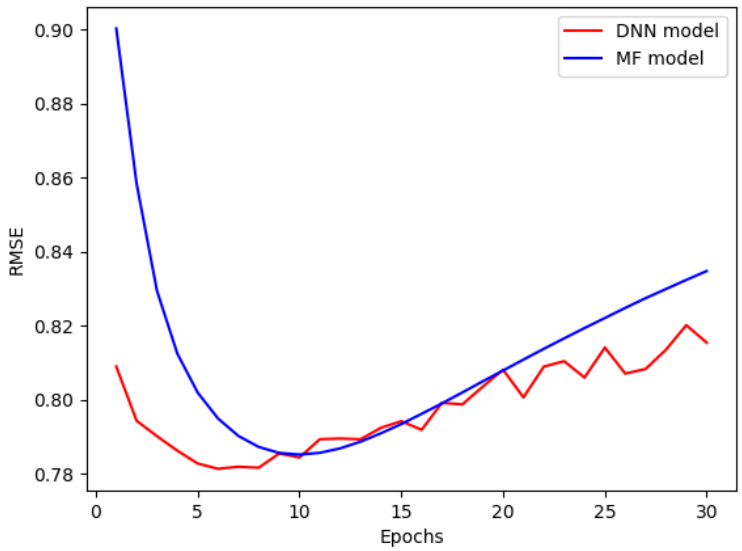
4. (1%)請試著用 DNN 來解決這個問題，並且說明實做的方法(方法不限)。並比較 MF 和 NN 的結果，討論結果的差異。

本題所建立的 DNN model 一樣將利用 UserID、MovieID 當成 input，透過 embedding 的方式建立各自的向量後，利用 keras 的 Concatenate 將兩向量接在一起後經過兩層 NN 分別含有 256、128 個 neuron，activation function 為 relu，每層皆使用 dropout(0.2)，最後接上一個 output layer 僅含有一個 neuron 來進行 regression 的預測，loss function 為 MSE，optimizer 為 Adam，使用 10%的 training data 作為 validation data。

將 training data 做完第一題的 normalization 後與 MF model 比較 validation data 的 RMSE，可以觀察到 DNN 收斂的比較早且能得到較低的 RMSE。

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 60)	362460	
embedding_2 (Embedding)	(None, 1, 60)	237180	
flatten_1 (Flatten)	(None, 60)	0	
flatten_2 (Flatten)	(None, 60)	0	
concatenate_1 (Concatenate)	(None, 120)	0	
dense_1 (Dense)	(None, 256)	30976	
dropout_1 (Dropout)	(None, 256)	0	
dense_2 (Dense)	(None, 128)	32896	
dropout_2 (Dropout)	(None, 128)	0	
dense_3 (Dense)	(None, 1)	129	

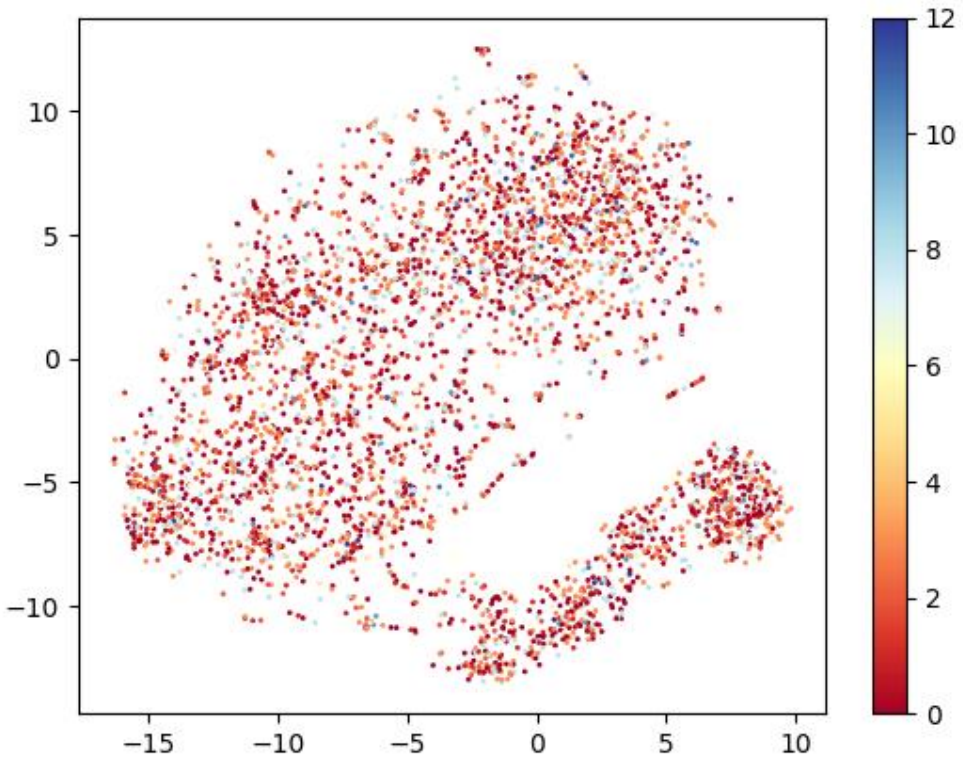
Total params: 663,641.0
Trainable params: 663,641.0
Non-trainable params: 0.0



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

   此題依照助教時間時所提供的建議，將 MovieID 內相似的類別歸為一類，Drama 與 Musical 算為同一類，Crime、Thriller 與 Horror 算為同一類，Animation、Children’s 與 Adbventure 歸為同一類，而剩餘的類別則各自獨立維持本來的類別。

   將 MF model 的 movie embedding layer 取出後，使用 sklearn 的 tsne 來將之降維至 2 維並依照上面的類別畫出下圖。可觀察到橘點與紅點常常聚在一塊，而他們各自代表了 Animation、Children’s、Adbventure 與 Fantasy 類別，從提供的 movie.csv 中可以觀察到此份資料有許多 data 同時具備此兩類別，所以降維後才會聚在一塊。



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。

本題仿照第四題的 DNN model，將 user.csv 提供的 gender、age、occupation 等資訊建立出相對應的 embedding layer 至 DNN model 內，與原先的 UserID、MovieID 利用 keras 的 Concatenate 在一起，接著接上內含 128 個 neuron 的 full-connected layer，activation function 為 relu，最後接上含有一個 neuron 的 output layer，optimizer 使用 Adam，loss function 為 mse。

結果如下，與原先的 MF model 相比較，validation data 在 RMSE 一開始就非常的低，相對於學習速度較慢，與 MF model 相同差不多在 15 個 epoch 後收斂。

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
input_3 (InputLayer)	(None, 1)	0	
input_4 (InputLayer)	(None, 1)	0	
input_5 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 50)	302050	
embedding_2 (Embedding)	(None, 1, 50)	197650	
embedding_3 (Embedding)	(None, 1, 50)	100	
embedding_4 (Embedding)	(None, 1, 50)	2900	
embedding_5 (Embedding)	(None, 1, 50)	1050	
flatten_1 (Flatten)	(None, 50)	0	
flatten_2 (Flatten)	(None, 50)	0	
flatten_3 (Flatten)	(None, 50)	0	
flatten_4 (Flatten)	(None, 50)	0	
flatten_5 (Flatten)	(None, 50)	0	
concatenate_1 (Concatenate)	(None, 250)	0	
dropout_1 (Dropout)	(None, 250)	0	
dense_1 (Dense)	(None, 128)	32128	
dropout_2 (Dropout)	(None, 128)	0	
dense_2 (Dense)	(None, 1)	129	

Total params: 536,007.0
Trainable params: 536,007.0
Non-trainable params: 0.0