

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

採用 embedding_dim=150, validation=0.1。

Normalize 的部分，是對「Rating」的欄位進行，並將 normalize 的結果存取，在 testing 的時候，再將 predict 出來的結果*std+mean 還原到 1~5 的分數。

Batch Normalize 則是加在 embedding layer 後面。

由於 RMSE 是將兩個值的差距做運算，因此在 normalize 之後，此值就會與原本的值有了壓縮後的變化，並不代表在壓縮前的 RMSE 也是這麼小，不能做參考。

	<i>Batch normalized</i>	<i>Trained epochs</i>	<i>Validation RMSE</i>	<i>Kaggle public</i>
<i>non-normalize</i>	X	7	0.858611	0.86037
	O	11	0.928736	0.93267
<i>normalize</i>	X	3	0.767135	0.86060
	O	9	0.783024	0.88133

2. (1%)比較不同的 latent dimension 的結果。

以下皆是 User 與 Movie 都採用 dropout=0.3，batch size=1024，並且沒有 normalize 的參數。

擔心太快 overfit，將 batch size 開得很大，發現小的 embedding dimension 很慢才走到收斂的 minimum，成績也沒有開大的 dim 來的好，或許是能夠還原出 movie : user 表的陣列維度較大。

Latent dimension	epochs	Validation RMSE	Validation MSE
20	63	0.861593	0.743685
50	30	0.856244	0.734204
100	17	0.857909	0.737266
150	13	0.854658	0.731921
200	9	0.858745	0.739262

3. (1%)比較有無 bias 的結果。

採用 embedding_dim=150, validation=0.1, dropout=0.3。發現沒有 bias 的效果稍微遜色一些，代表或許有些電影平均的評分都較高/低，或是某些使用者平均給分都偏高/低。

Bias	Validation RMSE	Validation MSE
O	0.861593	0.743685
X	0.862619	0.745700

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。

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 150)	906000	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 150)	592800	input_2[0][0]
reshape_1 (Reshape)	(None, 150)	0	embedding_1[0][0]
reshape_2 (Reshape)	(None, 150)	0	embedding_2[0][0]
dropout_1 (Dropout)	(None, 150)	0	reshape_1[0][0]
dropout_2 (Dropout)	(None, 150)	0	reshape_2[0][0]
concatenate_1 (Concatenate)	(None, 300)	0	dropout_1[0][0]; dropout_2[0][0]
dense_1 (Dense)	(None, 64)	19264	concatenate_1[0][0]
dropout_3 (Dropout)	(None, 64)	0	dense_1[0][0]
dense_2 (Dense)	(None, 64)	4160	dropout_3[0][0]
dropout_4 (Dropout)	(None, 64)	0	dense_2[0][0]
dense_3 (Dense)	(None, 64)	4160	dropout_4[0][0]
dropout_5 (Dropout)	(None, 64)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	65	dropout_5[0][0]

將 User 與 Movie 的 embedding layer 在下一層 Concatenate，之後接一串 NN 來做成 model，如左圖所示，當中測試數據是測試之後較佳的配置。

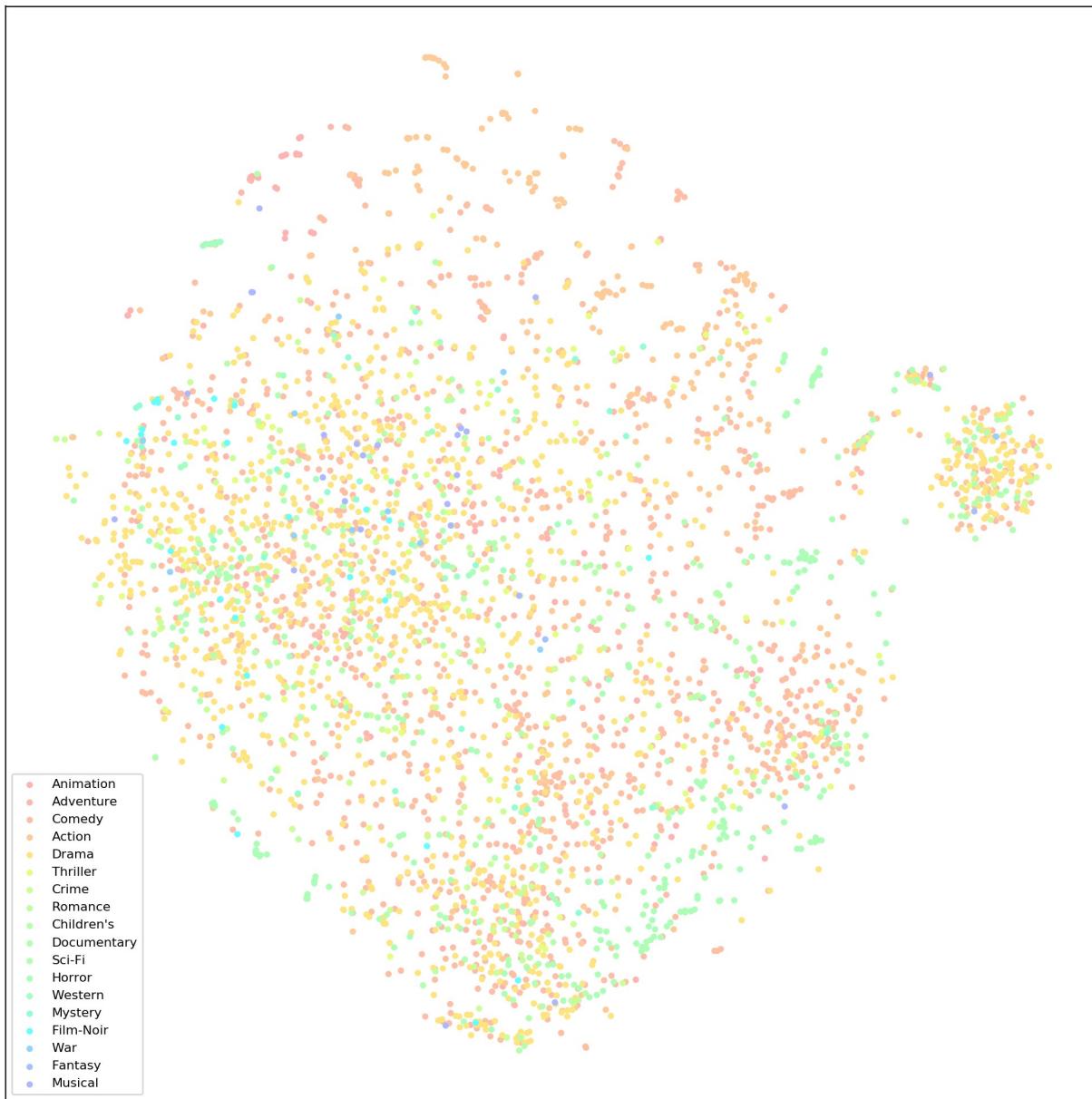
發現 MF 的效果較佳，訓練的 epoch 數也比較多(比較晚 overfit)，可能是我的 MF 參數調的較佳，較能還原出原本的對應關係，DNN 則是參數調不好，使他獲取的 feature 無法準確預測分數。

	Validation RMSE	Validation MSE
DNN	0.885913	0.785896
MF	0.854658	0.731921

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

我將 embedding_dim=150 有 bias 沒 normalize 的 model，取出第四層的 movie_embedding 使用 tsne 的 pca 降維，並使用 movies.csv 標註 category。繪出的分布圖如下：

可以發現 animation、adventure、children 分布在各處，其他類型都有較偏向分佈的區域。



(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。

我採用了 users.csv 中其他的 feature: **gender, age, occupation**，作為輸入。embedding layer 出來之後與 movie concatenate，做兩層的 DNN (units=64, 32) activation=relu，最好的結果是 validation rmse=0.9782，很不理想，結果去查資料發現，age=1 的資料居然高達 24474 筆，可能是一部分使得只採用 user 的資料效果不好的原因，

變更後的 model 結構如下：

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 4)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 4, 15)	90600	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 15)	59280	input_2[0][0]
reshape_1 (Reshape)	(None, 60)	0	embedding_1[0][0]
reshape_2 (Reshape)	(None, 15)	0	embedding_2[0][0]
dropout_1 (Dropout)	(None, 60)	0	reshape_1[0][0]
dropout_2 (Dropout)	(None, 15)	0	reshape_2[0][0]
concatenate_1 (Concatenate)	(None, 75)	0	dropout_1[0][0] dropout_2[0][0]
dense_1 (Dense)	(None, 64)	4864	concatenate_1[0][0]
dropout_3 (Dropout)	(None, 64)	0	dense_1[0][0]
dense_2 (Dense)	(None, 32)	2080	dropout_3[0][0]
dropout_4 (Dropout)	(None, 32)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	33	dropout_4[0][0]

Total params: 156,857
Trainable params: 156,857
Non-trainable params: 0