學號：B03902084 系級： 資工四 姓名：王藝霖

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

使用簡單的 model，來比較有無 normalize 的結果。

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 32)	193344	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 32)	126528	input_2[0][0]
flatten_1 (Flatten)	(None, 32)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 32)	0	embedding_2[0][0]
merge_1 (Merge)	(None, 1)	0	flatten_1[0][0] flatten_2[0][0]

Total params: 319,872
Trainable params: 319,872
Non-trainable params: 0

有 normalize validation mse: 0.934

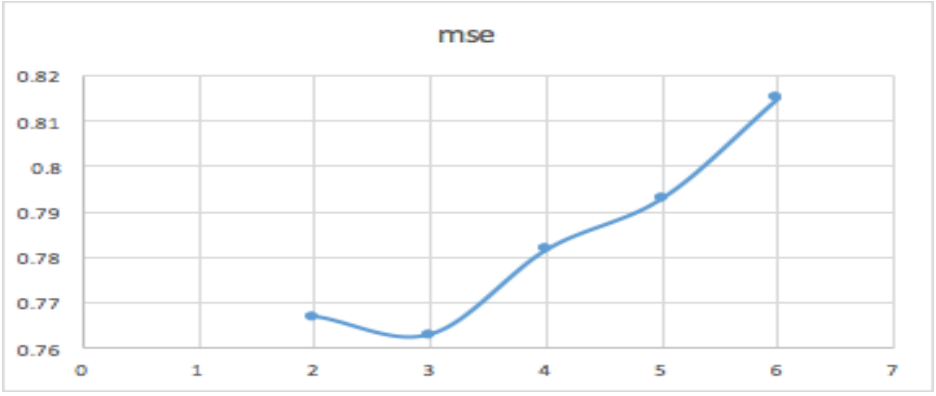
無 normalize validation mse: 0.896

normalize 的方法為 y = (y - mean) / std，用的是 numpy 提供的 std 和 mean，在 predict validation 時用的是 y_pred = y_pred * std + mean

2. (1%)比較不同的latent dimension的結果。

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 4)	24164	input_1[0][0]
embedding_3 (Embedding)	(None, 1, 4)	15812	input_2[0][0]
flatten_1 (Flatten)	(None, 4)	0	embedding_1[0][0]
flatten_3 (Flatten)	(None, 4)	0	embedding_3[0][0]
embedding_2 (Embedding)	(None, 1, 1)	6041	input_1[0][0]
embedding_4 (Embedding)	(None, 1, 1)	3952	input_2[0][0]
dot_1 (Dot)	(None, 1)	0	flatten_1[0][0] flatten_3[0][0]
flatten_2 (Flatten)	(None, 1)	0	embedding_2[0][0]
flatten_4 (Flatten)	(None, 1)	0	embedding_4[0][0]
add_1 (Add)	(None, 1)	0	dot_1[0][0] flatten_2[0][0] flatten_4[0][0]

Total params: 49,969
Trainable params: 49,969
Non-trainable params: 0



x 軸為 log2(dim)，y 軸為 mse

使用有 bias 的 model 來比較不同 latent dimension 的結果，發現其中在 dim = 2^3 = 8 時最好的結果 mse = 0.763

3. (1%)比較有無bias的結果。

使用上一題最好的結果 dim = 8，用同樣的架構（除了把 bias 拿掉之外），來比較有無 bias 的結果。

無 validation mse: 0.776

有validation mse: 0.763

有 bias 的結果比較好，除了多了能夠調整的參數之外，還有比較合理的原因，例如有些使用者傾向把所有的評分都評的比較低，在這種狀況下沒有 bias 的話比較沒辦法處理。

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, 8)	48328	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 8)	31624	input_2[0][0]
flatten_1 (Flatten)	(None, 8)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 8)	0	embedding_2[0][0]
concatenate_1 (Concatenate)	(None, 16)	0	flatten_1[0][0] flatten_2[0][0]
dense_1 (Dense)	(None, 128)	2176	concatenate_1[0][0]
dense_2 (Dense)	(None, 64)	8256	dense_1[0][0]
dense_3 (Dense)	(None, 1)	65	dense_2[0][0]

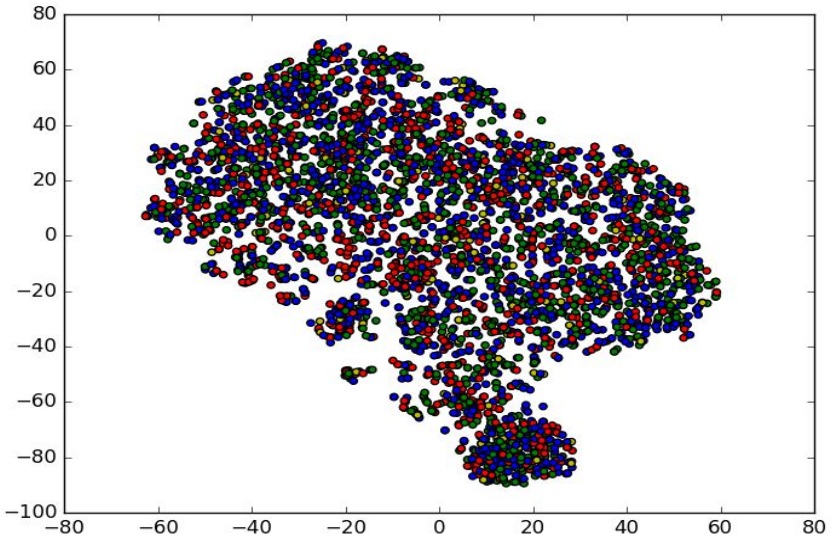
Total params: 90,449
Trainable params: 90,449
Non-trainable params: 0

dnn validation mse: 0.751

mf validation mse: 0.763

dnn 的方法是把 user 和 movie 的 embedding concate 起來，再丟到 dnn 中，得到最後的結果。dnn 的結果比 mf 稍微好一點，因為 mf 是把學到的 embedding 直接 dot 起來就當作輸出，但 dnn 除了把 embedding concate 起來之外，更多了兩層 hidden layer 的彈性，比較能夠調整 model，進而得到好一點的結果。

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

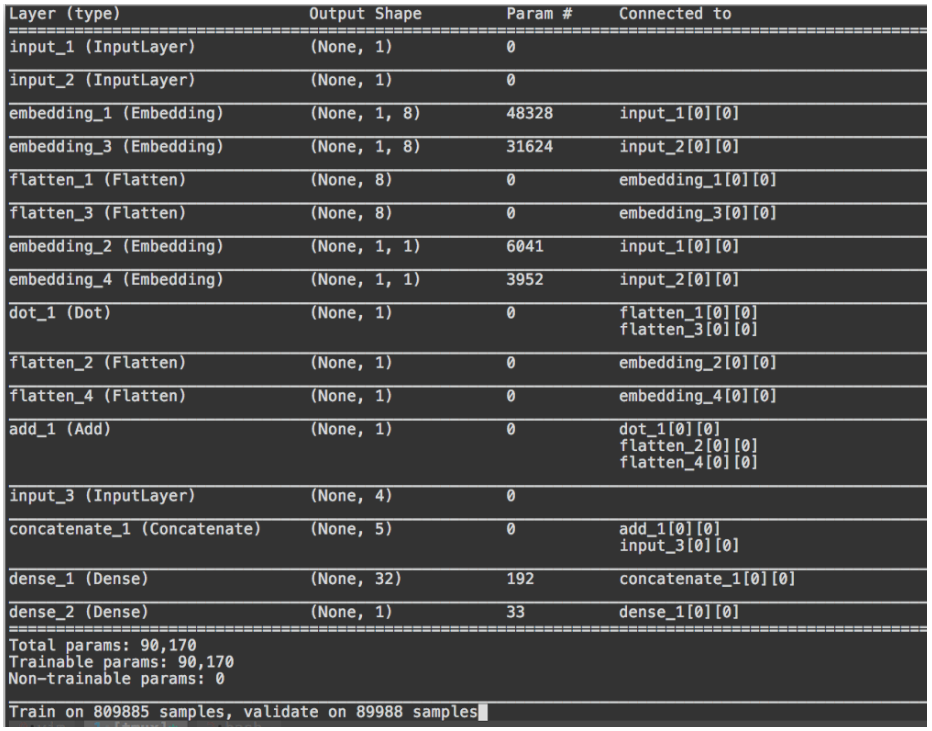



```
mapping = {
    'Drama': 'g',
    'Musical': 'g',
    'Romance': 'g',
    'Fantasy': 'g',
    'Thriller': 'r',
    'Horror': 'r',
    'Action': 'r',
    'Crime': 'r',
    'Western': 'r',
    'War': 'r',
    'Adventure': 'b',
    'Sci-Fi': 'b',
    'Animation': 'b',
    "Children's": 'b',
    'Comedy': 'b',
    'Mystery': 'b',
    'Film-Noir': 'y',
    'Documentary': 'y',
}
```

左邊是將 movie category 分類的方法。在圖正中間有一塊紅色的比例佔比較高，可能是因為我的 model 還學得不夠好，所以沒辦法把其他紅色都分在同一個位置，或是也有可能是因為我的 tag 分類也不是很精確，沒辦法抓到很好的分佈，也有另一個可能是 tsne 降維時落掉了一些分類的資訊，因此這張畫出來的圖並沒有非常明顯的分群，但若只看藍色和紅色，除了底下一大塊分不開的區域之外，是有一點藍色左上紅色中間的趨勢。

另外，因為最後的結果看的是 movie 和 user embedding dot 後的結果，因此若是光看 movie 的 embedding 的話，可能不會達到很好的分類結果。再者因為用的是 mse 的關係，model 會比較趨向 predict 中間值，而不是大膽的 predict 1 或 5

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。



Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 8)	48328	input_1[0][0]
embedding_3 (Embedding)	(None, 1, 8)	31624	input_2[0][0]
flatten_1 (Flatten)	(None, 8)	0	embedding_1[0][0]
flatten_3 (Flatten)	(None, 8)	0	embedding_3[0][0]
embedding_2 (Embedding)	(None, 1, 1)	6041	input_1[0][0]
embedding_4 (Embedding)	(None, 1, 1)	3952	input_2[0][0]
dot_1 (Dot)	(None, 1)	0	flatten_1[0][0] flatten_3[0][0]
flatten_2 (Flatten)	(None, 1)	0	embedding_2[0][0]
flatten_4 (Flatten)	(None, 1)	0	embedding_4[0][0]
add_1 (Add)	(None, 1)	0	dot_1[0][0] flatten_2[0][0] flatten_4[0][0]
input_3 (InputLayer)	(None, 4)	0	
concatenate_1 (Concatenate)	(None, 5)	0	add_1[0][0] input_3[0][0]
dense_1 (Dense)	(None, 32)	192	concatenate_1[0][0]
dense_2 (Dense)	(None, 1)	33	dense_1[0][0]

Total params: 90,170
Trainable params: 90,170
Non-trainable params: 0

Train on 809885 samples, validate on 89988 samples

有使用 validation mse: 0.746

無使用 validation mse: 0.763

我使用其他 feature 的方式為，把 mf 的結果，和其他的的 feature（經過 normalize）一起當作另一層 Dense layer 的 input，然後才得到最後的結果。結果如以上所列，有使用額外 feature 的結果有進步，但可能可以經由更深的 hidden layer 等方法達到更好的結果。