

- Normalize 的方式為算所有 ratings 的 mean 跟 standard deviation，然後利用 $\text{rating} = (\text{rating} - \text{mean}) / \text{std}$ 的方式 normalize。

Normalize 的 Kaggle score 為(0.88774, 0.88852)，對比(0.86882, 0.86534)，沒有 perform 上述 normalization 結果比較好。或許單純減掉 min 在除以(max-min)讓 normalize 完的 value 沒有負值會稍微好一些。

- 使用如 Q4 的 MF 結構，唯將 latent dimension 調整為 50 及 100，分別在 Kaggle 上的表現為(0.86640, 0.86631)及(0.87162, 0.87214)，對比 200 的結果為(0.88138, 0.88262)，似乎 50 有較好的結果，後來跟同學討論後覺得挺有道理，因為 word embedding 這麼多字也僅 mapping 到 250 維左右(當然跟 resource limitation 也有關係)，3000 多部電影跟 6000 多個 user 也許太高維反而資訊會發散。
- 若使用如 Q4 所示的 MF 結構有沒加入 bias 的結果在 Kaggle 上為(0.91971, 0.92143)，對比有 bias 的為(0.88138, 0.88262)，會發現加入 bias 會使結果提升許多，直觀解釋就如同作業投影片所述，某些 user 可能普遍給的分數會比較好。

在 training 的過程也會發現，沒有 bias 會很難 train，就算 40 多個 epoch 也僅能達到上述水準，但有 bias 的話大概在 30 左右就有不錯的結果了。

4. MF 的結構如下

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 1)	0	
input_4 (InputLayer)	(None, 1)	0	
embedding_7 (Embedding)	(None, 1, 200)	1208000	input_3[0][0]
embedding_8 (Embedding)	(None, 1, 200)	790400	input_4[0][0]
flatten_7 (Flatten)	(None, 200)	0	embedding_7[0][0]
flatten_8 (Flatten)	(None, 200)	0	embedding_8[0][0]
embedding_10 (Embedding)	(None, 1, 1)	6040	input_3[0][0]
dot_1 (Dot)	(None, 1)	0	flatten_7[0][0] flatten_8[0][0]
flatten_10 (Flatten)	(None, 1)	0	embedding_10[0][0]
embedding_9 (Embedding)	(None, 1, 1)	3952	input_4[0][0]
add_1 (Add)	(None, 1)	0	dot_1[0][0] flatten_10[0][0]
flatten_9 (Flatten)	(None, 1)	0	embedding_9[0][0]
add_2 (Add)	(None, 1)	0	add_1[0][0] flatten_9[0][0]

Total params: 2,008,392
Trainable params: 2,008,392
Non-trainable params: 0

將 user 以及 movie 分別 embed 到 200 維的空間並加上 bias，Dropout=0.5
 NN 的結構如下

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 200)	1208000	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 200)	776600	input_2[0][0]
embedding_5 (Embedding)	(None, 1, 34)	205360	input_1[0][0]
embedding_6 (Embedding)	(None, 1, 19)	73777	input_2[0][0]
flatten_1 (Flatten)	(None, 200)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 200)	0	embedding_2[0][0]
flatten_5 (Flatten)	(None, 34)	0	embedding_5[0][0]
flatten_6 (Flatten)	(None, 19)	0	embedding_6[0][0]
concatenate_1 (Concatenate)	(None, 453)	0	flatten_1[0][0] flatten_2[0][0] flatten_5[0][0] flatten_6[0][0]
dense_1 (Dense)	(None, 128)	58112	concatenate_1[0][0]
dense_2 (Dense)	(None, 64)	8256	dense_1[0][0]
dropout_1 (Dropout)	(None, 64)	0	dense_2[0][0]
dense_3 (Dense)	(None, 32)	2080	dropout_1[0][0]
dropout_2 (Dropout)	(None, 32)	0	dense_3[0][0]
dense_4 (Dense)	(None, 10)	330	dropout_2[0][0]
dropout_3 (Dropout)	(None, 10)	0	dense_4[0][0]
dense_5 (Dense)	(None, 1)	11	dropout_3[0][0]
Total params: 2,332,526			
Trainable params: 2,053,389			
Non-trainable params: 279,137			

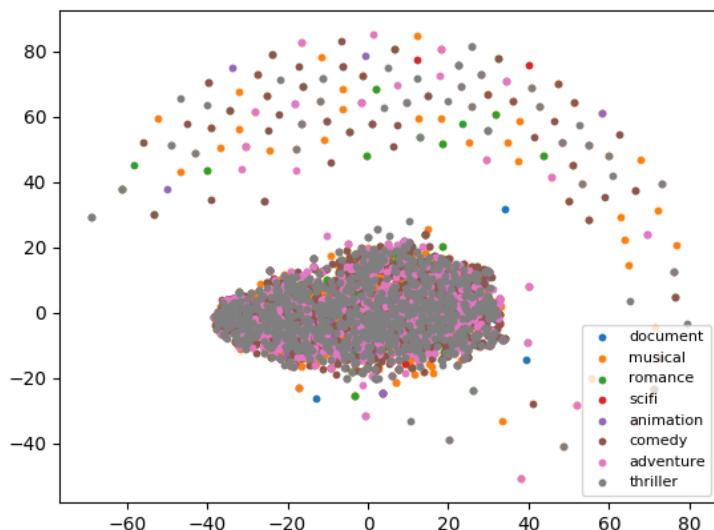
主要就是將 users.csv 以及 movies.csv 裡頭的 information 加進來。
 包含 User age, job, gender, postal code 轉換成美國的 10 個大區域 00, 01...到 09，
 另外 movie 的 year 以及 genre，並將 categorical 的 feature 弄成 one-hot
 encoding (除了一部電影屬於很多類別外)，而 numerical 則做一般的
 standardization。Dropout=0.5

MF 在 Kaggle 上的 private 以及 public 分數為 0.88138, 0.88262，NN 單一 model
 則為 0.86882, 0.86534。但可能因為 NN 有加入了額外的資訊，因為若單純用同
 樣的設定不加入資訊的話，NN 是有可能比 MF 來得不好。

5. 將所有類型分成 8 類，並因為 Western 太過廣泛，所以將之移除。

分法如下：

(Adventure, Action, War), (Musical, Drama), (Thriller, Mystery, Crime, Horror, Film-Noir), Comedy, Documentary, Romance, (Animation, Children's, Fantasy), Sci-Fi



感覺並沒有什麼 insight，可能是因為一部電影可能屬於好幾個類別 (multi-label)，或是 model 還不夠好，也有可能分法錯誤導致擠成一團。

6. 如 Q4 所概述，將所得到的 user 及 movie info vectors 跟 user 以及 movie 的 200 維 embedding concatenate 之後得到一個 453 維的 vector 在過數層 fc。
加入後的結果如 Q4 所示 0.86882, 0.86534，沒加的話 train 不大起來，validation loss 都會高個 0.1 不等。