學號：B03901161　系級：　電機四　姓名：楊耀程

1. (1%)請比較有無 normalize(rating)的差別。並說明如何 normalize.
(collaborator: b03901001 廖宜倫，b03901165 謝世暐，b03901096 周晁德)
如何 normalize:
在 training 時計算(train data 的)ratings 的平均與標準差，減去平均後再除以標準差，training data 的 rating 平均為 3.581，標準差為 1.116。
將這兩個統計數值記下來，testing 的時候，將 model output 的 predict value(預測的 rating)時，在乘以先前求得的標準差，還有加回平均，才是最終的 predict value。

In training

```
mu_ratings = np.mean(Ratings)
sig_ratings = np.std(Ratings)
print('mu: ',mu_ratings)
print('sigma: ',sig_ratings)
Ratings = (Ratings - mu_ratings)/sig_ratings
print('Ratings:', Ratings, ', shape =', Ratings.shape)
```

In testing

```
mu_ratings =  3.58171208604
sig_ratings = 1.11689766115

my_pred_value = my_pred_value*sig_ratings
my_pred_value = my_pred_value + mu_ratings
```

	沒有 normalize	有 normalize
kaggle public score	**0.8559**	**0.86**

討論: 經由上述的方式對 ranking 做 normalize 之後，kaggle 上的結果變爛了一些，但是整體而言影響沒有很大，可能是因為此處 ranking 的數字本來就不大，介於 1~5 之間，標準差也不大，因此做了 normalize 之後，數值的量值也沒有很大的改變。在這樣的情況下差異不大，是符合預期的結果。

2. (1%)比較不同的 latent dimension 的結果。
(collaborator: b03901001 廖宜倫，b03901165 謝世暐，b03901096 周晁德)

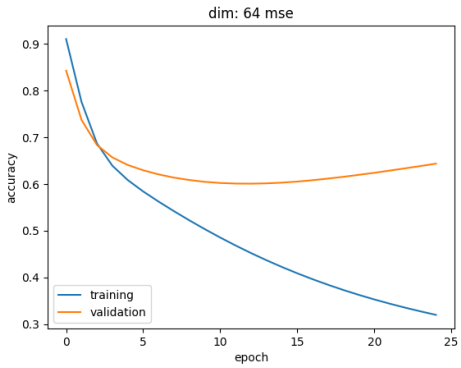
嘗試多種不同 latent dimension 的結果如下表，有經過 normalization

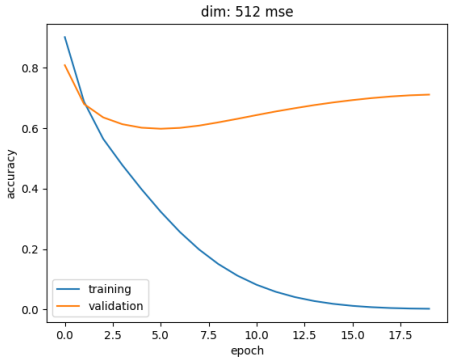
latent dim	8	32	64	128	256	512
mse on validation (with normalization	0.6048	0.6019	0.6008	0.5986	0.5988	0.5985
最小值出現的epoch數	42	16	13	10	8	5

討論:

可以發現有 latent dimension = 128 有最好的表現，其他 latent dimension 去跑，validation 的 mse 也不會太差。為公平比較，此處都是用 adamax +default learning rate 去跑，batch size 皆為 512，可以發現 latent dimension 越大，收斂的越快，也更容易 overfit。Latent dimension = 8 的跑了 42 個 epochs 才達到 validation 最低點，而 latent dimension = 512 的跑了 5 個 epochs 就達到最低點了。

下表為 latent dimension = 64 和 latent dimension= 512 的訓練過程作圖，可以更清楚看到他們訓練時收斂速度的差別。





3. (1%)比較有無 bias 的結果。

(collaborator: b03901001 廖宜倫，b03901165 謝世暐，b03901096 周晁德)

加上 Bias 的方式如下，除了本來的 MF 結構之外，users id and movies id 分別通過一維的 embedding layer (可以想成每個 id 給他一個 scalar 的值)，flatten 後，加上本來的 dot 後的結果。

加上 Bias 的 code 如下

```
dot_result = Dot(axes = 1, normalize=False)([flat_users,flat_items])
##################add bias
bias_users = Embedding(n_users, 1)(input_users)
flat_bias_users = Flatten()(bias_users)
bias_items = Embedding(m_items, 1)(input_items)
flat_bias_items = Flatten()(bias_items)

dot_result = Add()([dot_result, flat_bias_users, flat_bias_items])
```

	沒有 bias	有 bias
kaggle public score	**0.8673**	**0.8559**

討論: 有加上 bias 後，準確度有提高，可見加上 bias 是有用的，我的理解是 bias 提供了更多的資訊，有分別考慮到每個 id，單獨的情況，考量進去這樣的情況後，準確度就更高了一些。

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。

(collaborator: b03901001 廖宜倫，b03901165 謝世暐，b03901096 周晁德)

DNN 實作的方法:

兩個 input(user id, movie id)，分別通過各自的 embedding layer，與 MF 不同的是此處不是做 dot 後直接 output，而是做 concatenate(串接)起來，再通過 fully connected 後得到 output，Activation 使用的是 relu。(因為這邊的 ratings 沒做標準化，都是 1-5 的正數，所以用 relu 很合理)

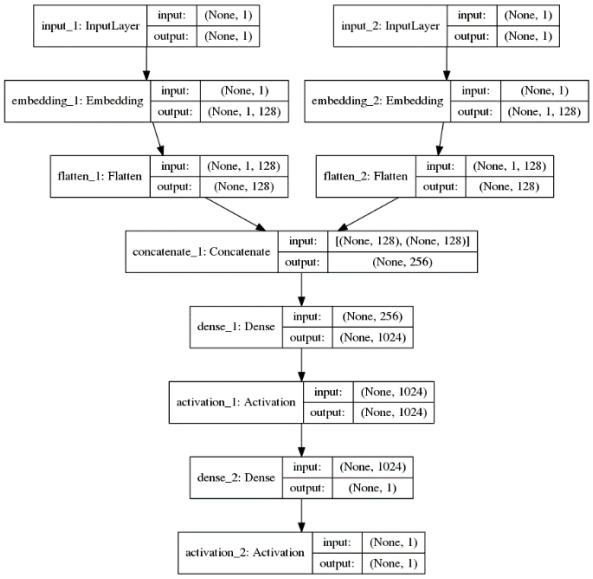
Latent dimension = 128

Model 架構如下，嘗試多種架構後，發現一層的 fully connected(如下圖)的表現較好，如果架兩層 fully connected，甚至三層 fully connected，都會 overfit，training error 很小但是 validation 誤差很大。

NN model

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 128)	773120	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 128)	505856	input_2[0][0]
flatten_1 (Flatten)	(None, 128)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 128)	0	embedding_2[0][0]
concatenate_1 (Concatenate)	(None, 256)	0	flatten_1[0][0] flatten_2[0][0]
dense_1 (Dense)	(None, 1024)	263168	concatenate_1[0][0]
activation_1 (Activation)	(None, 1024)	0	dense_1[0][0]
dense_2 (Dense)	(None, 1)	1025	activation_1[0][0]
activation_2 (Activation)	(None, 1)	0	dense_2[0][0]

Total params: 1,543,169
Trainable params: 1,543,169
Non-trainable params: 0

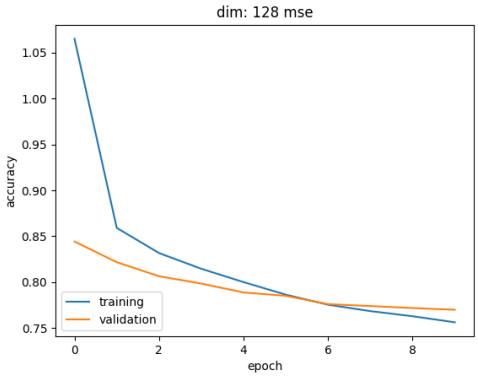


使用 Adamax，default learning rate (0.002)

Batch size = 512

800000 筆 data 拿來 train，剩下 90000 左右做 validation

訓練過程如下(此圖沒有 fine tune)



MF model: (為了和 NN 比較而已，為求精簡，此處不詳述訓練過程與參數)

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1)	0	
input_2 (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 128)	773120	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 128)	505856	input_2[0][0]
flatten_1 (Flatten)	(None, 128)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 128)	0	embedding_2[0][0]
embedding_3 (Embedding)	(None, 1, 1)	6040	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_2[0][0]
flatten_3 (Flatten)	(None, 1)	0	embedding_3[0][0]
flatten_4 (Flatten)	(None, 1)	0	embedding_4[0][0]
add_1 (Add)	(None, 1)	0	dot_1[0][0] flatten_3[0][0] flatten_4[0][0]

Total params: 1,288,968
Trainable params: 1,288,968
Non-trainable params: 0

結果的差異如下表:

	NN	MF
kaggle public score	**0.85738**	**0.8559**

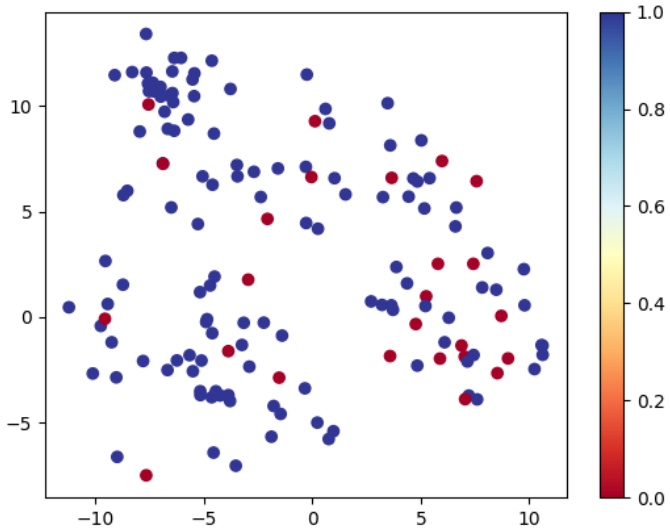
討論: 兩者皆有經過 fine tune (fine tune 時有 dropout)，想辦法把兩種方法最好的表現拿出來，我的結果是在 kaggle 上 MF 還是好一些，但是兩者都不錯，也都過了 Strong baseline。再次強調這邊的 NN 是一層 fully connected 的結果，兩層 fully connected 在 kaggle 上的分數是 0.89 多，NN 只要層數多了很容易 overfit。NN 和 MF 表現接近我覺得蠻合理的，前面 embedding layer 的結構，後面 MF 是 Dot，NN 則是 Concatenate。助教有提到，NN 的部份，只要調整得當，會比 MF 來的更好，但是我這次的經驗是 NN 參數並沒有想像中好調整，層數一多也很容易 overfit，我怎麼調都還是比用 MF 差一些(不過在 BONUS 的部分我用 NN+額外 feature 有做出超越 MF 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。

(collaborator: b03901001 廖宜倫，b03901165 謝世暐，b03901096 周晁德)

對於每個電影，很多電影類別都是不只一種，為了方便，有多種的話我只取第一個作為他的類別。

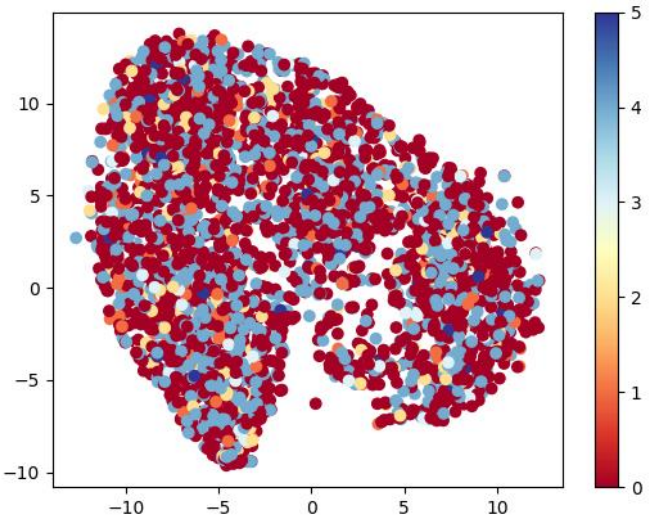
首先是只有畫兩個 class 的部分，我只取 Musical 和 Documentary，其他的資料點就不畫。希望少一點 class 能夠在圖中分比較開。取 Musical 和 Documentary 的原因是我覺得音樂劇很熱鬧，而記錄片通常沉悶，兩個性質剛好相反。



從圖中可以看到，紅色的點是 Musical，藍色的點是 Documentary，紅色的點較集中在畫面的右下角，藍色的點則較集中在左上角，這兩種類別的 embedding 降維確實有分開。

再來是畫多個 class 的部分，
我的分類法如下，分成六大類(全部的資料點都有畫)

```
class0 =['Drama','Romance','Comedy']
class1 =["Children's", 'Animation','Fantasy']
class2 =['Sci-Fi','Adventure','Mystery']
class3 =['Documentary','War']
class4 =['Crime','Thriller','Action','Horror','Western']
class5 =['Musical','Film-Noir']
```



從圖中，點都混在一起了，實在很難從畫面中看出怎麼分開。可能是我電影的 class 分的不夠好，或是 embedding matrix 沒學好，可能 embedding 學到的是: 相較於 movie 的類別，user 的過往給分，以及 movie 本身的評價，比類別更加重要。

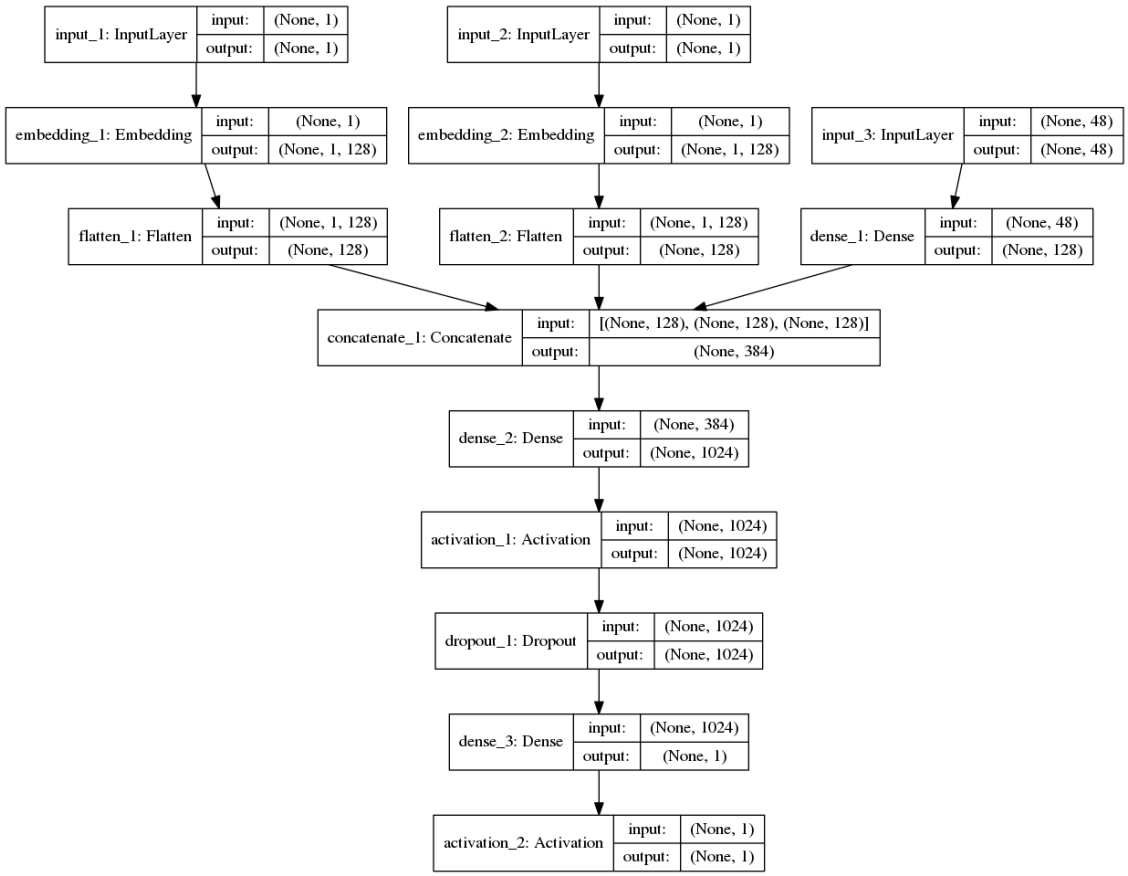
6. (BONUS)(1%)試著使用除了 rating 以外的 feature, 並說明你的作法和結果，結果好壞不會影響評分。

(collaborator:)

除了 rating 之外有使用的 feature:
Movies 中: 使用了 Genres
Users 中: 使用了 Gender, Age, Occupation
花了蠻多時間 parse 這些特徵，上述只有年齡是 continuous 的(但也是分為七類)，我認為七類不算太多，而且小於 18 歲的都歸類於 under 18，還是用 one

hot coding 比較合適，可能比較容易區別出兒童喜愛的電影(我個人認為大人的喜好跟年紀比較無關)，上述其他的特徵也都換成 one hot coding。至於 movie genres 的部分，有些有 multiclass 的 label，為避免重複計算，如果有多重 genre 標籤我就只取第一個。
Gender 有兩類，Age 有七類，Occupation 有二十一類，Genres 有十八類(只看第一個標籤的話)，做成 one hot coding 共有 2+7+21+18 = 48 維
因此我每個 user id, movie id 的組合，會搭配一個 48 維的 vector。這個 vector 會搭配 user id, movie id 一起作為 model 的輸入。

Model 架構如下:



Model 整體上是 NN based，除了原本的 embedding layer 之外，額外的特徵通過一層 Dense layer 後，與兩個 embedding layer 的 latent vector 一起串接在一起。

結果如下:

	有 feature(NN)	沒有 feature(NN)	沒有 feature(MF)
kaggle public score	**0.85421**	**0.85738**	**0.8559**

在 kaggle public 上，此 model 有很好的表現，可以看到比原本的 NN based(沒 feature)再提升了一些，準確度甚至超越了 MF based。可以證明此實作加上了 feature 後，對於 rating 的預測是有正面的影響的。