學號：B02705027 系級： 資管四 姓名：陳信豪

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

切同樣的 train/valid dataset，並運用同樣的 mf model 來跑。

差別只在於有無對 ratings 做 normalization。

patience = 10 epochs = 1000

batch\_size = 512 optimizer = 'adam'

normalization 的做法是對 y\_train (要訓練的 ratings label)

算平均和標準差，之後將 y\_train 和 y\_valid 減掉平均，再除以標準差。

訓練好的 model 預測出來的結果 y\_test，也是同樣處理。

*mean = np.mean(y\_train)*

*std = np.std(y\_train)*

*y\_train = y\_train - mean*

*y\_trian = y\_train/std*

*y\_valid = y\_valid - mean*

*y\_valid = y\_valid/std*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Train RMSE | Valid RMSE | Kaggle Public RMSE | Kaggle  Private RMSE |
| 有  normalize | 0.8780 | 0.8752 | 0.87344 | 0.87876 |
| 沒有  normalize | 0.7792 | 0.8656 | 0.86642 | 0.86933 |

從結果來看，做 normalization 無益於增加 model performance。

1. (1%)比較不同的latent dimension的結果。

切同樣的 train/valid dataset，並運用同樣的 mf model 來跑。

（沒有 normalization）

差別只在於 latent dimension。

patience = 10 epochs = 1000

batch\_size = 512 optimizer = 'adam'

Epoch through: Ealry Stopping 前歷經的epoch 數目

Epoch Take Per Time: 每個 Epoch 所需的時間

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| latent dimension | Epoch  through | Epoch  Take  Per Time | Train RMSE | Valid RMSE |
| 3 | 109 | 6s | 0.8269 | 08709 |
| 5 | 84 | 6s | 0.8038 | 0.8690 |
| 10 | 31 | 7s | 0.8062 | 0.8649 |
| 15 | 27 | 8s | 0.7867 | 0.8655 |
| 20 | 24 | 9s | 0.7792 | 0.8656 |
| 30 | 20 | 10s | 0.7800 | 0.8643 |
| 50 | 18 | 15s | 0.7662 | 0.8643 |
| 100 | 15 | 31s | 0.7310 | 0.8669 |
| 200 | 15 | 60s | 0.6641 | 0.8699 |

從上表我們可以看到隨著 latent dimension 增大

Epoch through 會越小 （越快找到最小點）

Epoch Take Per Time 會越大

Train RMSE 會越小

Valid RMSE 在 20.30左右就可以有很好的結果

latent dimension 再增大也沒有顯著的下降了。

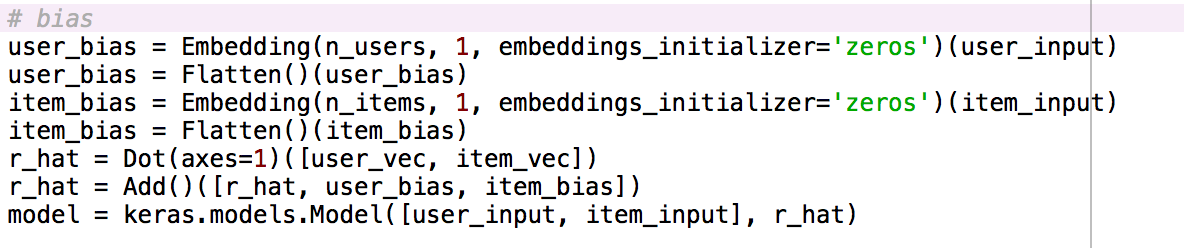
1. (1%)比較有無bias的結果。

切同樣的 train/valid dataset，並運用 mf model 來跑。

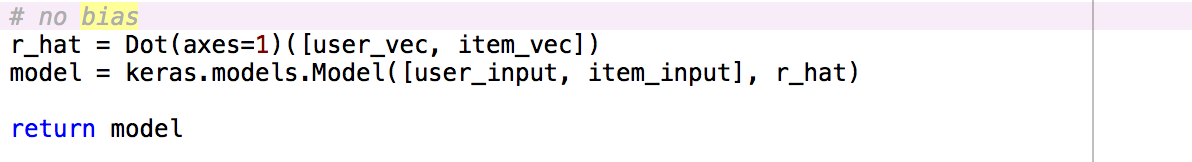
（沒有 normalization）

差別只在於有無 bias。

有 bias



無 bias

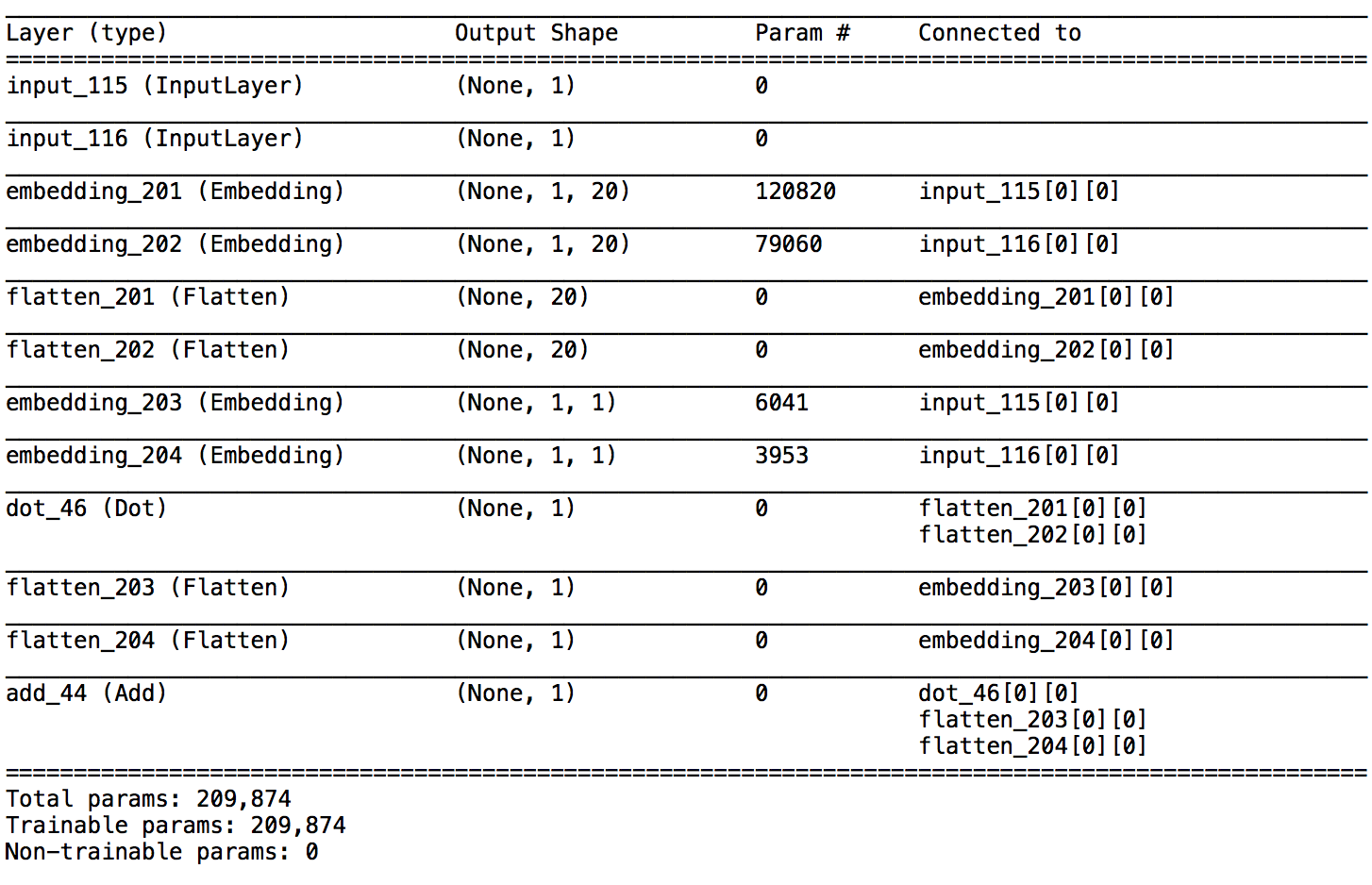


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Train RMSE | Valid RMSE | Kaggle Public RMSE | Kaggle  Private RMSE |
| 有  bias | 0.7792 | 0.8656 | 0.86642 | 0.86932 |
| 沒有  bias | 0.7743 | 0.8670 | 0.86796 | 0.87185 |

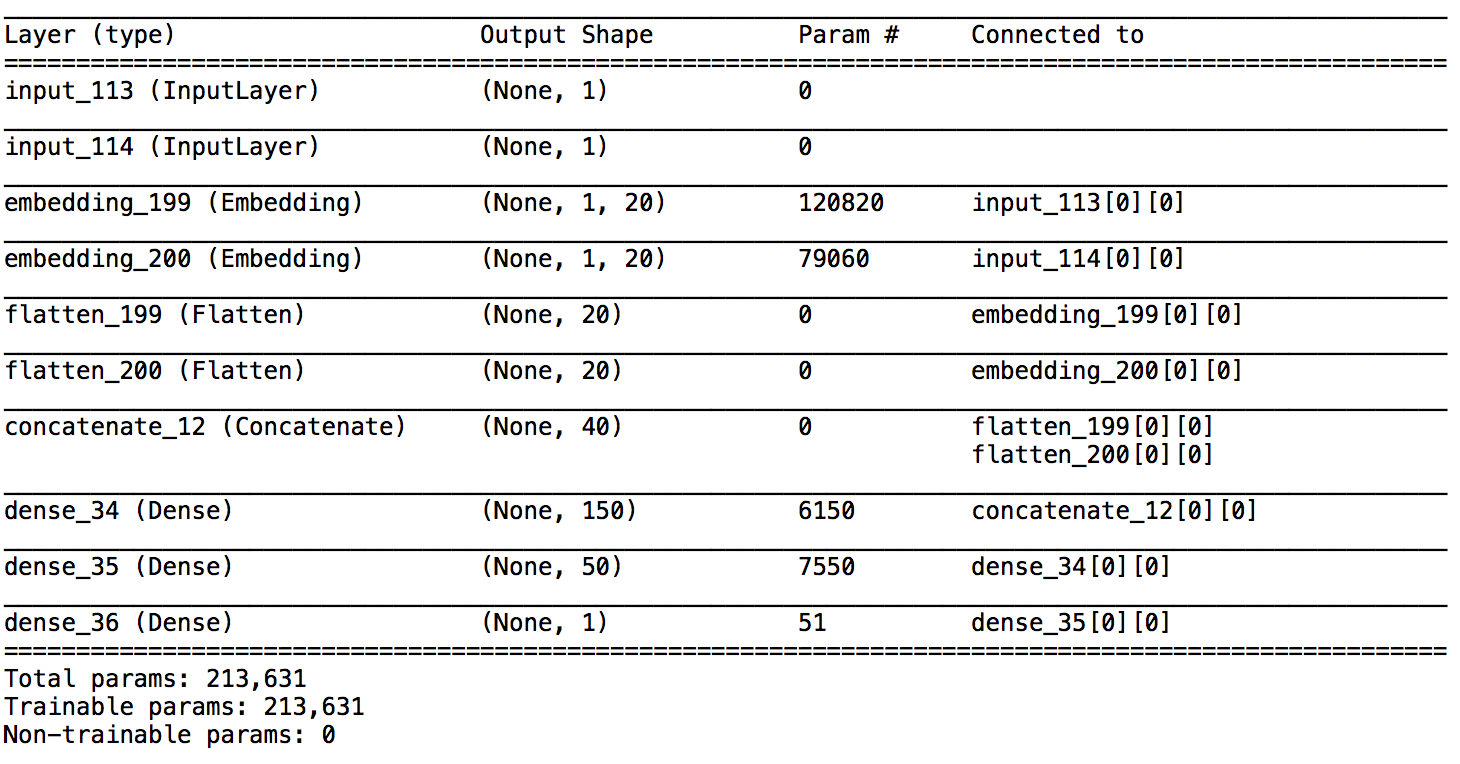
從 Validation 和 Test 的結果來看，有加上 bias 會使約果好一些。

1. (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。

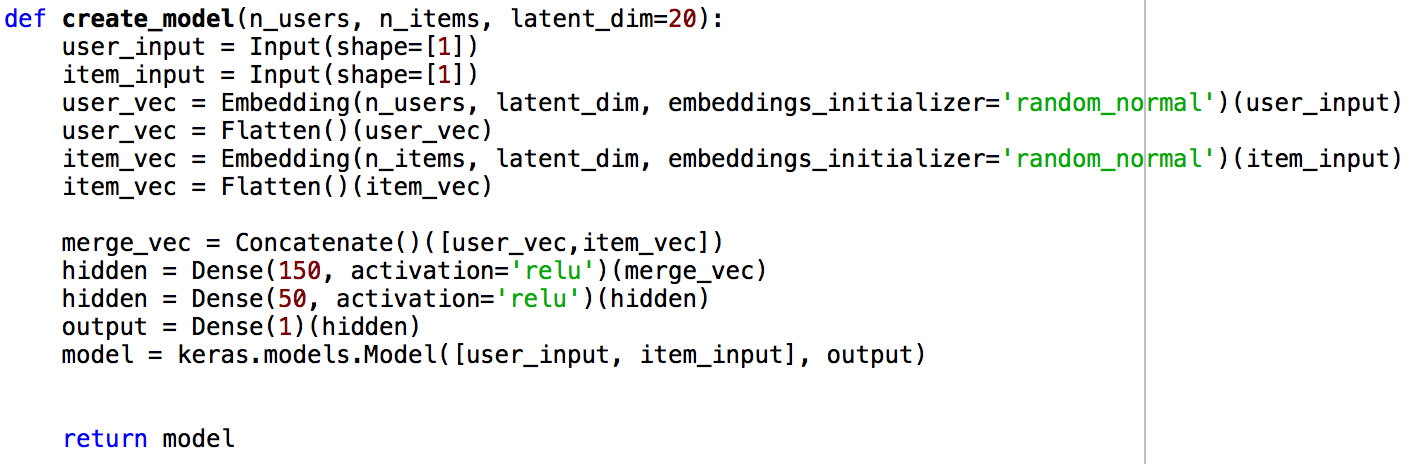
MF model (param 數：209874)



DNN model (param 數：213631)



DNN model



DNN model 的做法是在 embedding 和 flatten完 user 和 movie 後

透過 keras.layers.Concarenate 將兩者合併起來

然後直接餵給 Dense

在這邊疊了兩層 Dense(150)、Dense(50)

切同樣的 train/valid dataset，並分別運用 MF 和 DNN model 來跑。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Train RMSE | Valid RMSE | Kaggle Public RMSE | Kaggle  Private RMSE |
| MF | 0.7792 | 0.8656 | 0.86642 | 0.86932 |
| DNN | 0.8101 | 0.8783 | 0.87934 | 0.88253 |

在約略相同的 param 數目下，MF 的表現比 DNN 好。

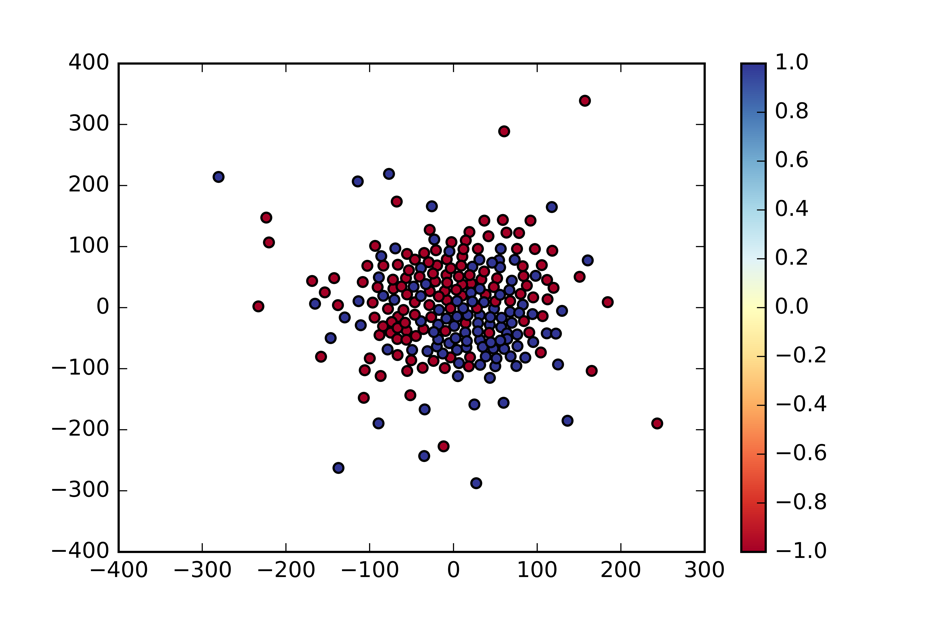
1. (1%)請試著將movie的embedding用tsne降維後，將movie category當作label來作圖。

我運用的 tsne 套件是sklearn.mainfold 的 TSNE

在做圖的時候只畫出有考慮的類別

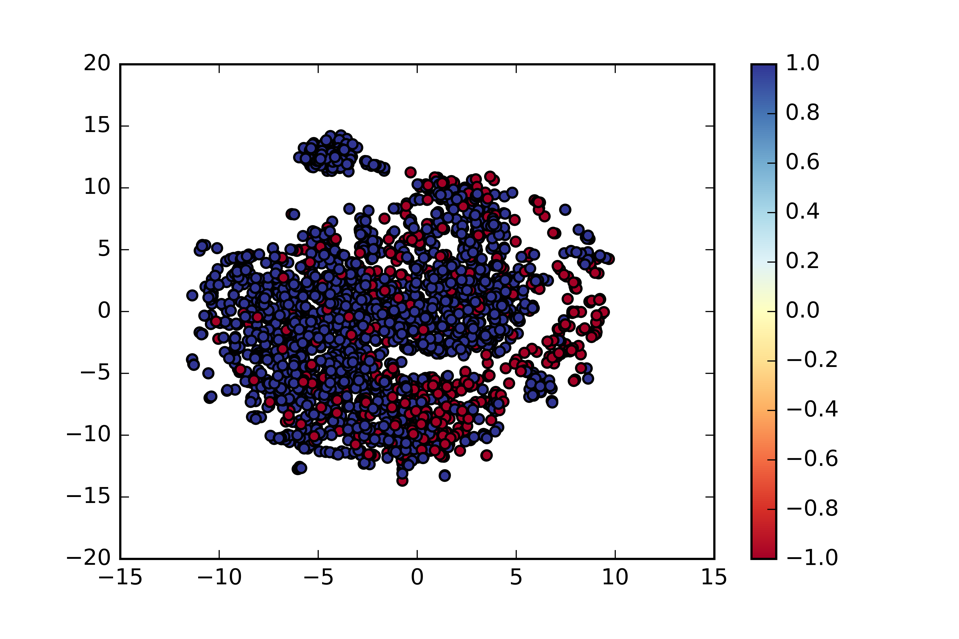
紅色：Documentary （偏左上）

藍色：Animation （偏右下）



紅色：Action, War, （偏右下）

藍色：Romance, Drama, Children‘s （偏左和中間）



1. (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。

我的 best model 就是採用所有feature 後 train 出來的。

假設今天的 latent dimension dim 是 embed\_dim

首先是電影種類，因為是 multi label  
所以將它轉成 binary vector，例如 [1,0,0,1,0,0…]

之後做 padding （後面補 0）

使其 vector 長度成為 embed\_dim

(因此 embed\_dim 不能少於電影種類)

客戶性別是M 和 Ｆ

同樣轉成長度為 2 的 vector，男是 [1,0]，女是[0,1]

之後同樣做 padding，使其 vector 長度成為 embed\_dim

其他如 age 和 occupation 也是如此

根據其種類的個數轉換成 n 維的 binary vector

之後做 padding，使其 vector 長度成為 embed\_dim

我的 model 的做法是

同樣先將 movie 和 user 做 embedding 和 flatten

出來後的維度是 embed\_dim

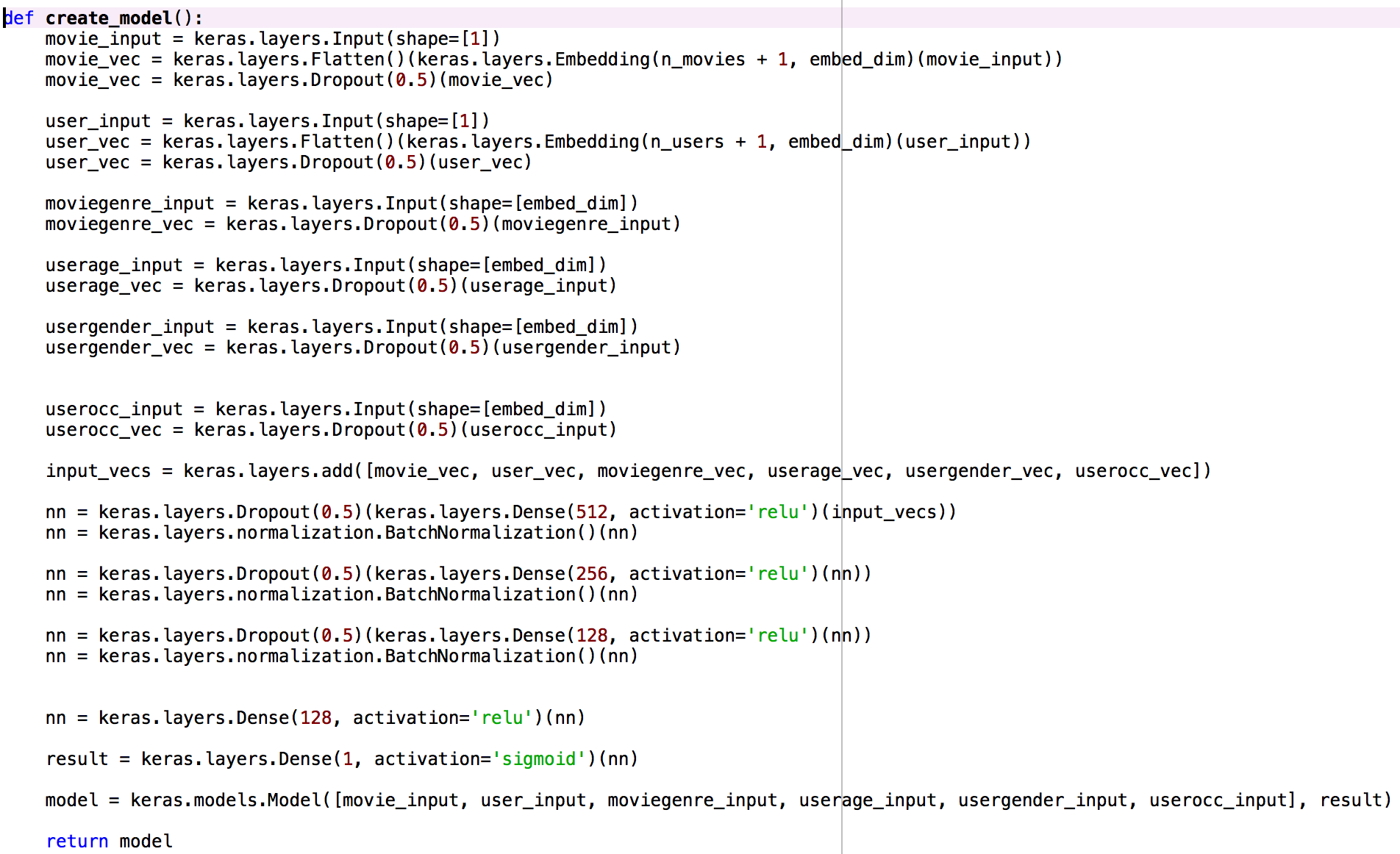
而其他的 feature 在餵近來之前，因為有做 padding

所以維度就已經是 embed\_dim 了

所以可以將這些 feature 和 movie 與 user 的 embedding 做 keras.layers.add 使其成為 input\_vecs

最後再疊上 NN 來完成 model

我的 model



在 Kaggle 上個人最佳成績來自於此 model

Kaggle Public: 0.85502

Kaggle Private: 0.85985