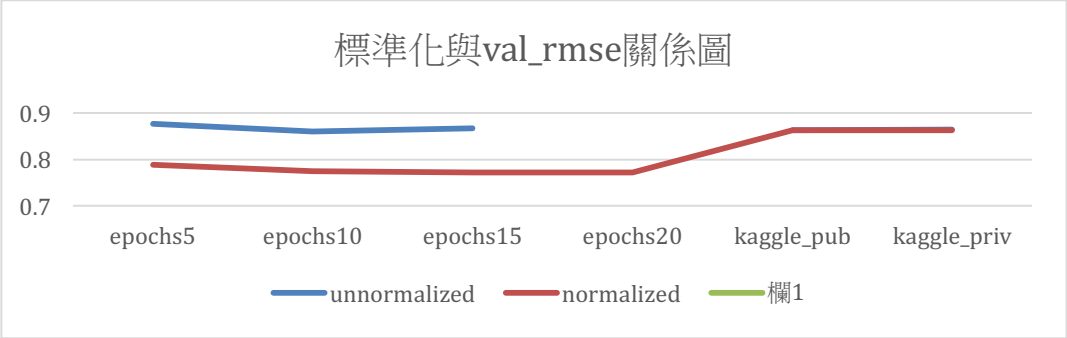
學號：B04902092 系級： 資工三 姓名：張均銘

1. (1%)請比較有無 normalize(rating)的差別。並說明如何 normalize. (collaborator:b04902021 陳弘梵)

   作法:

   使用 Matrix Factorization, 把 rating 資料 normalized(mean, std 另外開一個變數存，predict 時要恢復成 1-5)，另外採用 clip 的方式，避免在 train 的時候跑到太極端的值，predict 完後再*std+mean，也是採用 clip(1.,5.)的方式讓他的值在 1-5。

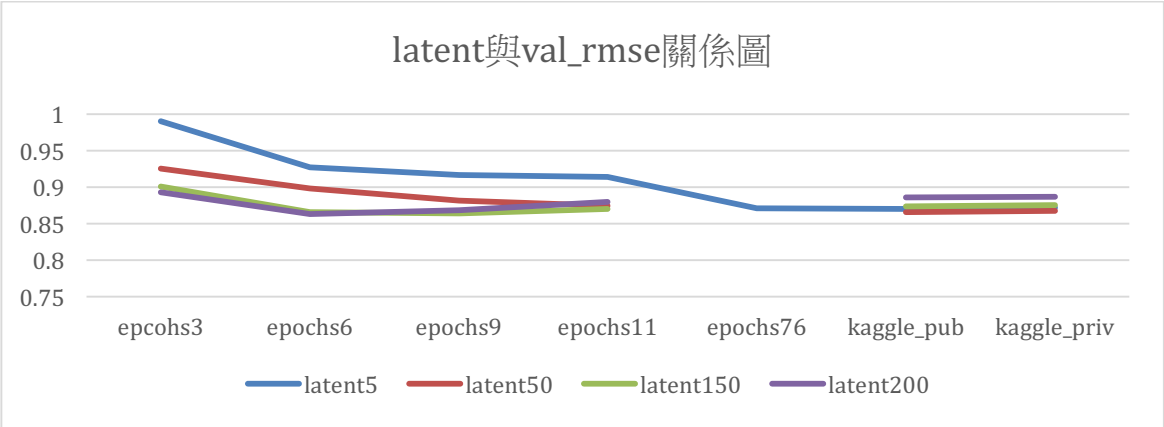



   觀察結果：有 normalized 的 val_rmse 看起來比較低(實際上是因為 normalized 後誤差本來範圍就比較小，乘回 std+mean 後差不多)，不知為何，normalized 得成績沒有好多少，與 unnormalized 幾乎依樣。差別就是 normalized 比較晚達到 earlystop, 而 unnormalized 的在 epochs15 左右就 early stop 了，有無 normalized 其實沒有差太多。

2. (1%)比較不同的 latent dimension 的結果。

   做法:

   模型採用跟上題依樣 Dropout rate 固定 0.5 沒有 normalized,validation 切 0.2 分成 latent = 5,50,150,200 來做，x 軸是 epochs 數、y 軸是 val_rmse(validation set 的 rmse)




   其中 latent5 在 30 epochs 中穩定下降，每次下降幅度很少，但最後其實產生

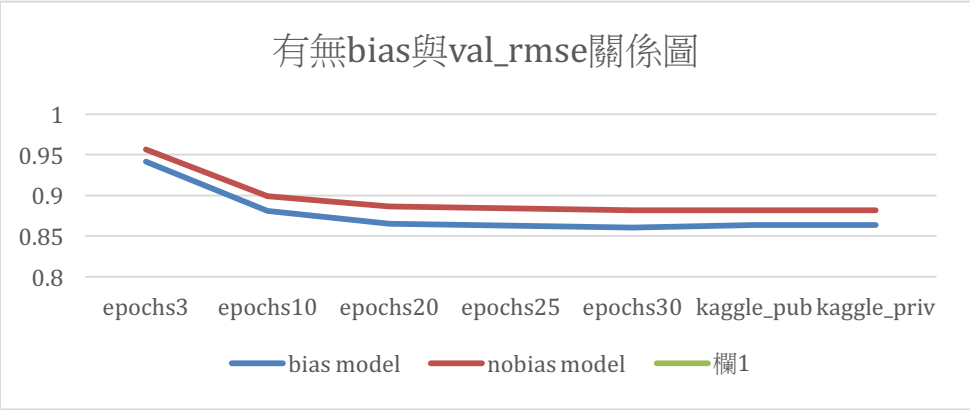
的結果比其他三種都略微好一點，latent50 在 epochs21 earlystop，最後結果大概是 val_rmse = 0.8619 左右，latent150 在 epochs10 earlystop, latent200 在 epochs9 就 early stop 了。

**說明：**

明顯參數比較多的 model 需要較少的 epochs 就能達到 minimum，但也更容易 overfitting，像 latent5 根本沒發生 earlystop(下降速度太慢)，在 latent150, 200 很明顯雖然很快就達到最低點，但是因為參數太多導致 overfit，後面的 val_loss 急劇上升。所以參數較多的 model 可以在較少的 epochs 數看到顯著的下降，但其產生的結果不一定比參數少的 model 還好，因為參數多 traindata 更容易 overfit，相對參數少的 model 雖然下降速度緩慢，但不容易 overfit，所以能 train 很多的 epochs，最後還會比參數多的 model 好一些。

3. **(1%)**比較有無 **bias** 的結果。

   model 跟第一個一樣，只是 droprate=0.1, validation = 0.2, latent=150



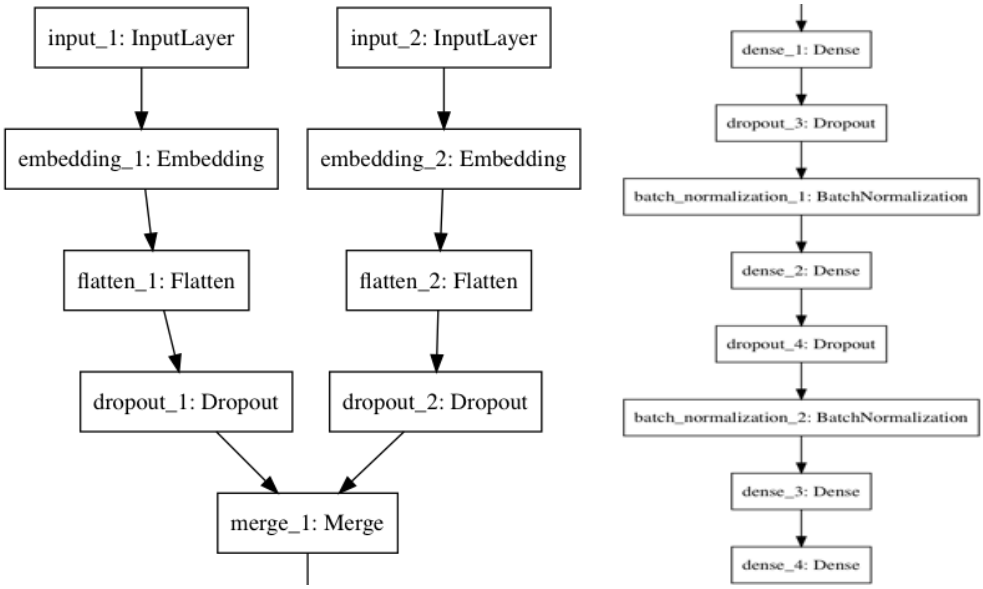

**說明：**

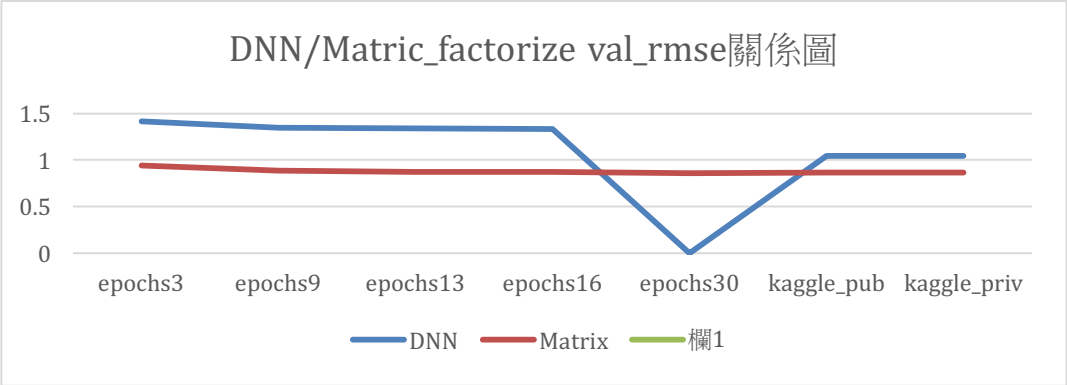
明顯的有 bias 的效果比較好，但兩者的下降幅度也都差不多。原因就是，每個 user 的給分習慣不同，有些人普遍喜歡給高分，有些人則是低分，所以 bias 的加入就能把它平移，所以就讓給高分跟給低分的 user feature 相近。使結果更接近 validation set，沒有 bias 的 model 就沒有對這部分做處理, 所以高分跟低分的 user 他們之間的 feature 就會差很多，一起 train 的話結果就會比較差。。

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。

   **DNN 實作:**

   將 user, movie 分別做 embedding 以後 concate 再一起後做兩層 dense，中間 Dropout 都是 0.5，最後輸出是 category 總共 5 個，就是代表該 user 對該 movie 的評價是屬於 1, 2, 3, 4, 5 中的哪個(categorical 的方式)，所以輸出方式並不會有小數。圖太長，切成兩張，左邊 merge 後進入右邊的 model




**說明:**

DNN epochs30=0 是因為 early stop，所以沒有該格資料。很明顯的，因為 DNN 只能夠將 output 結果分成五類，然後選擇機率最的那一類當成那個 user 對該 movie 給的分數，後來有做修改，改成某一類機率超過 20%的就抓來做平均，例如假設某一格機率(4 是 0.21,5 是 0.23，那該格的 output 變成 (4*0.21+5*0.23)/(0.21+0.23)結果會變好，kaggle pub 變成 0.949 但還是不夠。

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

**作法:**

採用 rmse 約 0.862~0.863 的 model，把 model load 進來後取 embedding layer,依助教的 code 拿去 TSNE fit，之後得到 x,y 軸，另外用 group 把 movies genre 分類

category1:[drama,musical]

category2:[thriller,horror,crime,mystery]

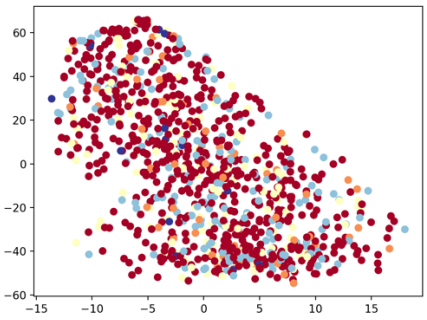
category3:[adventure,fantasy,action,western]

category4:[documentary, film-nor, war]

category5:[Animation, Children, Comedy]

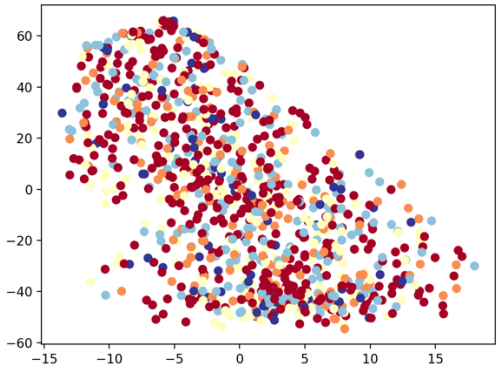
category6:[Romance]

分成這六類後，把 train.csv 資料讀進來 shuffle 過後做分類，就是依據他有上面 6 個 category 中每個分類幾個元素，然後這筆資料就屬於哪類。



**結果不好的原因：**

除了因為資料太多比，我只取 1000 筆作圖以外，就是 category 的分法造成問題，因為每個 category 元素個數不同，像 romance 只有一個可是其他有些組有 4 個，這樣萬一同時有 romance, adventure, fantasy 會被歸類在 action 類，因為他有那個 category 中的 3 個而只有 romance 中的一個，所以就要給每個 category 一個權重，如果有裡面一個元素要乘以多少權重，如 romance 權重就要*5 之類的，這樣做圖變成：



似乎只有增加其他點的個數，但整體還是很亂，推測原因是 embedding 時是依照 movieid 下去 embed，所以跟本身屬於什麼 genre 關係不大。

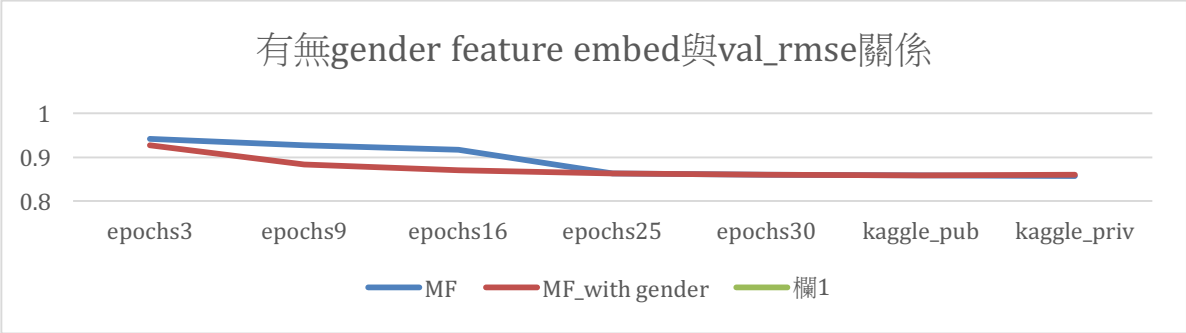
6. **(BONUS)(1%)**試著使用除了 **rating** 以外的 **feature**, 並說明你的作法和結果, 結果好壞不會影響評分。

**(collaborator:**b04902082 鍾偉傑**)**

**作法：**

加入 users.csv 的 gender 類別下去做 embedding，將 users.csv 的 gender，女生為 0，男生為 1, 利用字典方式建立 userid 和 gener 的關係

dict[userid]=gender，model 為第一題的 MF model，將剛剛得到的 user 的 gender 的 embed array embed 在 user bias 的地方下去 train。




途中會發現似乎兩者的準確利差不多，但原本沒有把 gender embed 到 user_bias 的 model 好一點，但基本上沒有太大的差距，最後 kaggle 出來於本 model 是 0.85782 而有 gender 的是 0.8579 也算是在誤差範圍，

**原因：**

就算 embed gender 進去當 bias，跟全部交由 keras embed，最後都會因為 fit 的過程對 bias 做 gradiant descent 使種方式都趨近於同一個值(fit train data 的最佳 bias),所以實際上把 gender embed 在 user_bias 有點像給 user_bias 不同的初始值，但並不影響 training 結果