ML2017 hw6 report

學號：B04902045 系級： 資工二 姓名：孫凡耘

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

把所有rating normalize到[0, 1]之間(即減掉平均再除以標準差)，train完之後predict時在用相同的標準差與平均transform回來。

hyperparameters:

CF model without bias

latent dimension: 120

optimizer: adamax

validation_split: 0.1

early stopping patience: 2

No normalization

LB: 0.86735

With normalization

LB: 0.86722

normalize之後有一點點的進步。

(1%)比較不同的latent dimension的結果。

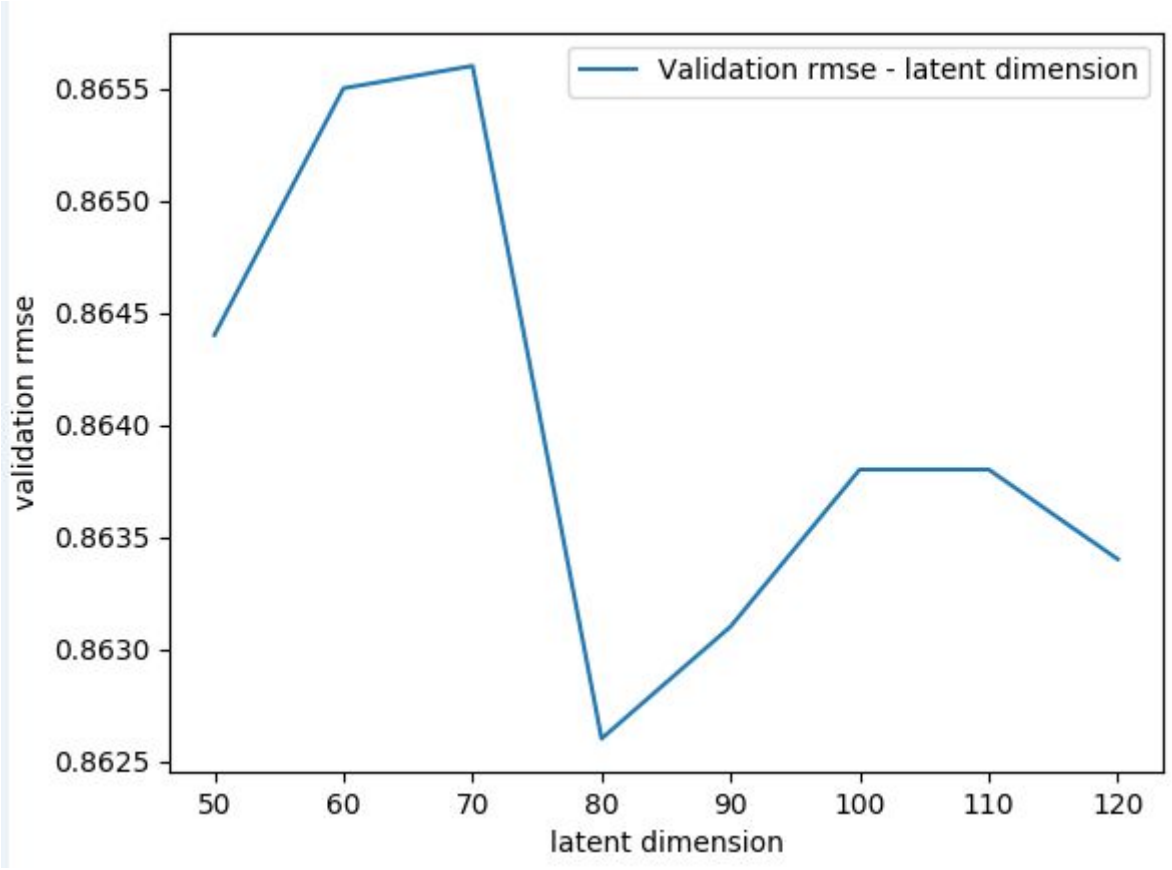
hyperparameters:

Model: matrix factorization with bias

optimizer: adamax

validation_split: 0.1

early stopping patience: 2



The optimal latent dimension seems to be around 80 according to my experiment.

(1%)比較有無bias的結果。

hyperparameters:

latent dimension: 120

optimizer: adamax

validation_split: 0.1

early stopping patience: 2

No bias

CV: 0.861

LB: 0.867

With bias

CV: 0.863

LB: 0.869

在這個dataset上，加上bias對結果並沒有太大幫助。我這邊只選兩次的結果做呈現，但每次random split的結果會讓model有些不同。不過大體上無法判定哪個model一定表現得比較好(當然若hyperparameters取的不一樣結果就不一定如此)。

(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。

Model Structure

```
class DeepModel(Sequential):

    def __init__(self, n_users, m_items, k_factors, p_dropout=0.1, **kwargs):
        P = Sequential()
        P.add(Embedding(n_users, k_factors, input_length=1))
        P.add(Reshape((k_factors,)))
        P.add(Dropout(0.1))
        Q = Sequential()
        Q.add(Embedding(m_items, k_factors, input_length=1))
        Q.add(Reshape((k_factors,)))
        Q.add(Dropout(0.1))
        super(DeepModel, self).__init__(**kwargs)
        self.add(Merge([P, Q], mode='concat'))
        self.add(Dropout(p_dropout))
        self.add(Dense(k_factors, activation='relu'))
        self.add(Dropout(p_dropout))
        self.add(Dense(1, activation='linear'))
```

將movie的embedding與user的embedding concat起來，餵到dnn。

hyperparameters:

（與上題相同以方便比較)

latent dimension: 120

optimizer: adamax

validation_split: 0.1

early stopping patience: 2

DNN hyperparameters:

output activation: linear

hidden layer activation: relu

Matrix Factorization without bias:

CV: 0.861

LB: 0.867

DNN with one hidden layer：

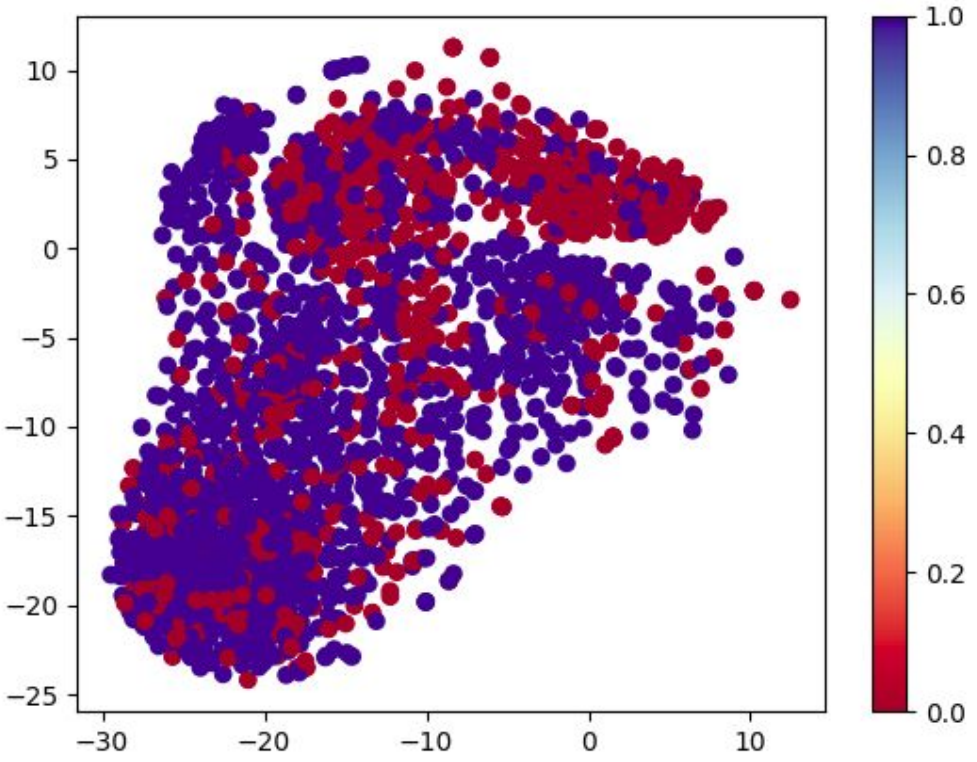
CV: 0.862

LB: 0.862

我認為dnn能夠表現得比較好是很合理的，畢竟matrix factorization的做法等於是"直接認定" Rating 可以用movie_embedding dot user_embedding + bias來預測。這個scenario很顯然dnn的function space比較大s(matrix facotrization所使用的近似函數是dnn所能表達的函數集合的子集)，如果train得好的話是能找到比matrix factorization更好的function去預測rating，不過當然也不代表這樣就一定能表現更好，畢竟dnn學出來也有可能是學到會把兩個embedding dot起來之類的。

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

為了好區分結果，將只要屬於Thriller, Horror, Crime其中一個即視為第1類，屬於Drama, Musical其中一個及屬於第2類（不使用同時屬於兩類的data)。因為標太多看不出效果，因次做圖時忽略不屬於這兩類的data。
red ⇒ type 1
blue ⇒ type 2



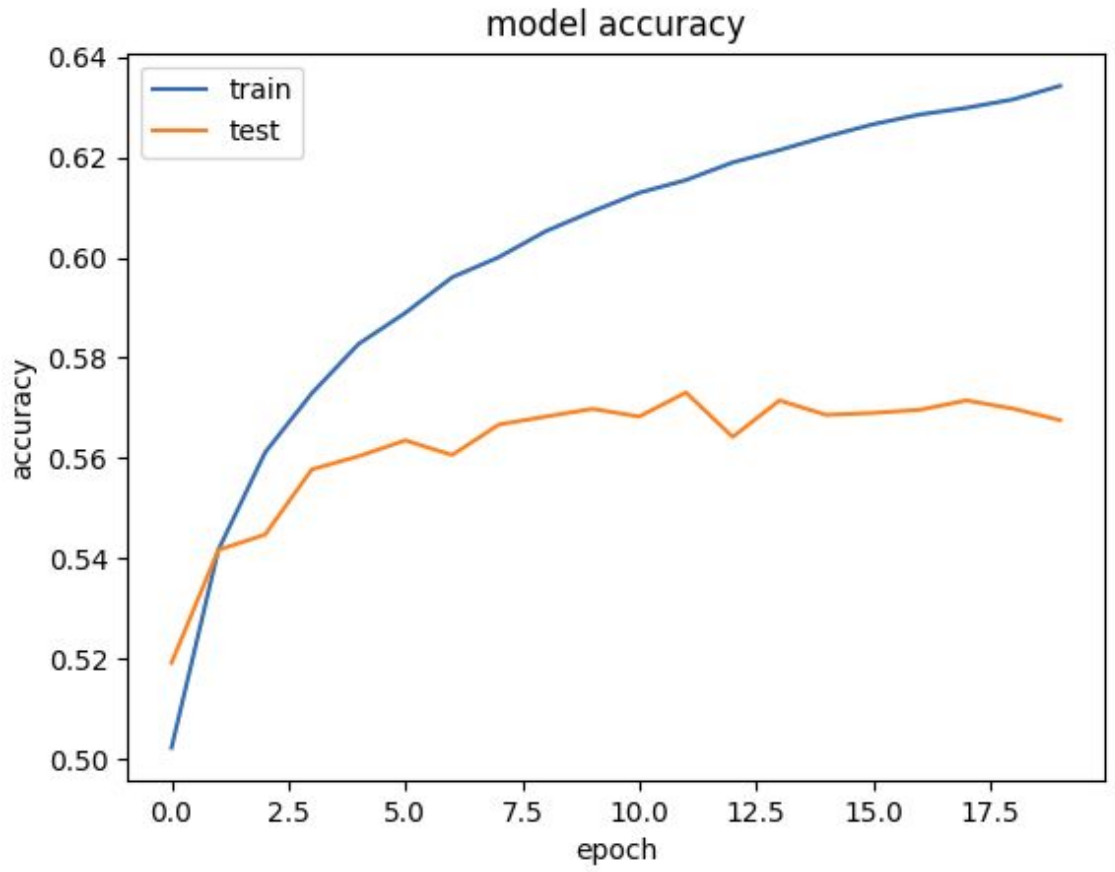
(BONUS)(1%)試著使用除了rating以外的feature, 並說明你的作法和結果，結果好壞不會影響評分。

除了將movie_emb跟user_emb concat起來(with latent dimension of 120)，再把movie的category(one hot encoding, 可以同時屬於很多個category)當作feature，形成一個feature vector of 258 dimension，並把他當成classification problem(5 個rating)，餵入dnn。

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	144896
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 256)	65792
dropout_3 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 5)	1285

Total params: 343,301
Trainable params: 343,301
Non-trainable params: 0

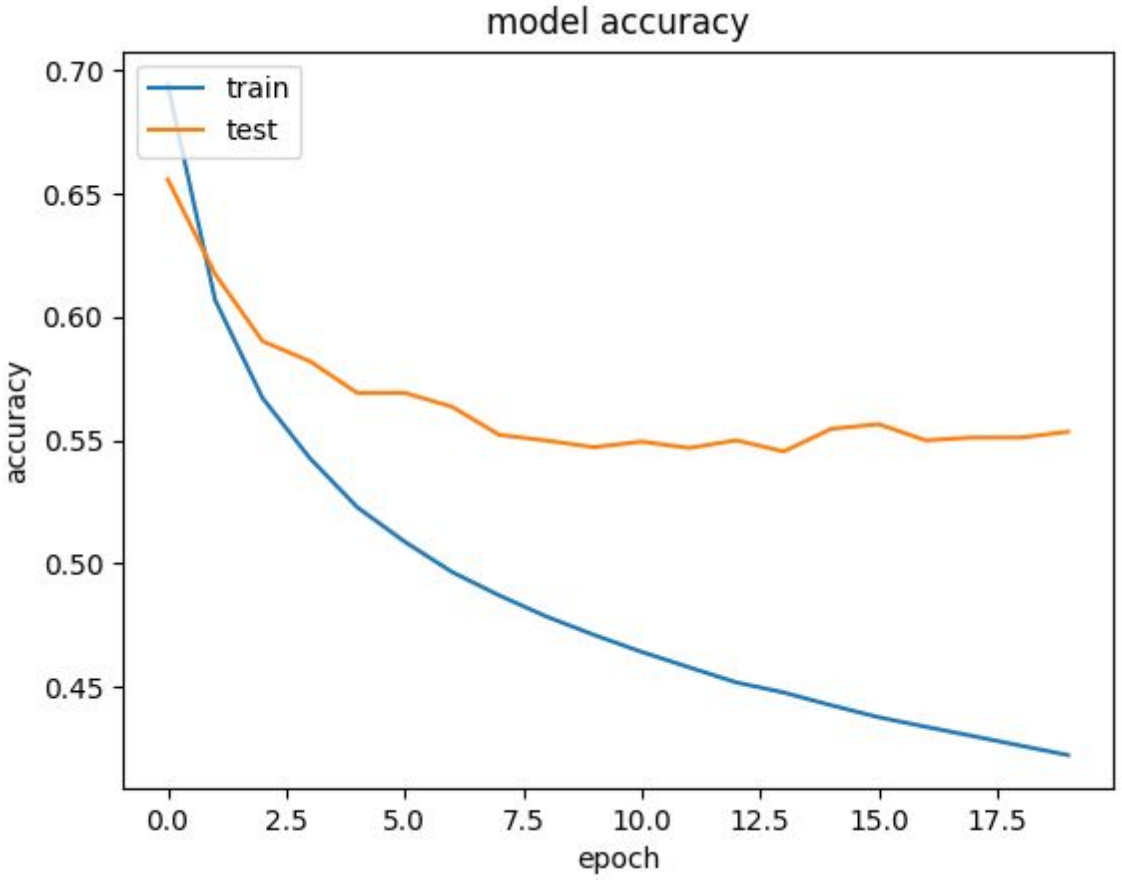
training過程以accuracy作為evaluation metric.



LB: 0.96532

不知道表現那麼差是不是我把他當成classification problem

所以我用用一樣的方法，但把model最後一層改為直接output一個rating



LB: 0.90264

不太確定為甚麼結果這麼不好，不過畢竟是拿前面model的embedding的結果來繼續train。很可能前面做出來的embedding結果不夠好導致dnn沒辦法表現更好。或許用上面的model的embedding當作initialize value，train的時候embedding跟這繼續train，這樣表現會比較好（不過也有可能movie's Genres這個feature，對於預測rating而言單純就不是有用的feature)。