ML2018SPRING HW6 Report

學號：r06921038 姓名：謝宗宏 系級：電機所碩一

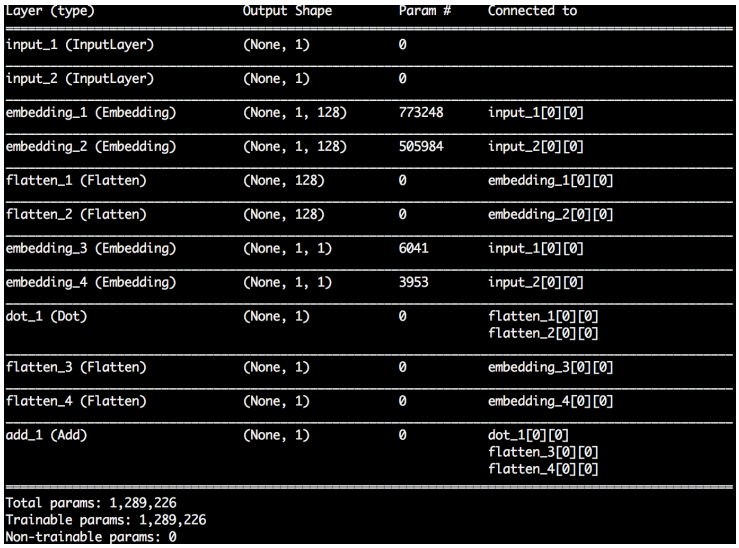
進行以下比較的 baseline model :

embedding dimension = 128

epochs = 100 with early stopping (tolerance 10)

batch size = 512

with normalization and bias



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)	773248	input_1[0][0]
embedding_2 (Embedding)	(None, 1, 128)	505984	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)	6041	input_1[0][0]
embedding_4 (Embedding)	(None, 1, 1)	3953	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: 1,289,226
Trainable params: 1,289,226
Non-trainable params: 0

每一項比較僅有在操縱變因上的調整。

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

如何 normalize:

normalized_rating = (rating – mean(rating)) / standard_deviation(rating)。

	with normalization	without normalization
Training loss	0.4816	0.5958
Validation loss	0.5558	0.7343
Kaggle public	0.86801	0.87287
Kaggle private	0.85971	0.86573

可看出有 normalize 的表現在 validation, public, private 上都是比較好的。

(2)比較不同的 embedding dimension 的結果。

dimension	64	128	256	512
Training loss	0.5058	0.4816	0.4810	0.3736
Validation loss	0.6124	0.5558	0.6249	0.6124
Kaggle public	0.87044	0.86801	0.86809	0.87176
Kaggle private	0.86380	0.85971	0.85821	0.86486

128 維的表現在 validation, public 上的表現都是比較好的，維度太小太大似乎都會讓表現變差。256 維在 private 的表現最好，但與 128 維相差不大。

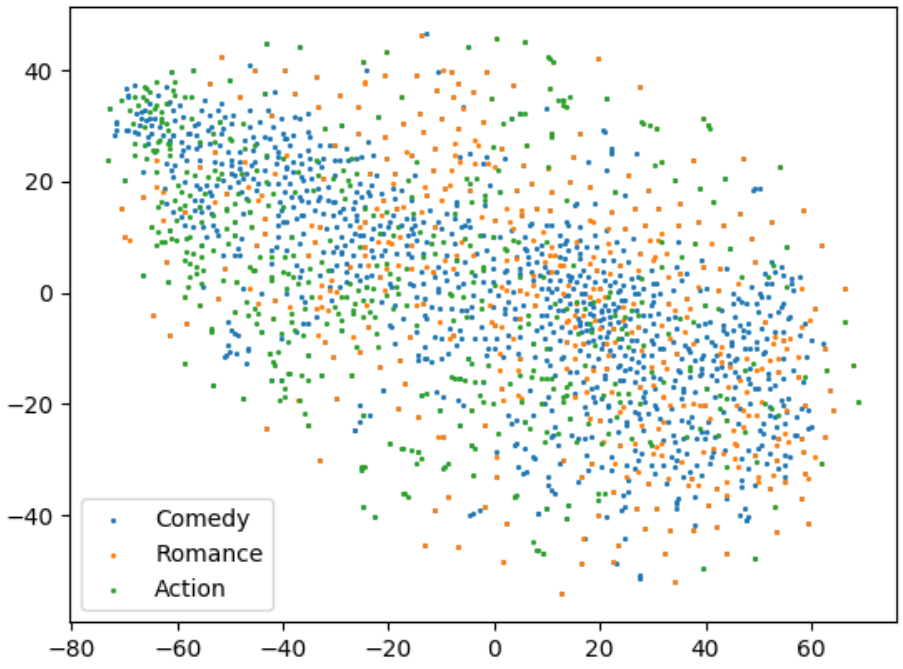
(3)比較有無 bias 的結果。

	With bias	Without bias
Training loss	0.4816	0.4572
Validation loss	0.5558	0.6750
Kaggle public	0.86801	0.87047
Kaggle private	0.85971	0.86303

可看出沒有 bias 的時候，在 validation, public, private 上表現都是比較差的。

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

將 64 維的 embedding 用 tsne 降到二維。使用的 categories: Comedy, Romance, Action。Romance 的分佈較為均勻鬆散，Comedy 的分佈較為集中於對角線上，Action 比 Romance 稍微集中一點，並且分佈略比 Comedy 偏左下。



(5)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。

使用 users.csv 中的 Gender, Age, Occupation ，將 Gender、Occupation 轉換成 one-hot encoding 的形式，Age 則是直接拿來使用。最後將 UserID, Gender (one-hot), Age, Occupation (one-hot) 全部 concatenate 起來變成代表一個 user

的 vector。

使用 movies.csv 中的 Genres，將 Genres 轉換成 bag of word 的形式，再與 movieID concatenate 起來變成代表一個 movie 的 vector。

將 user vector 和 movie vector 根據 train.csv 的對應 concatenate 起來變成一筆 training example，作為 model 的 input。而 model 的 output 仍為 rating。

training loss: 1.1897 , validation loss: 1.1824 , kaggle public: 1.08134 , kaggle private: 1.07740。

這個作法並不成功，畢竟只能算是硬把手邊的 feature 接起來就拿去 train 了，並沒有考慮那些 feature 是否重要，或是先將 feature 做一些比較適當的前處理，因此可能反而增加了不必要的 noise 。也有可能是使用的 DNN 架構並不適當。

Model 的架構如下。使用 nadam (default arguments), mse, batch size 512, epoch 100 with early stopping。

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 44)	0
dense_1 (Dense)	(None, 256)	11520
batch_normalization_1 (Batch	(None, 256)	1024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
batch_normalization_2 (Batch	(None, 128)	512
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
batch_normalization_3 (Batch	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 128)	8320
batch_normalization_4 (Batch	(None, 128)	512
dropout_4 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 256)	33024
batch_normalization_5 (Batch	(None, 256)	1024
dropout_5 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 1)	257

Total params: 97,601
Trainable params: 95,937
Non-trainable params: 1,664

References:

[1] https://nipunbatra.github.io/blog/2017/recommend-keras.html

[2] https://docs.google.com/presentation/d/1MJy5gr4uaSDy4RuTGugjQdxI-DnXih-FqdPDIJQeDHU/edit#slide=id.p8

[3] https://towardsdatascience.com/ensembling-convnets-using-keras-237d429157eb