

1. (1%) 請問 softmax 適不適合作為本次作業的 output layer? 寫出你最後選擇的 output layer 並說明理由。

softmax 並不適合作為本次作業的 output layer。

softmax 是會使最後一層所有 output 機率的結果合為 1，
如果是 multi class with 1 label 就適合 softmax

假設今天有 4 個 class，而 data 有 1 label
某一 data 經 model 出來的結果可能是 [0.05, 0.05, 0.1, 0.8]
我們能明顯判斷最後一項是對應的 label

但假設今天 data 有 3 個 label
某一 data 經 model 出來的結果可能是 [0.3, 0.3, 0.19, 0.21]
我們較不能明確判斷，更不用說在 multi label 的情況下，
我們是無法得知每個 data 有幾個 label，
也因此更無法判斷哪些是對應的 label

而本次作業是 multi class with multi label。
因此並不適合 softmax

2. (1%) 請設計實驗驗證上述推論。

跑兩個接近相同的 RNN model 在本次作業上，

取同樣的 training set 和 valid set。

兩者差別只在於最後一層是 softmax 或 sigmoid

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 186, 100)	5376800
gru_1 (GRU)	(None, 186, 256)	274176
gru_2 (GRU)	(None, 128)	147840
dense_1 (Dense)	(None, 64)	8256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 64)	4160
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 38)	2470
Total params:	5,813,702	
Trainable params:	436,902	
Non-trainable params:	5,376,800	
Train on 4467 samples, validate on 497 samples		

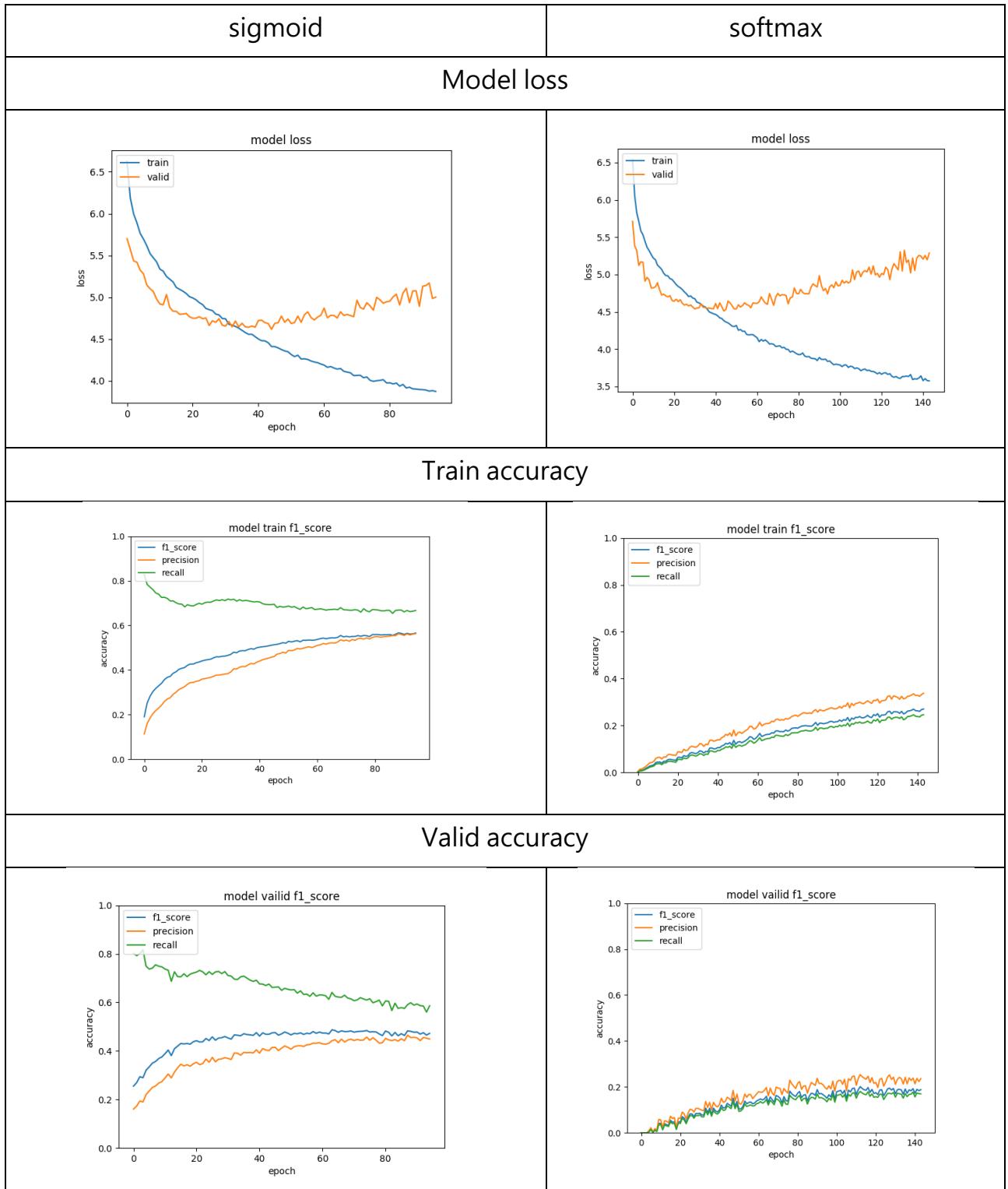
(model summary)

最後一層，dense_3，

一個選 softmax 做 activation，

一個選 sigmoid 做 activation。

結果：



從 Training accuracy 和 Valid accuracy 可以看出來 sigmoid 明顯比 softmax 好。

sigmoid valid f1 measure 約快 0.5 ,

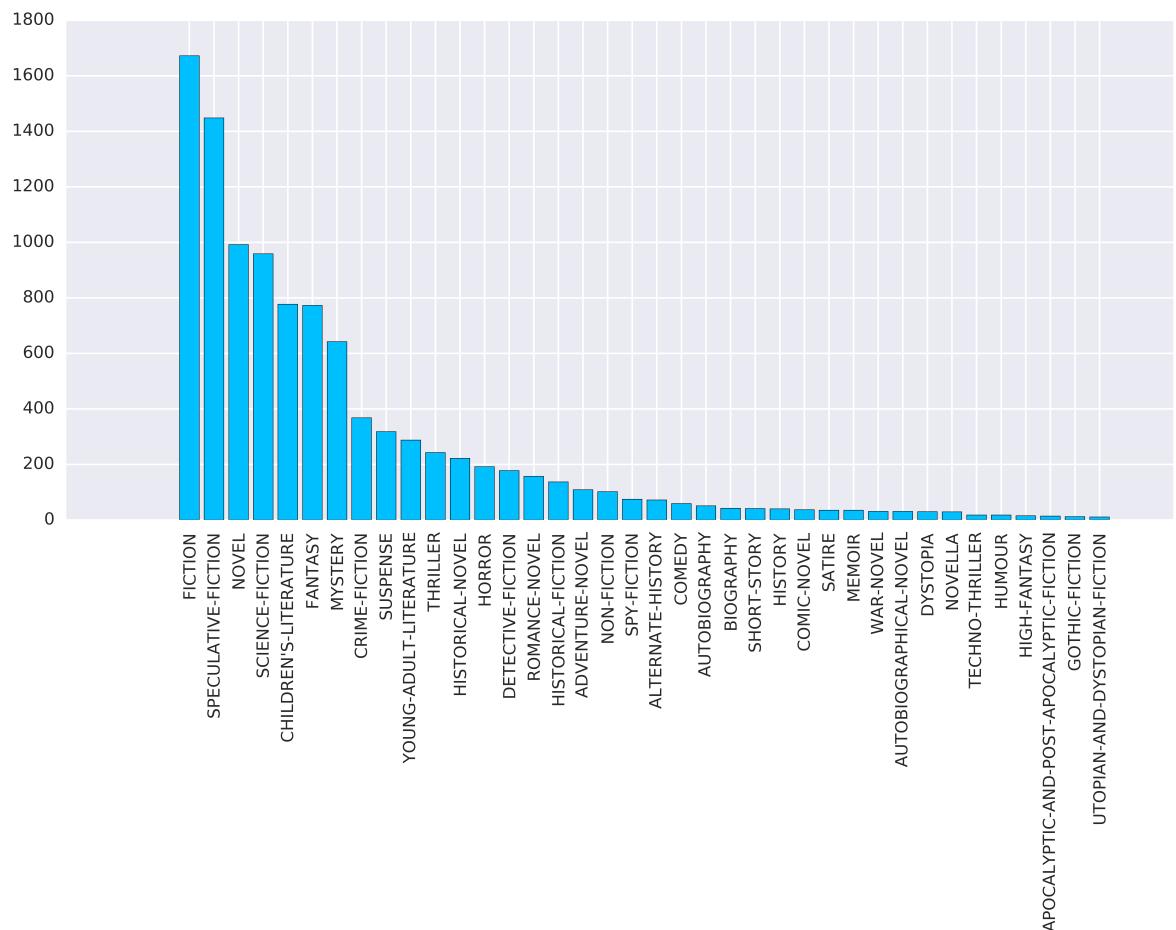
然而 softmax 只有 0.2 左右 ,

因此 softmax 較不適合本次作業。 (multi label)

3. (1%)請試著分析 tags 的分布情況(數量)。

統計 tags 出現次數，由大排到小，並且發現分佈其實很不均。

38 項中的前 10 項就佔了約 80 % 左右。

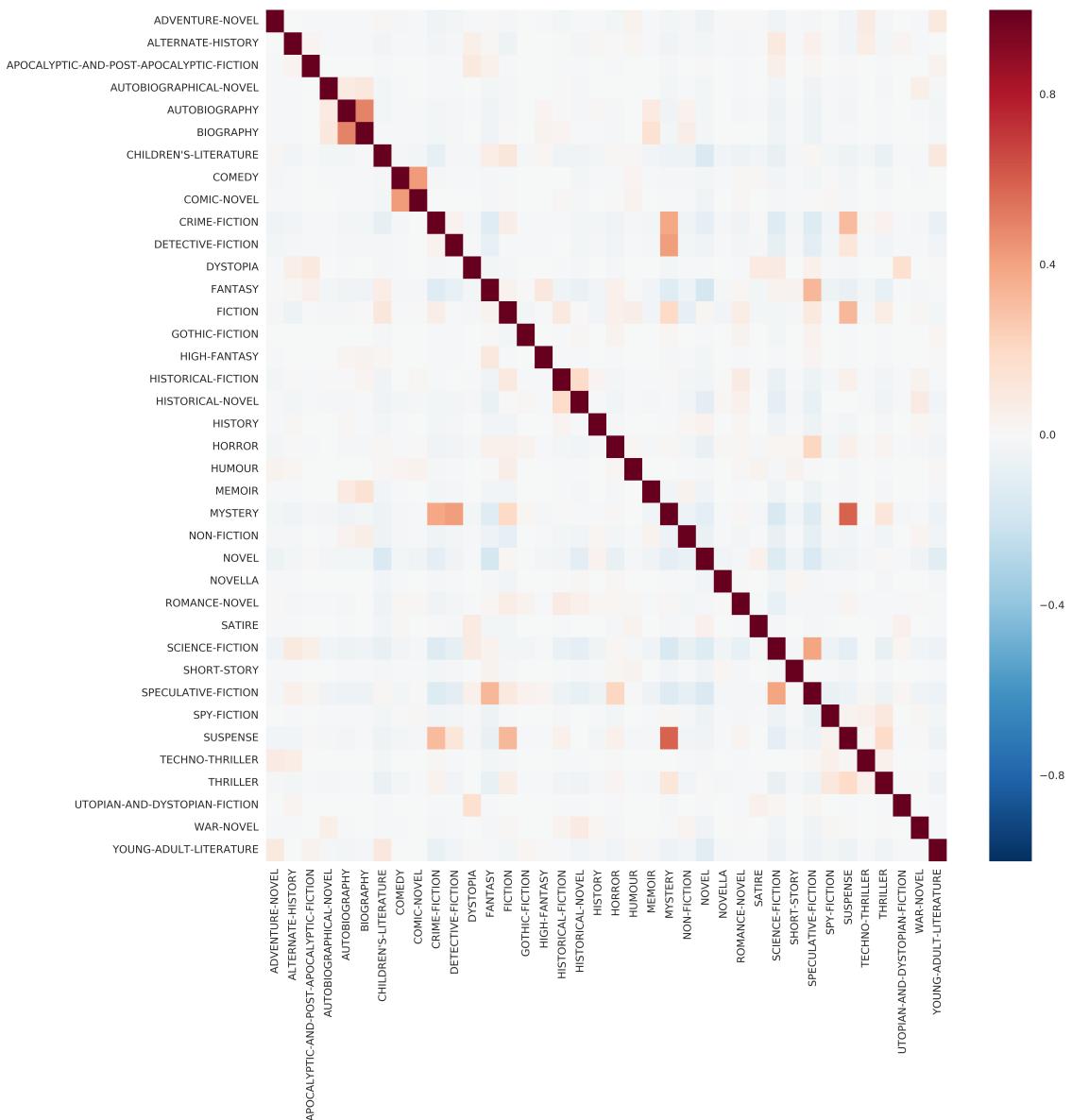


另外，透過將每個 label 以 binary 的方式儲存後，

我們可以將 label 轉成長度 38 的 binary list。

在所有 training 資料中，我們有 4964 個 label binary list。

接著可以算出 correlation matrix，並用 heatmap 觀察。



可以發現以下 tags 比較有正關聯。

(COMEDY · COMIC-NOVEL) 、

(BIOGRAPHY · AUTOBIOGRAPHY) 、

(MYSTERY · SUSPENSE)

4. (1%) 本次作業中使用何種方式得到 word embedding? 請簡單描述做法。

這次作業我的 embedding 來源是 glove

<https://nlp.stanford.edu/projects/glove/>

檔名 : glove.6B.100d.txt

(word vector 維度是 100)

首先將原本的 train_data.csv 資料讀進來後，

從 nltk 套件取 stop words 濾掉，並全部轉成小寫。

接著透過 keras 的 Tokenizer 把 word 做 index 的編碼。

(一個 word 對應到一個號碼的意思。)

tokenizer = Tokenizer()

tokenizer.fit_on_texts(all_texts)

接著把所有 data 文章轉成 index sequence，

並做 padding。 (將剩餘的空白做 0 編碼)

train_sequences = tokenizer.texts_to_sequences(train_texts)

X_train = pad_sequences(train_sequences)

然後讀 glove.6B.100d.txt，做出 embedding_dict。

透過 embedding_dict，我們給一個 word 後，

它可以找出對應的 word vector

有了 tokenizer.word_index 和 embedding_dict，

我們就能做出 embedding_matrix。

embedding matrix 的 row 代表每一個在 tokenizer 中的 word 的 vector，

並且每一個 word vector 是 100 維，所以會有 100 columns。

最後將 embedding matrix 餵進 keras model。

model.add(Embedding(num_words,

word_vec_dim,

weights=[embedding_matrix],

input_length=maxlen,

trainable=False))

5. (1%) 試比較 bag of word 和 RNN 何者在本次作業中效果較好。

此題的 bag of words 的 mode 採 tf-idf。

取同樣的 training set 和 valid set 下去做兩類型的 model。

RNN model:

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 186, 100)	5376800
gru_1 (GRU)	(None, 186, 256)	274176
gru_2 (GRU)	(None, 128)	147840
dense_1 (Dense)	(None, 64)	8256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 64)	4160
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 38)	2470
<hr/>		
Total params: 5,813,702		
Trainable params: 436,902		
Non-trainable params: 5,376,800		
<hr/>		
Train on 4467 samples, validate on 497 samples		

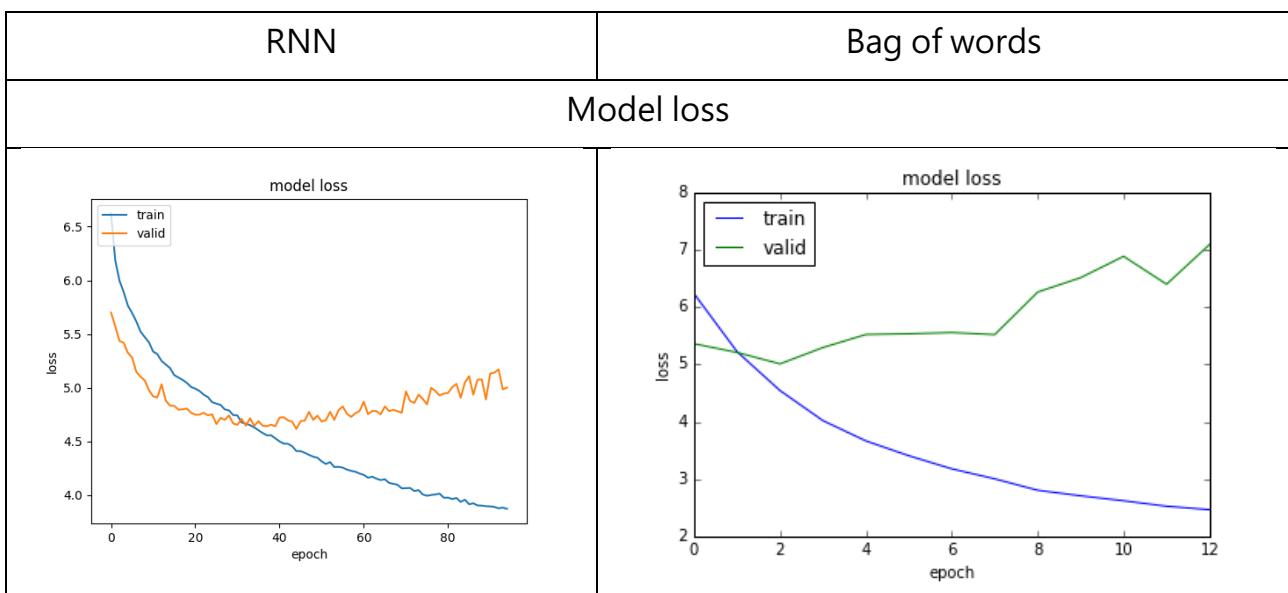
(RNN model summary)

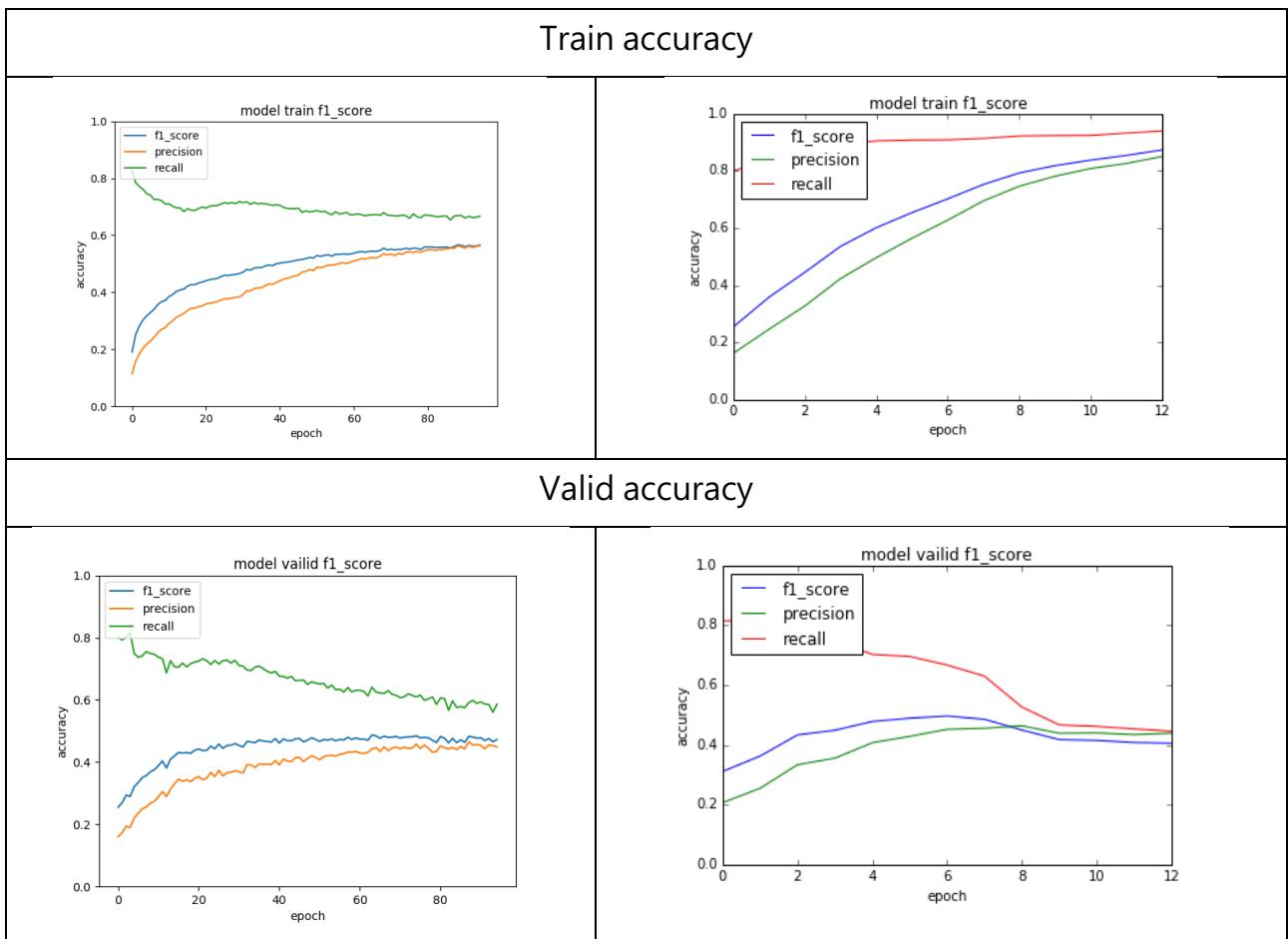
Bag of words model:

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 1024)	55059456
p_re_lu_1 (PReLU)	(None, 1024)	1024
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
p_re_lu_2 (PReLU)	(None, 512)	512
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
p_re_lu_3 (PReLU)	(None, 256)	256
dropout_3 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32896
p_re_lu_4 (PReLU)	(None, 128)	128
dropout_4 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 38)	4902
Total params: 55,755,302		
Trainable params: 55,755,302		
Non-trainable params: 0		
Train on 4467 samples, validate on 497 samples		

(Bag of words model summary)

結果：





兩者之間最大的差異在於 training 的速度

RNN 約 27 秒一個 epoch

Bag of words 約 4 秒一個 epoch

RNN 約到第 70 幾 epoch 才會到達 valid f1 measure 高峰

Bag of words 10 個 epoch 內就到達 valid f1 measure 高峰

經由測試 RNN 最後在 kaggle 上的分數是 public: 0.52281 / private: 0.49702

Bag of words 最後在 kaggle 上的分數是 public: 0.51808 / private: 0.49466

RNN 比 Bag of words 略好一些。