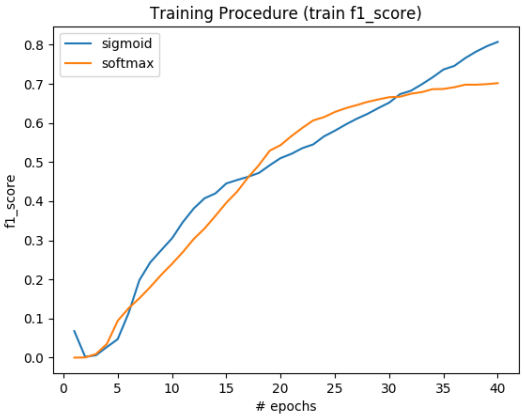
學號：B03705028 系級： 資管三 姓名：陳星宇

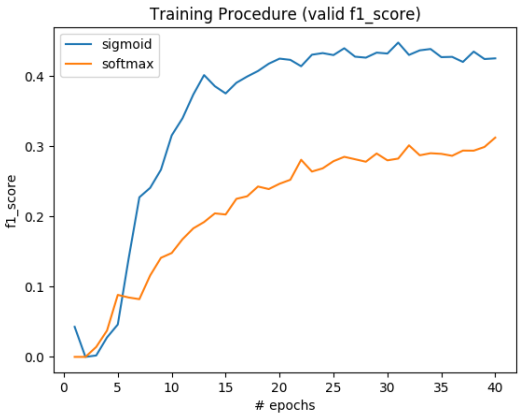
1. (1%)請問 softmax 適不適合作為本次作業的 output layer? 寫出你最後選擇的 output layer 並說明理由。

我認為 sigmoid 較適合作為本次作業 output layer 的 activation function。softmax 使得 output vector 的和為 1，隱含的意義是只有一個 class，被選為此篇文章的 label，較適合 multi-class and one-label 的情境。而 sigmoid 會使得 output vector 每一維的值都在 [0, 1] 這個區間，分別代表此篇文章屬於每一個 class 的機率，較適合本次作業 multi-class and multi-label 的情境。

2. (1%)請設計實驗驗證上述推論。

下圖為僅變更 output layer 的 activation function，兩個 model 的訓練過程。





由上圖可以發現，雖然兩者在 training f1_score 的表現差不多，但我們在乎的是其在 validation data 上的 f1_score，這時就可以發現 sigmoid 的表現明顯較佳。

3. (1%)請試著分析 tags 的分布情況(數量)。

下圖為每一個 tag 的名稱以及其佔所有 tags 數量的比例，以所占比例由高到低排序。由圖中可以發現，tags 出現的頻率相差懸殊，舉例來說，出現頻率最高的 FICTION 的頻率是出現頻率最低的 UTOPIAN-AND-DYSTOPIAN-FICTION 整整 162 倍，而出現頻率前十高的 classes 的百分比和，已經達到 80%，等於剩下 28 個 classes 只能瓜分剩下的 20%。為了解決 imbalanced classes 的問題，我先利用 sklearn.utils 算出 balanced class_weight，並在 model.fit() 的時候傳入 class_weights 的參數，使出現頻率較低的 class 不會直接被忽略，出現頻率較高的 class 也不會 dominate 預測結果。

Index	Tag	Percentage	Cumulative Percentage
1	FICTION	0.162756740971	0.162756740971
2	SPECULATIVE-FICTION	0.140952010124	0.303708751095
3	NOVEL	0.0965638080405	0.400272559136
4	SCIENCE-FICTION	0.0933515039424	0.493624063078
5	CHILDREN'S-LITERATURE	0.0756351601285	0.569259223206
6	FANTASY	0.0752457899348	0.644505013141
7	MYSTERY	0.0624939160907	0.706998929232
8	CRIME-FICTION	0.0358220578215	0.742820987053
9	SUSPENSE	0.0309549304001	0.773775917454
10	YOUNG-ADULT-LITERATURE	0.0280346539472	0.801810571401
11	THRILLER	0.023654239268	0.825464810669
12	HISTORICAL-NOVEL	0.021610045751	0.84707485642
13	HORROR	0.0186897692982	0.865764625718
14	DETECTIVE-FICTION	0.0173269736202	0.883091599338
15	ROMANCE-NOVEL	0.0152827801032	0.898374379441
16	HISTORICAL-FICTION	0.0133359291346	0.911710308576
17	ADVENTURE-NOVEL	0.0106103377786	0.922320646355
18	NON-FICTION	0.00992893993965	0.932249586294
19	SPY-FICTION	0.00730069113209	0.939550277426
20	ALTERNATE-HISTORY	0.00700866348681	0.946558940913
21	COMEDY	0.00574321035725	0.95230215127
22	AUTOBIOGRAPHY	0.00496446996982	0.95726662124
23	BIOGRAPHY	0.00408838703397	0.961355008274
24	SHORT-STORY	0.00399104448554	0.96534605276
25	HISTORY	0.00389370193712	0.969239754697
26	COMIC-NOVEL	0.00360167429183	0.972841428989
27	MEMOIR	0.00340698919498	0.976248418184
28	SATIRE	0.00340698919498	0.979655407379
29	AUTOBIOGRAPHICAL-NOVEL	0.00301761900127	0.98267302638
30	WAR-NOVEL	0.00301761900127	0.985690645381
31	DYSTOPIA	0.00292027645284	0.988610921834
32	NOVELLA	0.00282293390441	0.991433855738
33	HUMOUR	0.0017521658717	0.99318602161
34	TECHNO-THRILLER	0.0017521658717	0.994938187482
35	HIGH-FANTASY	0.00146013822642	0.996398325708
36	APOCALYPTIC-AND-POST-APOCALYPTIC-FICTION	0.00136279567799	0.997761121386
37	GOTHIC-FICTION	0.00116811058114	0.998929231967
38	UTOPIAN-AND-DYSTOPIAN-FICTION	0.00107076803271	1.0

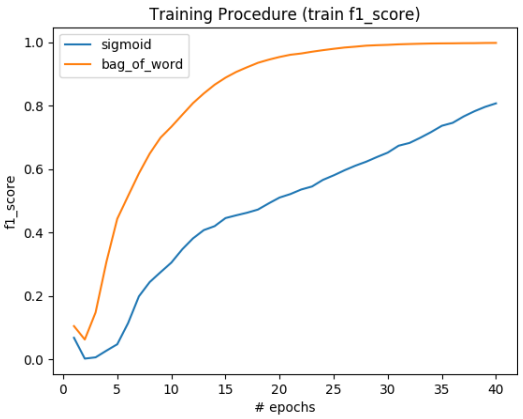
4. (1%)本次作業中使用何種方式得到 word embedding?請簡單描述做法。

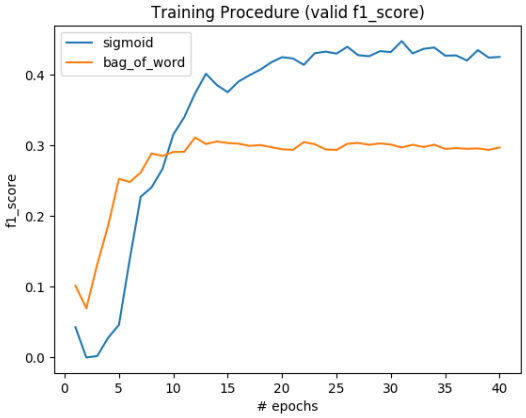
此次作業我以 count based 的 glove 達成 word embedding。

1. 以 keras 的 tokenizer 將每篇文章轉成 word index sequence
2. 將代表每一篇文章的 sequences pad 到相同長度
3. 以 glove.42B.300d 得到 word index 到 word vector 的 mapping
4. 將 model 的第一層 layer 設為 Embedding，傳入 sequences 作為 feature，再以 glove 的 weights 將 sequences 轉換為 vectors

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

我本來將 bag of word 跟 RNN 的 trainable 參數量調整到差不多，但這樣做出來，bag of word 的結果實在慘不忍睹，所以就稍微將 bag of word 的參數量增多，以下是兩個 model 的訓練過程。





由上圖可以發現，雖然 bag of word 的 training f1_score 進步得很快，但在 validation data 上的 f1_score 的 upper bound 明顯比 RNN 低，可能與 RNN 相比較容易有 overfitting 的情況。然而，這兩個 model 都沒加上 dropout layers，如果幫 bag of word model 加上 dropout layers，可能會使它在 validation data 上的 f1_score 有所提升。