學號：B03902042　　系級：資工三　　姓名：宋子維

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。

   首先，我的模型架構為將輸入的 token 轉換為 embedding 後，丟進 RNN，最後經過數個 dense layer 後輸出 38 維向量，因此，將書摘輸入，模型輸出向量的第 i 維為「是否在 class i 的機率」，其中 $i = 1, 2, \ldots, 38$，詳細模型架構如下：

```
model = Sequential()

model.add(Embedding(
      vocab_size,
            embedding_size,
            input_length=maxlen,
            weights=[embedding],
            trainable=False))

model.add(GRU(128, activation='tanh', dropout=0.3))

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(38, activation='sigmoid'))

model.compile(loss='categorical_crossentropy', optimizer='adam')
```

   softmax 並不適合作為這次作業的 output layer，因為 softmax 其實是對 output 做 normalization，使得最終的 38 維的輸出向量中，每個元素都落在 $[0, 1]$ 之間，且總和為 1，可以將其想像為，給定書摘，被「分類到 class i 的機率」。因此若使用 softmax 作為 output layer，當一筆書摘有兩個以上的 tag，Gradient Descent 想要最大化分類到其中一個 tag 的機率時，由於總和為 1，會壓縮到分類到其他 tag 的機率。

   因此，我最後選擇 sigmoid 作為 output layer。

2. (1%) 請設計實驗驗證上述推論。

我用了兩個和上述代碼相同架構的模型，但一個的 output layer 為 sigmoid，另一個為 softmax，並將他們的訓練過程中的 loss 和 F1 score 畫出來，得到 Figure 1圖表。從中可以發現，在相同 epoch 數下，雖然兩者的 loss 是差不多的，但很明顯的，在 F1 score 方面，以 sigmoid 做為 output layer 的模型優於以 softmax 為 output layer 的模型，前者的 F1 score 最高能到達 0.5，但後者卻只在 0.2 上下震盪。

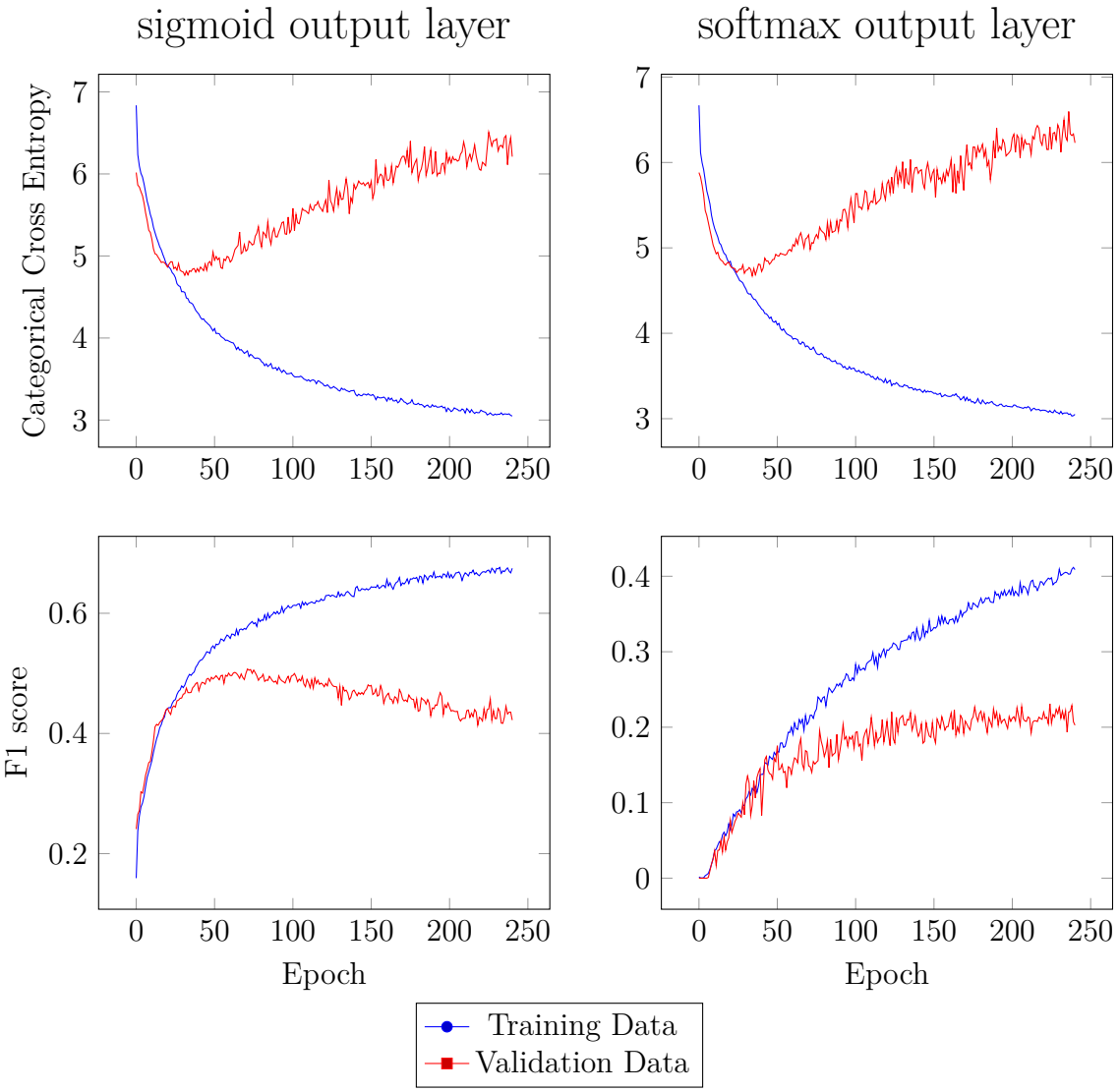

Figure 1: sigmoid 和 softmax output layer 訓練過程。

3. (1%) 請試著分析 tags 的分布情況 (數量)。

從 Figure 2能看出，tag 的分布非常極端，多則出現一千多次，如 FICTION、SPECULATIVE-FICTION；少則出現僅僅十多次，如 UTOPIAN-AND-DYSTOPIAN-FICTION、GOTHIC-FICTION 等。而 Figure 3中顯示出單筆書摘所擁有的 tag 數量，可以看出在訓練資料中，書摘最多有 8 筆 tag，然而，大部分的書摘有一至三個 tag。

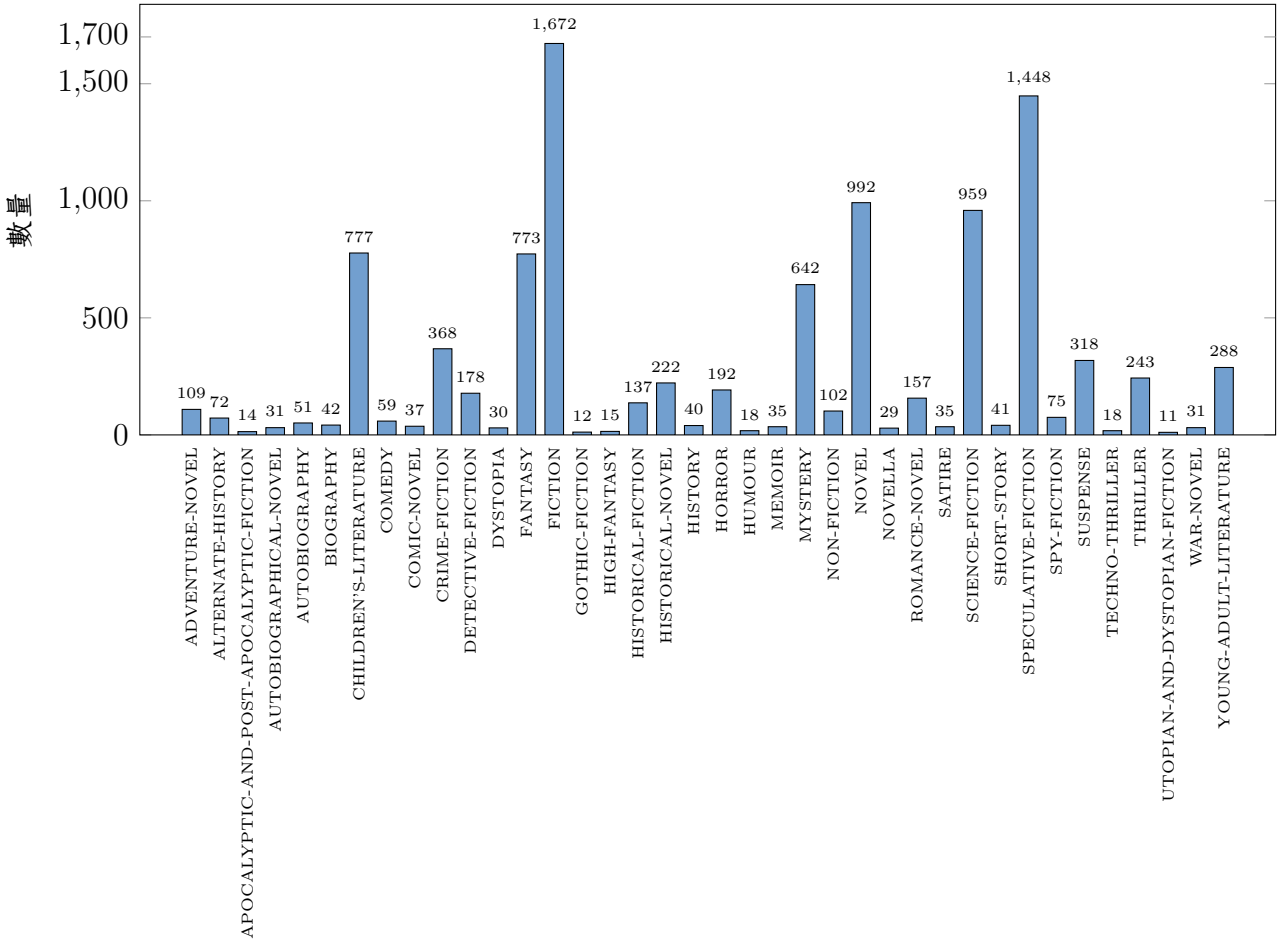

Figure 2: Tag 分布。(按照字母序排序)

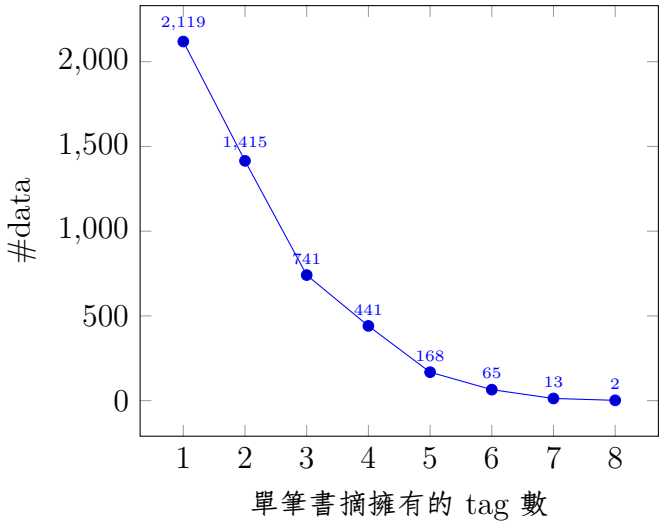

Figure 3: Tag 數與 data 量。

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

   我採用的是 pre-trained GloVe 100 維 word vectors，將其讀入後，放在 `keras` 的 `Embedding` layer 作為初始化的值，並在訓練過程中不更新它。

   GloVe 為一種 count-based 的作法，主要想法是，若 w_i 和 w_j 時常一起出現，則兩者的 word vector v_i 和 v_j 應該要越相近，故 GloVe 在訓練前，首先算出訓練資料的全域統計資訊 - **Co-occurence matrix** X，其中 X_{ij} 為 w_i 和 w_j 在 context window 中一起出現次數 (有加權過，離越遠的 word pair 會有比較小的權值)，在訓練過程中將 v_i 和 v_j 的內積值向 $\log X_{ij}$ 拉近。

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

   我使用的 bag of words 模型為 TF-IDF 加上 `LinearSVC`(Linear Support Vector Classifier)，詳細作法如下：

   - 將每段書摘用 TF-IDF 轉換成一個 feature vector。
   - 透過 One-Vs-All 策略，將 multilabel classification 轉換為 binary relevance 的問題。
   - 用 `LinearSVC` 進行分類。

   而最快速的做法即為透過 `sklearn` 套件建構模型，`Python` 代碼如下：

```python
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer

model = Pipeline([
  ('vectorizer', TfidfVectorizer(stop_words='english')),
  ('clf', OneVsRestClassifier(LinearSVC(C=0.0005, class_weight='balanced')))])
```

   從 Figure 2可以知道 tag 是非常不平衡的，因此若採用 `LinearSVC` 進行分類，必須加上 `class_weight='balanced'` 的選項，讓分類器依據 positive 和 negative 標籤數量調整 penalty `C`。

   至於 RNN 的模型，我是採用和第一題中的代碼一樣的架構。

   Table 1為 TF-IDF 和 RNN 在 public score 的表現，可以看到 TF-IDF 不管在分數抑或時間上都比 RNN 來得好，我想可能是訓練資料對 RNN 來說並不算多，且在 RNN 的模型架構中，我並沒有特別處理 tag 不平衡的情況，這兩點或許是造成 RNN 略遜於 TF-IDF 加 `LinearSVC` 的原因。

   若仔細分析他們在測資上的輸出，可以發現 TF-IDF 和 `LinearSVC` 的組合，在 1234 筆測資上有多達 61 筆沒有預測任何的 tag，而 RNN 僅有 2 筆未預測。為了修正未預測任何 tag 的情況，和增加模型的穩定性，我最終的模型為用多個 RNN (每個都只用 90% 的 training data 訓練) 和多個 TF-IDF (每個用的 feature 數量不同) 模型做 ensemble learning，採用的策略為 majority voting，若在 voting 後仍然有測資沒有任何預測值，則對該測資用單一個 RNN 模型的預測值做為輸出。

Model	Time	Public Score
RNN	30 minutes	0.50820
TF-IDF	30 senconds	0.51452

Table 1: Bag of words 和 RNN 在 public score 的比較。