學號：R05921033 系級： 電機所碩一 姓名：余政穎

**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。**

Ans:

首先，我先簡述除了 output layer 以外我的模型架構，資料處理的部份，先將 training data 用 Tokenizer 編碼，並用 padding sequence 使其長度相同，並將字詞出現頻率介於 4~4999 之間的字詞保留下來，其餘字詞去掉忽略不計，把必要的字詞保留(出現很少的對於預測幫助不大，出現很多的如 the, a, I 這類亦對預測無幫助的字詞)；第一層先使用 pre-trained 好的 word embedding vector (200 維)，輸入兩層大小為 256 的 GRU layers(activation function 為 ReLu，dropout 率為 0.5)，接著依序接大小為 512、256、128、64 的 DNN(activation function 為 PReLu，類似 LeakyReLu，只是 alpha 是動態調整的，dropout 率為 0.3)，並接到 output layer，output layer 的 threshold 出來若大於 0.4，則預測有該值對應的 class，若無則沒有，目標進行 multi-label and multi-class 的預測。

softmax 所代表的意義是，對於所有的 output，將 output layer 的值經過轉換後，使其加起來為 1，用來代表每一個 class 的可能的機率，對於 multi-label task 來說，由於一筆 test data(在此處即是一篇短文)，可能屬於多個不同的 class，用 softmax 所產生出來的結果，會使得最後出來僅有 1~2 個 class(在 threshold 設為 0.4 的情況下)，即使調低 threshold，由於總共有 38 種不同的 class，加上 softmax 會將越可能的 class 的值放大(假如原本兩筆 data 前一層出來是 1 跟 2，經過 softmax 後一個分子是 $e$、另一個是$e^2$，會差到 $e$ 倍)，預測出的 class 機率會互相影響，要達成準確的估計並不容易，故結論是，softmax 並不適合做為 output layer。

因此，我最後使用 sigmoid 作為 output layer，因為其輸入值只要夠大，output 要就可以大於 threshold，並不會受到其他 class 影響，對於此作業所需要達成的目標，multi-label and multi-class task 是頗為合適的，而最後跑出的結果，也確實不差，能夠在 kaggle 上的 leaderboard 達到 f1 score 分別為 0.52388 跟 0.50897。

**2. (1%)請設計實驗驗證上述推論。**

Ans:

為了確認第一題中所敘述的推論是合理的，基本上我以第一題的架構下去實作，但因為加兩層 GRU 對於 softmax 會訓練不起來，所以只使用一層 GRU，將 output layer 以外的所有參數固定進行訓練，並觀察 validation 的準確率、validation 的 loss、上傳 Kaggle 的結果。

| 最後一層 Activation | Validation f1 | Validation loss | Public f1 | Private f1 |
|---|---|---|---|---|
| softmax | 0.2031 | 4.8071 | 0.18372 | 0.14631 |
| sigmoid | 0.5389 | 4.5237 | 0.52388 | 0.50897 |

可以看到在各種數據上，softmax 當最後一層 activation，都比使用 sigmoid 當最後一層 activation 差很多，而且在越複雜的模型下，softmax 會訓練越不好(兩層 GRU 完全訓練不起來)；接著我們再檢查 predict 出來，於兩種模型下 test data 預測出平均的 tags 數量，以及原本 training data 中，每筆資料的 tags 數量平均個數。

| 最後一層 Activation | 原本 training data，tags 平均數量 | 預測 test data，tags 平均數量 |
|---|---|---|
| softmax | 2.0695 | 0.18785 |
| sigmoid | 2.0695 | 2.58219 |

我們可以很明顯的看到，使用 softmax 作為最後一層，預測出來的 test data 的 tags 數量非常少，相較於 sigmoid 還預測太多(表示應該要調低 threshold)，其 tags 數量預測的少的誇張，除非要把 threshold 設很低，但這樣又同時可能把太多不相關的也算進來，設定並不容易，因此 softmax 並不適合作為最後一層的 activation。

**3. (1%)請試著分析 tags 的分布情況(數量)。**

Ans:

這是我統計訓練資料中，每一個 tag 的分布狀況：

| Tags | HUMOUR | NOVEL | CRIME-FICTION | SUSPENSE |
|---|---|---|---|---|
| 數量 | 18 | 992 | 368 | 318 |
| Tags | HORROR | FANTASY | FICTION | NON-FICTION |
| 數量 | 192 | 773 | 1672 | 102 |
| Tags | UTOPIAN-AND-DYSTOPIAN-FICTION | SPECULATIVE-FICITON | TECHNO-THRILLER | CHILDREN'S LITERATURE |
| 數量 | 11 | 1448 | 18 | 777 |
| Tags | AUTOBIOGRAPHY | SPY-FICTION | SATIRE | COMEDY |
| 數量 | 51 | 75 | 35 | 59 |
| Tags | YOUNG-ADULT-LITERATURE | ROMANCE-NOVEL | SCIENCE-FICTION | HISTORICAL-FICTION |
| 數量 | 288 | 157 | 959 | 137 |
| Tags | COMIC-NOVEL | SHORT-STORY | HIGH-FANTSY | BIOGRAPHY |
| 數量 | 37 | 41 | 15 | 42 |
| Tags | MEMOIR | MYSTERY | HISTORY | NOVELLA |
| 數量 | 35 | 642 | 40 | 29 |
| Tags | DETECTIVE-FICTION | GOTHIC-FICTION | ALTERNATIVE-HISTORY | HISTORICAL-NOVEL |
| 數量 | 178 | 12 | 40 | 222 |
| Tags | DYSTOPIA | THRILLER | WAR-NOVEL | ADVENTURE-NOVEL |
| 數量 | 30 | 243 | 31 | 109 |
| Tags | AUTOBIOGRAPHICAL-NOVEL | | APOCALYPTIC-AND-POST-APOCALYPTIC-FICTION: | |
| 數量 | 31 | | 14 | |

可以看到，這裡面最多數量的 tags 是 FICTION，共 1672 個，最少數量的 tags 是 UTOPIAN-AND-DYSTOPIAN-FICTION，共 11 個，觀察整個 tags 的分布狀況，不難看出這些訓練資料中，tags 的數量差異甚大，表示有些資料我們其實並不容易讓模型辨認出來，因為本身有這個 label 的訓練資料就不夠多，要求模型在 test data 中猜出這些資料是屬於這個 label，其實是滿不合理的，應該設法平衡一下這樣的狀況。

同時，觀察 tags 的名稱，亦可看出有些 tag 彼此之間有高度相關性，甚至有些 tag 可說是屬於另一個 tag 之下，比方說：UTOPIAN-AND-DYSTOPIAN-FICTION 跟 DYSTOPIA 有高度相關、ADVENTURE-NOVEL 應屬於 NOVEL 之下，然而，在 labels 上這兩項直接相關的不太會被重複 tag 到，即使這兩者對於模型來說是有一定的相似性，故提升辨認的難度。

綜上所述，如果直接照訓練資料直接丟進去，而沒經過特別的處理，要有好的辨識度其實並不容易，若未來要真正使用機器學習解決問題，一些 data 跟 features 前處理是必要的。

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

Ans:

此次作業我使用 GloVe 訓練好的 model 得到 word embedding model，使用的為 200 維度的 Wikipedia 2014 + Gigaword 5 pre-trained model。轉換資料的過程依序為：使用 tokenizer.fit_on_texts 將每個字詞編號、用 tokenizer.texts_to_sequence 將每個文章轉成以數字表示的 sequence、用 pad_sequence 把上一步所得到的 sequence 調整成一樣長度、用 Glove 200 dimension 建立 embedding matrix、將 embedding matrix 和 sequence 放入 Embedding layer，把第三步得到的 sequence 投影到 200 維的空間上。

GloVe 所使用的訓練方式是根據 word-word co-occurrence matrix，若以矩陣X代表這個 matrix，$X_{ij}$代表的就是 word j 在 word i 的上下文中所出現的次數，$X_i = \sum_k X_{ik}$, $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$，後者表示 word j 在 word i 中出現的機率，亦可稱為兩個單詞之間的 co-occurrence 的機率。對於上述矩陣X進行分析$P_{ik}$與$P_{jk}$，可以知道對於一個固定的單詞 k，word i 與 word j 之間的比例關係，用來區分單詞之間彼此的相關性，以及大略知道相關詞之間的涵義為何。

至於 GloVe 中的訓練目標為讓兩個單詞的 vector 彼此之間的內積，等同於這兩個單詞的 co-occurrence 的機率取 log，會取 log 的原因為當我們分析$P_{ik}$與$P_{jk}$時，我們是看其比例(亦即$\frac{P_{ik}}{P_{jk}}$)，若將其取 log，便可以將其轉換為兩個 log term 的相減，如此一來，兩個單詞向量之間的相似性，便會與兩個單詞之間的相似性有正相關，使得其訓練出來的 word vector，彼此之間的相似性及類比能力得以被保留下來，此外，GloVe 的計算複雜度主要依賴於 X 矩陣中的非零項，故只要此項的數量不太大，計算複雜度也在可接受範圍以內。參照該篇 paper 的結論，由於 GloVe 的模型有使用到 word 之間彼此出現的次數，但同時又能保持單詞之間的 substructure，使其產生的 vector 能代表字的意義，基本上是融合了 count-base 的 prediction-base 模型。

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

Ans:

我將字詞出現頻率介於 4~4999 之間的字詞保留下來，其餘字詞去掉忽略不計，接著統計每筆資料中出現字詞的數量，以建立相應的 bag-of-word 的 vector，刪去字詞的目的是使 bag-of-word 建出來的 matrix 的 non-zero element 數量不會太多，同時也能把必要的字詞保留(出現很少的對於預測幫助不大，出現很多的如 the, a, I 這類亦對預測無幫助的字詞)。接著，由於不需要 RNN 的 layer，我便把第一層的 word embedding layer 和前兩層的 GRU 拿掉，改成接一層大小為 1024 的 DNN layer，此 layer 所使用的 activation function 為 PReLu，所有 DNN layer 的 dropout 率皆為 0.3，以減少 overfit 的狀況，其他設定同第一題。

以上述模型跑此次實驗，與第一題所跑出的模型，觀察 validation set 的正確率，以及丟上去 Kaggle 的 score(在 Kaggle deadline 後測，所以可得到 public 跟 private 的分數)

| 模型 | Validation f1 | Public f1 | Private f1 |
|---|---|---|---|
| Bag-of-Word | 0.4797 | 0.49563 | 0.46343 |
| RNN | 0.5389 | 0.52388 | 0.50897 |

從上表可知，Bag-of-Word 的結果比較不好，其實觀察訓練的過程，可以發現相較於 RNN，Bag-of-Word 模型的 training data 的 f1 score 會急遽上升，表示在這架構下 Bag-of-Word 容易 overfit，推測可能的原因是 Bag-of-Word 只考慮有什麼詞，而未考慮字跟字之間的語意，因此容易把某個字詞直接分類到某種 tag 的 novel 底下，但有些字詞可能是在多個 tags 中皆容易出現的，故可知，一般來說，RNN 模型在本次作業的目標上，還是比 Bag-of-Word 的模型來的好。