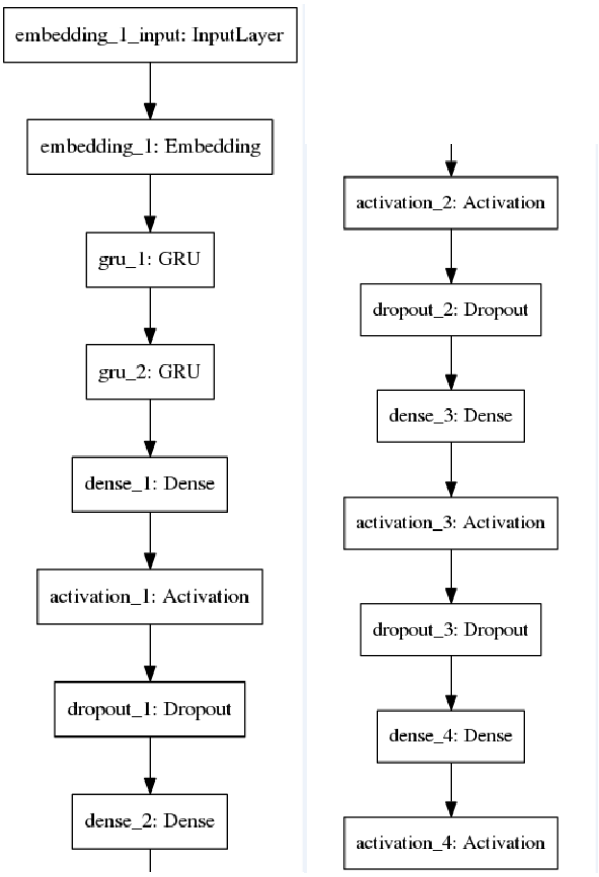
學號：B03505027　系級：電機三　姓名：劉亦浚

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。



(RNN model 架構)

經過 Softmax 函數，會使得 output 中每一個元素的範圍都在(0, 1)之間，且全部元素相加和為 1。此次作業因有 multi-label，所以 y_train(38dim) 的形式為[0, 1, 1, 0, … , 0, 0, 1]，若有相對應的 label 標為 1，無相對應的則標為 0，其元素相加和超過 1，所以此次作業使用 Softmax 則無法有效讓 output 趨近於 y_train，由此可知 Softmax 適用於單一 label 的 y_train (如[0, 0, 1, 0, 0])。

最後選擇 sigmoid 當作 output layer，其輸出在(0, 1)之間，但不限定元素相加和為 1，所以可多個元素的值可較高，其概念與 multi-label 較相符，因此可使 output 與 y_train 較相近。

2. (1%)請設計實驗驗證上述推論。

使用一層 GRU，三層 relu，其 neuron 數目相同，output layer 分別使用 softmax 與 sigmoid。觀察 validation data 的 predict 結果與 f1_score。

Softmax：

```
[  2.71946881e-02   2.49065727e-01   1.35570988e-01   4.53818124e-04
   4.12861675e-01   1.30307510e-01   4.94381283e-13   2.74979379e-06
   5.27019656e-05   5.29033276e-11   4.49811921e-07   1.69816001e-08
   1.74996076e-05   3.11425961e-14   1.11193259e-14   1.29561517e-09
   3.30588664e-04   3.36399826e-05   1.02148465e-06   5.30867965e-07
   9.62810276e-10   2.47334264e-10   5.44106129e-08   8.07443237e-14
   3.08651477e-04   1.23968002e-13   8.03093113e-15   2.45259728e-14
   8.09698337e-13   2.26891483e-09   1.02523090e-05   1.18333295e-13
   1.63128497e-10   1.90890545e-03   1.13840541e-03   1.42354939e-09
   5.57645490e-06   4.07344624e-02]
[  1.23095595e-06   1.77072256e-03   4.35153902e-01   5.52207172e-01
   2.68607550e-07   1.01598874e-02   1.16754061e-04   6.76500531e-06
   5.60213948e-07   2.08515739e-11   1.74619436e-06   3.66128484e-06
   1.40564282e-10   2.46129292e-14   2.45163445e-08   1.87503724e-06
   1.73196568e-08   4.29410749e-04   1.99745205e-06   1.09289759e-08
   2.26385073e-13   1.05706146e-14   4.88943108e-08   9.97589706e-13
   8.62716565e-09   1.11048414e-10   1.12744473e-10   1.95002858e-09
   1.56726415e-07   4.26826518e-05   6.51726870e-17   7.07547032e-20
   1.76066237e-17   1.01089630e-04   4.48341676e-11   2.77892130e-14
   2.32403825e-21   4.37578092e-20]
[  2.04541777e-07   7.78776768e-04   1.84317648e-01   8.11479747e-01
   4.95598783e-07   3.14922181e-06   1.17398549e-05   3.67927477e-12
   2.79123918e-03   9.83416953e-07   1.04915466e-06   1.46860532e-06
   1.68408856e-10   4.28189928e-10   7.62615482e-09   4.23579419e-04
   3.41976165e-12   5.50945551e-05   6.50304521e-09   3.53868546e-10
   1.03955095e-07   6.51774065e-15   9.76769776e-10   1.41011495e-14
   1.34661459e-04   3.70098685e-09   1.98392778e-11   8.76095818e-09
   8.69207928e-10   1.93326954e-08   1.94072758e-13   3.04667597e-18
   1.67312631e-17   3.42214124e-09   9.49634861e-14   4.47398402e-17
   1.78775350e-23   7.79836538e-22]
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0]
[0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0]
[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0]
```

隨機挑三個 x_val 來做 predict 並且與其 y_val 做比較(方框為 predict 結果)，發現 label 為 1 的相對應 predict 結果無法同時大於其他 tags predict 的值，與第一題相符，因其元素和相加為 1，無法同時存在許多較大的值，且通常只有一個值較大，所以應用在單一 label 較為恰當。

其 val_f1_score 只有 0.2613。

```
- loss: 2.9286 - f1_score: 0.4903 - val_loss: 6.2282 - val_f1_score: 0.2613
```

Sigmoid：

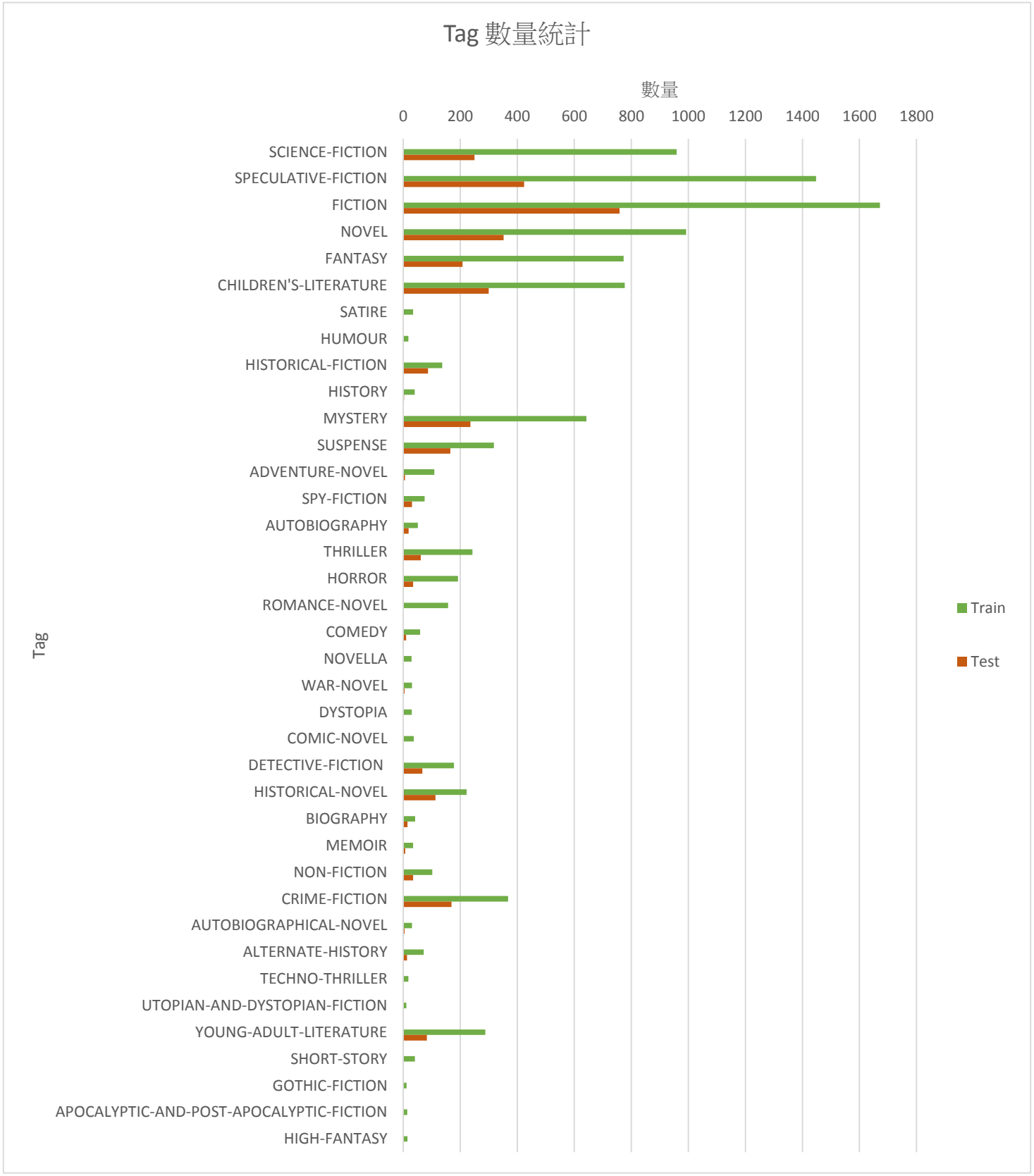
```
[  1.48165459e-03   4.17741872e-02   3.58466983e-01   2.27293268e-01
   1.09975748e-02   4.15536493e-01   1.05366502e-02   5.49099892e-02
   3.11706960e-02   1.08190603e-03   1.09933484e-02   6.44995365e-03
   5.51102217e-03   1.45637969e-05   9.10754595e-03   1.68341189e-03
   4.26035025e-04   3.56338322e-02   3.75929698e-02   2.46868376e-02
   8.45187024e-05   1.22237261e-05   1.78876910e-02   2.69687356e-04
   2.62030656e-03   6.15613628e-03   3.91301140e-03   3.03895387e-04
   1.11202826e-03   4.22515422e-02   8.17437194e-06   1.17282390e-07
   4.04385929e-07   4.94819283e-01   3.27745639e-02   1.04347698e-03
   4.03290096e-06   5.96245309e-06]
[  3.28797032e-04   1.92563869e-02   5.42938888e-01   1.76906019e-01
   3.96914293e-05   3.75239807e-03   1.66967395e-04   1.33581361e-05
   1.04118073e-04   6.39521787e-08   8.83319318e-01   4.39896762e-01
   3.01022665e-05   3.87595198e-03   1.25714763e-08   6.04706332e-02
   6.92743197e-06   7.02630268e-06   2.35441985e-05   3.89355712e-08
   2.37277291e-08   5.97514713e-08   2.72416015e-04   3.32368195e-01
   1.52145556e-04   6.16617157e-08   1.24432395e-10   9.10854396e-08
   4.31479812e-01   2.04057983e-07   2.14621121e-08   3.89773686e-06
   1.10659680e-12   2.01646235e-05   9.14607323e-09   7.26409041e-07
   6.41024109e-11   2.89251587e-14]
[  1.28773246e-02   2.74196774e-01   5.09127676e-01   2.13327989e-01
   4.75930795e-02   2.22216602e-02   5.28341625e-03   2.55181035e-03
   3.05474252e-01   5.01511879e-02   3.32332671e-01   3.59529763e-01
   5.49195940e-03   7.96433073e-03   4.68105573e-04   2.02300280e-01
   5.29241450e-02   3.46339941e-01   6.92330720e-03   2.19359342e-02
   3.75971347e-02   4.99755086e-04   1.92409474e-03   3.00453277e-03
   5.25148392e-01   3.60395387e-03   4.31687688e-04   3.42691143e-04
   1.24388613e-01   5.34337992e-03   6.21336699e-03   1.05429674e-03
   8.33753438e-05   3.79398023e-03   1.58535072e-03   5.02151670e-03
   4.07657266e-04   5.62094610e-05]
[0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0]
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0]
```

隨機挑三個 x_val 來做 predict 並且與其 y_val 做比較(方框為 predict 結果)，可明顯發現 label 為 1 的相對應 predict 結果值都頗大，比其他 tags predict 的值大出許多，與第一題相符，不限制相加和為 1，可同時存在多個較高的值，其預測結果與 multi-label 較為相符。

其 val_f1_score 有 0.5091，高於 softmax 的分數許多。

```
- loss: 3.4997 - f1_score: 0.6181 - val_loss: 5.5264 - val_f1_score: 0.5091
```

3. (1%)請試著分析 tags 的分布情況(數量)。



可看出大範圍的分類如 FICTION(虛構)、NOVEL(夾雜寫實與虛構)等都有大量文本屬於這些 tag，而較主流且熱門的種類如 SCIENCE、SPECULATIVE、FANTASY、MYSTERY 等 tag 數量也很多；較冷門的 tag 如 GOTHIC(哥德次文化)、DYSTOPIA(反烏托邦)等較少文本屬於此類；我在設定 thresh 時採用 Matthews correlation coefficient(y_val、y_pred)來自動調整，最後發現數量較多的 tag 擁有的 thresh 較高；反之較少數量的 tag 擁有較低的 thresh。

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

使用別人 train 好的 glove.6B.200d.txt 得到 word embedding。Glove 是屬於 count-based 的方法，透過收集大量的文本，觀察 word 與 word 之間是否常一起出現，若時常一同出現，表示兩個 word 有較高的關係，其 word 的 vector 也會較相近(內積較大)。利用一個 word co-ocurrence matrix 記綠 word 與其他 word 同時出現的關係，在此 word 前後放上固定的 window_size，收集此範圍內的資料，且離此 word 越遠的 word 也會乘上較小的 weight，更能區分 words 之間的關係程度。

$$w_i^T w_j + b_i + b_j = log(X_{ij})$$

兩兩 word 的 constraints，w 為 vector 參數，b 為 bias

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

調整參數使之越接近 constraints，其 cost function 也越低

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

使用 RNN 的效果較好。

Model 架構：

bag of word 後面接 DNN(三層 relu、output layer 用 sigmoid)。

數據：

```
loss: 3.6433 - f1_score: 0.6193 - val_loss: 5.0681 - val_f1_score: 0.4790
```

分析：

採用 bag of word 每一個 epoch 訓練的非常快速，因為訓練過程與 RNN 不同，沒有時序的概念，且前幾個 epoch 分數上升很快，不像 RNN 前期分數提升很慢，但使用 bag of word 也會很快停在分數 0.47~ 0.48 左右，RNN 最後則會停在 0.51 左右。

結論：

使用 bag of word 可以很快訓練出一個 model；RNN 則相當的耗時，但以準確的角度來看，還是使用 RNN 能得到較好的結果。