

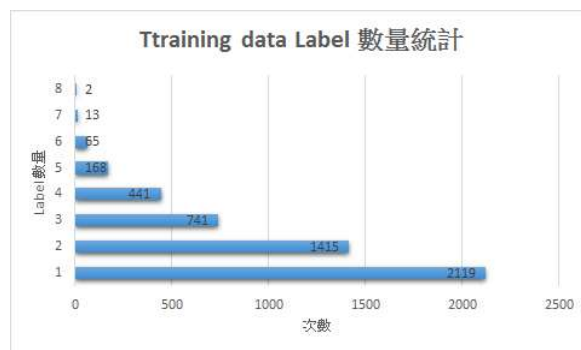
1. (1%)請問softmax適不適合作為本次作業的output layer? 寫出你最後選擇的output layer並說明理由。

- ❖ 不適合
- ❖ 使用sigmoid
 - 因為softmax對應到的output是相加為1，而sigmoid是每一個對應到0~1，本題為multi label multi class，每一個label被選到的機率為0~1比較make sense，若使用softmax的話threshold會比較難調(每篇文章的label數量不一樣)。
 - 若文章X的label數量為K個，則softmax threshold必須 $<1/K$ 才有可能準確預測
 - 不設一個很小的threshold的原因: 在預測時並不會那麼精準，若threshold設太小可能會造成一些錯誤的label被輸出

```
model = Sequential()
model.add(Embedding(num_words,
                    embedding_dim,
                    weights=[embedding_matrix],
                    input_length=max_article_length,
                    trainable=False))
model.add(GRU(128, recurrent_dropout=0.25, dropout=0.25))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(38, activation='sigmoid'))
```

2. (1%)請設計實驗驗證上述推論。

- ❖ 下表為Label數量統計，由於softmax是總和為一，所以在不知道數量的情況下沒辦法設出一個好的threshold



● threshold : 0.4

```
Epoch 10/40: 7e - loss: 1.3098 - f1_score: 0.7397 - precision: 0.7863 - recall: 0.6927
Epoch 11/40: 7e - loss: 1.3094 - f1_score: 0.7248 - precision: 0.7911 - recall: 0.6983
Epoch 12/40: 7e - loss: 1.3090 - f1_score: 0.7240 - precision: 0.7961 - recall: 0.6973
Epoch 13/40: 7e - loss: 1.3088 - f1_score: 0.7221 - precision: 0.7892 - recall: 0.6980
Epoch 14/40: 7e - loss: 1.3091 - f1_score: 0.7380 - precision: 0.7782 - recall: 0.6945
Epoch 15/40: 7e - loss: 1.3037 - f1_score: 0.7205 - precision: 0.7804 - recall: 0.6961
Epoch 16/40: 7e - loss: 1.4977 - f1_score: 0.7235 - precision: 0.7840 - recall: 0.6994
Epoch 17/40: 7e - loss: 1.4945 - f1_score: 0.7228 - precision: 0.7806 - recall: 0.6994
Epoch 18/40: 7e - loss: 1.5888 - f1_score: 0.7308 - precision: 0.7746 - recall: 0.6991
Epoch 19/40: 7e - loss: 1.6050 - f1_score: 0.7299 - precision: 0.7766 - recall: 0.6972
```

● threshold : 0.2

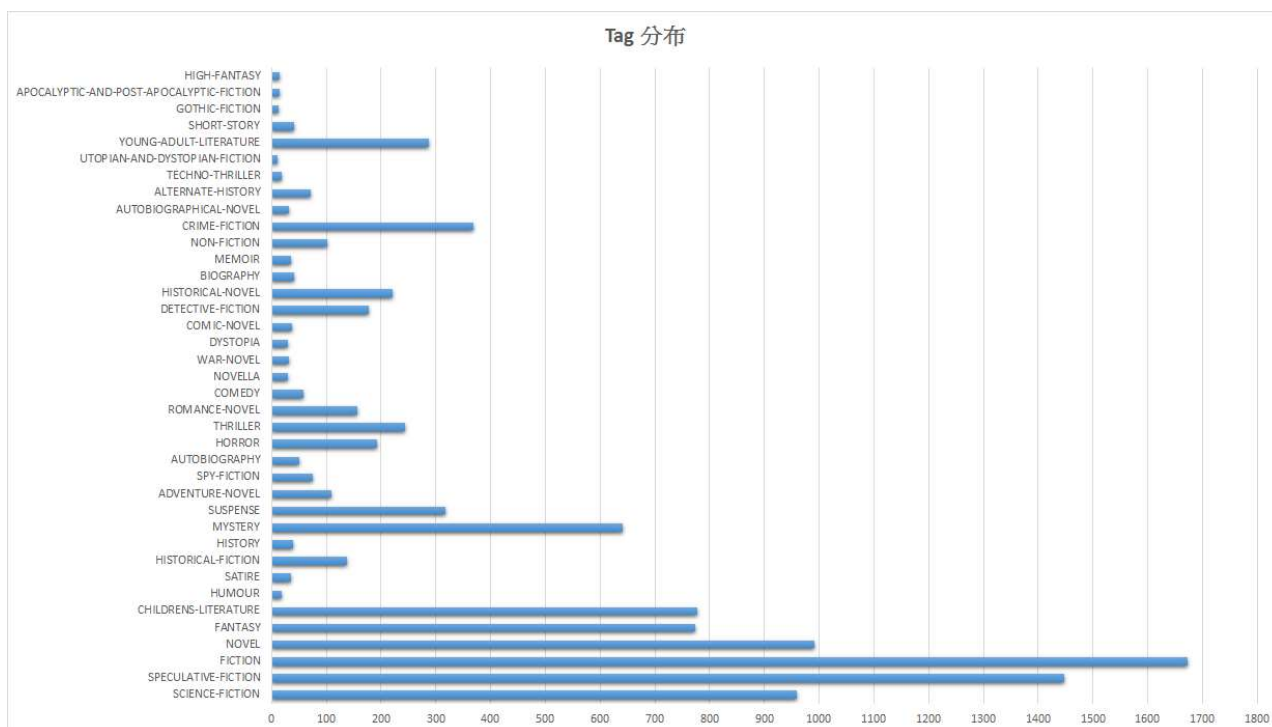
```
Epoch 10/40: 7e - loss: 1.4992 - f1_score: 0.9613 - precision: 0.9958 - recall: 0.9461
Epoch 11/40: 7e - loss: 1.5056 - f1_score: 0.9609 - precision: 0.9958 - recall: 0.9458
Epoch 12/40: 7e - loss: 1.5062 - f1_score: 0.9612 - precision: 0.9977 - recall: 0.9446
Epoch 13/40: 7e - loss: 1.5046 - f1_score: 0.9602 - precision: 0.9956 - recall: 0.9435
Epoch 14/40: 7e - loss: 1.4988 - f1_score: 0.9597 - precision: 0.9957 - recall: 0.9433
Epoch 15/40: 7e - loss: 1.4932 - f1_score: 0.9618 - precision: 0.9954 - recall: 0.9469
Epoch 16/40: 7e - loss: 1.4892 - f1_score: 0.9624 - precision: 0.9936 - recall: 0.9468
Epoch 17/40: 7e - loss: 1.4873 - f1_score: 0.9628 - precision: 0.9930 - recall: 0.9468
Epoch 18/40: 7e - loss: 1.4885 - f1_score: 0.9620 - precision: 0.9930 - recall: 0.9477
Epoch 19/40: 7e - loss: 1.4833 - f1_score: 0.9627 - precision: 0.9936 - recall: 0.9482
Epoch 20/40: 7e - loss: 1.4816 - f1_score: 0.9630 - precision: 0.9940 - recall: 0.9522
Epoch 21/40: 7e - loss: 1.4832 - f1_score: 0.9645 - precision: 0.9957 - recall: 0.9511
```

● threshold 0.2 validation

```
Epoch 10/40: ETA: 0s - loss: 3.6179 - f1_score: 0.6446 - precision: 0.7253 - recall: 0.6475Epoch 00009: val_f1_score did not improve
Epoch 11/40: ETA: 0s - loss: 3.5229 - f1_score: 0.6605 - precision: 0.7355 - recall: 0.6638Epoch 00010: val_f1_score did not improve
Epoch 12/40: ETA: 0s - loss: 3.5185 - f1_score: 0.6613 - precision: 0.7358 - recall: 0.6649 - val_loss: 4.8622 - val_f1_score: 0.3499 - val_precision: 0.3874 - val_recall: 0.3733
Epoch 13/40: ETA: 0s - loss: 3.4171 - f1_score: 0.6836 - precision: 0.7609 - recall: 0.6833Epoch 00011: val_f1_score improved from 0.35773 to 0.36010, saving model to best.hdf5
Epoch 14/40: ETA: 0s - loss: 3.4219 - f1_score: 0.6835 - precision: 0.7614 - recall: 0.6828 - val_loss: 4.8904 - val_f1_score: 0.3601 - val_precision: 0.4012 - val_recall: 0.3837
Epoch 15/40: ETA: 0s - loss: 3.3454 - f1_score: 0.6980 - precision: 0.7746 - recall: 0.6967Epoch 00012: val_f1_score improved from 0.36010 to 0.36026, saving model to best.hdf5
Epoch 16/40: ETA: 0s - loss: 3.3467 - f1_score: 0.6975 - precision: 0.7732 - recall: 0.6970 - val_loss: 4.8964 - val_f1_score: 0.3603 - val_precision: 0.4002 - val_recall: 0.3858
Epoch 17/40: ETA: 0s - loss: 3.2859 - f1_score: 0.7056 - precision: 0.7859 - recall: 0.7028Epoch 00013: val_f1_score improved from 0.36026 to 0.38445, saving model to best.hdf5
Epoch 18/40: ETA: 0s - loss: 3.2765 - f1_score: 0.7067 - precision: 0.7869 - recall: 0.7039 - val_loss: 4.9527 - val_f1_score: 0.3845 - val_precision: 0.4254 - val_recall: 0.4152
Epoch 19/40: ETA: 0s - loss: 3.2303 - f1_score: 0.7246 - precision: 0.8010 - recall: 0.7207Epoch 00014: val_f1_score improved from 0.38445 to 0.39131, saving model to best.hdf5
Epoch 20/40: ETA: 0s - loss: 3.2243 - f1_score: 0.7247 - precision: 0.8009 - recall: 0.7209 - val_loss: 4.9144 - val_f1_score: 0.3913 - val_precision: 0.4375 - val_recall: 0.4206
Epoch 21/40: ETA: 0s - loss: 3.1928 - f1_score: 0.7328 - precision: 0.8096 - recall: 0.7258Epoch 00015: val_f1_score did not improve
Epoch 22/40: ETA: 0s - loss: 3.1926 - f1_score: 0.7331 - precision: 0.8097 - recall: 0.7263 - val_loss: 4.9431 - val_f1_score: 0.3889 - val_precision: 0.4271 - val_recall: 0.4196
Epoch 23/40: ETA: 0s - loss: 3.1948 - f1_score: 0.7306 - precision: 0.8095 - recall: 0.7211Traceback (most recent call last):
```

- ❖ 由上圖可以發現，threshold必須越小才越能預測出一些label數量多的文章
- ❖ 然而就算threshold設小一點validation的performance也不好

3. (1%)請試著分析tags的分布情況(數量)。



- ❖ 分布非常不平均，38個class中就有18個數量小於100
- ❖ 觀察預測出來的檔案後，的確這18類幾乎沒有出現，於是我嘗試將這18個class挑出另外train一個不同參數得model再將其append入答案之中

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

- ❖ 一開始使用word2vec套件，將training data & testing data當作corpus適用CBOW model進行訓練，獲得dim 100的word vector
- ❖ 最後在RNN的部分選擇使用glove的dim 100的檔案，因為我認為只用training data & testing data沒辦法有效的分類文字

❖ glove訓練概念

- The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus.
- GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning.

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

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

❖ 模型設計:

- 先將stopword去除，tokenizer後將每篇文章裡的單字設成1，獲得一個長度為單字總數的list
- 接著丟給DNN train一波

```

model = Sequential()
model.add(Dense(512, input_dim=num_words, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.7))
model.add(Dense(len(tag_list), activation='sigmoid'))

```

❖ 數據:

平均1個epoch所需時間

- RNN: 10s
- bag of word: 4s

Kaggle public score

- RNN: 0.50685
- bag of word: 0.51440

訓練速度



❖ 分析:

- 我認為bag of word的表現比較好適合理的，因為這一題是文章分類，而不同總類的文章單字會差很多，比如說自傳有些裡面就直接包含biography了，或是戰爭通常有war weapon等等。
- 不過由於資料量不夠多，這題用bag of word也沒有非常理想，比如說有同樣兩個字苗速同一個意思，但是training的時候沒有出現過這個單字的話，就無法有效辨別，這是比word vector差的一個點。

- code: bag_of_word.py