

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。

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

Softmax 的公式:  $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$ ，即是將向量中的每一個值取 exponential 之後再取 normalize，而因為是取了 exponential，所以 softmax 會特別注重於向量中”最大”的那一個值，而不是向量中”較大”的值的分布。我想在這次作業中，我們要做的是 multi-class ”multi-label”的預測，一本書可能有很多個 tag，因此 softmax 可能比較不適合做這次作業的 output layer。

至於我在這次作業中最後選擇的 output layer 為”sigmoid”這個 activation

function， $S(x) = \frac{1}{1 + e^{-x}}$  (sigmoid)這個 function 剛好會將原本 range 比較大的一組數據，normalize 到 0~1 的區間，因此只要設定一個好的 threshold，就能將一組數據中大於此 threshold 的值取出，當作這本書所 predict 出來的 tag!!

2. (1%)請設計實驗驗證上述推論。

為了比較第一題中 sigmoid 或者 softmax 適不適合當作這次作業的 output layer，我將 model 中的最後一層的 activation function 分別使用了 sigmoid 和 softmax(其餘皆保持不變)，並且做了一些比較:

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

 (最後一層為 sigmoid)

```
model.add(Dense(38,activation='softmax'))
```

 (最後一層為 softmax)

※Kaggle 上 f1\_score 的比較:

b03902090_aynk	0.50453	10
----------------	---------	----

(sigmoid : 0.50453)

Your Best Entry ↑

Your submission scored 0.23640, which is not an improvement of your best score. Keep trying!

(softmax : 0.23640)

※將 testing data 的 output 做比較:

在這題中，我去觀察了 testing data 在 model 最後一層的 output，下圖其中一筆 testing data 經過 softmax layer 之後所得出來的 38 維 vector:

```
Y_pred: [ 1.24330760e-03  3.53208557e-02  8.29910263e-02  2.14129761e-02
 9.55171604e-03  1.79436698e-03  2.07441244e-05  2.02783762e-04
 2.04939380e-01  1.13273719e-02  1.82342564e-03  1.65015960e-03
 8.81705608e-04  3.23704990e-05  2.96559709e-04  1.17911572e-04
 2.01548406e-04  1.04755126e-02  8.61214823e-04  4.78846574e-04
 9.85335559e-03  9.00080158e-06  4.73601540e-04  6.20798528e-06
 6.00731134e-01  1.00534805e-03  1.78749047e-04  4.03379418e-05
 8.73604367e-05  4.94662090e-04  6.55824377e-04  2.14072975e-06
 2.09129144e-06  2.60237372e-04  5.17348642e-04  4.29433276e-05
 3.10330438e-06  1.27820349e-05]
```

由上圖可以發現，38 維 vector 中的值都差得非常多，幾乎都是分布在 0.001 以下的值，唯有其中一維的值為 0.6007，所以 output 的 tag 就是這維所對應到的 tag，但在經過 softmax 之前，0.20493 這一維的 value 和 0.6007 的 value 是相近的，因為經過了 softmax，所以最後這兩維的值差距被放大，導致只會 output 出 0.6007 這個 tag(在 threshold 一樣為 0.3 的條件之下)。

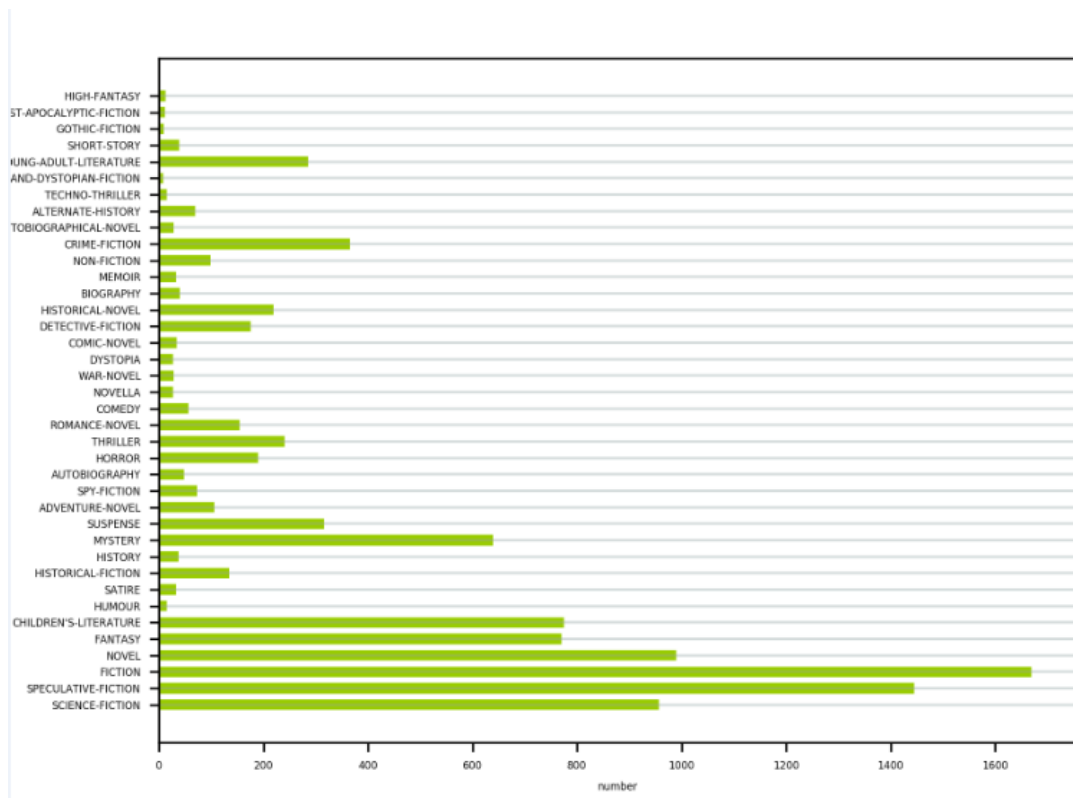
```
Y_pred: [ 1.37685239e-03  9.04553086e-02  2.74351954e-01  1.92939609e-01
 1.20735392e-02  1.76246348e-03  1.80645642e-04  3.90414079e-03
 3.07507813e-01  5.34295514e-02  4.56062518e-03  1.56815397e-03
 1.19639153e-03  3.56921955e-04  3.14981118e-03  1.21764629e-03
 3.34298308e-03  3.54408771e-02  7.35525182e-03  6.86471723e-03
 1.71250813e-02  4.13875008e-04  1.58337899e-03  1.55870039e-05
 6.27034903e-01  9.52281337e-03  4.42578085e-03  8.09035904e-04
 6.91086927e-04  2.42504803e-03  2.65899859e-03  9.06892892e-05
 9.04956105e-05  2.25852622e-04  1.87593373e-03  1.78886679e-04
 1.41825265e-04  2.82106594e-05]
```

(同樣一筆 testing data 經過 sigmoid layer 之後所得的 38 維 vector)

由上圖可以發現，值的大小分布不在差距那麼大，而大於 0.1 的值就有 4 個，大於 0.3 的值也有兩個，也就是說，在用 sigmoid 的情況下，可能在 activation 前的相近的值，通過了 sigmoid 並不會被明顯放大!

由上述觀察中，更可以確定 sigmoid function 比 softmax function 更適合使用在 multi-class multi-label 的 task 之中!!

### 3. (1%)請試著分析 tags 的分布情況(數量)。



上圖為這次 training data 中所 label 出來的 tag 分布情況，其實可以發現，這次的 training data tag 分布相當的不平均，因此在 predict testing data 的時候可以發現 output 出來的 tag “Fiction”這個 tag 也佔了大多數，所以可以改進的方法可能是讓 training data 裡的 tag 分布平均一點，這樣訓練起來的效果也會好很多!

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

最原本我是嘗試用 random 的 embedding matrix，在 train model 的過程中一起更新 matrix 裡的參數，但試了好幾種方法效果都不是太好，因此我最後用了 pre-trained 好的 word embedding，gensim 和 glove 我都分別試過了，最後是發現 glove42B 這份 corpus 的詞彙量比較多，比較不會出現找不到詞而變成零向量的情況，且 train 的效果也最好!

我的做法是，先將 glove42B.300.txt 這份檔案中的資料變成 dictionary，  
ex: dict['my'] = (out vector)，再利用 keras 中的 Tokenizer 找出 data 中的所有字彙量，利用這個 map 關係建立一個總字彙量的 embedding\_matrix，維度為 (len(word), embedding\_dim) !在 training 的時候，將 這個 pre-trained 好的 embedding\_matrix feed 到 model 中第一層的 embedding layer!!

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

```

### tokenizer for all data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_corpus)
word_index = tokenizer.word_index
### convert word sequences to index sequence
print ('Convert to index sequences.')
train_sequences = tokenizer.texts_to_sequences(X_data)
test_sequences = tokenizer.texts_to_sequences(X_test)
print('sequences to matrix.')
train_bow = tokenizer.sequences_to_matrix(train_sequences, mode='freq')
test_bow = tokenizer.sequences_to_matrix(test_sequences, mode='freq')

```

(利用 Tokenizer 裡的 sequences\_to\_matrix 實作 bag of word)

```

model = Sequential()
model.add(Dense(768, input_shape = ((51867,)), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(38, activation='sigmoid'))

```

(將做完 bag of word 的 sequence 丟進 DNN 裡面作訓練)

這次用了 keras tokenizer 裡面的"sequence\_to\_matrix"這個 method 去實作 bag of word，另外 API 裡面 mode 參數設定為"freq"，將 sequence 分別變成 bag of word 的形式之後，丟進一個四層的 DNN 裡，去做 training，在 validation score 最後的結果大概是 0.49 左右，在 Kaggle 上的 f1 score 為 0.46810，比 RNN 的 performance 來的差一點。

Your Best Entry ↑

Your submission scored **0.46810**, which is not an improvement of your best score. Keep trying!