學號：B03705027 系級： 資管三 姓名：鄭從德

(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。

```
def RNN_model():
    model = Sequential()
    model.add(Embedding(embedding_matrix.shape[0],embedding_matrix.shape[1],
                        weights=[embedding_matrix],input_length = x.shape[1], trainable=False))
    model.add(GRU(512, recurrent_dropout = 0.5, dropout=0.5, return_sequences=True, activation='relu', implementation=2))
    model.add(GRU(256, recurrent_dropout = 0.5, dropout=0.5, return_sequences=True, activation='relu', implementation=2))
    model.add(GRU(128, recurrent_dropout = 0.5, dropout=0.5, activation='relu', implementation=2))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(39, activation='sigmoid'))
    model.add(Dropout(0.3))
    model.compile(loss='binary_crossentropy', optimizer=opt,metrics=[fmeasure])

    return model
```

上圖為我的 model 架構。

這次的作業不適合用 softmax 作為 activation function，因為 softmax 會讓每一個 tag 出現的機率加總等於一，其中隱含的意義就是，當一個 tag 出現的機率變高，就壓縮了其他 tag 出現的機會。但這顯然不適用於這次作業的狀況，因為這次以文句判斷分類的任務中，我們是透過不同的關鍵字找出文章地分類，所以儘管少數分類可能會有互斥的關係(如兒童讀物跟青少年讀物)，基本上每一個類別是獨立判斷的(第三題中會再次確認這一點)。我最後是利用 sigmoid function 作為 output layer 的 activation function，因為他可以使每個類別的機率收斂到 0,1 之間，能夠獨立的判斷屬於不同類別的機率。

(2) 請設計實驗驗證上述推論。

```
def RNN_model():
    model = Sequential()
    model.add(Embedding(embedding_matrix.shape[0],embedding_matrix.shape[1],
                        weights=[embedding_matrix],input_length = x.shape[1], trainable=False))
    model.add(GRU(512, recurrent_dropout = 0.5, dropout=0.5, return_sequences=True, activation='relu', implementation=2))
    model.add(GRU(256, recurrent_dropout = 0.5, dropout=0.5, return_sequences=True, activation='relu', implementation=2))
    model.add(GRU(128, recurrent_dropout = 0.5, dropout=0.5, activation='relu', implementation=2))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(39, activation='softmax'))
    model.add(Dropout(0.3))
    model.compile(loss='binary_crossentropy', optimizer=opt,metrics=[fmeasure])

    return model
```

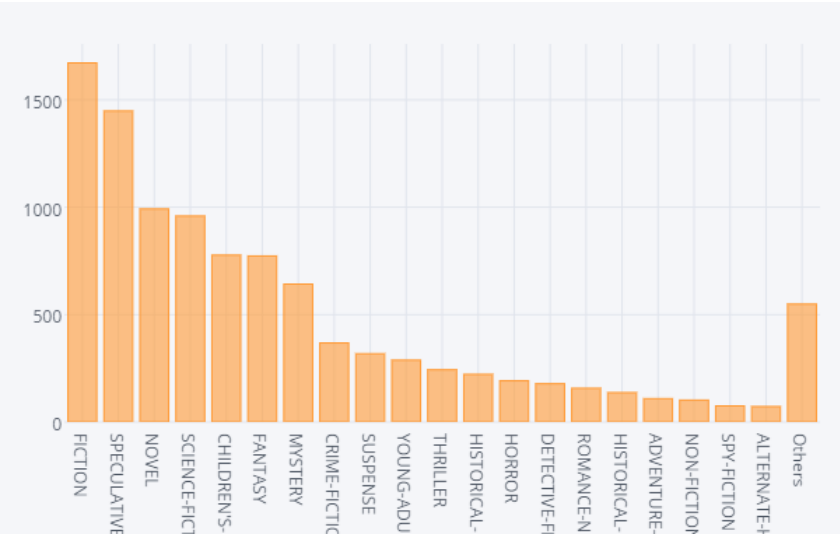
這是我嘗試使用 softmax 的 RNN model。在 threshold 方面，我使用一個迴圈跑過

從 0.01 -> 0.25 每個 tag 的 threshold(見下圖，我原本的 RNN 也是用這個手法)，找到可以讓 validation set 的 f1 score 最高的一組 threshold。在 softmax 的情況下，找出來的 threshold 會比 sigmoid 低上許多。

```
170  threshold = np.arange(0.01,0.25,0.01,dtype=float)
171  acc = []
172  accuracies = []
173  best_threshold = np.zeros(out.shape[1])
174  for i in range(out.shape[1]):
175      y_prob = np.array(out[:,i])
176      for j in threshold:
177          y_pred = [1 if prob>=j else 0 for prob in y_prob]
178          acc.append(f1_score(y_validation[:,i], y_pred, average='micro'))
179      acc    = np.array(acc)
180      index = np.where(acc==acc.max())
181      accuracies.append(acc.max())
182      best_threshold[i] = threshold[index[0][0]]
183      acc = []
```

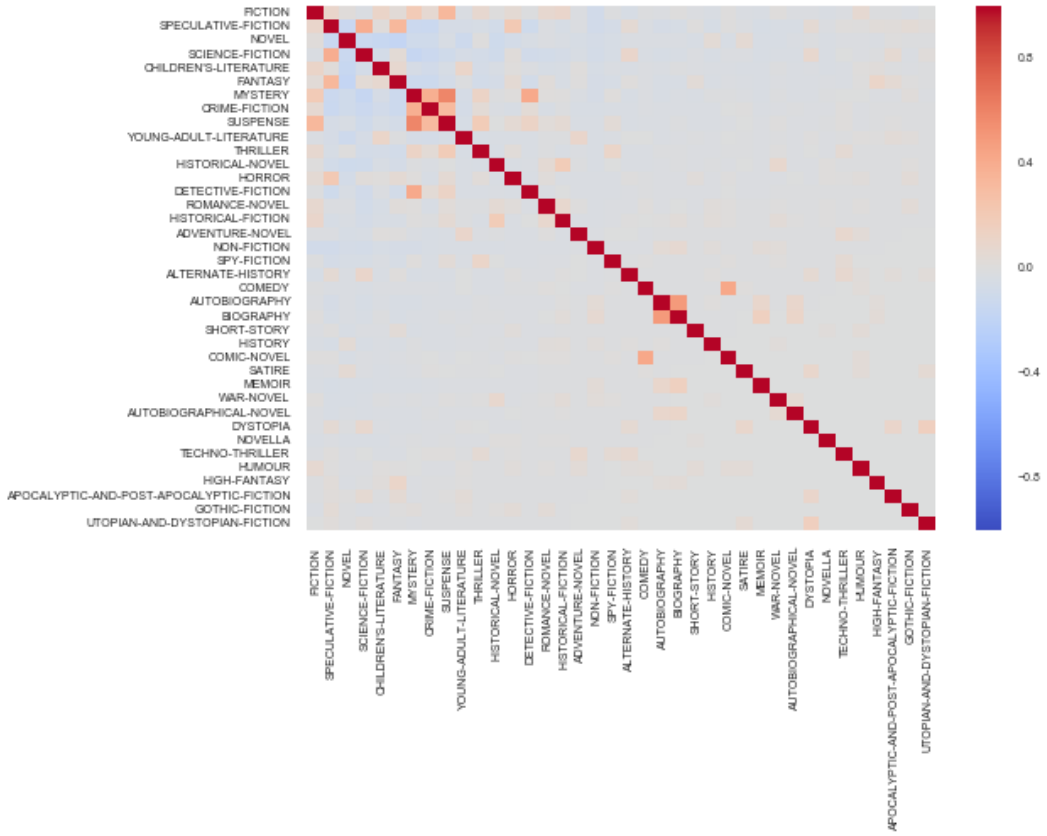
我做出來最後的結果，在同樣的 model 下使用 softmax 上傳 kaggle 之後，f1 分數僅為 0.4538 分，觀察 output file 之後也會發現，平均 predict 出來的 tags 數要比 softmax 少很多 (有許多空格)。這可能是因為不同的 tag 在 softmax 的情況下會彼此壓縮彼此的機率，因此難以出現多種 tags 都超過 threshold 的情形。整體而言，我覺得使用 sigmoid 還是比較好，在我設計的 RNN model 下 sigmoid 也有著比較好的表現。

(3) 請試著分析 tags 的分布情況(數量)。



在每個 tag 個別分布的部分，可以從左圖中看出在所有的 tag 中，最多的前四個 tag 就占了 49.3%，代表幾乎每本書都有這四個 tag 中的一個，是屬於範圍比較大的分類。

而剩下的 34 個種類加起來占了另外 50%，這些就是比較 specific 的分類。



在於 tag 的相關性部分，我繪製了 38 個 tag 彼此 correlation 的 heatmap。圖中我們可以看出除了與自身完全正相關以外，只有少數 tags 互相有依存關係，最明顯的為 (Speculative-Fiction, Science-Fiction) , (Mystery, Suspense), (Mystery, Detective-Fiction), (Autobiography, Biography)等。而只有前面個大類有互相排斥的關係，像是 Novel 和 Science-Fiction(只有左上角有點藍)，在後面叫小的分類中，跟其他 tag 是沒有互斥關係的(可以想像成比較 specific 的分類更可以一次擁有多個)。這也印證了我們在問題一中，所描述個別機率幾乎是獨立的假設。

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

這次的作業中我嘗試使用了 word2vec 以及 Glove，其中 word2vec 是自己拿這次 training set 及 testing set 所有的字進行 skip-gram 的演算法得出來的。實驗結果仍然輸 Glove 滿多的，而在嘗試的 Glove 幾個模型當中，又以 42B、300D 的表現最佳(是我常是最大的語料集了)。

Glove 是 count base 的 embedding 方式，在我下載的資料集中已經是一個 word to vector 的 dictionary。其中我選擇的是 300d，也就是每個字對應成一個三百維的 vector。我只要將每個字 tokenize 之後的 index 與他的 vector 做成一個 dictionary，再把我有用到的字的 vector 放入一個叫大的 matrix 中(程式碼中的 embedding_matrix)，即可丟到 RNN model 中的 embedding layer 進行訓練了。

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

我實作了 bag of word，將每一段文字做成一個 20 萬維的矩陣，已每個字的出現字數作為 feature 丟入一個六層的 DNN 中進行訓練(因為前面的 RNN model 也都是六層)。訓練的結果在本地端測試 f1 score 最高僅 0.43，丟上 Kaggle 的成績最高僅 0.39。

```
def DNN_model():
    model = Sequential()
    model.add(Dense(1024, activation='relu',input_dim=200000))
    model.add(Dropout(0.5))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(39, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=opt,metrics=[fmeasure])
    return model
```

從結果來看，RNN 比 bag of word 好上許多，可能原因有二：第一個是每個字的相關性是 bag of word 沒有考慮到的，在英文中可能詞性變化造成的些微改變皆會在 bag of word 中被視為不同的字，或是有一些意義相同的字，我們再 RNN 中可以在 embedding 的步驟中實現，BOW 則無法。第二個原因是 RNN 可以考慮字詞的前後關係，這在文字中也可能特別重要，在一整篇敘述中，形容詞的前後可能決定了是悲劇或喜劇等等，這些特性都是 bag of word 無法考慮的。