學號：B02705027 系級： 資管四 姓名：陳信豪

* 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。

softmax 並不適合作為本次作業的 output layer。

softmax 是會使最後一層所有 output 機率的結果合為 1，

如果是 multi class with 1 label 就適合 softmax

假設今天有 4 個 class，而 data 有 1 label

某一 data 經 model 出來的結果可能是 [0.05, 0.05, 0.1, 0.8]

我們能明顯判斷最後一項是對應的 label

但假設今天 data 有3 個 label

某一 data 經 model 出來的結果可能是 [0.3, 0.3, 0.19, 0.21]

我們較不能明確判斷，更不用說在 multi label 的情況下，

我們是無法得知每個 data 有幾個 label，

也因此更無法判斷哪些是對應的 label

而本次作業是 multi class with multi label，

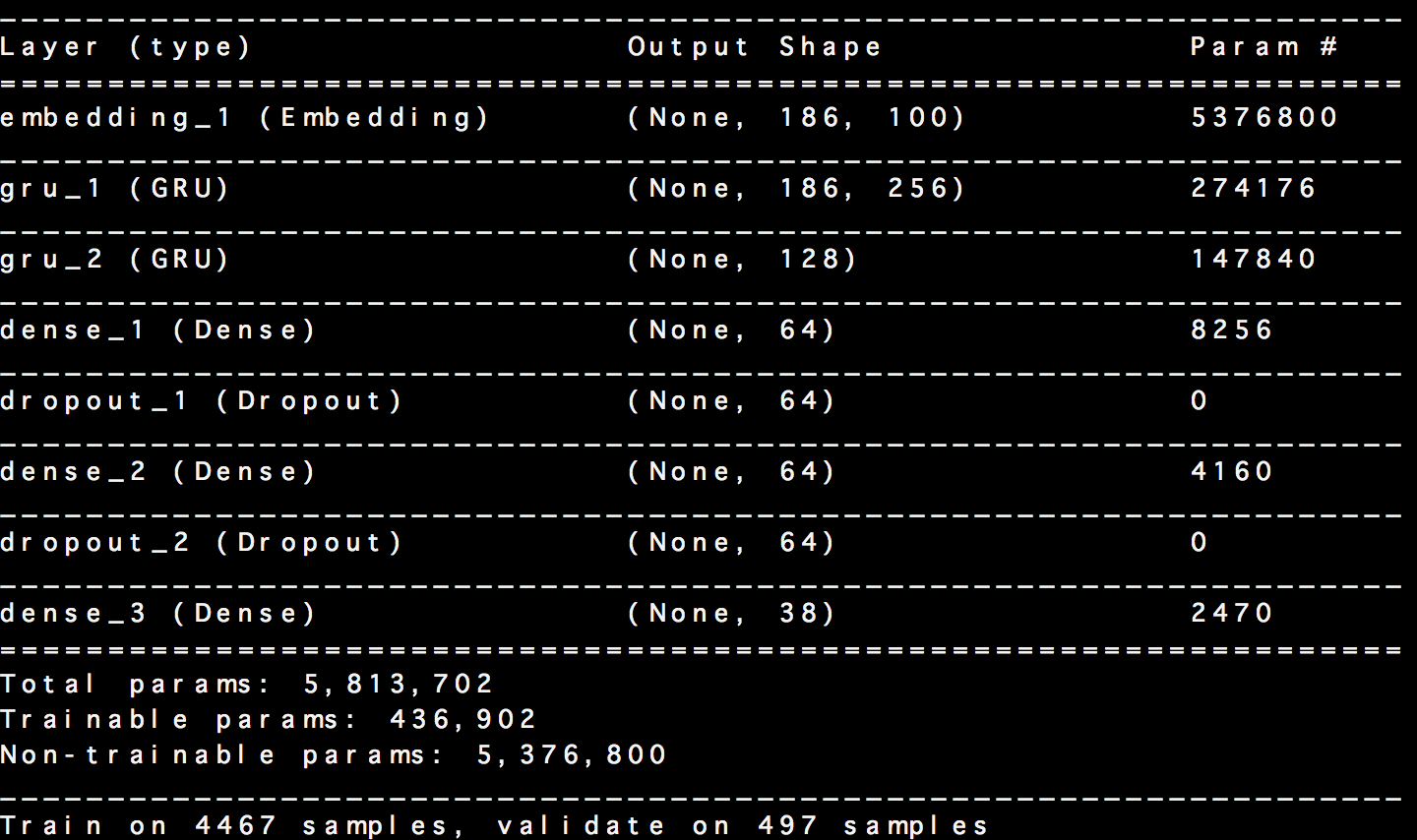
因此並不適合 softmax

* 1. (1%)請設計實驗驗證上述推論。

跑兩個接近相同的 RNN model 在本次作業上，

取同樣的 training set 和 valid set。

兩者差別只在於最後一層是 softmax 或 sigmoid



（model summary）

最後一層， dense\_3，

一個選 softmax 做 activation，

一個選 sigmoid 做 activation。

結果：

|  |  |
| --- | --- |
| sigmoid | softmax |
| Model loss | |
| rnn_model_loss.png |  |
| Train accuracy | |
| rnn_train_accuracy.png |  |
| Valid accuracy | |
| rnn_valid_accuracy.png |  |

從 Training accuracy和 Valid accuracy 可以看出來 sigmoid 明顯比 softmax 好。

sigmoid valid f1 measure 約快 0.5 ，

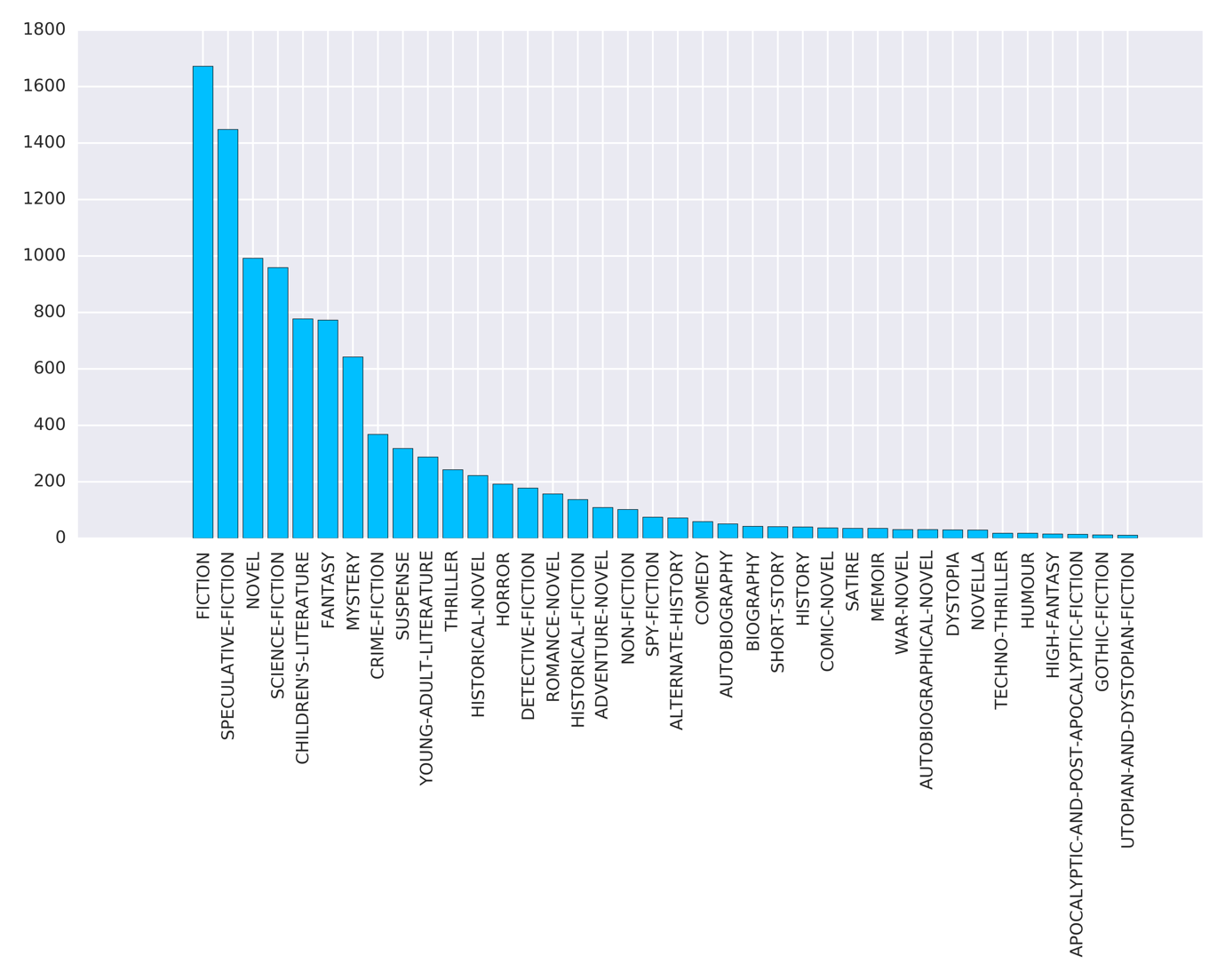
然而 softmax 只有 0.2 左右，

因此 softmax 較不適合本次作業。（multi label）

* 1. (1%)請試著分析tags的分布情況(數量)。

統計 tags 出現次數，由大排到小，並且發現分佈其實很不均。

38 項中的前 10項就佔了約 80 % 左右。

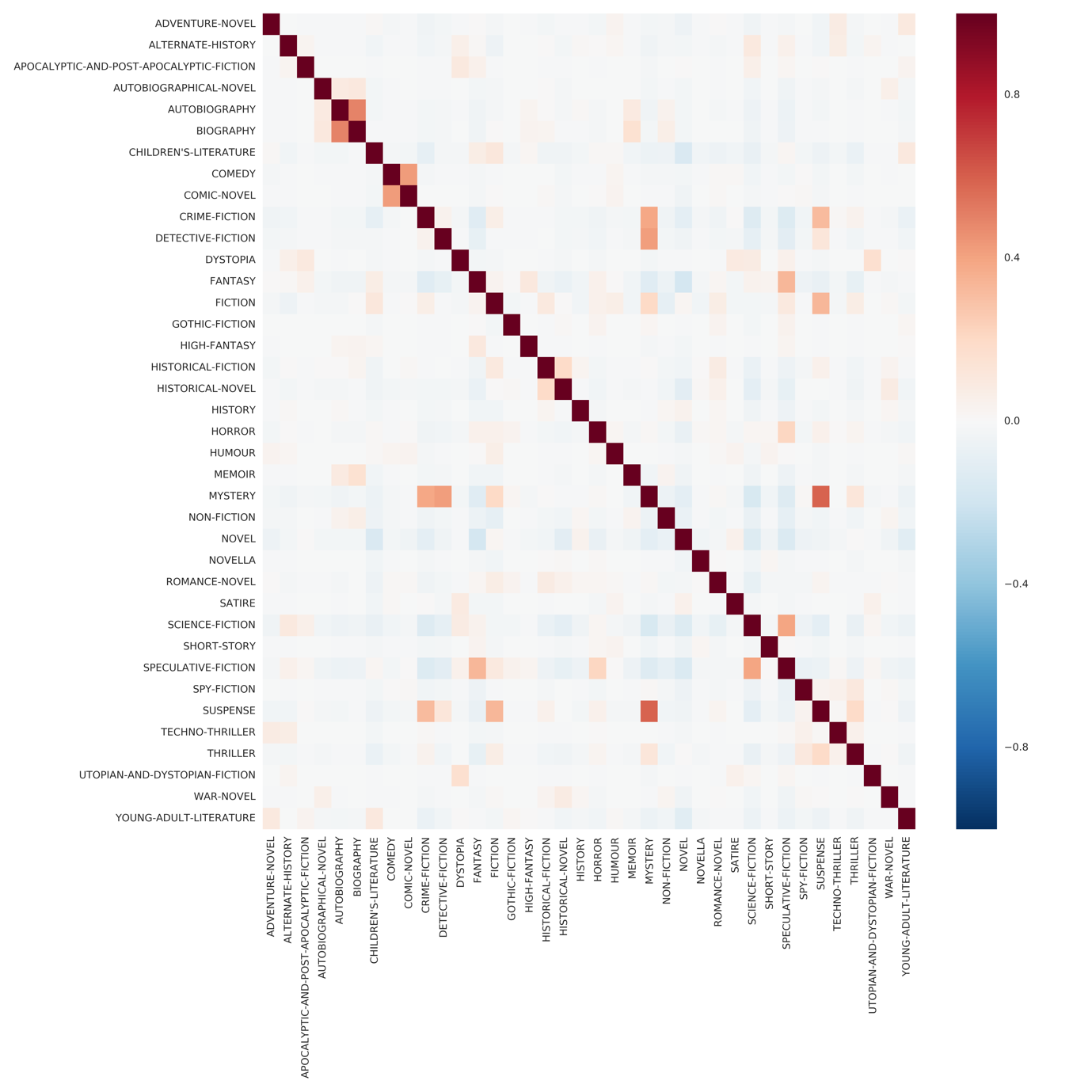


另外，透過將每個 label 以 binary 的方式儲存後，

我們可以將label 轉成長度 38 的 binary list。

在所有 training 資料中，我們有 4964 個 label binary list 。

接著可以算出 correlation matrix，並用 heatmap 觀察。



可以發現以下 tags 比較有正關聯。

（COMEDY，COMIC-NOVEL）、

（BIOGRAPHY、AUTOBIOGRAPHY）、

（MYSTERY、SUSPENSE）

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

這次作業我的 embedding 來源是 glove

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

檔名：glove.6B.100d.txt

(word vector 維度是 100)

首先將原本的 train\_data.csv 資料讀進來後，

從 nltk 套件取 stop words 濾掉，並全部轉成小寫。

接著透過 keras 的 Tokenizer 把 word 做 index 的編碼。

（一個 word 對應到一個號碼的意思。）

*tokenizer = Tokenizer()*

*tokenizer.fit\_on\_texts(all\_texts)*

接著把所有 data 文章轉成index sequence，

並做 padding。 （將剩餘的空白做 0 編碼）

*train\_sequences = tokenizer.texts\_to\_sequences(train\_texts)*

*X\_train = pad\_sequences(train\_sequences)*

然後讀 glove.6B.100d.txt，做出 embedding\_dict。

透過 embedding\_dict，我們給一個 word 後 ，

它可以找出對應的 word vector

有了 tokenizer.word\_index 和 embedding\_dict，

我們就能做出 embedding\_matrix。

embedding matrix 的 row 代表每一個在 tokenizer 中的 word 的vector ，

並且每一個 word vector 是 100 維，所以會有 100 columns 。

最後將 embedding matrix 餵進 keras model。

*model.add(Embedding(num\_words,*

*word\_vec\_dim,*

*weights=[embedding\_matrix],*

*input\_length=maxlen,*

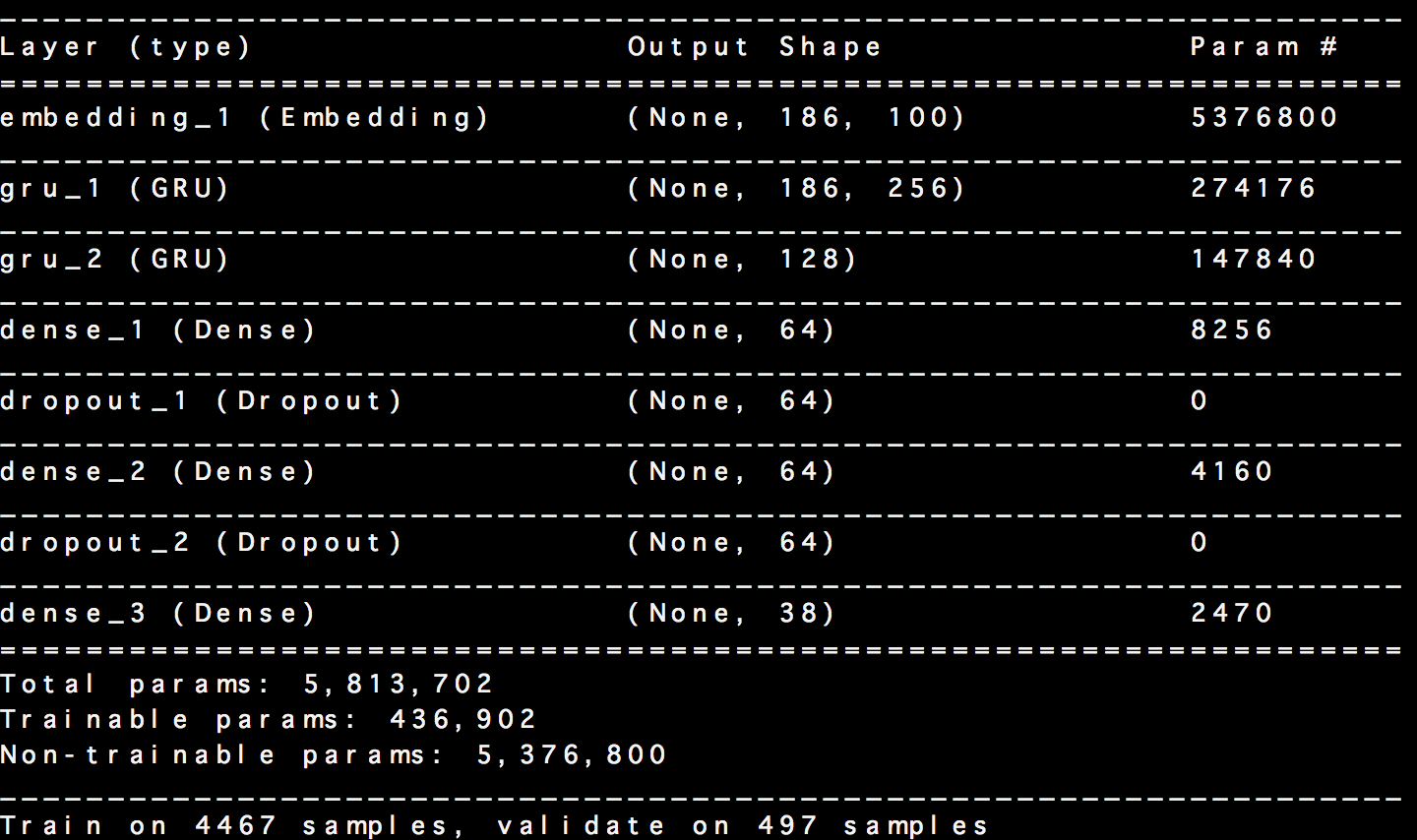
*trainable=False))*

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

此題的 bag of words 的 mode 採 tf-idf。

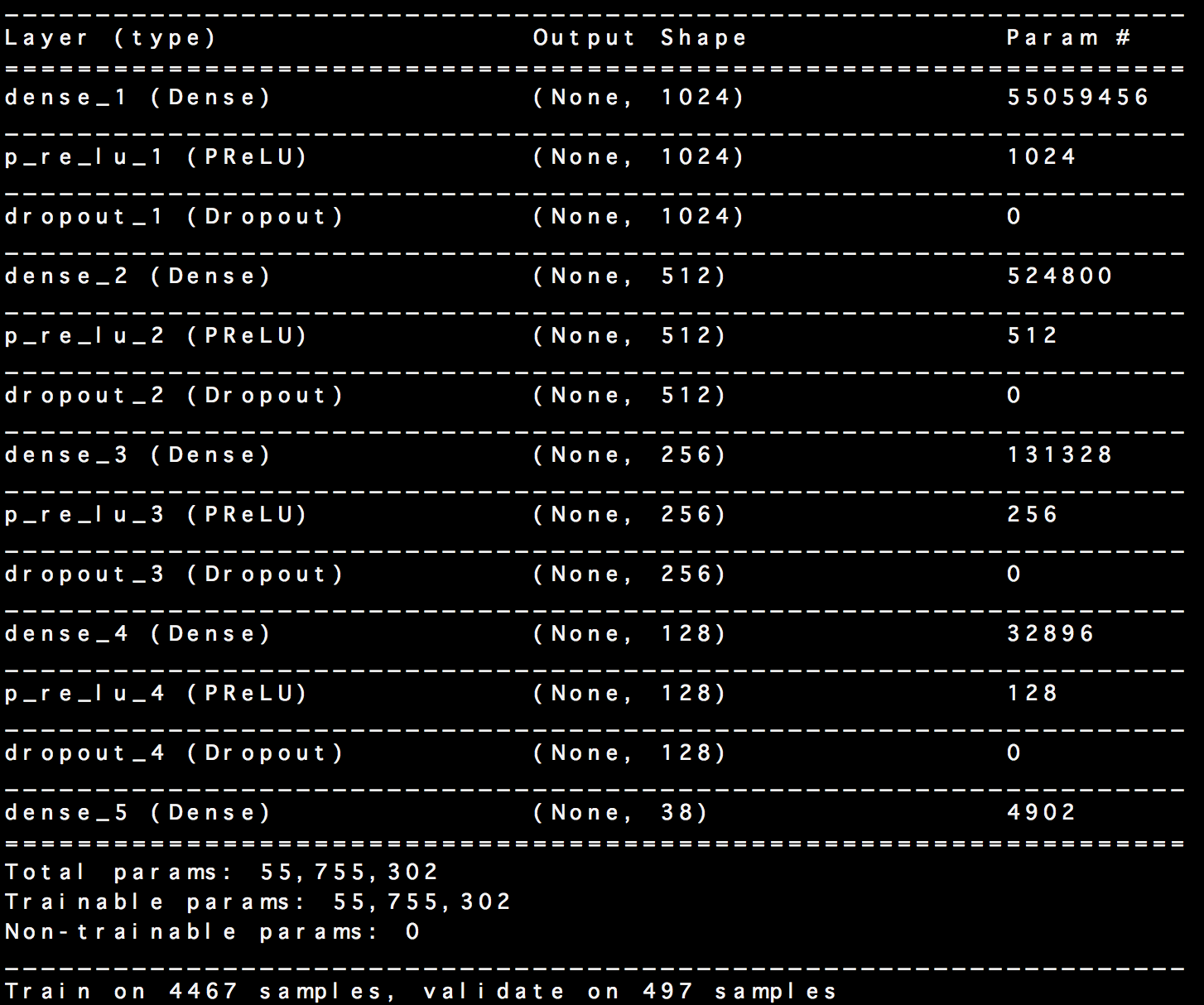
取同樣的 training set 和 valid set 下去做兩類型的 model。

RNN model:



（RNN model summary）

Bag of words model:



（Bag of words model summary）

結果：

|  |  |
| --- | --- |
| RNN | Bag of words |
| Model loss | |
| rnn_model_loss.png | bag_model_loss.png |
| Train accuracy | |
| rnn_train_accuracy.png | bag_train_accuracy.png |
| Valid accuracy | |
| rnn_valid_accuracy.png | bag_valid_accuracy.png |

兩者之間最大的差異在於 training 的速度

RNN 約27 秒一個 epoch

Bag of words 約 4 秒一個 epoch

RNN 約到第70 幾 epoch 才會到達 valid f1 measure 高峰

Bag of words 10 個 epoch 內就到達 valid f1 measure 高峰

經由測試 RNN 最後在 kaggle 上的分數是 public: 0.52281 / private: 0.49702

Bag of words最後在 kaggle 上的分數是 public: 0.51808 / private: 0.49466

RNN 比 Bag of words 略好一些。