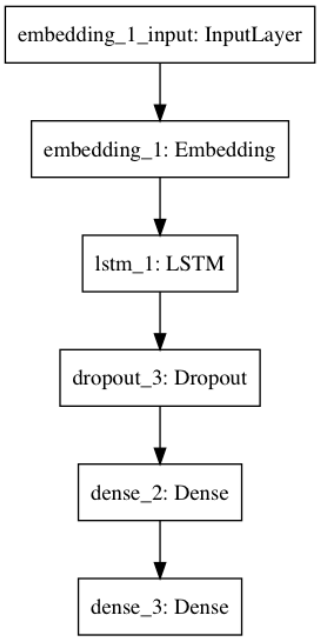
學號：B04902092 系級： 資工三 姓名：張均銘

1. (1%) 請說明你實作的 RNN model, 其模型架構、訓練過程和準確率為何？

訓練過程:



同時讀進 labeled 的 train data 和 test data 一起做 keras.tokenize，label 另外存然後 np.utils.categorical(2)把它變成兩維。

採用 keras 的 toeknizer 來做，tokenize 的 num_word =25000，設定 alldata 這個變數是 traindata concatenate testdata 後一起做 fit_on_texts，之後再把 alldata 切回 train 跟 test data，拿 train data 去做 text_to_sequence，在把 train data padding 成長度 40 就用 model 來 train。

參數:

Total params: 3,660,546，Trainable params: 3,660,546

在 LSTM 內有兩個參數:dropout=0.3, recurrent_dropout=0.1,另外切了 0.05 的 validation_set

這次的發現是，在 traindata 數目小的時候，validation_set 不能切太高，因為本身資料就不多，在切的話 train 起來效果更不好，反而像這樣 0.05train 出來的效果是最好的。

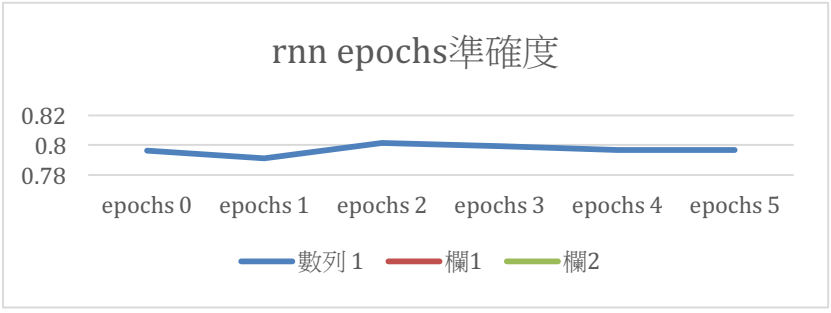
Num_word 也是越多越好，較意外的是，進行 cutword(把句子中不會影響情緒的字刪掉，如 a,the...)之類的刪掉準確度反而下降，這就不知道為什麼了。還有普遍不管 embedding layer 還是 LSTM，設定 kernel_initializer 準確度都會降，也是無解。

準確度：

本機 val_acc: 0.7968

kaggle public 分數: 0.79458

private 分數 : 0.79419

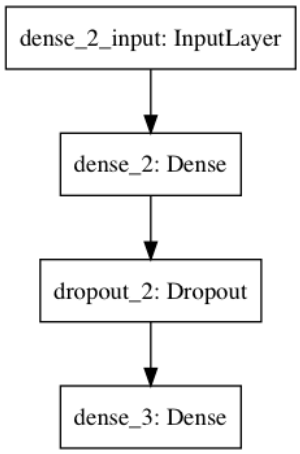


2. (1%) 請說明你實作的 BOW model, 其模型架構、訓練過程和準確率為何？ (Collaborators: 鍾偉傑(B04902082))

預處理：

採用 sklearn 的 countvectorizer 來做 BOW，採用 count 的形式，就是先讀 train data 和 test data 後建字典，每個 train data 的維度就是字典大小，然後字典對每筆資料會 count 該字典裡的個別字出現幾次，這樣就做好 BOW 的預處理

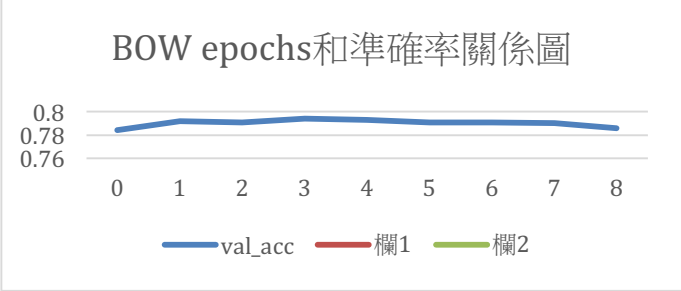
架構:



訓練過程:

同 RNN，使用 checkpoint 和 earlystop 兩個 callback, 30 個 epochs 大概跑到第 9 個就停，Dense layer 512 採用 relu，中間一個 dropout(0.5)
然後取 val_acc 最高的 model 來 predict

準確率:



kaggle 分數 public: 0.78758 private: 0.78896

3. (1%) 請比較 bag of word 與 RNN 兩種不同 model 對於"today is a good day, but it is hot"與"today is hot, but it is a good day"這兩句的情緒分數，並討論造成差異的原因。

RNN:

同第一題的 code，預處理字典用 train+test data 建成，num_word=25000,本身 model valid_acc=0.802。

RNN	Today is a good day,but it is hot	Today is hot,but it is a good day
0	0.42719039	0.17795284
1	0.57280958	0.82204711

BOW:(同第二題使用的模型,把這兩個句子拿下去一起建 dictionary,fit_ontransform)

BOW	today is a good day, but it is hot	today is hot,but it is a good day
0	0.29117536	0.29117536
1	0.7088244	0.7088244

差異:

BOW 兩個句子的機率完全依樣，RNN 第一句偏向開心，第二句偏向不開心
以 BOW 來說合理，因為他玩只看字出現的次數，兩句字出現的次數完全依樣，所以預測出來結果當然完全依樣。

原因:

RNN 則用 LSTM，會紀錄前面句子的狀態丟到下一個 input，所以句子的字的順序不同，就會得到不同的結果，而由上面的結果看來，由第二句，可以猜測 but 的影響非常大，我的 RNN 模型感覺是著重在 but 後面的那個句子，像第一

句是 but it is hot 可能句子就會覺得那個是重點導致判斷的機率很接近，但第二句 it is a good day 就很明顯判斷出整句是開心的，RNN 模型感覺前面的句子漢字的影響力比較小，越接近後面的句子和字的影響力越大。

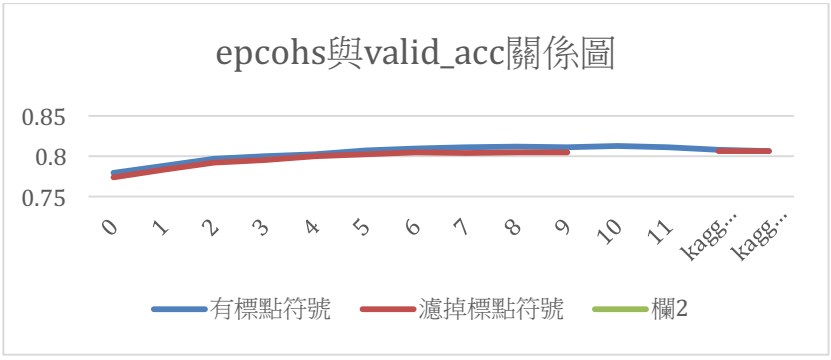
4. (1%) 請比較"有無"包含標點符號兩種不同 tokenize 的方式，並討論兩者對準確率的影響。

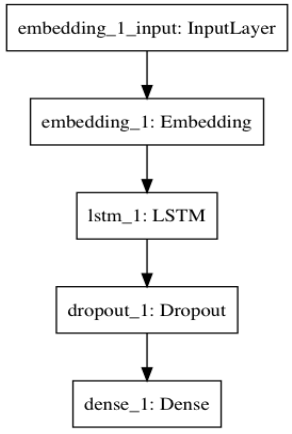
標記方法:

使用 keras.preprocess 的 text_to_word_sequence 函式來分資料，可藉由參數 filters 來決定要不要把標點符號移除。

RNN 模型:

先用 gensim word2vec 建立字典和字的模型，然後藉由 embedding layer 傳進 rnn，rnn 只有 embedding, lstm 這樣的架構。





在 kaggle 方面其實兩者差距沒有太大，沒有濾掉標點符號的好一點，雖然 private 比濾掉標點符號的差一點，但 public 就相對高了 0.0013，我覺得標點符號其實對情感分析是蠻重要的，像是…常常表示無言基本上應該就是不開心，那沒有高太多的原因是因為有些還是要依前後文，像「!」同時有可能是開心或生氣的句子，所以反而會造成誤判。應該重點還是看 traindata 標點符號的分佈。

5. (1%) 請描述在你的 semi-supervised 方法是如何標記 label，並比較有無 semi-surpervised training 對準確率的影響。

標記方法:

先使用 RNN 得出一個 model 在拿這個 model 來 predict unlabel data，predict 完後取部分 unlabel data 來當作第二個 model 的 input 來預測 testdata，試了兩種 model:取 predict unlabel 機率最高的前 60 萬個，或者取 predict 機率中間的萬個 unlabel data 來做點個 model(report 取中間 60 萬個 unlabel data 來做 model)， unlabel data 的 label 是由 train data 的產生的，所以再拿 unlabel data 來 train 時 val_acc 都會飆的異常的高。所以其實預測的結果有很大的部分是依據原本 train data label 的特性來 predict，所以其實不會和 train data 拿去 train 的結果差太多，而我做出來取中間 60 萬筆 unlabel data train 出來的 model predict 結果都比沒有 semi 的結果好 0.006，估計原因是，unlabel 那 60 萬筆多了一些單字是 train data 沒出現過的，也就是說有些單字只出現在 unlabel 和 test data，那這樣 unlabel data

的 model 就能做出比較好的判斷。但這跟取的 unlabel data 數目和種類有關。

如果改取 predict unlabel data 中機率最高的 60 萬筆來製作 semi 的 model，那就會跟 train data 類型太像，semi 出來結果就跟 train data 差不多，因為資料不夠 diverse。但是還是會好一點(畢竟 data 60 萬筆是原本 train data 得 3 倍)