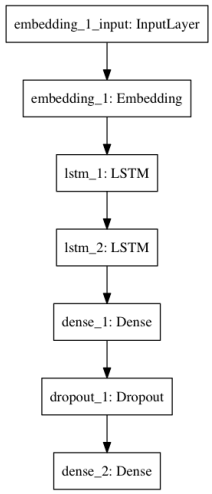
學號：B04902011 系級：資工三 姓名：張立暐

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

1) 模型架構




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 40, 100)           2000100
_________________________________________________________________
lstm_1 (LSTM)                (None, 40, 256)           365568
_________________________________________________________________
lstm_2 (LSTM)                (None, 256)               525312
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,924,005
Trainable params: 2,924,005
Non-trainable params: 0
_________________________________________________________________
```

如上圖所示，經過預處理的訓練資料先通過 embedding 將每個 token 轉成 100 維的向量，接著通過兩層 LSTM、一層 Dense，最後通過 sigmoid 輸出在 0 到 1 之間的值。詳細參數：

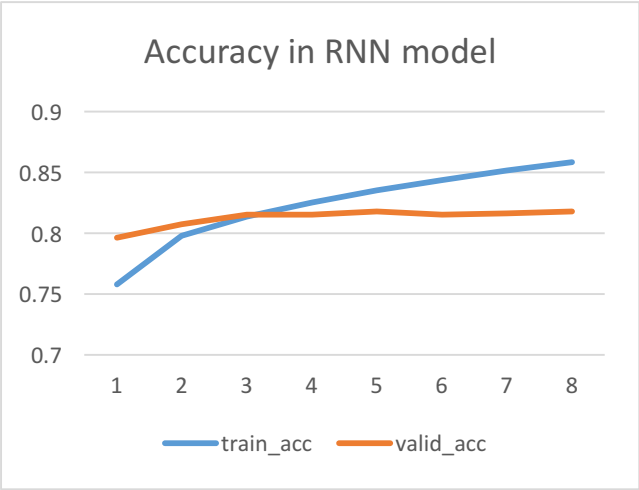
- Embedding：input_dim=20001（字典大小加一）；output_dim=100；input_length=40（padding 後長度）；weights 有透過 gensim.models.Word2Vec 對『經由 gensim.utils.tokenize 進行斷詞並去除特殊字元與標點符號的訓練資料』進行 pretrain 得到的詞向量做初始化，並開啟 trainable 使其可以繼續 fit。
- LSTM：units=256；return_sequence 依序為 True, False；dropout_rate=0.3。
- Dense：units=128；activation 為 relu。
- Dropout：rate=0.3。

2) 訓練過程

透過 keras.preprocessing.text.Tokenizer 建立詞典後透過 Tokenizer.text_to_sequence 將訓練資料中斷詞並以一一以編號表示，為了使 input 大小一致，使用 keras.preprocessing.\sequence.pad_sequence 進行自動補（切）齊成長度為 40 的 token sequence。取出前 10% 的資料作為 validation 使用。訓練時共跑 20 個 epoch（但有使用 earlystopping），batch_size 為 512。而 loss function 為 binary_crossentropy，optimizer 為 adam。另外有使用 keras.callbacks.ModelCheckpoint 來保留 val_acc 較高的 model。

3) 準確率

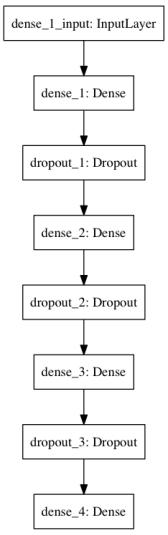
右圖為訓練過程中準確率走勢圖，可以看到由於 batch size 對於資料量來說較小造成每個 epoch 更新參數的次數較多，故僅 3 個 epoch 時就已經趨近收斂。而 validation accuracy 最高達到 **0.81915**。



而 kaggle 上最好的成績是由數個類似的 RNN model（包含單、雙層、雙向 RNN 及不同的 hidden layer size）進行 ensemble 後得到的，其綜合準確率在 kaggle 上得到 **0.82399** / **0.82475** 的準確率。

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

   1) 模型架構



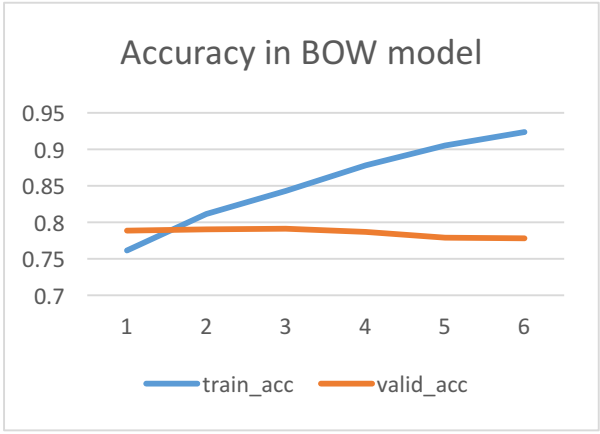

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               1280256
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,321,473
Trainable params: 1,321,473
Non-trainable params: 0
_________________________________________________________________
```

   如上圖所示，經過預處理的訓練資料依序通過三層 Dense Block，最後通過 sigmoid 輸出在 0 到 1 之間的值。詳細參數：

   - Dense：units 依序為 256, 128, 64；activation 為 relu。
   - Dropout：rate 皆為 0.3。

   2) 訓練過程

   一樣透過 keras.preprocessing.text.Tokenizer 建立詞典，但這邊改用 Tokenizer.\
   text_to_matrix 將訓練資料轉成 BOW 的形式。其餘過程皆與上題相似。



   3) 準確率

   右圖為訓練過程中準確率走勢圖，validation accuracy 最高達到 **0.79205**。另外可以看出此 model 很快就開始收斂且開始 overfitting。

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。**

   使用前兩題所描述的 model 進行 predict，得到的原始輸出如下表：

	RNN	BOW
today is a good day, but it is hot	0.29801	0.49847
today is hot, but it is a good day	0.94922	0.49847

   可以看出兩句話在 BOW model 之下有一樣的輸出，顯然是因為使用 BOW 的形式表示一段文字時，只要每個詞出現的順序一樣，對應出來的向量也就一樣，故兩句話雖然詞序不同，但轉成

BOW 後相同，predict 出來的值當然也相同。而透過 RNN model 因為 LSTM 有記憶前面出現過的詞這個特性，故可以考慮到詞序的影響，所以能夠判斷兩句話在情緒上的差異。

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。**
   兩者皆使用第 1 小題中所描述模型架構，而包含標點符號的 tokenize 方式是藉由在 keras.\preprocessing.text.Tokenizer 參數中將 filters 從預設值改成 '\t\n'，並在 pretrain Word2Vec model 時改用 split(' ') 將訓練資料做斷詞且保留標點符號。兩者準確率如下：

	train_acc	valid_acc	public	private
有標點	0.8486	0.8266	0.8271	0.8258
無標點	0.8434	0.8192	0.8197	0.8199

   可以看出有標點符號所訓練出來的結果在各方面的數據上都較沒標點符號有更好的準確率。可以推論在有標點符號的情況下，RNN 可以獲得更多資訊（例如逗號表示文字的斷句、驚嘆號與問號表示文字的語氣等），造成其可以更準確的判斷一段文字的情緒。

5. **(1%) 請描述在你的 semi-supervised 方法是如何標記 label，並比較有無 semi-supervised training 對準確率的影響。**
   我在 semi-supervised 方法中標記 label 的方法是將原先 train 好的 model 去對所有的無 label 的資料進行 predict，取輸出值大於 0.9 和小於 0.1 的資料分別標記上 1 和 0 的 label。標記後合併至 training set 中，並將 model 對新的 training set 進行 fit 兩個 epochs 後再次對無 label 的資料進行 predict……，以此類推並進行 10 次以上動作，最後透過 ModelCheckpoint 將中途 val_acc 最好的 model 記錄下來。分別在有無標點的 RNN model 中測試後，各項準確率如下：

	Before Semi	After Semi
RNN (without punctuation)		
train accuracy	0.8434	0.9750
validation accuracy	0.8192	0.8231
public score	0.8197	0.8202
private score	0.8199	0.8185
RNN (with punctuation)		
train accuracy	0.8486	0.9755
validation accuracy	0.8266	0.8304
public score	0.8271	0.8284
private score	0.8258	0.8283

   可以看出經由 semi-supervised training 過後，準確率較原先有些微提升，但提升有限。
   對於 semi-supervised 所帶來準確率的提升，可以猜想其原因是我們透過在一定信心下將原先沒有 label 的資料標記上 label 以增加訓練資料量，這些增加的訓練資料可能可以補充原有訓練資料中所不能完整提供的資訊，讓 RNN 可以學到更多資訊以增加其判斷準確率。