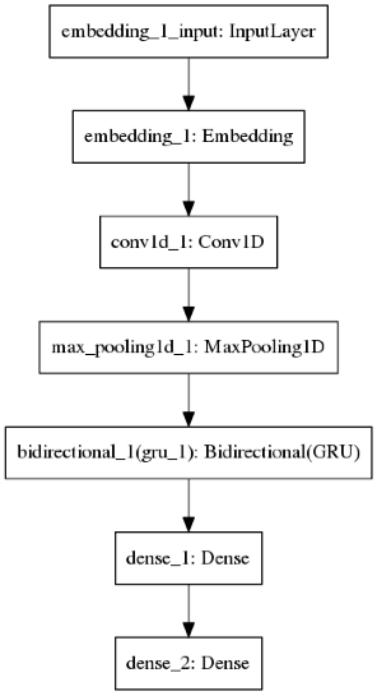
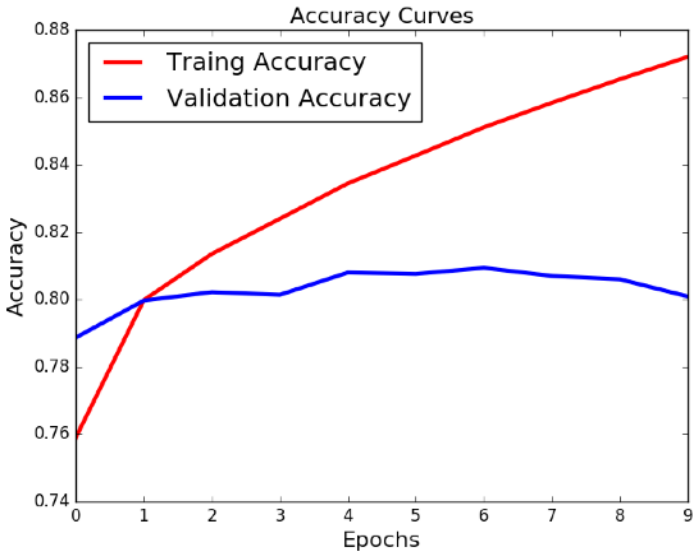
學號：r06946004 系級： 資科學程碩一 姓名：蔡尚錡

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？**

這次作業我實作了一個 RNN model，首先把training data裡面句子的每個單字，依照出現頻率的高到低標出從1開始的index，也就是頻率最高的標為1，第二高的標為2以此往下類推，但是出現頻率少於五次以下的一律不使用，如此一來每個句子都可以轉成一個由index組成的sequence，再將這些 sequence 丟進 keras 的embedding layer 讓每個單字可以用一個長度300的word embedding來表示。接下來我在丟入rnn做training之前先將 embedding sequence 丟進一個CNN的layer，用一維的CNN和max pooling 來學習一些正向或負向的句子裡面不變的 feature，再將從CNN的挑出的 feature 作為 rnn 的 input sequence ，因此後面接上了一個 bidirectional 的GRU layer，之所以不選擇simpleRNN 是怕會有 gradient vanishing的問題，而 LSTM 訓練的參數數量容易讓實作時間比較長，因此最後選擇GRU，memory 大小設為256，計算完長度 20 的 input sequence 之後會得到一個 hidden vector，把它傳入一個長度是1024的 fully-connected network ，最後 output一個0到1之間的值。

訓練的過程我使用了 keras 裡面的 adadelta 作為 optimizer，loss function則是採用 keras 裡面的 binary_crossentropy，batch_size則是設為 32，訓練了大概 10 個epochs就會收斂了，最後大概validation accuracy會收斂在0.805至0.81之間，而上傳的test accuracy也大概落在0.808左右。

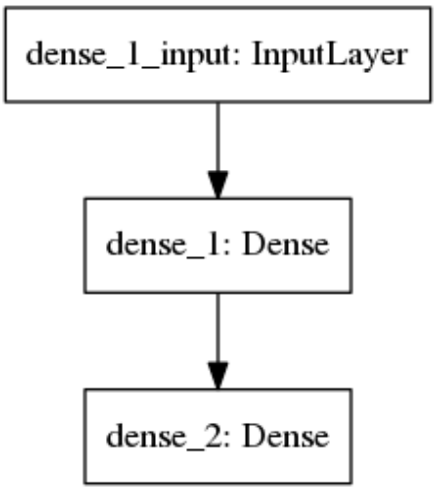


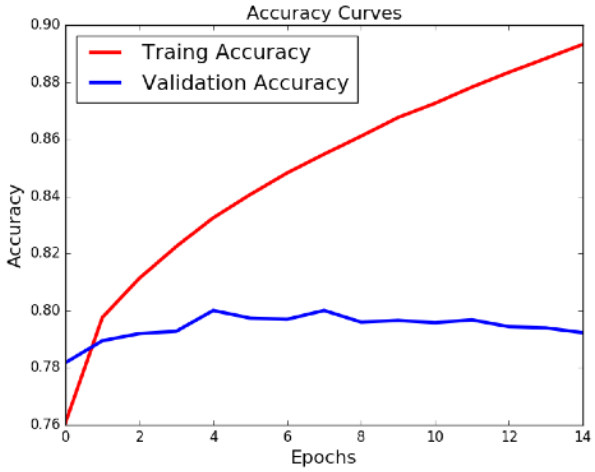


2.(1%) **請說明你實作的 BOW model，其模型架構、訓練過程和準確率為何？**

BOW的model 前置處理跟RNN model大同小異，一樣把training data整理出一個用出現頻率的高低標記成 index 的 dictionary，用這個 dictionary 把每個句子轉成一個數字sequence，次數五以下的捨棄，接下來就是把每個句子裡面的每個字所代表的數字，對應到一個初始都是0的陣列的 index 位置，把該 index 在陣列裡面的數字加1，代表這個單字出現過一次，如此一來一句的每個單字都標記過後，這個陣列就可以做為該句子的BOW representation，把這個 representation 後面接到一個長度為512的 fully connected network，最後 output一個0到1之間的值。

訓練的過程我使用了 keras 裡面的 adadelta 作為 optimizer，loss function則是採用 keras 裡面的 binary_crossentropy，batch_size則是設為 64，訓練了大概 15 個epochs，最後大概validation accuracy會收斂在0.796至0.801之間，而上傳的test accuracy也大概落在0.799左右





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。**

在 RNN 的 model 裡面可以得到這兩句話的情緒分數分別是0.994、0.998，而在 BOW 的 model 裡面得到這兩句話的情緒分數分別是 0.626、0.626，會有這樣的差異是因為 BOW 是針對一句話裡面出現過哪些單字，並不考慮字之間順序性的關係，因此這兩句話儘管長相不一樣，但因為都出現過一樣的單字，所以他們的 BOW representation 會長的一樣，也導致預測出來的結果是相同的。另一方面 RNN model 會考慮到字與字之間順序性的關係，把這樣的關係所透露出來的資訊作為預測考量，因此這兩句話所用的字都一樣，但是因為出現的順序不同，會有不同的預測數值，但都是正向的分數，而第二句會比較高應該是因為 good 這個字出現在字尾，更加肯定地表示了今天是個好日子，所以導致正向分略高。

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。**

比較了有無包含標點符號的兩種tokenize的方式之後可以發現，不包含標點符號時大概是accuracy = 0.804左右，加入標點符號之後則可以上升到 accuracy = 0.808 左右，兩者都是在不做 semi-supervised 的情況之下得到的結果。由此可見，在建立字典時把標點符號考慮進去是有益於 performance 的提升，如果仔細觀察 label data 裡面的句子，可以發現某些標點符號其實數量相當多，也確實可以表達某些意義，例如許多句子裡面有用驚嘆號來表達強烈情緒，可能是特別高興或興奮，或許就能幫助機器判斷出正面句子；而有些句子裡面有 ’ … ‘ 之類的符號，可能是用來表達無奈或悲傷的情緒，或許就能幫助機器判斷出負面句子，所以加入標點符號確實是有用的。但根據我的試驗結果，大概是加入 ’ ? ’ 、 ’ . ’ 、 ’ ! ’ 這三種有比較強的效果提升，如果是加入 ’ = ’ 、 ’ : ’ 、 ’ , ’ 就沒有太顯著的效果。

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。**

我先將一個用 label data 做好 training 且有不錯 performance 的 model load 進來，還有之前 train 這個 model 時建立好的dictionary 也 load 進來，用這個 dictionary 把 unlabel 的 data 一句一句用 index 標記成 sequence ，再丟進 model 去 predict 結果，根據每一句 predict 出來的結果我設定 0.85 跟 0.15 作為 threshold，大於 0.85 就將這句的 label 設為 1，而小於 0.15 的句子就把它的 label 設為 0，介於兩者之間的句子就捨棄，代表 model 還沒有足夠的信心判斷它的結果。這些標好的句子大概有80多萬句我隨機取了一半左右的數量作為額外的 training data 跟原來的 label data 合併起來作為新的 training data 重新 train 了一個 model來 predict 最後 testing data 的結果。

比較兩者semi-supervised 跟 supervised 兩種 training 方法的結果，可以看出來加入了從 unlabel data 裡面取得的新 data 之後，可以讓 model 的prediction accuracy 從 0.808 左右上升到 0.812左右，算是有小幅的上升，由此可見藉由 semi-supervised 取得 unlabel data 部分的句子做為額外的 training 資訊是有用的，但上升幅度不高我認為應該跟我的 label 方式有些關聯，只單純設定 threshold 並 random 取出部分句子似乎不夠嚴謹，因為 80多萬句裡面可能會有很多多出來的 noise 或是不相關的單字，或許可以多取幾個 model 並對這80多萬的句子用投票的方式來決定每一句的真正 label ，再來決定這句是否可以加入 trainging data 似乎比較嚴謹，這樣一來應該可以讓準確率更加提升。