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？
(Collaborators: b04901022 b04901040 b04507009)

答:

a. 模型架構

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 32, 128)	849536
gru_1 (GRU)	(None, 32, 512)	984576
gru_2 (GRU)	(None, 32, 512)	1574400
gru_3 (GRU)	(None, 32, 256)	590592
gru_4 (GRU)	(None, 32, 256)	393984
gru_5 (GRU)	(None, 32, 256)	393984
gru_6 (GRU)	(None, 256)	393984
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 1)	129

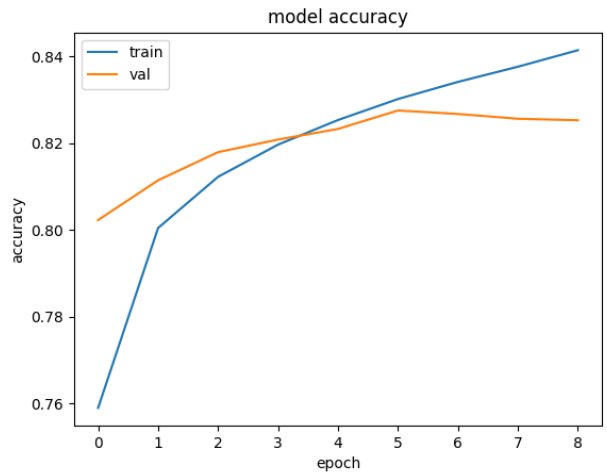
Total params: 5,230,593
Trainable params: 5,230,593
Non-trainable params: 0

Train on 180000 samples, validate on 20000 samples

模型介紹:
(1) 架構
1-1 Embedding Layer:
我使用 gensim 做為 word2vec 的輔助工具，將 train_label, train_nolabel, test 的文字整併為一個檔案後經由 gensim 做出 word2vec,並將這些 weight 直接用來初始化我的 embedding layer(設定 trainable)
1-2 RNN layers:
經過嘗試，我認為 GRU 的效果最好，因此直接疊代三層 512 unit & 三層 256 unit 的 GRU 做為將句子轉換的 hidden layer
1-3 DNN layers:
我疊代兩層 128 unit 的 dense 來做 DNN，再多疊反而會造成 training accuracy 上不去

上述模型是我在 kaggle 上得分最高的模型，其 kaggle 分數 0.82901, validation acc = 0.8287

b. 訓練過程與準確率



(1) 訓練過程:
如右圖所示，training acc 一直持續上升沒有下降，而 val_acc 則在第五 epoch 後開始下降。由於我有設定 earlystop(patience=3)，因此在第 8 epoch 時結束訓練
(2) 準確率:
val_acc max = 0.82755, kaggle 上顯示的 acc = 0.82901

c. 討論

(1) 在使用 gensim 做為 word2vec 前，我也使用過 glove 做為輔助預訓練 word2vec 的工具。雖然在自己的 validation acc 上表現跟 gensim 差不多(最高可以到 82.915%)，但是實際放到 kaggle 上後卻都不理想(最好只有 81.314%)。因此雖然網路的論文大多使用 glove，此次作業最終模型我還是選擇 gensim。
而直接使用 keras embedding layer 結果更是慘不忍睹，最好的模型 val acc 只有 80.0%上下，推測原因是因為由 twitter 爬下來的文句有太多口語用法的字(如 luv, anddddd 等)還有語法不正確比例比較高，因此在只有 20 萬筆資料之下能訓練出的 embedding layer 轉換能力非常有限。
(2) 在這次的實做中，使用 bidirectional layer 效果很有限，疊代超過兩層就會導致 val acc 下降，推測原因為正常英語語句的語法大多是單方向的(由左向右)，因此使用雙向的 RNN 貌似意義不大
(3) 直接使用 LSTM 其實也可以得到不錯的效果，成效與 GRU 相差不大，但是因為我訓練出使用 GRU 的模型好像都比較強一點，因此之後大多的模型我都以 GRU 實做。
(4) preprocessing 會大大影響最後的結果，最簡單的例子就是"字頻篩選"，選擇出現頻率比較高的字進行訓練可以得到明顯得進步(上升約 1.5%)，另外 test 的語句切割要把 id 斷乾淨，不然正確率會下降約 2%

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: no)

答:

a. 模型架構

```
EPOCH  20 BATCH_SIZE  256 SEMI_EPOCH  3 SEMI_BATCH_SIZE  128
(200000, 8266)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 2048)              16930816
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_7 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 257
=================================================================
Total params: 21,063,425
Trainable params: 21,063,425
Non-trainable params: 0
_________________________________________________________________
Train on 180000 samples, validate on 20000 samples
```

模型介紹:

(1) preprocessing:

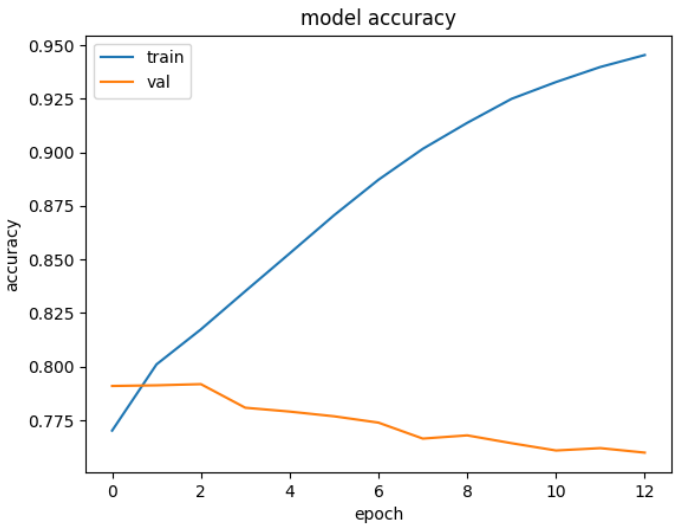
我使用 tokenizer 進行 preprocessing，實做非常單純，就是將整併好的文字檔丟入 tokenizer 讓其自動幫忙進行文字字典的建立。再將每個字都換成字典中的編號即可。

(2) DNN:

Input 為轉換為編號後的句子，不再有 embedding layer 而是直接以這些編號進行訓練。我的模型疊代了七層的 hidden layer，前 5 層每層都有 0.2~0.3 的 dropout

b. 訓練過程與準確率

訓練過程如下圖所示:



最終的模型 val acc 取最好時的準確率，為 78.96%

可以看到模型可能因為 BOW 的性質而很快的就 overfit 了(但其實也可能是我的模型參數過多，不過我不管如何調整 dropout 跟 layer 數好像都是這個趨勢)。不管疊代多少層，dropout 設定多少，通常在 5 epoch 以內都會 overfit。

c. 討論

BOW 的實做比起 RNN 輕鬆很多，概念也比較簡單。不過他最大的缺點就是沒有使用記憶單元(如 LSTM)，這會使訓練時至多只能針對文字組合進行機率分析，判斷某些字的出現會使機率更偏向 label 0/1，但是字的順序的資料就因為只論字數不論順序的性質而被捨棄了，對於語意分析無疑是一個很大的致命傷。

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。

(Collaborators: RNN: b04901022 b04901040 b04507009)

答:

a. 模型選擇: BOW 模型選擇和第 2 題相同的模型；而 RNN 的模型選擇疊代 6 層 GRU 的模型，如下圖所顯示:

```
EPOCH  20 BATCH_SIZE  128 SEMI_EPOCH  5 SEMI_BATCH_SIZE  256
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 128)         2560128
_________________________________________________________________
gru_1 (GRU)                  (None, None, 512)         984576
_________________________________________________________________
gru_2 (GRU)                  (None, None, 512)         1574400
_________________________________________________________________
gru_3 (GRU)                  (None, None, 512)         1574400
_________________________________________________________________
gru_4 (GRU)                  (None, None, 256)         590592
_________________________________________________________________
gru_5 (GRU)                  (None, None, 256)         393984
_________________________________________________________________
gru_6 (GRU)                  (None, 256)               393984
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 8,105,089
Trainable params: 8,105,089
Non-trainable params: 0
_________________________________________________________________
Train on 180000 samples, validate on 20000 samples
```

b. 預測結果

下表為顯示的預測結果:

BOW	RNN
`result of bow:  [[ 0.96888363  0.96888363]]` 1st 預測結果 = 2nd 預測結果=0.96888363 兩者預測結果相同，都傾向”正面”結果	`result of RNN:  [[ 0.13557912  0.96956831]]` 1st 預測結果=0.1356, 2nd 預測結果=0.9696 兩者預測結果不同，第一個為”負面”，第二個為”正面”

c. 結果探討

就結果而言，用 RNN 判斷出來的結果比較符合實際結果，即第一句是 label=0 而第二句是 label=1。先討論有關 BOW 自己的兩個預測出的機率是相等的理由，其原因為這兩句話所使用到的字與各字的字數是一模一樣的，因此對於 BOW 模型而言，input 是一模一樣的，它無法辨識這兩者的差異因而給予相同的預測。追根究柢乃是因為 BOW 的 input 其實已經犧牲了文字間的順序性，因此無法辨識由語句語法造成的語意歧異。反觀 RNN，因為其有記憶單元在內，因此 input 丟入的 word vector 順序是會影響輸出結果的，因此它可以學到語法對語意造成的影響。這也是 BOW 跟 RNN 兩個模型最大的不同。

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。

(Collaborators: RNN: b04901022 b04901040 b04507009)

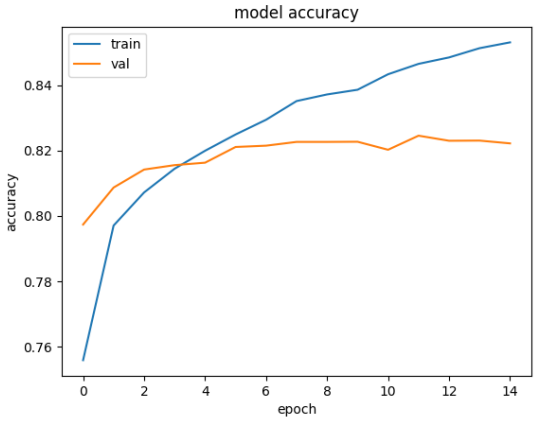
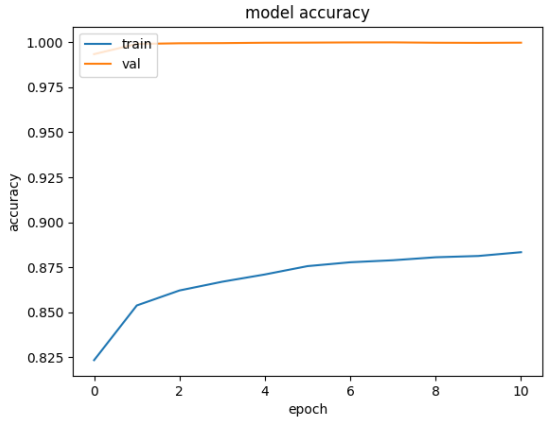
答:

a. 無標點符號模型

(a-1) 模型架構: 取跟第一題中的模型相同的參數，唯一不同的是在用 gensim 做 embedding 時先把所有的標點符號都濾除了。

(a-2) 模型訓練過程與正確率:

模型訓練過程如下圖 4:

圖 4(for Prob.4)	圖 5(for Prob.5)
	

其最好的 validation accuracy 在 epoch=12 時，val_acc=82.455% ，比有標點的低
而丟到 kaggle 後顯示成績為 82.461%, 亦遜於有標點的結果。

b. 討論

在英文中(由其 twitter 這種社群文章中)，情緒與標點符號的關連很大。因此，如果將所有飄演猜號都去除，可能造成無法更準確的判斷情緒。舉例如 “!!” 就可能有驚訝之類的感覺在裡面，貨多或少可以幫助判斷。但是也並非所有符號都需要，應該是要用 preprocess 的方式把出現率很低的特殊符號去除以免影響訓練進行。

簡而言之，有重要的標點符號(如?, !, ?!等等)可以保留，他們帶有比較強烈的語意色彩；而一些少見的特殊符號(如一些接近亂碼的字)就可以用字頻篩選或是手動剔除的方式來去除。

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。
(Collaborators: RNN: b04901022 b04901040 b04507009)

答:

a.如何標記 label:

我設定當用 labeled data 訓練出的模型去預測 unlabeled data 時得到超過 0.9 的機率值時，將其直接設定為 label=1; 相反的，當預測出的結果低於 0.1 的機率值時，就將其 label = 0。每次利用預測完重新標註的資料再重新訓練一次模型。

b. 有無 semi-supervise 對準確率的影響:

上圖 5 為 semi-supervised 的訓練過程，使用第一題中的模型做 pretrained model(只做一次):
跟據自己做出的所有結果顯示，訓練資料數大不一定是好的。在經過多次的 semi-supervise 訓練之後，得到的正確率往往會下降。目前比較好的一次是只使用 200000 筆資料做 semi-supervise，然後只做一次(reproduce 的模型盡量不要再拿去做 2nd semi-supervise train，偏差好像會越來越大)。目前我最好的模型仍然是只用 labeled data 做出的結果，可能原因可能是 corpus 的語法比較複雜，然後社交語言常常比較含糊的緣故。

簡而言之，semi-supervise learning 在我的這次作業中表現比較普通，沒有可以顯著帶動我的正確率上升的跡象。而且隨著資料量取越大，越容易 overfit(但是本例中無 overfit，應該是因為我的這次訓練新增加的資料量不過 90000 左右，比較少一點)，而用這種學習方式，val_acc 都很接近 1，因此可能會失去用 val_acc 判斷模型效果的依據

而不用 semi-supervise 好像影響不大，但是製作 word2vec 時其實有用到 unlabeled 的資料，或許適當的使用可以幫助增加準確率。我的最佳模型並沒有使用 semi-supervise learning，但是我有將其附於我的 train.py 黨之後。