學號：b06901045　系級： 電機三　　姓名：曹林熹

1. (1%) 請說明你實作的 RNN 的模型架構、word embedding 方法、訓練過程 (learning curve)和準確率為何？ (盡量是過 public strong baseline 的 model)

ANS：

看了助教的 sample code，我對可以調的參數統整如下：

可以改的地方

1. def train_word2vec(x): 訓練 word to vector 的 word embedding

```
model = word2vec.Word2Vec(x, size=250, window=5, min_count=5, workers=12, iter=10, sg=1) #原本的
```

2. class LSTM_Net(nn.Module):

```
def __init__(self, embedding, embedding_dim, hidden_dim, num_layers,  dropout=0.5, fix_embedding=True)
```

3. training

```
def training(batch_size, n_epoch, lr, model_dir, train, valid, model, device)
```

4. Ensemble Learning

```
Reference: http://violin-tao.blogspot.com/2018/01/ml-ensemble.html
```

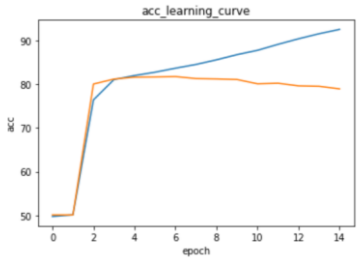
在 1. 部分，我的 word embedding 只改了 min_count = 6，讓篩選更嚴苛，此外其他都相同。

在 2. 部分，參數與 sample code 相同。

在 3. 部分，n_epoch = 15，其他也都與 sample code 相同。

在 4. 部分，我的 RNN 模型並未使用 ensemble 方法，因此暫無討論。

首先看到我們單純使用 label data 在 validation set ([:20000] 代表前 20000 筆資料) 上的表現，可以發現大約在 epoch = 6，val_curve (黃色) 有最好的表現。此作法使用 unsupervised，但是我接著仍使用 semi_supervised 方式。



我把原來用 label 資料 train 好的模型對 1,200,000 unlabel 資料預測，當我們判斷語句類型 > 0.8 or 語句類型 < 0.2 時，會把他們分別判斷為 1 or 0。

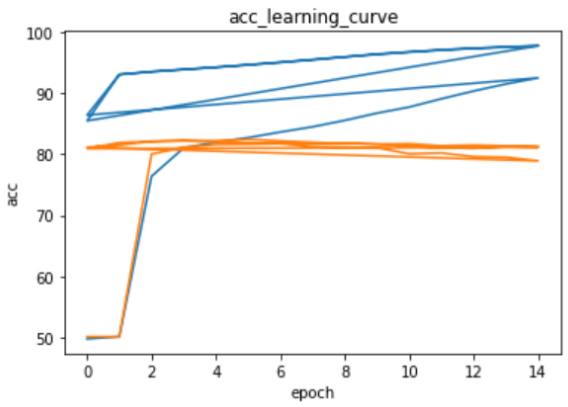
num of good = 735354（語句類型 => 0.8 or 語句類型 <= 0.2）
num of bad = 443260（語句類型 < 0.8 or 語句類型 > 0.2）

```
loading training_nolabel data ...
Get embedding ...
loading word to vec model ...
get words #21853
total words: 21855

load model ...
num of good =  735354
num of bad =  443260
save csv ...
Finish semi_supervised
```

我們把這些 ”嚴格篩選” 的 label 資料加到原來 label 好的 label 資料，再對這些新的 label 資料重新 train 一次。我把 val_set 設為原來 20000 筆，拿 [20000:500000]筆資料(總共有 735354+200000 = 935354，但我怕資料量太大所有只用部分）來 train，可以看到 val_acc 都約在 82% 左右，而在 epoch = 5，有 val_acc = 82.259 的高正確率。

```
LSTM_Net(
  (embedding): Embedding(37497, 250)
  (lstm): LSTM(250, 150, batch_first=True)
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=150, out_features=1, bias=True)
    (2): Sigmoid()
  )
)
```

```
loading data ...
Get embedding ...
loading word to vec model ...
get words #37495
total words: 37497

start training, parameter total:9615601, trainable:241351
```




```
Train | Loss:0.15807 Acc: 94.258
Valid | Loss:0.46718 Acc: 82.220
epoch = 4 , time cost =  32.08139371871948  sec(s)
-----------------------------------------------

Train | Loss:0.14786 Acc: 94.678
Valid | Loss:0.44863 Acc: 82.459
saving model with acc 82.459
epoch = 5 , time cost =  31.777169942855835  sec(s)
-----------------------------------------------

Train | Loss:0.13646 Acc: 95.060
Valid | Loss:0.48446 Acc: 82.181
epoch = 6 , time cost =  31.785238027572632  sec(s)
-----------------------------------------------

Train | Loss:0.12544 Acc: 95.531
Valid | Loss:0.49649 Acc: 81.947
epoch = 7 , time cost =  31.577646732330322  sec(s)
```

可以看到準確率有上升，使用此模型去對 test_set 預測後上傳 kaggle，可以過 strong baseline。

2. (2%) 請比較 BOW+DNN 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"這兩句的分數(過 softmax 後的數值)，並討論造成差異的原因。

ANS:

Id 0: 判斷負面

['today', 'is', 'a', 'good' ,'day', ',', 'but', 'it', 'is', 'hot']

Id 1: 判斷正面

['today', 'is', 'hot', ',', 'but', 'it', 'is', 'a', 'good', 'day']

BOW:

Id 0: 0.587395 / Id 1: 0.587395

兩個會相同是因為使用 BOW 並沒有順序性，而且在判斷語意並不會很有自信，沒有超過八成以上。且兩者皆判斷正面。

RNN:

Id 0: 0.221068 / Id 1: 0.997012

兩個會不同是因為使用 RNN 有順序性，而且在判斷語意並很有自信，兩者要碼有超過八成以上，或是低於兩成。兩者判斷皆正確。

3. (1%) 請敘述你如何 improve performance（preprocess、embedding、架構等等），並解釋為何這些做法可以使模型進步，並列出準確率與 improve 前的差異。（semi supervised 的部分請在下題回答）

ANS：

在 preprocess 中，我發現使用較大的 sen_len 會有比較好的效果，單純改 sen_len =20 至 40， 在 kaggle 的表現可從 0.80448 到 0.82112。將 sen_len 調大後，我們可以一次得到較長的句子，單次判斷的參數比較多，去計算時能得到較多的資訊。

4. (2%) 請描述你的 semi-supervised 方法是如何標記 label，並比較有無 semi-supervised training 對準確率的影響並試著探討原因（因為 semi-supervise learning 在 labeled training data 數量較少時，比較能夠發揮作用，所以在實作本題時，建議把有 label 的 training data 從 20 萬筆減少到 2 萬筆以下，在這樣的實驗設定下，比較容易觀察到 semi-supervise learning 所帶來的幫助）。

ANS：

我使用原來 train 好的 model 預測 unlabel 資料，重新 train 加上去的使用資料只有大於等於 0.8、小於等於 0.2 的預測資料，此部分在第一題有介紹。經過 semi-supervised 後，我在 kaggle 上的排名上升 50 名，也如願過 strong baseline (acc = 0.82740，總共多用了 100000 筆 unlabel 資料，外加上原來 200000 筆 labeled)。使用 semi 後，我們的資料量變多，所以參考的資料變多，訓練出來的模型相較準確。
我接著試著把所有 unlabeled 資料都丟到我的 train model 內，因此資料量超大。看看資料量極端大能否使 val acc 更好。(val_set 1000 筆，training_set 980282 筆)

```
loading data ...
Get embedding ...
loading word to vec model ...
get words #39545
total words: 39547

start training, parameter total:10128101, trainable:241351
```

沒想到居然比之前差了一點， kaggle 也才 0.823 多，應該是 overfitting 造成的問題。

```
Train | Loss:0.05789 Acc: 98.255
Valid | Loss:0.60886 Acc: 81.055
epoch = 12 , time cost =  65.30304002761841  sec(s)
---------------------------------------------

Train | Loss:0.05531 Acc: 98.318
Valid | Loss:0.58796 Acc: 81.250
epoch = 13 , time cost =  65.28082847595215  sec(s)
---------------------------------------------

Train | Loss:0.05330 Acc: 98.374
Valid | Loss:0.64242 Acc: 81.445
saving model with acc 81.445
epoch = 14 , time cost =  65.61391735076904  sec(s)
---------------------------------------------

Train | Loss:0.05149 Acc: 98.436
Valid | Loss:0.65408 Acc: 80.664
epoch = 15 , time cost =  64.76785278320312  sec(s)
```

我接著把原本的 labeled 資料只挑 10000 筆出來重 train，其餘皆為我的 unlabeled 資料被 label 後產生出的資料 (semi) ，用來看出 semi-supervised 特性。出乎我意料，在 val_acc 表現進步了，但是 kaggle 卻是退步的。因此原本的 label data 也不能太少，才可以得到較好的模型。

```
loading data ...
Get embedding ...
loading word to vec model ...
get words #39545
total words: 39547

start training, parameter total:10128101, trainable:241351
```

```
[85] Train | Loss:0.03525 Acc: 99.015
     Valid | Loss:0.79415 Acc: 82.684
     epoch = 13 , time cost =  15.979857921600342  sec(s)
     ---------------------------------------------

     Train | Loss:0.03230 Acc: 99.131
     Valid | Loss:0.77286 Acc: 82.654
     epoch = 14 , time cost =  15.921482563018799  sec(s)
     ---------------------------------------------

     Train | Loss:0.02916 Acc: 99.198
     Valid | Loss:0.82345 Acc: 83.000
     saving model with acc 83.000
     epoch = 15 , time cost =  16.061305046081543  sec(s)
     ---------------------------------------------

     Train | Loss:0.02804 Acc: 99.262
     Valid | Loss:0.85143 Acc: 82.644
     epoch = 16 , time cost =  16.086255073547363  sec(s)
     ---------------------------------------------
```

22	b06901045_DPGOD		0.82785	18	now

Your Best Entry ↑

Your submission scored 0.82011, which is not an improvement of your best score. Keep trying!