Embedding

2018 年 8 月 3 日

1 文字的深度學習 - 詞嵌入 + MLP

1.1 介紹

對於文字的深度學習，我們一樣遇到一個很嚴重的問題，當時我們在單純貝氏的時候，我們是把文字變成一個超高維度的向量，假設直接拿來這裡使用的話，可以想見一定是個不適用的方法，因為太稀疏了，所以你每一個訓練都跟沒訓練一樣，而且參數 w 的數量也多到不可思議

所以這裡我們要做一件很特別的事，也就是『降低維度』

1.2 降低維度

事實上，仔細研究我們的文字，我們會發現，其實我們可能不需要這麼大的維度，因為很多文字的意思是一樣的啊，譬如在討論好感度的文章，『喜歡』和『愛』應該是同一個維度的東西，只是程度的不同吧

沒錯，我們在處理文字的時候，通常做的第一件事就是降低維度，下面我們來看看我們對於降低維度 (詞嵌入) 的處理步驟

1.3 詞嵌入 (Embedding)

詞嵌入是我們給每個詞降低維度比較專業的稱呼，我們一起來看看最基本的處理步驟

1.3.1 選擇你的輸入維度

首先，你要決定你的最大輸入維度，白話的來說，就是建立一個跟維度一樣大的字典，舉個例子，假設我有一個句子

This is an apple.

我會建立一個四個字的字典，而且賦予他們一個數字

1	2	3	4
This	is	an	apple

我們就可以把上面的句子換成數字

1 2 3 4

跟之前單純貝氏不一樣的是，我們現在要對每一個詞處理，而不是對一整篇文章處理

所以其實上面的數字序列你可以表示成跟字典維度一樣大的向量序列

[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]

接下來再來做出我們下一步的處理

1.3.2 降低輸入維度

舉個例子

如果你可以有一個轉換的方式，不管是根據辭意或者根據討論議題，把它轉換成

[2, 1], [1, 2], [3, 3], [3, 2]

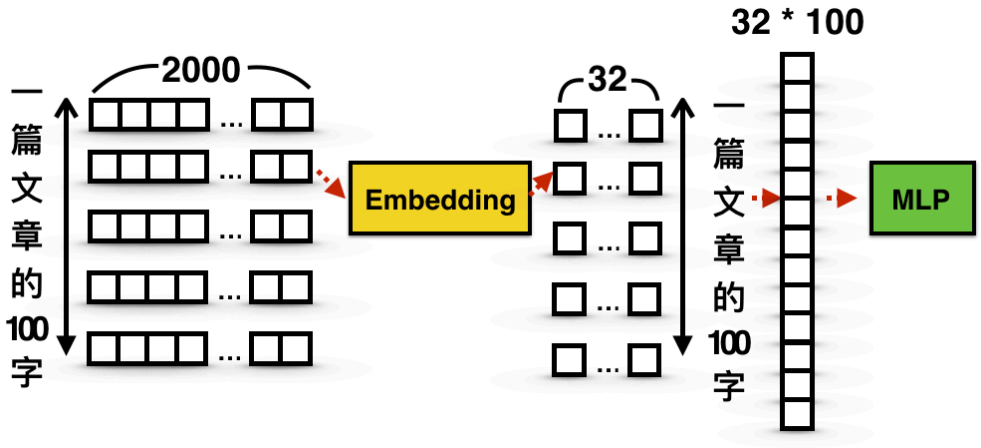
每一個詞都開始變成兩個維度的向量，你就可以開始做很多事了

譬如你的 kNN 會開始有比較好的效果 (稀疏度減少)，可以開始把詞歸類成不同群體

又或者你可以把特徵攤開，做一個深度的 MLP

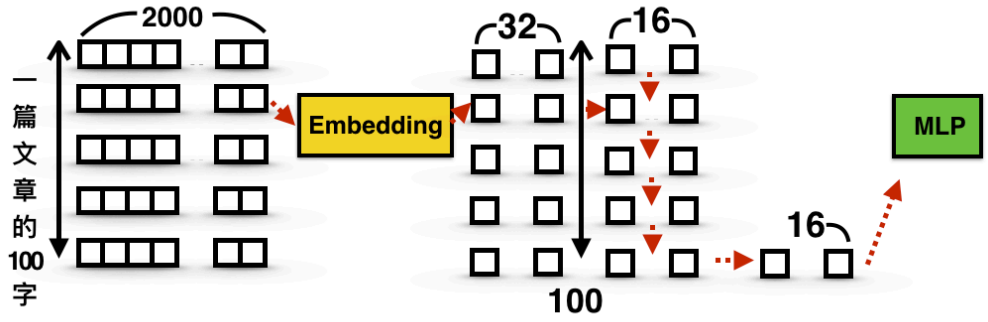
1.4 加入 MLP 做出分類

利用我們詞嵌入和 MLP 加在一起，就變成下面這樣

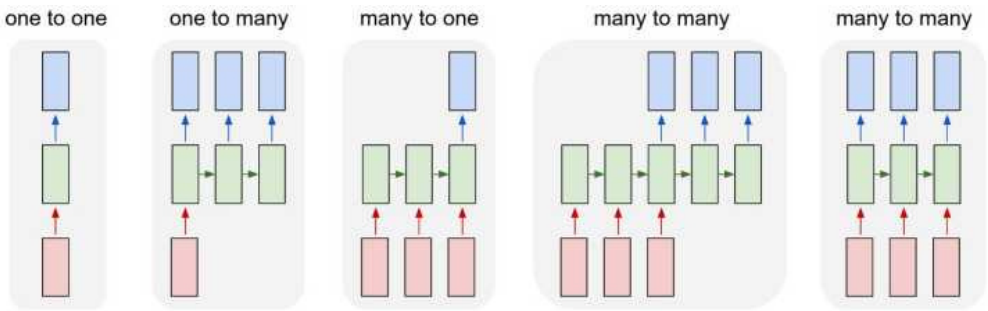


1.5 RNN 記憶單元

利用完詞嵌入和 MLP，你已經可以得到一個不錯的結果了
但是你應該會有一個疑惑，我們有些詞是要根據上下文來推斷他的意思
或者要加在一起考慮 (看到後面的『美食』應該把剛剛記憶中的『英國』拿出來)
這是我們的 RNN 記憶單元所負責的事物
簡單來說，就是把每一個詞化成狀態 (可能是地點，喜歡...之類) 神經元
然後把狀態一直累加下去，直到最後把最後累加的狀態輸出
我們最常使用的 RNN 由於只輸出最後的狀態，所以我們叫它 many-to-one 的 RNN



另外還有不同的記憶單元 (ex. many-to-many 每個都輸出，代表我想知道看到每個字的記憶狀態)



我們就不在這介紹了

1.6 Step1. 資料預處理

這裡我們選用 imdb 的資料庫，但我們不直接用 keras 裡面的 imdb，因為裡面已經把原始的文字處理掉了，我們從網路下載原版的 imdb 資料庫

imdb 資料集已經幫你把每一篇 imdb 影評標註成『正面』和『負面』

imdb 資料集裡總共有 25000 訓練資料 (一半正一半負) 25000 測試資料 (一半正一半負)

我們就要藉由 Embedding 加上 MLP 做出二元的分類

下載網址是: http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

```
In [1]: # urlretrieve 是一個方便的東西
        # 他直接結合 urlopen + file.write 幫你做完儲存工作
        from urllib.request import urlretrieve
        import os
        # MAC 要加入這段, SSL 證書才不會被視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context

        url = "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
        # 如果 data 資料夾不存在就創一下
        if not os.path.exists("./data"):
            print("data 資料夾不存在, 現在幫你創唷")
            os.mkdir("./data")
        # 還沒下載過就下載一下
        filepath = "./data/imdb.tar.gz"
        if not os.path.exists(filepath):
            print("還沒下載過資料, 現在幫你下載唷")
            urlretrieve(url, filepath)
        else:
            print("已下載過")
```

已下載過

我們可以使用內建的 tarfile 幫我們解壓縮

In [2]:
```
import tarfile
if not os.path.exists("data/aclImdb"):
    print("還沒解壓縮過, 現在幫你解壓縮")
    tfile = tarfile.open(filepath, 'r')
    tfile.extractall('data')
else:
    print("已解壓縮過")
```

已解壓縮過

整理一下我們的訓練資料

In [3]:
```
import pandas as pd
# 為了顯示的漂亮, 我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)
pd.set_option('display.max_columns', 10)

train_df = pd.DataFrame(columns = ["content", "sentiment"])

# 走過 pos 的資料夾, 把資料夾整理好
pos_path = "data/aclImdb/train/pos"
for fpath in os.listdir(pos_path):
    if not fpath.startswith("."):
        fpath = os.path.join(pos_path, fpath)
        f = open(fpath, "r", encoding = "utf-8")
        content = f.read()
        s = pd.Series([content, 1], index = ["content", "sentiment"])
        train_df = train_df.append(s, ignore_index = True)

# 走過 neg 的資料夾, 把資料夾整理好
neg_path = "data/aclImdb/train/neg"
for fpath in os.listdir(neg_path):
```

```
        if not fpath.startswith("."):
            fpath = os.path.join(neg_path, fpath)
            f = open(fpath, "r", encoding = "utf-8")
            content = f.read()
            s = pd.Series([content, 0], index = ["content", "sentiment"])
            train_df = train_df.append(s, ignore_index = True)

train_df
```

Out[3]:

	content	sentiment
0	Bromwell High is a cartoon comedy. It ran at t...	1
1	Homelessness (or Houselessness as George Carli...	1
2	Brilliant over-acting by Lesley Ann Warren. Be...	1
3	This is easily the most underrated film inn th...	1
4	This is not the typical Mel Brooks film. It wa...	1
5	This isn't the comedic Robin Williams, nor is ...	1
6	Yes its an art... to successfully make a slow ...	1
...	...	...
24993	Although the production and Jerry Jameson's di...	0
24994	Capt. Gallagher (Lemmon) and flight attendant ...	0
24995	Towards the end of the movie, I felt it was to...	0
24996	This is the kind of movie that my enemies cont...	0
24997	I saw 'Descent' last night at the Stockholm Fi...	0
24998	Some films that you pick up for a pound turn o...	0
24999	This is one of the dumbest films, I've ever se...	0

[25000 rows x 2 columns]

整理測試資料

```
In [4]: test_df = pd.DataFrame(columns = ["content", "sentiment"])

        pos_path = "data/aclImdb/test/pos"
        for fpath in os.listdir(pos_path):
            if not fpath.startswith("."):
                fpath = os.path.join(pos_path, fpath)
                f = open(fpath, "r", encoding = "utf-8")
                content = f.read()
```

```
            s = pd.Series([content, 1], index = ["content", "sentiment"])
            test_df = test_df.append(s, ignore_index = True)

    neg_path = "data/aclImdb/test/neg"
    for fpath in os.listdir(neg_path):
        if not fpath.startswith("."):
            fpath = os.path.join(neg_path, fpath)
            f = open(fpath, "r", encoding = "utf-8")
            content = f.read()
            s = pd.Series([content, 0], index = ["content", "sentiment"])
            test_df = test_df.append(s, ignore_index = True)

    test_df
```

Out[4]:

	content	sentiment
0	I went and saw this movie last night after bei...	1
1	Actor turned director Bill Paxton follows up h...	1
2	As a recreational golfer with some knowledge o...	1
3	I saw this film in a sneak preview, and it is ...	1
4	Bill Paxton has taken the true story of the 19...	1
5	I saw this film on September 1st, 2005 in Indi...	1
6	Maybe I'm reading into this too much, but I wo...	1
...	...	...
24993	This is one dreary, inert, self-important bore...	0
24994	Awful, awful, awful times a hundred still does...	0
24995	I occasionally let my kids watch this garbage ...	0
24996	When all we have anymore is pretty much realit...	0
24997	The basic genre is a thriller intercut with an...	0
24998	Four things intrigued me as to this film - fir...	0
24999	David Bryce's comments nearby are exceptionall...	0

[25000 rows x 2 columns]

把我們訓練資料找出最常見的 2000 字拿來建立出 2000 維度的字典

```
In [5]: from keras.preprocessing.text import Tokenizer
        token = Tokenizer(num_words=2000)
        token.fit_on_texts(train_df["content"])
```

```
# 我省略了這裡的印出，讀者可以把註解秀出字典的樣子
# token.word_index
```

```
Using TensorFlow backend.
```

利用字典把我們整理好的 DataFrame 每一個詞變成數字

```
In [6]: x_train_seq = token.texts_to_sequences(train_df["content"])
        x_test_seq = token.texts_to_sequences(test_df["content"])
        pd.DataFrame(x_train_seq)
```

Out[6]:

	0	1	2	3	4	...	1706	1707	1708	1709	1710
0	309	6	3	1069	209	...	NaN	NaN	NaN	NaN	NaN
1	39	14	739	44	74	...	NaN	NaN	NaN	NaN	NaN
2	526	117	113	31	1957	...	NaN	NaN	NaN	NaN	NaN
3	11	6	711	1	88	...	NaN	NaN	NaN	NaN	NaN
4	11	6	21	1	797	...	NaN	NaN	NaN	NaN	NaN
5	11	215	1	1714	1693	...	NaN	NaN	NaN	NaN	NaN
6	419	91	32	495	5	...	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
24993	259	1	362	2	1514	...	NaN	NaN	NaN	NaN	NaN
24994	2	23	3	1501	890	...	NaN	NaN	NaN	NaN	NaN
24995	946	1	127	4	1	...	NaN	NaN	NaN	NaN	NaN
24996	11	6	1	240	4	...	NaN	NaN	NaN	NaN	NaN
24997	10	216	233	311	30	...	NaN	NaN	NaN	NaN	NaN
24998	46	105	12	22	1259	...	NaN	NaN	NaN	NaN	NaN
24999	11	6	28	4	1	...	NaN	NaN	NaN	NaN	NaN

```
[25000 rows x 1711 columns]
```

你發現每一篇文章的長度不一樣，所以我們截長補短把每一篇文章取 100 個字來做訓練 (訓練的時候無法處理長短不同的文章)

```
In [7]: from keras.preprocessing import sequence
        x_train_pad = sequence.pad_sequences(x_train_seq, maxlen = 100)
        x_test_pad = sequence.pad_sequences(x_test_seq, maxlen = 100)
        pd.DataFrame(x_train_pad)
```

```
Out[7]:          0     1    2    3     4   ...    95    96   97    98    99
        0       30     1  169   55    14   ...    48     3   12     9   215
        1       27   553    7    7   134   ...    77    22    5   335   405
        2        8  1640   23  330     5   ...     9    60    6   176   396
        3       88    19    1  249    91   ...    35    73   14     3   482
        4       73   326   71   88     4   ...   196   253   65   528    70
        5       24     7    7   79  1180   ...   155    36    7     7     1
        6        0     0    0    0     0   ...     2   839    3   343    62
        ...    ...   ...  ...  ...   ...  ...   ...   ...  ...   ...   ...
        24993  222    54  757    4     9   ...  1246  1586  848    30    36
        24994  203     2  841  130    23   ...    37    12    2     7     7
        24995    3   195  135   10   298   ...     5   103    3   411    76
        24996   75    14   10   83   194   ...    41  1568   37     4     1
        24997    9     6   49  846    18   ...    80    11   17    96    75
        24998    1    55    9  211     5   ...   105     8  260  1195   794
        24999  683    49  344   39   106   ...   126    55   11     6  1350

        [25000 rows x 100 columns]
```

秀個大家看一下經過我們 padding 成為 100 以後的資料長成如何

```
In [8]: x_train_pad[0]
```

```
Out[8]: array([  30,    1,  169,   55,   14,   46,   82,   41,  392,  110,  138,
                 14,   58,  150,    8,    1,  482,   69,    5,  261,   12,    6,
                 73,    5,  632,   71,    6,    1,    5,    1, 1534,   34,   67,
                 64,  205,  140,   65, 1230,    1,    4,    1,  223,  901,   29,
                 69,    4,    1,   10,  693,    2,   65, 1534,   51,   10,  216,
                  1,  387,    8,   60,    3, 1467,  800,    5,  177,    1,  392,
                 10, 1237,   30,  309,    3,  353,  344,  143,  130,    5,   28,
                  4,  126, 1467,    5,  309,   10,  532,   12,  108, 1468,    4,
                 58,  555,  101,   12,  309,    6,  227,   48,    3,   12,    9,
                215], dtype=int32)
```

1.7 Step2. 建立模型

這裡我們一樣使用 Sequential 模型

但我們開始用一個新的 Layer

1.7.1 Embedding 層

Keras 內建詞嵌入層，可以幫你把 2000 維度的每一個詞向量換成 32 維度的詞向量

```
In [9]: from keras.models import Sequential
        from keras.layers.core import Dense, Dropout, Flatten
        from keras.layers.embeddings import Embedding

        model = Sequential()
        model.add(Embedding(output_dim=32, input_dim=2000, input_length=100))
        model.add(Dropout(0.2))
        model.add(Flatten())
        model.add(Dense(units=256, activation='relu'))
        model.add(Dropout(0.35))
        # 注意一下，因為我們是二元分類，最後的激勵函數選擇 sigmoid
        # sigmoid(正 + 負 =100%) softmax(類別全部 =100%)
        model.add(Dense(units=1, activation='sigmoid'))

In [10]: model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 32)           64000
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 32)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               819456
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 883,713
Trainable params: 883,713
Non-trainable params: 0
```

```
_________________________________________________________________
```

1.7.2 Param 個數

我們一起看看上方 summary 的參數個數

1. Embedding 層的 64000: 2000(輸入詞維度) x 32(輸入詞維度) = 64000
2. Flatten 層的 3200 輸入維度: 32(詞維度) * 100(文章的字數) = 3200

```
In [11]: import numpy as np
         # 特別注意一下, 因為我們只是二元分類, 所以這裡的 loss 選擇 binary_crossentropy
         model.compile(loss='binary_crossentropy',
                       optimizer='adam',
                       metrics=['accuracy'])
         from keras.datasets import imdb
         y_train = train_df['sentiment']
         train_history = model.fit(x_train_pad, y_train,
                                   batch_size = 100,
                                   epochs = 3,
                                   verbose = 2,
                                   validation_split = 0.2)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/3
 - 4s - loss: 0.4770 - acc: 0.7572 - val_loss: 0.5236 - val_acc: 0.7576
Epoch 2/3
 - 4s - loss: 0.2715 - acc: 0.8897 - val_loss: 0.5791 - val_acc: 0.7472
Epoch 3/3
 - 4s - loss: 0.1595 - acc: 0.9420 - val_loss: 0.5526 - val_acc: 0.7932
```

根據 val_loss，我們可以知道大概在 3-4 個 epoch 的時候，模型就已經完成了訓練
繼續訓練下去只是增加過擬合的程度

```
In [12]: y_test = test_df['sentiment']
         # 正確率是 list 第二個元素
         model.evaluate(x_test_pad, y_test)
```

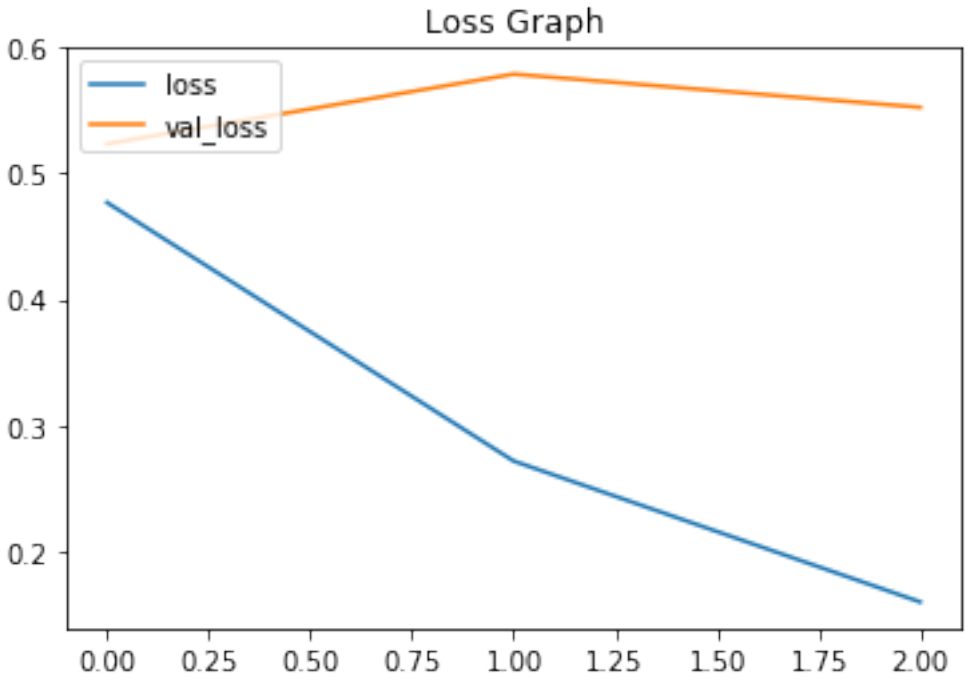
```
25000/25000 [==============================] - 1s 43us/step
```

```
Out[12]: [0.45064449694633485, 0.82524]
```

你可以看到正確率大概已經在 83%，是一個非常不錯的結果了

```
In [13]: import matplotlib.pyplot as plt
         %matplotlib inline
         plt.plot(train_history.history["loss"])
         plt.plot(train_history.history["val_loss"])
         plt.title("Loss Graph")
         plt.legend(['loss', 'val_loss'], loc="upper left")
```

```
Out[13]: <matplotlib.legend.Legend at 0x138f91550>
```



1.7.3 中間層輸出

我想讓你看看到底 2000 維的向量是怎麼被轉成 32 維的向量，又是長怎麼樣的 keras 可以藉由 Model 拿出你剛剛模型的某幾層，創立出一個新的模型

```
In [14]: # 模型的 Layer List
         model.layers
```

```
Out[14]: [<keras.layers.embeddings.Embedding at 0x11c447e10>,
          <keras.layers.core.Dropout at 0x11c447198>,
          <keras.layers.core.Flatten at 0x104a62b00>,
          <keras.layers.core.Dense at 0x11ae4d780>,
          <keras.layers.core.Dropout at 0x11c4653c8>,
          <keras.layers.core.Dense at 0x11c4882e8>]
```

```
In [15]: # 拿最初的 input 和第一層 (embedding) 的輸出拿來當新模型
         from keras.models import Model
         embedding_layer_model = Model(inputs=model.input,
                                       outputs=model.layers[0].output)
         # 把第一筆文章拿來給你看轉換後的維度
         em = embedding_layer_model.predict(x_test_pad[0:1])
         em
```

```
Out[15]: array([[[ 0.04700758,  0.01377721,  0.10867094, ..., -0.02624216,
                    0.03142255, -0.09424717],
                  [-0.00131393, -0.02347459,  0.00862844, ...,  0.01718462,
                    0.00600311, -0.01461102],
                  [ 0.05301251,  0.06678709, -0.00761796, ..., -0.01124587,
                   -0.02221869,  0.01999949],
                  ...,
                  [ 0.02881096, -0.03530543, -0.01820028, ..., -0.0320297 ,
                   -0.0836475 , -0.07444255],
                  [-0.06361311, -0.01159843,  0.03845826, ..., -0.01422905,
                    0.0371515 , -0.09605152],
                  [ 0.08404704, -0.07338575,  0.00265952, ...,  0.01804166,
                    0.01961992, -0.04136086]]], dtype=float32)
```

```
In [16]: print("維度:", em.shape)
         print("第一個詞被轉換過的向量:", em[0][0])
```

```
維度: (1, 100, 32)
第一個詞被轉換過的向量: [ 0.04700758  0.01377721  0.10867094  0.08715586 -0.00013417  0.0251381
  0.0326918   0.03093791 -0.01146789  0.05069069 -0.00337323  0.0022066
  0.01615707  0.03905306 -0.02227833  0.019749   -0.09394031  0.09523332
  0.08358718 -0.0468246   0.03339849 -0.00626512  0.04425366  0.06629214
  0.03313974  0.06694432  0.01610038  0.00765152  0.0225146  -0.02624216
```

```
  0.03142255 -0.09424717]
```

1.8 Step3. RNN

試試看 RNN 吧

RNN 參數個數: 32 x 16 (每一個詞都由 32 維度轉成 16 維度狀態) + 16 x 16 (往下傳遞的記憶) + 16 (輸出的參數) = 784

```
In [26]: from keras.layers import SimpleRNN
```

```
model = Sequential()
model.add(Embedding(output_dim=32, input_dim=2000, input_length=100))
model.add(Dropout(0.2))
# RNN: 記憶 16 個狀態
model.add(SimpleRNN(units=16))
model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.35))
# 注意一下, 因為我們是二元分類, 最後的激勵函數選擇 sigmoid
# sigmoid(正 + 負 =100%) softmax(類別全部 =100%)
model.add(Dense(units=1, activation='sigmoid'))
# 特別注意一下, 因為我們只是二元分類, 所以這裡的 loss 選擇 binary_crossentropy
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 100, 32)           64000
_________________________________________________________________
dropout_13 (Dropout)         (None, 100, 32)           0
_________________________________________________________________
simple_rnn_5 (SimpleRNN)     (None, 16)                784
_________________________________________________________________
dense_13 (Dense)             (None, 256)               4352
_________________________________________________________________
```

```
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 257
=================================================================
Total params: 69,393
Trainable params: 69,393
Non-trainable params: 0
_________________________________________________________________
```

你可以看到透過 RNN，我們還蠻常可以得到一個更好的結果
這裡就得到了約 1% 的正確率提升 (0.84)

```
In [27]: train_history = model.fit(x_train_pad, y_train,
                                   batch_size = 100,
                                   epochs = 3,
                                   verbose = 2,
                                   validation_split = 0.2)
         y_test = test_df['sentiment']
         # 正確率是 list 第二個元素
         model.evaluate(x_test_pad, y_test)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/3
 - 5s - loss: 0.4810 - acc: 0.7669 - val_loss: 0.6841 - val_acc: 0.6846
Epoch 2/3
 - 4s - loss: 0.3380 - acc: 0.8588 - val_loss: 0.4044 - val_acc: 0.8340
Epoch 3/3
 - 4s - loss: 0.2888 - acc: 0.8803 - val_loss: 0.3504 - val_acc: 0.8634
25000/25000 [==============================] - 4s 163us/step
```

```
Out[27]: [0.37006479842185974, 0.84144]
```

1.9 結語

我們已經完成了文字的基本預測，利用詞嵌入模型加上我們之前學會的深度學習，完成情緒的預測

但其實我們的選取都選的比較小 (計算量較小)

你可以自行調整字典的大小 (ex: 2000 -> 5000)，調整我們文章取出的字數 (ex: 100 -> 500) 再預測看看