

Bonus-D：使用開源的中文預訓練詞向量進行話題分類

在第 6 章的 10 分類話題任務，我們的神經網路止步於約 74% 的準確率，如果想要更上一層樓，就得做點改變。在這裡我們便會帶各位讀者使用開源的預訓練詞向量來看看是否能突破當前的瓶頸，讓神經網路的準確率再創新高。

騰訊 AI 實驗室開源的中文詞向量

本實驗將會使用由騰訊 AI 實驗室所推出的簡體中文詞向量，此資料集不僅規模龐大、質量高，而且還是開源的，能幫助不少以中文為母語的開發者。

這個詞向量包含了 800 多萬的詞彙量，每個詞的向量維度則為 200。其中收錄了不少簡體中文的流行用語，像是 "神吐槽"、"因吹斯听"（來自英文的 interesting）等等，訓練詞向量的語料則來自於騰訊新聞、中文網頁及中文小說語料。

由於我們先前的社區問答資料也是簡體中文的，因此使用騰訊中文詞向量會是相當合適的選擇，首先我們來下載這個詞向量並看看它的格式吧。

開源詞向量的格式

要下載這個詞向量可以開啟瀏覽器，在網址列輸入以下連結：https://ai.tencent.com/ailab/nlp/data/Tencent_AILab_ChineseEmbedding.tar.gz。這是一個大小高達 6 GB 多的壓縮檔，解壓縮後則是 16 GB 多的文本資料，下載前請先確認電腦的剩餘空間。另外也能使用 Keras 的資料下載工具 `get_file()` 進行下載，可以看到剩餘下載時間並自動解壓縮（詳細使用方法請參見 5-5-1 節），以下為使用 `get_file()` 下載到工作區的程式：

BonusD.0.py 下載公開 (騰訊) 預訓練詞向量


```
from tensorflow.keras.utils import get_file
import os

path = os.getcwd()+os.sep+'Tencent_AILab_ChineseEmbedding.tar.gz'
#若是在 Colab 上執行, 建議在掛載 Google 雲端硬碟後, 指定路徑在雲端硬碟的資料夾中

origin = 'https://ai.tencent.com/ailab/nlp/data/Tencent_AILab_ChineseEmbedding.tar.gz'
cache_dir = os.getcwd()+os.sep+'cache' ← 解壓縮的路徑
#若是在 Colab 上執行, 建議指定路徑在雲端硬碟中

path = get_file(path,
                origin,
                cache_dir=cache_dir,
                extract=True
                )
```

get_file() 要指定路徑為絕對路徑, 所以如果只知道相對路徑, 可以使用此方法

 **tips** 建議讀者可以在 Colab 上掛載 Google 雲端硬碟, 並將以上程式中的路徑改為雲端中的資料夾, 並在 Colab 上執行, 這樣一來便能將檔案下載到雲端中, 之後的程式也能直接在 Colab 上執行, 節省運算時間 (Colab 的詳細使用方法請參見附錄 A)。

如果是使用連結下載的話, 請將下載好的檔案解壓縮, 用上述程式下載的話, 檔案會解壓縮到 cache/datasets (讀者指定路徑的 datasets 資料夾中)。解壓縮完的檔案為 "Tencent_AILab_ChineseEmbedding.txt", 大小為 16 GB 多, 由於一般的文字編輯軟體無法處理這麼大的檔案, 所以必須使用程式來讀取：

BonusD.0.py(續) 讀取部分的詞向量檔案

```
wv_path = 'cache/datasets/Tencent_AILab_ChineseEmbedding.txt'
with open(wv_path,encoding='utf-8-sig') as file:
    for _ in range(5): ← 讀取前 5 行
        print(file.readline())
```

顯示出來的文字即為文本檔案中的前 5 行：

```

8824330 200 ← 第 1 行為該詞向量檔案的資訊, 其中第 1 個
                數字代表詞彙量、第2個數字則是詞向量維度
</s> 0.002001 0.002210 -0.001915...(共有 200 個數字) ← 第 2 行開始為詞與
的 0.209092 -0.165459 -0.058054...(共有 200 個數字)    對應的向量, 文字與
    0.128825 -0.267995 0.000795...(共有 200 個數字)    數字間皆以空格分隔
, -0.098885 -0.335186 -0.092543...(共有 200 個數字)

```

這是大多數公開詞向量的儲存格式, 而且皆以文本檔案儲存, 因此不管開發者使用什麼框架 (例如非 tensorflow/keras) 皆能將其轉換為符合自己需要的格式。

實例：使用騰訊的開源詞向量進行 10 類話題分類

了解資料格式後, 我們之後便能處理這個檔案, 並將詞向量載入自行建立的模型中。

將資料集轉成簡體字

我們在 6-3-0 節製作的資料集已將簡體轉為繁體, 而騰訊的詞向量為簡體的, 所以也須使用簡體的資料集, 讀者可以將 6-3.1.py 中設定斷詞工具 jieba 為繁體字庫的程式碼註解掉, 並將簡體轉繁體的部分移除, 再重新執行一次。或建立以下 Python 程式並執行, 直接將處理好的資料集 (dataSet.txt) 由繁體轉回簡體：

BonusD.1.py 將簡體資料集轉為繁體

```

from opencc import OpenCC

f = open('dataSet.txt', encoding='utf-8-sig')
new_f = open('simp_dataSet.txt', mode = 'w', encoding='utf-8-sig')

cc_t2s = OpenCC('t2s')
for i,line in enumerate(f):
    print('\r已處理',i,'個句子',end='')
    new_f.writelines(cc_t2s.convert(line))

f.close()
new_f.close()
print('轉換完成')

```

執行後便會得到 "simp_dataSet.txt" 的簡體資料集, 此檔案也會附在範例程式中, 因此讀者可以直接使用。

接著請開啟書附範例的 6-4.4.py 並另存新檔為 BonusD.2.py, 然後進行以下修改。首先更改調整區中的 topics 為簡體字：

BonusD.2.py	更改調整區
<pre># %% ----- 調整區 ----- # ...(略) topics = ['音', '政治', '游', '影', '法律', ← 改成簡體字 '情感', '健康', '教育', '美食', '旅行']</pre>	

載入資料集的 topic_contents 也改成簡體字：

BonusD.1.py(續)	更改載入資料集
<pre># ----- 載入資料集 ----- # # 將資料依據話題種類存入 topic_contents = {'影': [], '法律': [], '游': [], '音': [], '政治': [], '情感': [], '健康': [], '教育': [], '美食': [], '旅行': [],} ← 改成簡體字</pre>	

刪除載入 .pickle 檔的程式片段, 因為原本建立的 Tokenizer 物件是繁體的, 所以要重新建立一個簡體的 Tokenizer, 我們可以在載入資料集時同時建立簡體版的 Tokenizer。更改載入資料集的程式如下：

BonusD.1.py(續)	更改載入資料集
<pre># %% ----- 載入資料集 ----- # ...(略) # 載入資料集 from tensorflow.keras.preprocessing.text import Tokenizer tokenizer = Tokenizer(num_words=max_words, oov_token='N') ← 建立一個新的 Tokenizer</pre>	

[接下頁](#)

↓ 使用簡體資料集

```

with open("simp_dataSet.txt", 'r', encoding='utf-8-sig') as file:
    word_seq = []
    num_texts = 0
    for data in file:
        topic, content = data.split('@@')    ← 用 @@ 將資料切開
        content = content.strip()           ← 去除前後的換行字元與空白
        topic_contents[topic].append(content) ← 依據話題分類儲存
        word_seq.append(content)
        num_texts += 1
        if num_texts%1000 == 0: ← 累計 1000 個句子後再給 Tokenizer 進行處理
            tokenizer.fit_on_texts(word_seq)
            print('\r已處理', num_texts, '個句子', end='')
            word_seq = []
    if word_seq:
        tokenizer.fit_on_texts(word_seq)
        print('\r已處理', num_texts, '個句子', end='')
        del word_seq
    print('\nToken建立完畢')

```

將詞向量轉為權重矩陣

建立神經網路前，要先將開源詞向量檔案中的數字儲存成一個權重矩陣，這樣之後就能載入神經網路中的詞嵌入層。製作權重矩陣的方法如下：

- ❶ 建立一個 shape 為 (詞彙數, 詞向量維度) 的空權重矩陣。
- ❷ 逐行讀取公開詞向量檔案，並且與 Tokenizer 進行比對。
- ❸ 若讀取到的詞有在詞彙對照表中，並在前 20000 個常用詞中，那麼便將該詞後面的 200 個數字存入權重矩陣中 (4-4-3 節的文字轉詞向量有提到權重矩陣中第幾列就代表是第幾個詞的詞向量，所以只要將該詞轉數字，就知道權重矩陣中第幾列是它的向量，再將 200 個數字存放到那一列即可)。

以下為製作詞嵌入層權重矩陣的程式：

BonusD.1.py(續) 載入公開的預訓練詞向量

```
wv_path = 'cache/datasets/Tencent_AILab_ChineseEmbedding.txt'
with open(wv_path, encoding='utf-8-sig') as file:

    head = file.readline().strip().split()  ← 取出第一行, 即詞
    total_words = int(head[0])               向量檔案的資訊
    embedding_dim = int(head[1])

    print('總共有', total_words, '個詞, 詞向量維度為', embedding_dim)

    # 先製作空的詞嵌入層權重矩陣, 其 shape 為 (詞彙數, 詞向量維度)
    embedding_matrix = np.empty((max_words, embedding_dim))
    match_num = 0  ← 用來記錄吻合的詞彙量
    pg_percent = 0 ← 用來記錄進度比例

    for i, line in enumerate(file):
        values = line.split()
        word = values[0]
        num = tokenizer.word_index.get(word)  ← 將詞轉數字
        if num is not None and num < max_words:  ←
            try:                                判斷詞是否有在詞彙對照表中, 並在前20000個常用詞中
                vector = np.asarray(values[1:], dtype='float32') ←
                embedding_matrix[num] = vector  ←
                match_num += 1                  將向量存入 取出詞的向量
                                                權重矩陣中 (200 個數字)
            except:
                print('\n格式錯誤:', line)
        if i >= (total_words / 100) * pg_percent:
            pg_percent += 1
            print('\r已處理 %i%s 的詞' % (pg_percent, '%'), end='')
    print('\r已處理 100% 的詞')

    print('吻合的詞向量數量:', match_num)
```

↓

吻合的詞向量數量: 19606

執行完畢後可以看到吻合的詞彙量高達 19606 個, 代表此公開詞向量的涵蓋範圍真的很廣。

建立神經網路並載入詞向量

再來建立神經網路, 使用 1D CNN + LSTM 的結構, 並載入剛剛建立好的權重矩陣:

BonusD.1.py(續) 建立神經網路

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense, LSTM,
Dropout, Conv1D, MaxPooling1D

model = Sequential()

model.add(Embedding(max_words, 100,
                    input_length=maxlen,
                    name='word2vec'))
model.add(Dropout(0.25))
model.add(Conv1D(filters=256,
                 kernel_size=5,
                 padding='valid',
                 activation='relu',
                 strides=1))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(256, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(topics), activation='softmax'))

model.layers[0].set_weights([embedding_matrix]) ← 載入剛剛建立的權重矩陣
model.layers[0].trainable = False ← 凍結詞嵌入層

model.summary()
```

最後讓神經網路進行訓練, 可以得到 78% 的驗證準確率, 可見使用大型的預訓練詞向量對於此分類問題真的有幫助。

結果報告

另外我們也延續 6-4 節的實驗，以不同的神經網路結構進行測試，並將結果放入原本的實驗報告中進行比較：

神經網路	Answer (驗證準確率)	Embedding (驗證準確率)	騰訊詞向量 (驗證準確率)
僅 Embedding 層	65 %	74 %	71 %
Dense	68 %	73 %	75 %
1D CNN	70 %	70 %	77 %
LSTM	73 %	72 %	79 %
1D CNN + LSTM	73 %	72 %	78 %

可見騰訊的預訓練詞向量在大多神經網路結構中，都取得比其它兩者更高的成績，這說明了使用大量且高多樣性的語料庫，訓練出來的詞向量能做到更好、更通用的轉換方式，讓分類效果更上一層樓。