資料探勘研究與實務 Data Mining Research & Practice

HW4

0853426 陳紹雲

步驟一：讀入資料並刪除換行符號

In [4]:
```
import numpy as np
import pandas as pd

res = [];count = 0
with open('data/training_label.txt', 'r' , encoding='utf-8') as fn:
    for line in fn:
        line=line.strip('\n')
        if line != "":
            line_list = str(line).split("+++$+++")
            line_list[1] = line_list[1].strip()
            res.append(line_list)
            count += 1
            if(count>=10000):
                break
train = pd.DataFrame(res, columns=["sentiment", "review"])

res = []
with open('data/testing_label.txt', 'r' , encoding='utf-8') as fn:
    for line in fn:
        line=line.strip('\n')
        if line != "":
            line_list = str(line).split("#####")
            line_list[1] = line_list[1].strip()
            res.append(line_list)
test = pd.DataFrame(res, columns=["sentiment", "review"])
```

步驟二：製作 token 函式，並去除 stopword，改用 SnowballStemmer

```
def preprocess(text, stem=True):
    # Remove link,user and special characters
    text = re.sub("@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+", ' ', str(text).lower()).strip()
    tokens = []
    for token in text.split():
        if token not in stop:
            if stem:
                tokens.append(SnowballStemmer(language="english").stem(token))
            else:
                tokens.append(token)
    return " ".join(tokens)
```

```
[nltk_data] Downloading package stopwords to /home/iebi/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
train.review = train.review.apply(lambda x: preprocess(x))
test.review = test.review.apply(lambda x: preprocess(x))
```

```
total = pd.concat([train.review, test.review], axis=0, ignore_index=True)
```

步驟三：製作單詞文本以及給先設定 model 參數

```python
documents = [text.split() for text in total]
```

```python
import gensim
W2V_SIZE = 300
W2V_WINDOW = 7
W2V_EPOCH = 32
W2V_MIN_COUNT = 5
w2v_model = gensim.models.word2vec.Word2Vec(
                                            size=W2V_SIZE, # 一次讀進去的單字量
                                            window=W2V_WINDOW, # 滑動視窗 一次抓幾個字
                                            min_count=W2V_MIN_COUNT, # 出現>min_count 才算進去
                                            workers=8)
```

```python
w2v_model.build_vocab(documents)
```

步驟四：Tokenizer

```python
%%time
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train.review)

vocab_size = len(tokenizer.word_index) + 1
print("Total words", vocab_size)
```

```
Total words 10778
CPU times: user 152 ms, sys: 31.9 ms, total: 184 ms
Wall time: 147 ms
```

步驟五：padding/truncate 到 maxlength （dataframe.review 文本

```python
%%time
SEQUENCE_LENGTH = 300
x_train = pad_sequences(tokenizer.texts_to_sequences(train.review), maxlen=SEQUENCE_LENGTH)
x_test = pad_sequences(tokenizer.texts_to_sequences(test.review), maxlen=SEQUENCE_LENGTH)
```

```
CPU times: user 109 ms, sys: 0 ns, total: 109 ms
Wall time: 109 ms
```

步驟六：向量累加製作 embedding_matrix

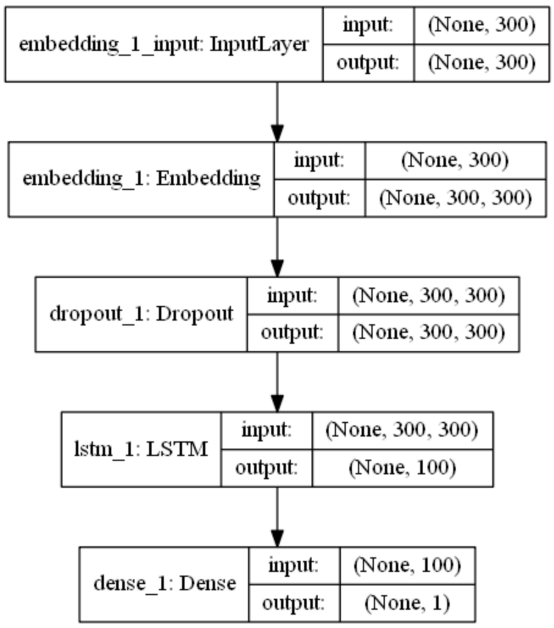
```
embedding_matrix = np.zeros((vocab_size, W2V_SIZE))
for word, i in tokenizer.word_index.items():
    if word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word] # 向量值給embedding_matrix
print(embedding_matrix.shape)
```

```
(10778, 300)
```

步驟七：建模

```
model = Sequential()
model.add(Embedding(vocab_size, W2V_SIZE, weights=[embedding_matrix], input_length=SEQUENCE_LENGTH, trainable=False))
model.add(Dropout(0.5))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

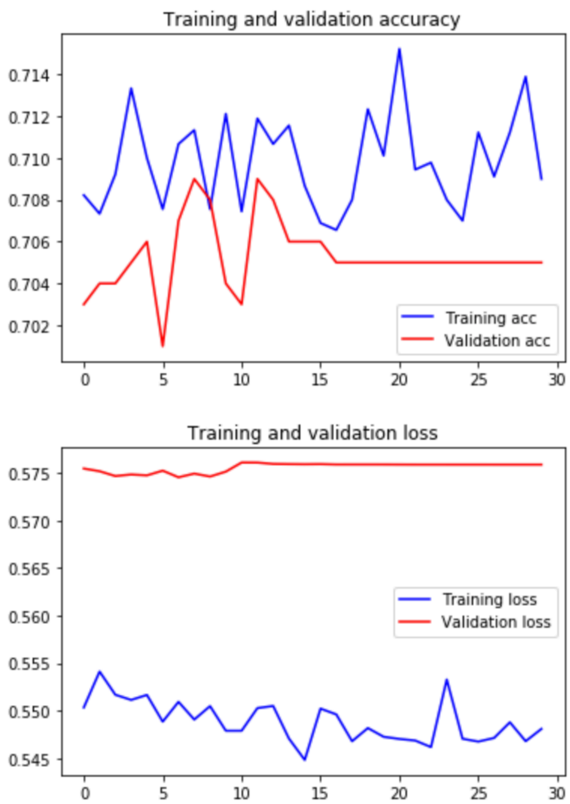


步驟八：Training

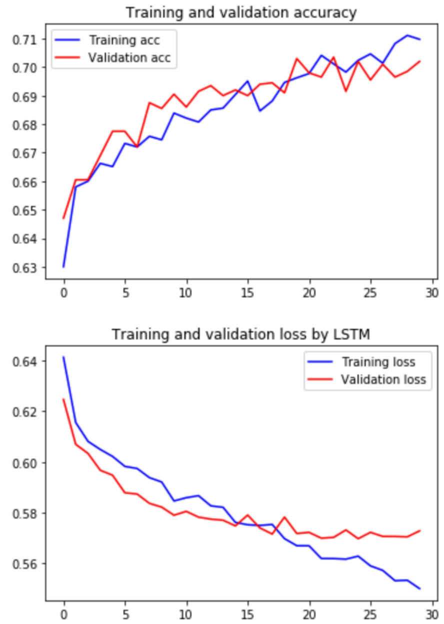
```
model.compile(loss='binary_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])
callbacks = [ ReduceLROnPlateau(monitor='val_loss', patience=5, cooldown=0),
              EarlyStopping(monitor='val_acc', min_delta=1e-4, patience=5)]
```

```
%%time
SEQUENCE_LENGTH = 300
EPOCHS = 8
BATCH_SIZE = 256
history = model.fit(x_train, y_train,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_split=0.1,
                    verbose=1,
                    callbacks=callbacks)
```

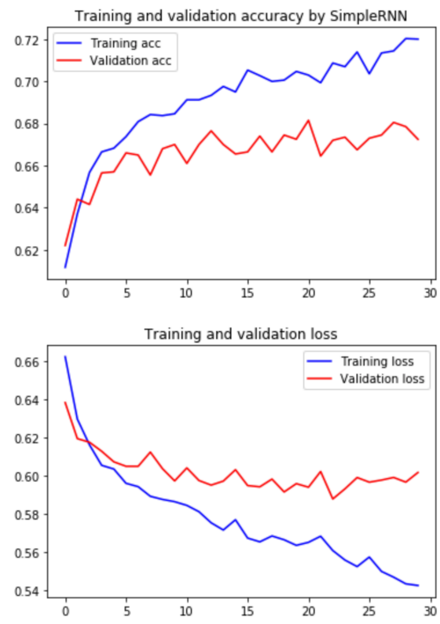
結果：



Overfitting Result



LSTM result



SimpleRNN

討論：

我覺得跑出來的結果有點驚訝，起初我只用 epochs = 8 ，跑出來看不太出來，調成 epochs=30 後才發現大概在週期 10 以後開始收斂，但結果實在是太爛。把 callbacks 參數 ReduceLR 註解掉，validation loss 卻不會動，所以我想是過擬合了，之後改用輸入層 Dropout(0.2)，隱藏層 Dropout(0.5)，才正常收斂，大概在 epochs=15 的時候 val_loss 才沒繼續下降，代表收斂。

RNN vs. LSTM

因為繹安學長說將 LSTM 層直接改成 SimpleRNN 就可以跑了，但起初因為我用原先的 LSTM(Dropout(0.2))跟 recurrent_dropout，發現 plot 不出任何值來，後來我將 dropout 拿掉後，才有值出來，所以我在猜可能是因為 SimpleRNN 的梯度遺失的關係，所以造成以上結果。另外效能方面，SimpleRNN 訓練的速度比 LSTM 快，但是在準確率和 loss 上，比 LSTM 差很多。

```
%%time
score = model.evaluate(x_test, y_test, batch_size=BATCH_SIZE)
print()
print("ACCURACY:",score[1])
print("LOSS:",score[0])
```

```
90/90 [==============================] - 0s 602us/step

ACCURACY: 0.7555555701255798
LOSS: 0.5381515622138977
CPU times: user 113 ms, sys: 21.8 ms, total: 135 ms
Wall time: 56.5 ms
```

```
%%time
score = model_RNN.evaluate(x_test, y_test, batch_size=BATCH_SIZE)
print()
print("ACCURACY:",score[1])
print("LOSS:",score[0])
```

```
90/90 [==============================] - 0s 412us/step

ACCURACY: 0.6111111044883728
LOSS: 0.6173274517059326
CPU times: user 89.7 ms, sys: 0 ns, total: 89.7 ms
Wall time: 38.8 ms
```