HW4 sentiment analysis

CNN/LSTM

310706043 肇綺筠

- **資料前處理**

1. Data Balanced, 讓正負樣本的數量都一樣多，有助於 model performance 的提升

```
[20] #making the lenght of neg and positive review the same
     #讓正面負面評論的數量一樣多有助於data balanced
     new_d = new_d.sample(frac=1)
     pos_df=new_d.loc[new_d.sentiment==1,:][:2384]
     neg_df=new_d.loc[new_d.sentiment==0,:][:2384]
     text_1= pd.concat([pos_df,neg_df])
     text_1 = text_1.sample(frac=1) #打亂review的順序
     pd.value_counts(text_1.sentiment)
```

```
1    2384
0    2384
Name: sentiment, dtype: int64
```

2. 使用 word2vec,把建好的語料庫丟進去訓練 w2v 模型，並將轉換好的詞向量存進 embedding matrix，做為後面 model embedding 層的 pretrained weight

```
[23] corpus = [] #record all the text in reviews

     for i in text_1.cleaned_review:
         corpus.append(i.split()) #each row one review sentence
```

```
[27] from gensim.models import Word2Vec
     w2v_model = Word2Vec(corpus,min_count=10,window=20 ,workers=10)
     w2v_model.save('w2v.model')
     w2v_k = Word2Vec(corpus_testk,min_count=10,window=20 ,workers=10)
```

Embedding matrix 後面 model 會用到

```
[40] embedding_matrix = np.zeros((vocab_size, 100))
     for word, i in tok.word_index.items():
         if word in w2v_model.wv.vocab:
             embedding_matrix[i] = w2v_model.wv.word_vec(word)#訓練過後各個文字的詞向量取出來，丟進embeddi
     print('Null word embeddings: %d' % np.sum(np.sum(embedding_matrix, axis=1) == 0))
```

3. 為了讓輸入資料能通過神經網路，須 tokenized, 把字詞都轉換成 digital list，最後再 padding 成一樣長度（這裡設成和最長 review 一樣）

```
#shuffle完review資料之後重新給text_1的內容編碼(index from 0...n)
#ref:https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reset_index.html

text_1 = text_1.sample(frac=1).reset_index(drop=True)
tok = Tokenizer() #建立token給每個review中的文字
tok.fit_on_texts(text_1.cleaned_review.astype(str))#依照review中某文字出現的次數做排序
vocab_size = len(tok.word_index) + 1
encd_rev = tok.texts_to_sequences(text_1.cleaned_review.astype(str)) #encode reviews,把每個revie
print(vocab_size)
```

```
#把review sequence都padding成一樣長度
from keras_preprocessing.sequence import pad_sequences
pad_rev= pad_sequences(encd_rev, maxlen=max_rev_len2) #encd_rev是每個review的數字list,需把他們都pad
pad_rev.shape #各個review現在已經成為長度為476的數字list, 分別對應text_1之中的sentiment(label)
```

```
(4768, 870)
```

- **模型訓練(CNN, LSTM)**

1. CNN 網路架構
   含 Embedding 層、兩層卷積、Dropout 設為 0.7、通過 relu 提升 model flexibility, 最後過 sigmoid。epoch 為 10

▾ CNN

```
# define model
model_cnn = Sequential()
model_cnn.add(Embedding(vocab_size, 100,weights=[embedding_matrix], input_length=max_rev_len2))
model_cnn.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model_cnn.add(Dropout(0.7))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model_cnn.add(Dropout(0.7))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(10, activation='relu'))
model_cnn.add(Dense(1, activation='sigmoid'))
print(model_cnn.summary())
```

```
Epoch 6/10
120/120 - 1s - loss: 0.4480 - accuracy: 0.7999 -
val_loss: 0.4816 - val_accuracy: 0.7998 - 996ms/epoch - 8ms/step
Epoch 7/10
120/120 - 1s - loss: 0.3539 - accuracy: 0.8492 -
val_loss: 0.4690 - val_accuracy: 0.7987 - 1s/epoch - 9ms/step
Epoch 8/10
120/120 - 1s - loss: 0.3028 - accuracy: 0.8760 -
val_loss: 0.4352 - val_accuracy: 0.8103 - 958ms/epoch - 8ms/step
Epoch 9/10
120/120 - 1s - loss: 0.2656 - accuracy: 0.8925 -
val_loss: 0.4638 - val_accuracy: 0.7883 - 824ms/epoch - 7ms/step
Epoch 10/10
120/120 - 1s - loss: 0.2252 - accuracy: 0.9124 -
val_loss: 0.4886 - val_accuracy: 0.7715 - 761ms/epoch - 6ms/step
```

▲上述可見 model 有過擬合情況，epoch 8 的時候效果最佳。也許可考慮減少 conv 層數、改變 optimizer 調整學習綠的方法，或是加入更多訓練資料防止 model fit 在小資料結構上

2. LSTM 網路架構

   含 Embedding 層、兩層 LSTM、Dropout 設為 0.5、最後過 sigmoid。epoch 為 15

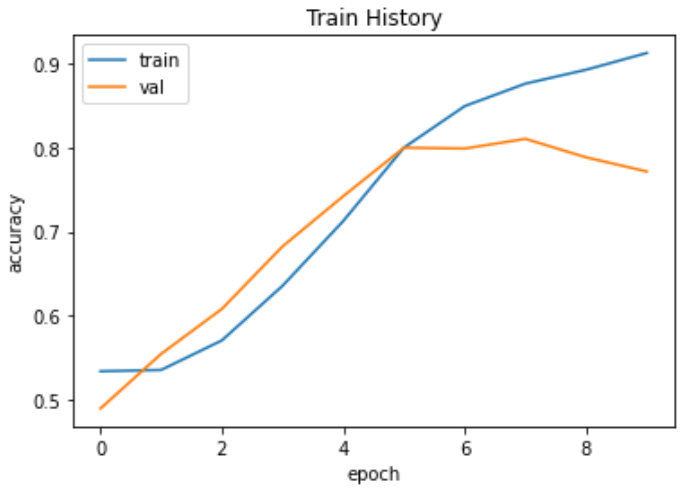
```
model_lstm = Sequential()
model_lstm.add(Embedding(vocab_size,100,weights=[embedding_matrix], input_length=max_rev_len2,t
model_lstm.add(LSTM(100,return_sequences=True))
model_lstm.add(Dropout(0.5))
model_lstm.add(LSTM(100))
model_lstm.add(Dropout(0.5))
model_lstm.add(Dense(1, activation='sigmoid'))

model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model_lstm.summary())
```

```
Epoch 12/15
120/120 - 9s - loss: 0.5742 - accuracy: 0.6972 -
val_loss: 0.5885 - val_accuracy: 0.7055 - 9s/epoch - 79ms/step
Epoch 13/15
120/120 - 8s - loss: 0.5563 - accuracy: 0.7139 -
val_loss: 0.5700 - val_accuracy: 0.7002 - 8s/epoch - 69ms/step
Epoch 14/15
120/120 - 8s - loss: 0.5569 - accuracy: 0.7139 -
val_loss: 0.5687 - val_accuracy: 0.7107 - 8s/epoch - 68ms/step
Epoch 15/15
120/120 - 8s - loss: 0.5441 - accuracy: 0.7213 -
val_loss: 0.5943 - val_accuracy: 0.6813 - 8s/epoch - 68ms/step
```

▲上述可見 epoch14 的時候效果最佳，過擬合情況沒有 CNN 來得嚴重。但 Accuracy 表現比 CNN 差，需要的 epoch 數要更高一點才會有更好的精準度

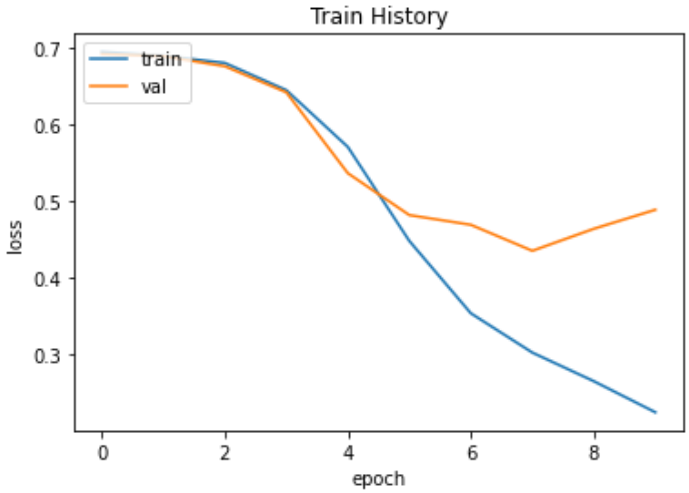
- **Acc, Loss 比較**

1. CNN Acc&Loss 變化



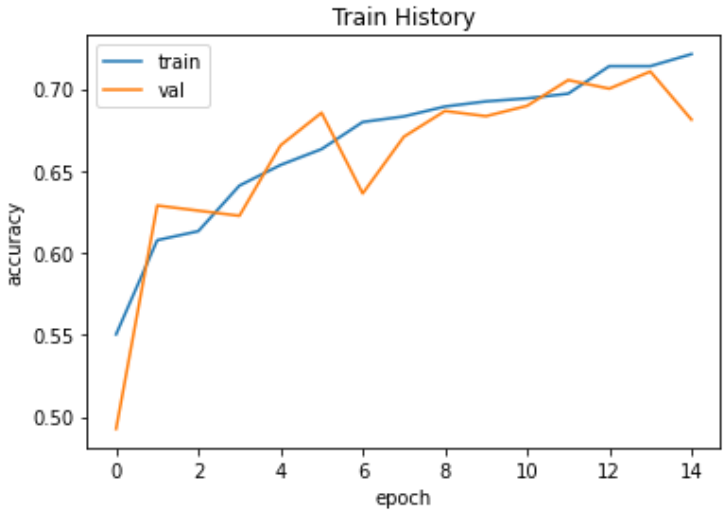

```
Test Accuracy CNN:0.7714884877204895
```

▲ Accuracy of CNN



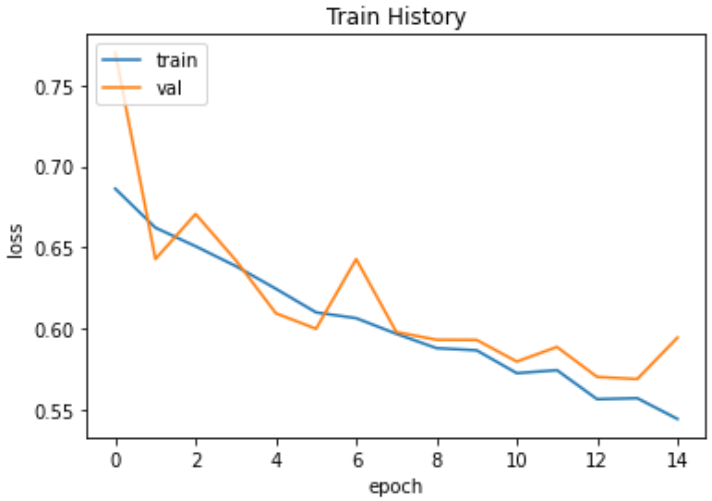
▲ Loss of CNN

2. LSTM Acc & Loss 變化




```
Test Accuracy LSTM:0.6813417077064514
```

▲ Accuracy of LSTM



▲ Loss of LSTM