

# ML 2019 SPRING HW6

## B06902074 資工二 柯宏穎

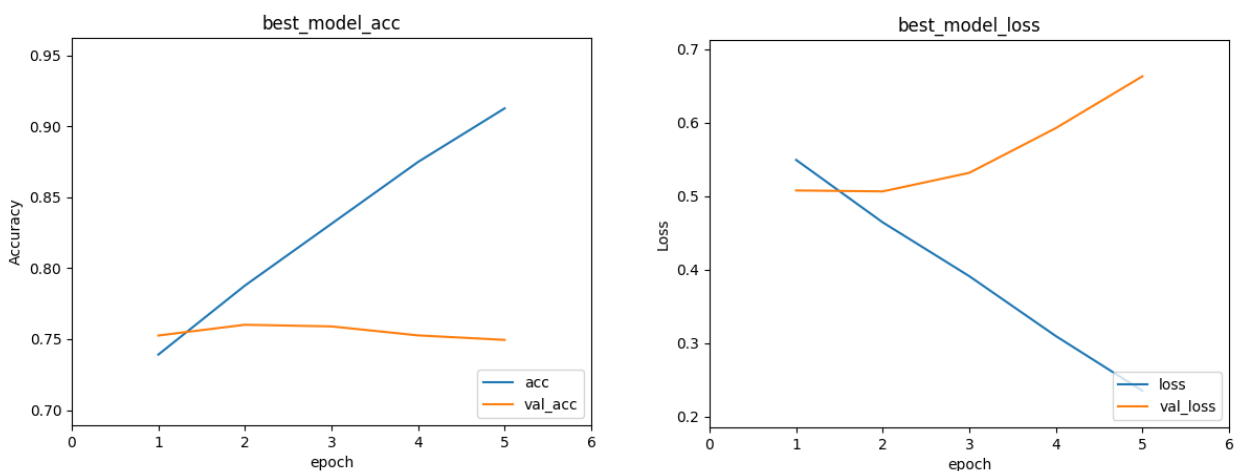
(Reference: <https://www.kaggle.com/jerrykuo7727/embedding-rnn-0-876>)

1. 請說明你實作之 **RNN** 模型架構及使用的 **word embedding** 方法，回報模型的正確率並繪出訓練曲線。

我的模型架構很簡單，我利用 *Keras* 的 *Embedding Layer*，*GRU* (*CuDNNGRU*) 與 *Dense Net* 組成。

```
embedding_layer = Embedding(input_dim=embedding_matrix.shape[0],
                             output_dim=embedding_matrix.shape[1], weights=[embedding_matrix], trainable=True)
model = Sequential()
model.add(embedding_layer)
model.add(Bidirectional(CuDNNGRU(128, return_sequences = True)))
model.add(Bidirectional(CuDNNGRU(128, return_sequences = False)))
model.add(Dense(2, kernel_regularizer = regularizers.l1_l2(l1 = 3e-3, l2 = 5e-3),
                 activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

*Word Embedding* 的部分，我先將 *gensim* 所建成的 *word vector* 做成一個可相對應的表，這個表的第0項代表原本字典裡沒出現過的字詞，方便我後續做 *training*。之後將 *training data* 轉成我所製作的表的相對應 *index*。再使用 *keras* 的 *pad\_sequence()* 將每段語句切成適當的長度，方能丟進 *model*。



由圖中可看到，通常超過3個 *epoch* 後，就 *overfitting* 了，再繼續做下去只會浪費時間且得到更糟糕的結果。此做法最終做出來的結果約在 0.7582/0.7553 (public score/private score) 左右。

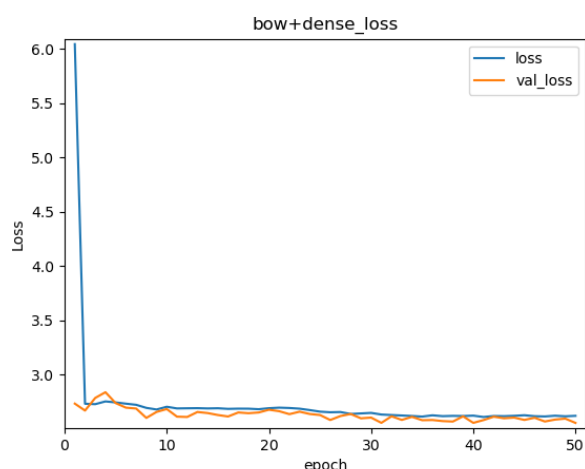
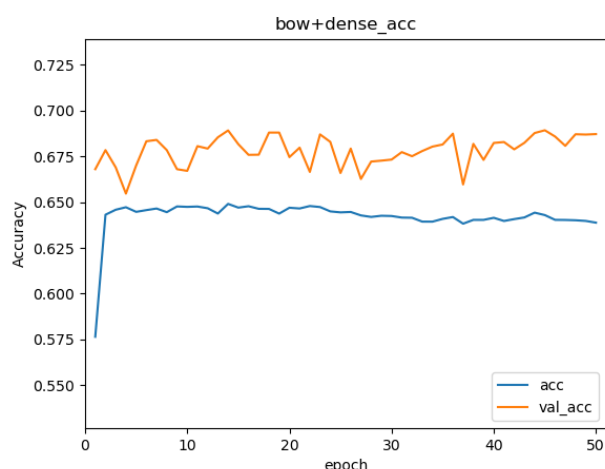
2. 請實作 **BOW+DNN** 模型，敘述你的模型架構，回報模型的正確率並繪出訓練曲線。

*BOW* 即是將所有字詞轉成 *one hot* 來做訓練，但其需消耗大量的記憶體來儲存，電腦有點不堪負荷。

```

model = Sequential()
model.add(Dense(256, input_dim = X_bow.shape[0], kernel_regularizer =
regularizers.l1_l2(l1 = 3e-3, l2 = 5e-3), activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.75))
model.add(Dense(64, kernel_regularizer = regularizers.l1_l2(l1 = 3e-3, l2 = 5e-3),
activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.75))
model.add(Dense(16, kernel_regularizer = regularizers.l1_l2(l1 = 3e-3, l2 = 5e-3),
activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.75))
model.add(Dense(2, kernel_regularizer = regularizers.l1_l2(l1 = 3e-3, l2 = 5e-3),
activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

```



基本上到後面都是浮動數值了，最終結果約0.6851/0.6769(public score/private score).

### 3. 請敘述你如何 **improve performance** (preprocess, embedding, 架構等)，並解釋為何這些做法可以使模型進步。

*Preprocess*的部分，我在做切詞時，即會將空白與BX(Dcard tag其他層留言)進行切除，明顯地，這些字串對判斷並無實質意義，我們盡可能地減少noise，降低訓練的難度。

用*word2vec*時，我也適當地增加*window*的數量(7-10個)，讓他能跟更多的語句做連結，*public score*有些為地上升(0.7561 → 0.758)，但若設定太長，罵人的話通常簡短有力，反而會使得信心度下降，並沒有太好的結果。

### 4. 請比較不做斷詞 (e.g., 以字為單位) 與有做斷詞，兩種方法實作出來的效果差異，並解釋為何有此差別。

以字做斷詞，所做出來每個字之間的關聯性會降低。我使用*word2vec*的*most\_similar*來看相似度，明顯地發現許多奇怪的字，也因為每個詞都只有一個字，我們辦法拿正常的詞下去測試。不過我拿笨這個字，會出現解，糕等奇怪的字詞，可能是因為笨通常與笨蛋的形式出現，其相關性就與蛋綁在一起了。

不過做後做出來的*public score*仍有0.7485，其實並沒有糟糕太多，可能因為我在做*word2vec*時，仍然會去考慮前後字詞，此狀況下，與我們自己斷詞不會相差太多，才會造成此現象。

5. 請比較 **RNN** 與 **BOW** 兩種不同 **model** 對於 "在說別人白痴之前，先想想自己"與"在說別人之前先想想自己，白痴" 這兩句話的分數（**model output**），並討論造成差異的原因。

為求方便，假設"在說別人白痴之前，先想想自己"為第一句，"在說別人之前先想想自己，白痴"為第二句。

*BOW + RNN*:

	<b>0</b>	<b>1</b>
第一句	0.39614993	0.60385007
第二句	0.39614993	0.60385007

*RNN*:

	<b>0</b>	<b>1</b>
第一句	0.55454963	0.44545034
第二句	0.42188346	0.57811650

我認為第一句並不屬於惡意留言，不過許多 *RNN* model 所做出來的判斷為惡意留言，*BOW*所做的判斷其實也非常不確定，只有極些為的差距。

其實從斷詞來看，這兩句話切出完全一樣的詞，只是順序不同。所以使用 *BOW*時，每一句話都是每個斷詞的 *vector* 的和，只在意其內容，與順序無關，故只要斷詞相同，預測出來的結果就完全相同。