ML 2019 SPRING HW6

B06902074 資工二 柯宏穎

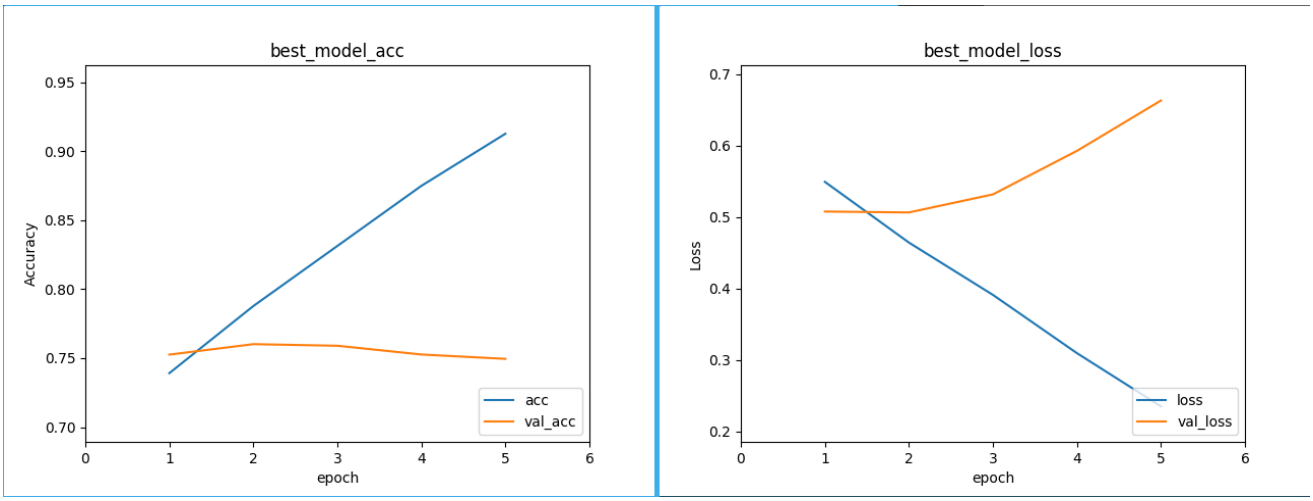
(Reference: https://www.kaggle.com/jerrykuo7727/embedding-rnn-0-876)

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$kearas$的$pad_sequence()$將每段語句切成適當的長度，方能丟進$model$。

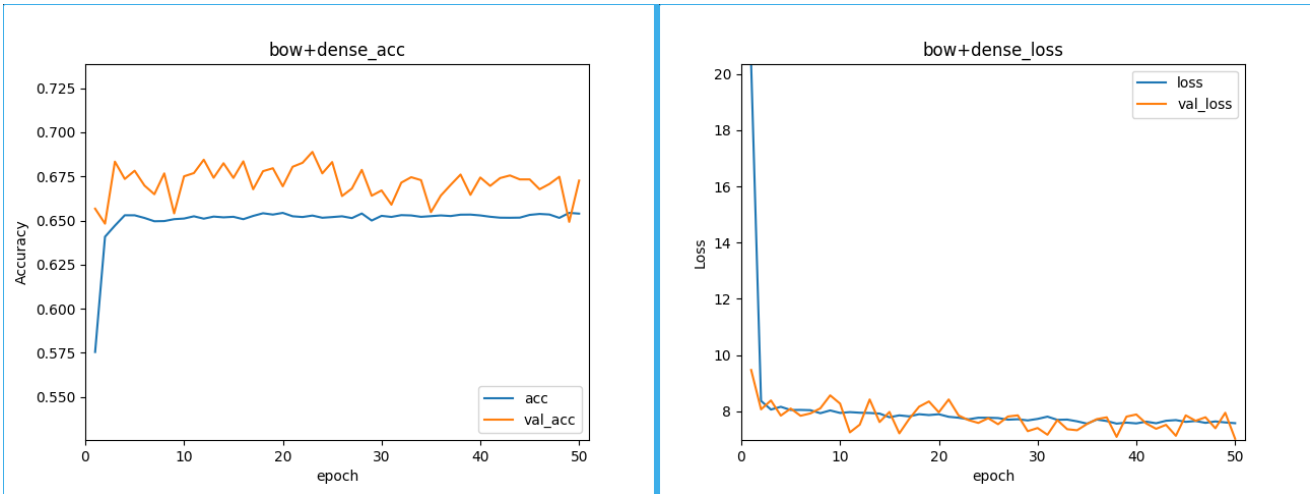


由圖中可看到，通常超過3個$epoch$後，就$over fitting$了，再繼續做下去只會浪費時間且得到更糟糕的結果。此做法最終做出來的結果約在0.758(public score)左右。

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。於是我同樣地使用$embedding\ layer$，其效果差不多，我都先將字詞轉成特定的$index$，與原本的做法差不多，不過我將$trainable$設成$False$，讓那層完全不做$training$的動作。後面直接接$Dense\ Layer$，最後的$public\ score$約0.6850。

```
embedding_layer = Embedding(input_dim=embedding_matrix.shape[0],
output_dim=embedding_matrix.shape[1], weights=[embedding_matrix], trainable=False,
input_length = PADDING_LENGTH)
model = Sequential()
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(256, 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'])
```



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 $\rightarrow$ 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，會出現孵，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。

RNN:

	0	1
第一句	0.55454963	044545034
第二句	0.42188346	0.5781165

$BOW + Dense$:

	0	1
第一句	0.50054080	0.4994592
第二句	0.42022213	0.5797779

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。不過第二句話的白痴是在逗號後面，被獨立出來，就非常像在罵人，故其被判成惡意留言的信心度就較大，只是也沒有到非常懸殊。

語言，前後文必定會有夠影響，這些偏中性的詞，我們自己也可能搞錯，電腦能做到這樣的預測已經非常厲害了。