ML HW4

B05902001 資工三 廖彥綸

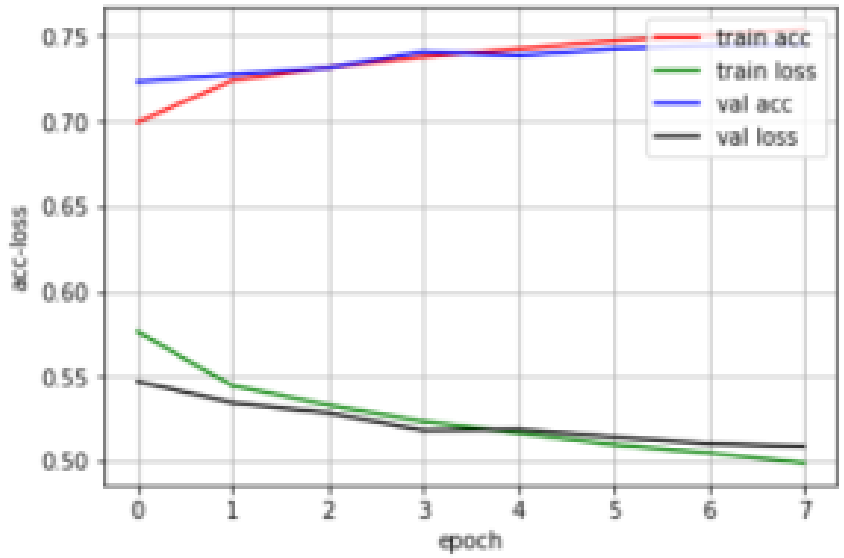
1. 請說明你實作之 RNN 模型架構及使用的 word embedding 方法,回報模型的正確率並繪出訓練曲線。請實作 BOW+DNN 模型,敘述你的模型架構,回報正確率並繪出訓練曲線。

RNN 模型：

Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	(None, 120, 250)	9829250
lstm_8 (LSTM)	(None, 256)	519168
dense_22 (Dense)	(None, 256)	65792
dropout_15 (Dropout)	(None, 256)	0
dense_23 (Dense)	(None, 256)	65792
dropout_16 (Dropout)	(None, 256)	0
dense_24 (Dense)	(None, 1)	257

Total params: 10,480,259
Trainable params: 651,009
Non-trainable params: 9,829,250

訓練曲線：



在 val data(取 train_x 120000 筆資料中的前 10000 筆) embedding 時加入 test_data 的句子，有 0.7540 的正確率。Kaggle 上的 public scores 為 0.75165 疊代

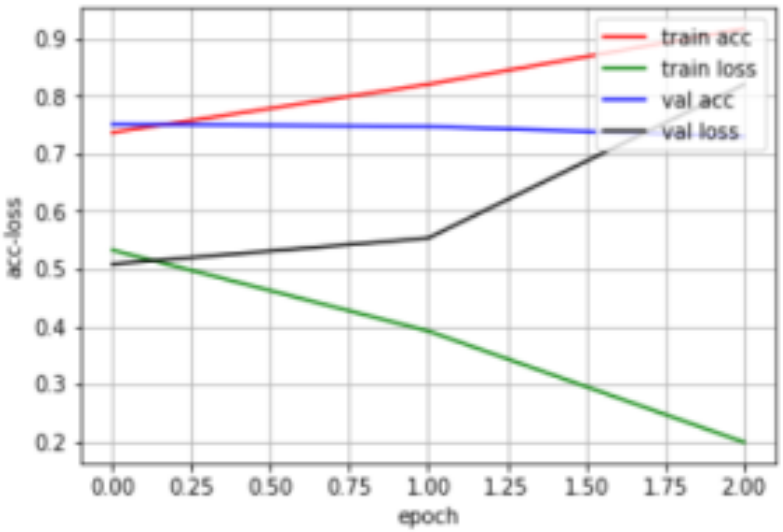
時的正確率緩慢上升，但在較少次數即到達收斂。

其他參數包括：使用 jieba 分詞，Word2Vec 使用 iter = 5, min_window = 5，embedding size 選擇 250 維，並沒有做 OOV 的處理，而是直接捨棄 OOV 的字。Model learning rate = 0.001, weight decay = 1e-6. 加上 early stop 和 callback。

BOW+ DNN 模型：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 256)               5120256
_________________________________________________________________
dense_6 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 257
=================================================================
Total params: 5,252,097
Trainable params: 5,252,097
Non-trainable params: 0
```

訓練曲線：



在 val data(取 train_x = 120000 筆資料中的前 10000 筆) 有 0.7451 的正確率。Kaggle 上的 public scores 為 0.7506，只花了 1 個 iter 就到達收斂(代表 BOW 較容易產生 over training 的問題)

其他參數包括：使用 jieba 分詞，使用 keras 的 Tokenizer word to matrix 功能，只取前 22000 個高頻單字。Model learning rate = 0.001, weight decay = 1e-6. 加上 early stop 和 callback。

2. 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。

RNN 模型的改良上在 word embedding 花費較多時間，word embedding 的好壞決定了模型的上界，而之後 build model 的工作只是盡可能使其接近上界。使用 Word2Vec 時，嘗試多種疊代次數、embedding 的維度、加入 min_count 過濾使用頻率極低的字(設 min_count = 5，如果一個詞在 240000 句中出現次數少於 5 次，則視它為不重要)，幫助模型更簡潔一些。

BOW 模型則調控保留的字數量，嘗試的結果為保留 20000 字上下效果不錯。

3. 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。

模型皆使用第一大題所描述的 RNN 模型，唯一的差別是是否用 jieba 進行斷詞。使用 jieba 斷詞後結果：kaggle 的 public scores: 0.75165。未斷詞的結果：public scores: 0.74850。雖然未做斷詞已經有很好的表現，但進行斷詞後會有些許的進步。中文表達意思以詞為單位比用字為單位更能表達一句話的涵義，例如：很多惡意留言都有”母豬”這個詞，但如果未做分詞只用”母”和”豬”，做比對會使分類更混淆。因為”母”後面可能是接”親”；”豬”後面可能是接”肉”，兩者至少再單詞上都沒有貶意。

增加的分數不如預期的原因可能是：1. Jieba 分詞使用繁體中文的準確率較低。2. 網路用語很多 OOV、錯字、諧音等…導致分詞上的困難。

4. 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。

使用第一大題所描述的 RNN 模型。前話得到的分數是 0.37941903，後者則得到 0.56197035，所以判斷後者為惡意言論，前者則否。雖然兩句話的字組成完全相同，但 RNN 還考慮了語序的問題，讓兩者出現判斷上的差異。

使用第一大題描述的 BOW 模型。兩句話的得分皆為 0.6911724，即皆判斷為惡意言論。BOW 實作時並不考慮字的順序，將兩句話會用相同向量表示，而得到相同的結果。推測是”白癡”被判斷為惡意程度高的詞而被判斷為負面言論。

5.

T = 1:

$u_1^n = 1(n = 0, 1, 2...8, 9)$

f_1: decision stump between 7 and 8.

$\epsilon_1 = 0.3$

$\alpha_1 = 0.42$

T = 2 :

0	1	2	3	4	5	6	7	8	9
0.65	1.53	0.65	0.65	0.65	1.53	1.53	0.65	0.65	0.65

u_2^n is the line below.

f_2: decision stump between 4 and 5.

$\epsilon_2 = (0.65 + 1.53)/(1.53 * 3 + 0.65 * 7) = 0.239$

$d_2 = 1.784$

$\alpha_2 = 0.58$

T=3:

0	1	2	3	4	5	6	7	8	9
0.36	2.73	0.36	0.36	0.36	0.86	0.86	1.16	0.36	0.36

u_3^n is the line below.

f_3: decision stump between 0 and 1.

$\epsilon_2 = (0.36 * 3 + 1.16)/(2.73 + 0.36 * 5 + 0.86 * 2 + 1.16) = 0.3$

$d_3 = 1.53$

$\alpha_3 = 0.42$

final classifier = sign($0.42f_1(x) + 0.58f_2(x) + 0.42f_3(x)$**)**

6.

t = 1

z = 3 + 0, z_i = 100 -10 = 90, z_f = -100+110 = 10, z_o = -10

pass input, store in forget layer, but not pass output layer.

forget_layer = 3, output = 0.

t = 2

z = -2 + 0, z_i = 100 -10 = 90, z_f = -100+110 = 10, z_o = 100-10 = 90

pass input, store in forget layer, pass output layer.

forget_layer = 1, output = 1.

t = 3

z = 4 + 0, z_i = 200 -10 = 190, z_f = -200+110 = -90, z_o = 100-10 = 90

pass input, clear forget layer, pass output layer.

forget_layer = 0, output = 4.

t = 4

z = 0 + 0, z_i = 100 -10 = 90, z_f = -100+110 = 10, z_o = 100-10 = 90

reset, output = 0
forget_layer = 0, output = 0.

t = 5
z = 2 + 0, z_i = 100 -10 = 90, z_f = -100+110 = 10, z_o = 0-10 = -10
pass input, store in forget layer, bit not pass output layer.
forget_layer = 2, output = 0.

t = 6
z = -4 + 0, z_i = 0 -10 = -10, z_f = 110 = 110, z_o = 100-10 = 90
not pass input, store in forget layer, pass output layer.
forget_layer = 2, output = 2.

t = 7
z = 1 + 0, z_i = 200 -10 = 190, z_f = -200+110 = -90, z_o = 100-10 = 90
pass input, clear forget layer, pass output layer.
forget_layer = 0, output = 1.

t = 8
z = 2 + 0, z_i = 100 -10 = 90, z_f = -100+110 = 10, z_o = 100-10 = 90
pass input, store in forget layer, pass output layer.
forget_layer = 2, output = 2.

t	1	2	3	4	5	6	7	8
output(y_t)	0	1	4	0	0	2	1	2