一、請清楚標示你選的題目 (1)

ML-2017Fall-Final-TV-conversation

二、 Team name, members and your work division (1)

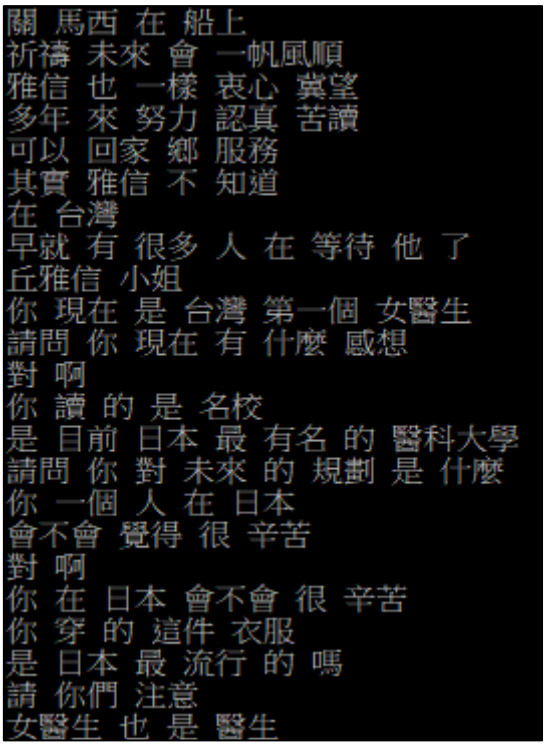
Team name: NTU_r06922002_Overfitting

Members and Work division:

學號	姓名	Work Division
R06922002	姚嘉昇	Build models and dictionary
R06942054	潘仁傑	Fine tune models

三、 Preprocessing/Feature Engineering (3)

1. 建立新的詞典
   A. 下載結巴(jieba)斷詞台灣繁體版本中的詞典 (dict.txt)。
      Reference: https://github.com/ldkrsi/jieba-zhTW/blob/master/jieba/dict.txt
   B. 下載結巴(jieba)斷詞簡體版本所提供的繁體詞典 (dict.txt.big)。
      Reference: https://github.com/fxsjy/jieba/blob/master/extra_dict/dict.txt.big
   C. 將上述詞典利用簡轉繁的工具，將所有簡體轉為繁體 (dict.txt+dict.txt.big.TW)
      最後將兩個詞典合而為一，就是我們最後給 jieba 用來切詞的詞典 (切詞方式參考助教的手把手)。
2. 依序讀入 1_train.txt ~ 5_train.txt。
3. 利用 jieba 對每一行分別切詞後，再一行、一行寫入 trainSeg.txt (如下圖)。



圖、切詞後的句子 (trainSeg.txt)

四、 Model Description (At least two different models) (7)

1、 jieba + gensim w2v + Cosine similarity

A. jieba

結巴的字典是使用台灣繁體版以及簡體版本翻為繁體後的新詞典(dict.txt+dict.txt.big.TW)，並使用 jieba.cut(line, cut_all=False)對每一行進行分詞產生並一行、一行寫入 trainSeg.txt，再丟給之後的 w2v 進行 training。

B. gensim word2vec

使用 word2vec.Text8Corpus 直接將 trainSeg.txt 轉成 sentences 的 list，再使用 word2vec.Word2Vec 進行詞向量的訓練，其中我們參數設定如下表所示。

size	window	sample	negative	hs	sg	Iter	workers	min_count
64	24	1e-4	10	0	0	300	8	3

C. Cosine similarity

i. 讀入 w2v 之 model、test.csv 的對話及選項。

ii. 刪除換行符號。

iii. 用結巴對 dialogue 進行分詞。

iv. 將對話中的所有詞向量加總後取平均，代表整個對話的向量，若是沒有字典中沒有的字詞，則 random 產生一個向量作為代表，並永遠代表這個詞。

v. 對 options 進行分詞，每一個選項也都將所有的詞向量取平均，若是沒有字典中沒有的字詞，則 random 產生一個向量作為代表，並永遠代表這個詞。

vi. 個別計算 dialogue 與每一個 options 向量的 cosine similarity，選出最高相似度的選項作為答案。

2、 Jieba + gensim word2vec+ Deep + Cosine similarity

A. jieba

結巴的字典是使用官方詞典，並使用 jieba.cut(line, cut_all=False)對每一行進行分詞產生並一行一行寫入 trainSeg.txt。

B. gensim word2vec

使用 word2vec.Text8Corpus 直接將 trainSeg.txt 轉成 sentences 的 list，再使用 word2vec.Word2Vec 進行詞向量的訓練，其中我們參數設定如下。

size	window	sample	negative	hs	sg	iter	workers	min_count
200	5	1e-4	10	0	0	15	8	5

C. 切 window

將 trainSeg.txt 切成每四句一個 window(三句對話及一句答案)，並將前三句使用 jieba 切詞後轉成詞向量，對三句話所有詞取平均，作為 deep 的 input。

D. Deep FC 架構如下圖

```
model = Sequential()
model.add(Dense(units = 4096,input_dim=200, kernel_initializer = 'glorot_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 2048, kernel_initializer = 'glorot_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1024, kernel_initializer = 'glorot_normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(200, activation = 'tanh'))

# Compile model
model.compile(loss='mse', optimizer='adam')
```

E.Cosine similarity

將 deep 預測的 vector 與選項的詞向量平均做 cosine similarity，找出與預測最相似的選項。

3、 Jieba + seq2seq

A. jieba

結巴的字典是使用官方詞典，並使用 jieba.cut(line, cut_all=False)對每一行進行分詞產生 input_texts.obj 、 target_texts.obj 、 input_characters.obj 、 input_token_index.obj 、 target_token_index.obj。

B. encoder (LATENT_DIM=256)

```
# Define an input sequence and process it.
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(LATENT_DIM, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]
```

C. Decoder (LATENT_DIM=256)

```
# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm = LSTM(LATENT_DIM, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_state
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

D. optimizer='rmsprop'

E. loss='categorical_crossentropy'

F. testing

利用 dialogue 的所有句子當作 seq2seq 的 input，而 seq2seq 的 output 則是預測出來的下一句話，利用這句話與 6 個 options 個別計算 cosine similarity，進而選出最高 similarity 的作為答案。

五、 Experiments and Discussion (8)

1、 jieba + gensim w2v + cosine similarity

我們試著調整 gensim w2v 的參數，嘗試各種參數配合並記錄其 Kaggle 的分數。紀錄如下表所示。目前最好的 gensim word2vec 參數如下， 而我們就下表調整其他參數來選擇最好的 word2vec，最後挑分數前十高的 model ensemble 起來，在 public 得到 0.55810 的分數。

size=64, window=24, sample=1e-4, negative=10, hs=0, sg=0, iter=300, workers=8, min_count=3

表、Gensim word2vec 參數微調

size	window	min count	iter	Public score
64	24	3	300	0.53950
128	24	3	300	0.53715
96	16	3	300	0.53715
320	16	3	300	0.52450
64	16	3	300	0.53675
128	16	3	300	0.53873
256	16	3	300	0.53636
384	24	3	300	0.52292
384	16	3	300	0.52371
384	16	3	50	0.50632
384	16	3	15	0.49920
128	16	3	15	0.50711
256	12	3	15	0.49683
256	20	3	15	0.49960
256	24	3	15	0.50948
256	16	3	15	0.51343
256	10	3	15	0.49604
256	5	3	15	0.46719
256	8	1	15	0.48853
256	8	3	15	0.48853
128	8	5	15	0.48221
512	8	5	15	0.45849
384	8	5	15	0.48063
256	8	5	15	0.48379

2、 Jieba + gensim word2vec+ Deep + Cosine similarity

目前只嘗試兩種 Deep 的架構，每層之間都有加 BatchNormalize、RELU 及 Dropout 0.5，最後一層使用 tanh，compile 使用 rmse 及 adam。發現分數如此慘淡後，就放棄該 model 的架構，改向發展 seq2seq。

A. 4096-2048-1024：public score 0.25

B. 4096-4096-4096：public score 0.16

3、 Jieba + seq2seq

因為我們使用 keras 來 train seq2seq，每 train 一個 epochs 都要非常久，80 個 epochs 就要 8 個小時以上，雖然這個方法可以達到 0.42，但是我們沒有那麼多時間來 fine tune 這個 model，所以後來就放棄這方法了。

看完了其他組別的作法後，或許可以改用 pytorch 來 train seq2seq 來增加 training 速度，就能夠使用 seq2seq 來實作這個題目了！

4、 N-fold

我們切了 5、10、15 個的 n-fold，丟上去 Kaggle 皆為 0.45~0.46

5、 Ensemble

我們將 0.51 以上的 8 個 model 做 ensemble，得到 public score 0.55271，為 1/21 晚上 11 點的第三名。1/22 當天又將 10 個 public score 有 0.52 以上的 model ensemble 起來後，達到第二名的 0.55810。

6、 可以改善或嘗試的部分

A. w2v 的參數中 sg 設為 1

我們的 model 都是將此 sg 設為 0，看了前 10%的發表我們發現，也許可以嘗試將 sg 設為 1 再 fine tune，前十名的組別也有些許組將此參數設為 1。

B. 將 jieba 的字典改為中研院的字典

C. 使用 pytorch 建立 seq2seq 的 model

因 top 10 %組別有提到使用 pytorch 建立 seq2seq 的模型會跑比較快，才能夠有充分的時間 fine tune，該組就是使用 seq2seq 達到前十名的成績。

D. OOV 的處理方式

我們是隨機賦予新詞一個向量，也許可以改為忽略，或是 0 向量，也可以透過 gensim 找出與其相似的作為代替，都是值得嘗試的方向。