Q1: Data processing.
a. How do you tokenize the data.
1.Intent classification:

- Data:
  使用 Keras 函式庫中的 Tokenizer,將 sequence data 分成 array of string(word), 並去除標點符號,全部轉成小寫,mapping 成數字,最後再將全部 sequence data 以 0 padding 成一樣長度。
- Label:
  使用 Keras 函式庫中的 Tokenizer,所有 Label 分成 array of string(word), 並去除標點符號,全部轉成小寫,mapping 成數字。

2.Slot tagging:

- Data:
  使用 Keras 函式庫中的 Tokenizer,將 sequence data 分成 array of string(word), 並去除標點符號,全部轉成小寫,mapping 成數字,之後在將全部 sequence data 以 0 padding 成一樣長度。最後再 reshape 成(number of data, max length, 1)。
- Label:
  使用 Keras 函式庫中的 Tokenizer,將 Label 分成 array of string(word), 並去除標點符號,全部轉成小寫,並設定 oov_token=<OOV>代表不在選取範圍內的 word,mapping 成數字,之後再將全部 Label 以 0 padding 成一樣長度。最後再以 np_utils.to_categorical 轉成 one-hot 格式。

b. The pre-trained embedding you used.
glove.840B.300d.txt

Q2: Describe your intent classification model.
a. Model
1.RNN
先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後將過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300
。接著經過 RNN 學習每個 time step 的資訊,輸出最後一個 time step 的 hidden state, 其 hidden size 為 256 。最後經過全連接層搭配 softmax ,產生 150 個類別的機率分佈。

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 28, 300)           3000000

 batch_normalization (BatchN  (None, 28, 300)          1200
 ormalization)

 simple_rnn (SimpleRNN)      (None, 256)               142592

 dense (Dense)               (None, 150)               38550

=================================================================
Total params: 3,182,342
Trainable params: 3,181,742
Non-trainable params: 600
_________________________________________________________________
```

公式:

$$h_t = W * h_{t-1} + Ux_t + b$$

$$y_t = g(V * h_t)$$

W,U,V:weight matrix
h_t:hidden state at time step t
g:activation function
b:bias
y_t:output at time step t

2.GRU
先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後將過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過 GRU 學習每個 time step 的資訊,輸出最後一個 time step 的 hidden state, 其 hidden size 為 256 。最後經過全連接層搭配 softmax ,產生 150 個類別的機率分佈。

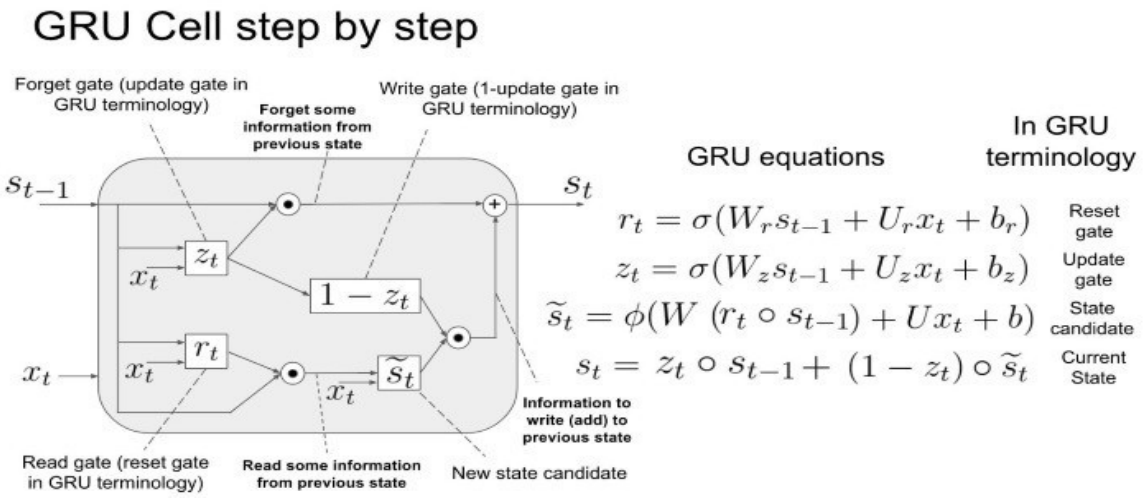
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 28, 300)           3000000

 batch_normalization_1 (Batc  (None, 28, 300)          1200
 hNormalization)

 gru (GRU)                   (None, 256)               428544

 dense_1 (Dense)             (None, 150)               38550

=================================================================
Total params: 3,468,294
Trainable params: 3,467,694
Non-trainable params: 600
_________________________________________________________________
```



GRU 由更新閥以及遺忘閥組成,可以藉此調控要保留以及捨棄的資訊,為 LSTM 的簡化版。

3.LSTM

先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後將過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過 LSTM 學習每個 time step 的資訊,輸出最後一個 time step 的 hidden state, 其 hidden size 為 256 。最後經過全連接層搭配 softmax ,產生 150 個類別的機率分佈。

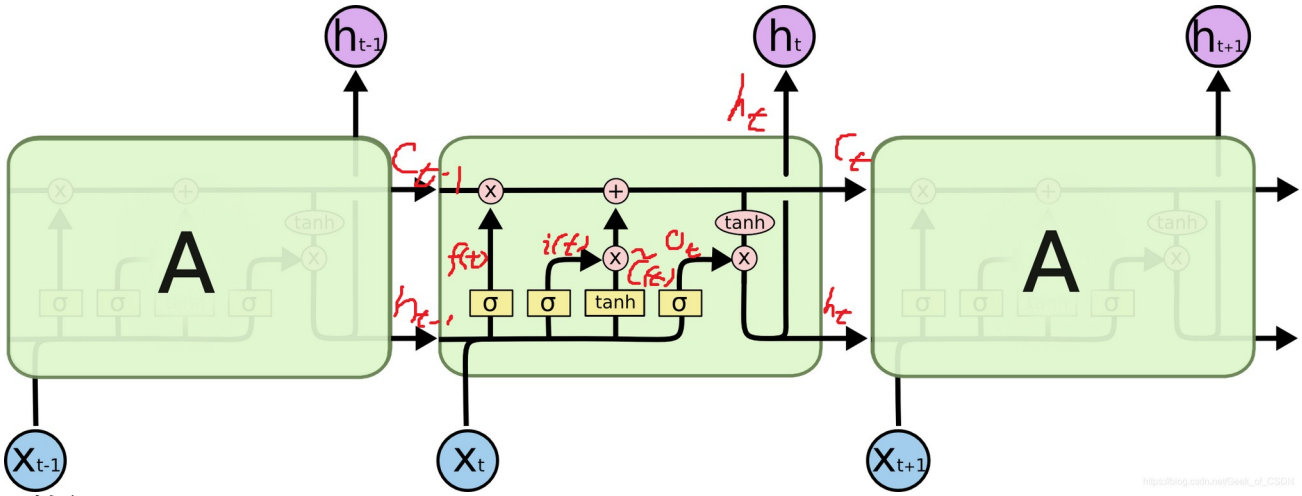
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 28, 300)           3000000

 batch_normalization_2 (Batc  (None, 28, 300)          1200
 hNormalization)

 lstm (LSTM)                 (None, 256)               570368

 dense_2 (Dense)             (None, 150)               38550

=================================================================
Total params: 3,610,118
Trainable params: 3,609,518
Non-trainable params: 600
```



輸入閘:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

遺忘閘:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$

輸出閘

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

長記憶:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

短記憶:

$$h_t = o_t * \tanh(C_t)$$

4.BILSTM
先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後經過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過 BILSTM 層,其中由兩個由不同順序出入資料的 LSTM(hidden size=256)分別學習每個 time step 的資訊,之後將各自的結果串接在一起(hidden size=512),輸出最後一個 time step 的 hidden state, 其 hidden size 為 512 。最後經過全連接層搭配 softmax ,產生 150 個類別的機率分佈。

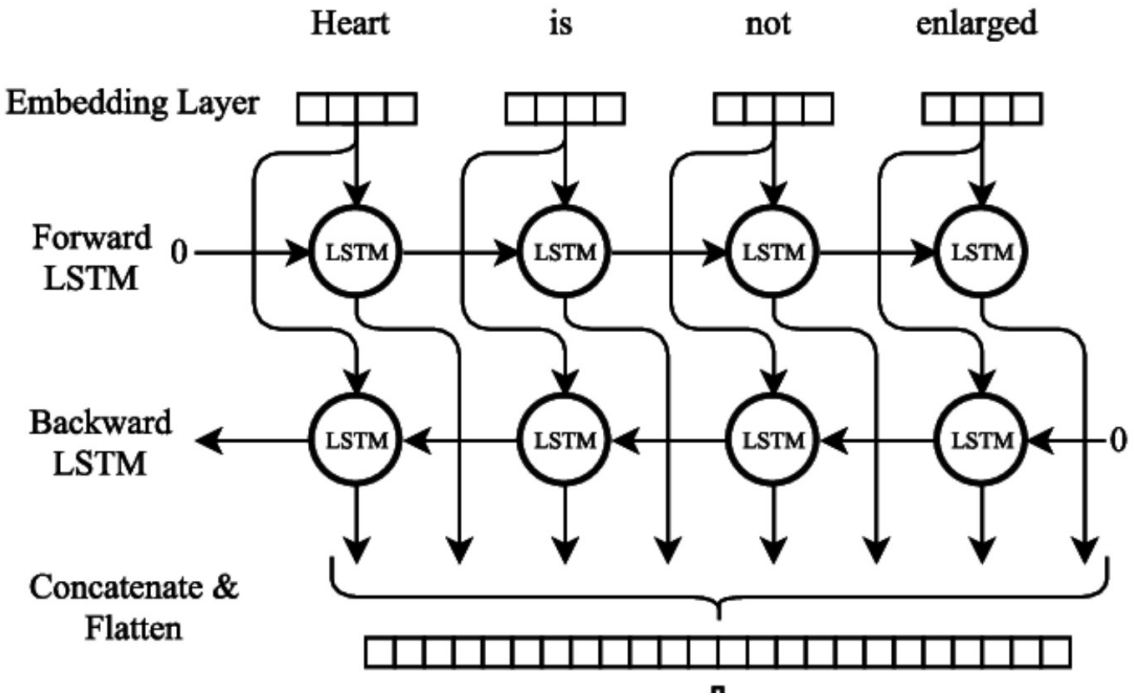
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_3 (Embedding)      (None, 28, 300)           3000000

 batch_normalization_3 (Batc  (None, 28, 300)           1200
 hNormalization)

 bidirectional (Bidirectiona  (None, 512)               1140736
 l)

 dense_3 (Dense)              (None, 150)               76950

=================================================================
Total params: 4,218,886
Trainable params: 4,218,286
Non-trainable params: 600
_________________________________________________________________
```



5.2BILSTM
先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後經過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過第一個 BILSTM 層,其中由兩個由不同順序出入資料的 LSTM(hidden size=256)分別學習每個 time step 的資訊,之後將各自的結果串接在一起(hidden size=512),輸出每個 time step 的 output,隨後經過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 512,再經過第二個 BILSTM 層,輸出最後一個 time step 的 hidden state, 其 hidden size 為 512 。最後經過全連接層搭配 softmax ,產生 150 個類別的機率分佈。

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 28, 300)           3000000

 batch_normalization_4 (Batc  (None, 28, 300)          1200
 hNormalization)

 bidirectional_1 (Bidirectio  (None, 28, 512)          1140736
 nal)

 batch_normalization_5 (Batc  (None, 28, 512)          2048
 hNormalization)

 bidirectional_2 (Bidirectio  (None, 512)              1574912
 nal)

 dense_4 (Dense)             (None, 150)               76950

=================================================================
Total params: 5,795,846
Trainable params: 5,794,222
Non-trainable params: 1,624
_________________________________________________________________
```

b. Performance

	public score	private score
RNN	0.87955	0.88533
GRU	0.87822	0.888
LSTM	0.89377	0.88755
BILSTM	0.89822	0.89244
2BILSTM	0.924	0.92355

由結果可見,RNN 之結果較差,因 RNN 無法記得太久以前的資訊。LSTM 能記住較久以前的資訊,故結果比 RNN 好。BILSTM 同時考慮了兩個不同的方向,結果又再好一些。2BILSTM 更加能夠捕捉完整資訊,故有最佳結果。此 2BILSTM 在 untrainable embedding , vocabulary size 為 10000(包含全部 word) 之條件下進行訓練,有本次 intent classification task 中最佳結果。

c. Loss function

Sparse categorical crossentropy:
Loss=sparse_categorical_crossentropy(y_true, y_pred)

$$L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

特色:
1）和 categorical crossentropy 一樣，不過可以接受非 one-hot 的 vector 作為 y 值
2）配合 softmax 用
3）可用於多分類問題

d. The optimization algorithm (e.g. Adam), learning rate and batch size.
使用 Adam 作為 optimizer,learning rate 設定成 0.001,batch size 為 64

Q3: Describe your slot tagging model.
a. Model
1.RNN
先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後將過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過 RNN 學習每個 time step 的資訊,輸出每個 time step 的 hidden state, 其 hidden size 為 256 。最後經過全連接層搭配 softmax ,產生 10 個類別的機率分佈。

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 35, 300)           3000000

 batch_normalization_4 (Batc  (None, 35, 300)          1200
 hNormalization)

 simple_rnn (SimpleRNN)      (None, 35, 256)           142592

 time_distributed_2 (TimeDis  (None, 35, 10)           2570
 tributed)

=================================================================
Total params: 3,146,362
Trainable params: 3,145,762
Non-trainable params: 600
_________________________________________________________________
```

2.GRU
先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後將過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過 GRU 學習每個 time step 的資訊,輸出每個 time step 的 hidden state, 其 hidden size 為 256 。最後經過全連接層搭配 softmax ,產生 10 個類別的機率分佈。

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 35, 300)           3000000

 batch_normalization_5 (Batc  (None, 35, 300)          1200
 hNormalization)

 gru (GRU)                   (None, 35, 256)           428544

 time_distributed_3 (TimeDis  (None, 35, 10)           2570
 tributed)

=================================================================
Total params: 3,432,314
Trainable params: 3,431,714
Non-trainable params: 600
_________________________________________________________________
```

3.LSTM

先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後將過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過 LSTM 學習每個 time step 的資訊,輸出每個 time step 的 hidden state, 其 hidden size 為 256 。最後經過全連接層搭配 softmax ,產生 10 個類別的機率分佈。

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 35, 300)           3000000

 batch_normalization_8 (Batc  (None, 35, 300)          1200
 hNormalization)

 lstm_6 (LSTM)               (None, 35, 256)           570368

 time_distributed_6 (TimeDis  (None, 35, 10)           2570
 tributed)

=================================================================
Total params: 3,574,138
Trainable params: 3,573,538
Non-trainable params: 600
_________________________________________________________________
```

4.BILSTM

先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後經過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過 BILSTM 層,其中由兩個由不同順序出入資料的 LSTM(hidden size=256)分別學習每個 time step 的資訊,之後將各自的結果串接在一起(hidden size=512),輸出每個 time step 的 hidden state, 其 hidden size 為 512 。最後經過全連接層搭配 softmax ,產生 10 個類別的機率分佈。

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 35, 300)           3000000

 batch_normalization_7 (Batc  (None, 35, 300)          1200
 hNormalization)

 bidirectional_4 (Bidirectio  (None, 35, 512)          1140736
 nal)

 time_distributed_5 (TimeDis  (None, 35, 10)           5130
 tributed)

=================================================================
Total params: 4,147,066
Trainable params: 4,146,466
Non-trainable params: 600
_________________________________________________________________
```

5.2BILSTM

先經過一層 embedding layer 將每個 word 被 embedded 成一個 300 維的向量。之後經過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 300。接著經過第一個 BILSTM 層,其中由兩個由不同順序出入資料的 LSTM(hidden size=256)分別學習每個 time step 的資訊,之後將各自的結果串接在一起(hidden size=512),輸出每個 time step 的 output,隨後經過 batch normalization layer 對每個 batch 進行正規化,輸出維度為 512,再經過第二個 BILSTM 層,輸出每個 time step 的 hidden state, 其 hidden size 為 512 。最後經過全連接層搭配 softmax ,產生 10 個類別的機率分佈。

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 35, 300)           3000000

 batch_normalization_2 (Batc  (None, 35, 300)          1200
 hNormalization)

 bidirectional_2 (Bidirectio  (None, 35, 512)          1140736
 nal)

 batch_normalization_3 (Batc  (None, 35, 512)          2048
 hNormalization)

 bidirectional_3 (Bidirectio  (None, 35, 512)          1574912
 nal)

 time_distributed_1 (TimeDis  (None, 35, 10)           5130
 tributed)

=================================================================
Total params: 5,724,026
Trainable params: 5,722,402
Non-trainable params: 1,624
_________________________________________________________________
```

b. Performance

	public score	private score
RNN	0.67935	0.69131
GRU	0.69222	0.71864
LSTM	0.67184	0.69667
BILSTM	0.79249	0.79849
2BILSTM	0.8016	0.80117

由結果可見 RNN/GRU/LSTM 少了逆向的資訊學習,準確率比起有逆向資訊的 BILSTM 和 BILSTM 差很多,可以見得逆向資訊對於詞性判斷是重要的。

c. Loss function

Categorical crossentropy：

Loss=categorical_crossentropy(y_true, y_pred)

當預測值與實際值愈相近，損失函數就愈小，反之差距很大，就會更影響損失函數的值

$$L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

d. The optimization algorithm (e.g. Adam), learning rate and batch size.

使用 Adam 作為 optimizer,learning rate 設定成 0.001,batch size 為 64

Q4: Sequence Tagging Evaluation

1.RNN

```
              precision    recall  f1-score   support

        date       0.64      0.58      0.61       206
  first_name       0.89      0.84      0.86       102
   last_name       0.80      0.55      0.65        78
      people       0.58      0.63      0.60       238
        time       0.80      0.84      0.82       218

   micro avg       0.70      0.69      0.70       842
   macro avg       0.74      0.69      0.71       842
weighted avg       0.71      0.69      0.70       842

token_acc: 0.9466480800912432
joint_acc: 0.696
```

2.GRU

	precision	recall	f1-score	support
date	0.69	0.61	0.65	206
first_name	0.92	0.87	0.89	102
last_name	0.79	0.68	0.73	78
people	0.59	0.61	0.60	238
time	0.81	0.81	0.81	218
micro avg	0.73	0.70	0.71	842
macro avg	0.76	0.72	0.74	842
weighted avg	0.73	0.70	0.71	842

token_acc: 0.9479153465973894
joint_acc: 0.708

3.LSTM

	precision	recall	f1-score	support
date	0.62	0.66	0.64	206
first_name	0.90	0.84	0.87	102
last_name	0.78	0.91	0.84	78
people	0.58	0.58	0.58	238
time	0.84	0.82	0.83	218
micro avg	0.71	0.72	0.72	842
macro avg	0.74	0.76	0.75	842
weighted avg	0.71	0.72	0.72	842

token_acc: 0.948422253199848
joint_acc: 0.71

4.BILSTM

	precision	recall	f1-score	support
date	0.76	0.72	0.74	206
first_name	0.93	0.93	0.93	102
last_name	0.94	0.78	0.85	78
people	0.72	0.73	0.72	238
time	0.82	0.81	0.82	218
micro avg	0.80	0.78	0.79	842
macro avg	0.83	0.80	0.81	842
weighted avg	0.80	0.78	0.79	842

token_acc: 0.96629071093651
joint_acc: 0.787

5.2BILSTM

	precision	recall	f1-score	support
date	0.73	0.75	0.74	206
first_name	0.95	0.93	0.94	102
last_name	0.86	0.69	0.77	78
people	0.76	0.69	0.72	238
time	0.87	0.86	0.86	218
micro avg	0.81	0.78	0.79	842
macro avg	0.83	0.78	0.81	842
weighted avg	0.81	0.78	0.79	842

token_acc: 0.9655303510328223
joint_acc: 0.786

Joint accuracy = correct sequences/number of sequences
Token accuracy = correct tokens/number of tokens

Seqeval 會把每個 sequence predict/ground truth 拆解為(tag, begin, end)的序列

Precision = $\frac{TP}{TP+FP} = \frac{TP}{\text{預測}(tag, begin, end)\text{序列為該}tag\text{數量}}$

Recall = $\frac{TP}{TP+FN} = \frac{TP}{\text{真實}(tag, begin, end)\text{序列為該}tag\text{數量}}$

F1 score = $\frac{2}{\frac{1}{Precision}+\frac{1}{Recall}}$

Supprt = 真實(tag, begin, end) 為該 tag 的數量

micro avg = $\sum_{i=1}^{N} \frac{1}{N} x \frac{TPi+TNi}{TPi+TNi+FPi+FNi}$

macro avg = $\frac{\sum_{i=1}^{N} TPi}{\text{總樣本數量}}$

weighted avg = $\sum_{i=1}^{N} \frac{TPi+FNi}{\sum_{j=1}^{N} TPi+FNi} x \frac{1}{N} x \frac{TPi+TNi}{TPi+TNi+FPi+FNi}$

Q5: Compare with different configurations

a.Intent classification:

1.Bonus tricks:
使用 Conv1D(filters=256,kernel_size=3,activation='relu',padding='same')
加上一層/兩層 BILSTM

	public score	private score
CNN+BILSTM	0.91733	0.90755
CNN+2BILSTM	0.88933	0.86622

CNN+BILSTM 結果比 BISLSTM 好。但 CNN+2BILSTM 結果比 2BILSTM 差,可能是因為 overfitting。

2.Different vocabulary size:
因為 2BLSTM 在之前的實驗中有最佳的結果,故使用 2BILSTM 做 Ablation study

- Trainable embedding with oov:

vocabulary size(max_words)	public score	private score
6002	0.91866	0.92044
5002	0.908	0.90933
4002	0.91866	0.91422
3002	0.91644	0.91644
2002	0.91022	0.90044
1002	0.88888	0.87288

- Trainable embedding without oov:

vocabulary size(max_words)	public score	private score
6002	0.91911	0.92044
5002	0.89777	0.89733
4002	0.90888	0.90977
3002	0.91111	0.91066
2002	0.908	0.896
1002	0.89911	0.89955

- Untrainable embedding without oov:

vocabulary size(max_words)	public score	private score
6002	0.91422	0.90488
5002	0.91288	0.912
4002	0.91866	0.91466
3002	0.90266	0.89733
2002	0.89644	0.89733
1002	0.89911	0.89955

由此時間可知 embedding layer weights 是否可以 train 以及是否使用 oov token 對於 intent classification task 影響不大。vocabulary size 則是愈多,準確度愈高。

b.Slot tagging:

1.Bonus tricks:

使用 Conv1D(filters=256,kernel_size=3,activation='relu',padding='same')
加上一層/兩層 BILSTM

	public score	private score
CNN+BILSTM	0.76032	0.77116
CNN+2BILSTM	0.8016	0.81189

可以見得 CNN 對於準確率提昇無明顯幫助,因為 CNN 無法有效提取完整 sequence data 特徵。使用 CNN+2BILSTM model ,在 Trainable embedding with oov tokens ,vocabulary size=10000 之條件下,為本次 slot tagging task 中 private score 最高之設置。

2.Different vocabulary size:
因為 2BLSTM 在之前的實驗中有最佳的結果,故使用 2BILSTM 做 Ablation study

- Trainable embedding with oov tokens:

vocabulary size(max_words)	public score	private score
5002	0.78284	0.79367
4002	0.77694	0.78938
3002	0.73887	0.7508
2002	0.79839	0.81243
1002	0.81662	0.80707
502	0.79624	0.80332

- Untrainable embedding with oov tokens:

vocabulary size(max_words)	public score	private score
5002	0.78766	0.78188
4002	0.78713	0.77706
3002	0.79142	0.78349
2002	0.80268	0.79474
1002	0.79839	0.79796
502	0.78176	0.79689

由上表可以觀察出,embedding layer weights 是否可以 training 對 slot tagging 任務影響不大。另外,由其他實驗可的使用 oov token 比起不使用之分數高,這是因為若不使用 oov token ,會導致後面的資料被移到前面,其所對應的 tag 則為錯誤的 tag,造成準確率下降。使用 2BILSTM model ,在 Trainable embedding with oov tokens ,vocabulary size=1002 之條件下,為本次 slot tagging task 中 public score 最高之設置。

c.Other observations:

- mask 和 glove embedding 對兩個 task 都會提昇預測準確率
- 兩個 task 皆是 BILSTM 模型有最佳結果
- 在進行 slot tagging 任務時,activation function 使用 relu, leaky relu , swish 之效果無明顯差別。optimizer 使用 rmsprop 會得到很糟糕的結果。dropout 層會使結果變差。

References:

1.https://dotblogs.com.tw/greengem/2017/12/17/094023
2. https://medium.com/holler-developers/slot-filling-using-sequence-models-78ecd928b300
3. https://keras.io/api/
4. https://ithelp.ithome.com.tw/articles/10194201