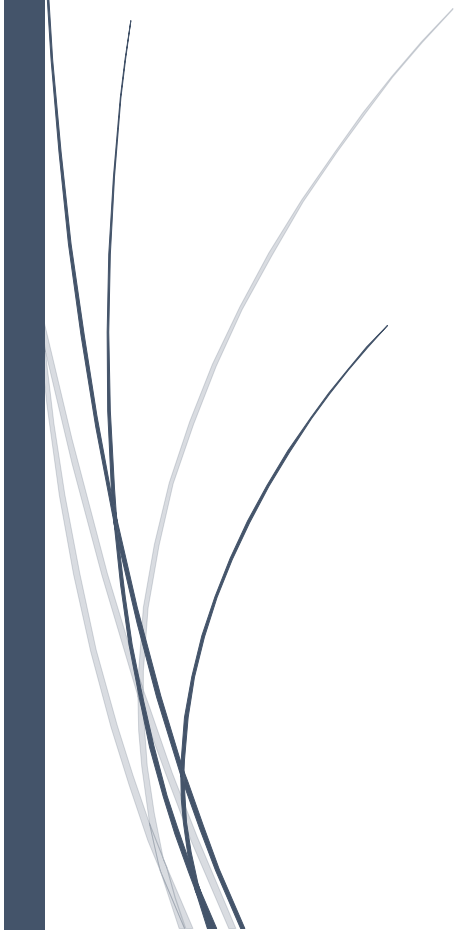


醫病訊息決策與對話語料分析競賽 秋季賽：醫病資料去識別化 成果報告



隊名：沒事就來遛一遛

指導教授：

國立成功大學系統及船舶機電工程學系 李坤洲教授

隊員：

國立成功大學系統及船舶機電工程所 顏振宇

國立成功大學系統及船舶機電工程所 賴煜翔

國立成功大學系統及船舶機電工程所 周禮宏

國立成功大學系統及船舶機電工程所 施俊宇

目錄

壹、摘要	2
貳、演算法說明	3
參、工具說明	5
肆、流程說明	6
伍、組態說明	13
陸、外部資源與參考文獻.....	14

壹、摘要

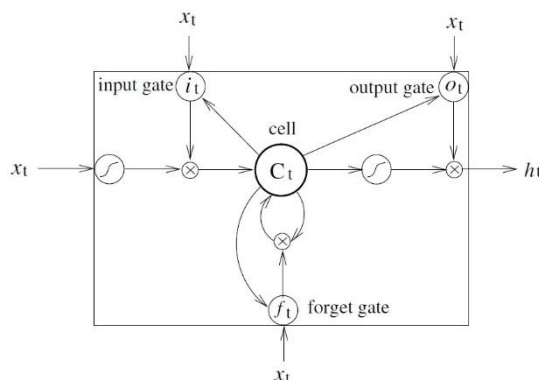
近年來本實驗室的研究跟人工智慧有相當大的關連，特別是深度學習的部分。除了研究上的應用，我們在研究所期間也修了幾門深度學習相關的課程，剛好在本學期開始時看到醫病訊息決策與對話語料分析競賽，而且又是成功大學主辦的，這激發了我們挑戰的動力，雖然我們之前沒做過自然語言處理的應用，但相信透過此競賽可以獲得寶貴的經驗以及知識，也可以了解自己和其他參賽隊伍的實力差距。

經過不斷的嘗試，我們最後決定使用 BI-LSTM-CRF 模型來進行訓練，這訓練的過程經歷了許多編寫程式上的困難，也有許多不同想法的碰撞，究竟是該把時間花在訓練集標註的正確性以及擴充，還是花在調整深度學習模型的參數調整呢？雖然辛苦，但看到排行榜的分數有些微的提升就覺得這一切的努力是有價值的，此報告書會介紹我們在訓練資料集與 BI-LSTM-CRF 模型做了什麼調整，而最後得到怎麼樣的成果。

貳、演算法說明

本組使用 Bi-LSTM-CRF Network，應用 BI-LSTM-CRF 模型於 NLP 基準序列標記數據集，由於具有雙向 LSTM 組件，該模型可以同時使用過去和未來的輸入特徵。

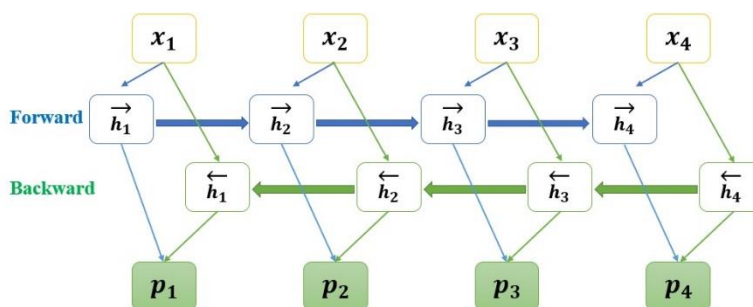
RNN(Recurrent Neural Network)能夠在輸入、輸出序列之間的映射過程中利用上下文有關的訊息，但能夠存取的訊息有限，使得隱藏層的輸入對於輸出的影響隨著神經網路不斷遞歸而衰減，為了解決此問題，創造了 LSTM (Long Short-Term Memory)，將 RNN 隱藏層中的神經元替代為 LSTM 元件，如圖一，更擅長查找和利用數據中的遠程依賴關係。



圖一、單個 LSTM 儲存元件

Bidirectional LSTM Networks，在序列標記任務中，在特定時間範圍內，可以使用過去、未來的輸入功能，因此可以使用雙向的 LSTM 網路，如此一來，可以在特定時間範圍內，有效地利用過去特徵 (Forward) 和未來特徵 (Backward)，如圖二。

使用反向傳播(backpropagation through time)訓練雙向 LSTM 網路，先在數據的開頭和結尾處進行特殊處理，對整個句子進行前進(Forward)和後退(Backward)操作，只需要在每個句子開始時將隱藏狀態重置為 0，批次處理實現，可以同時處理多個句子。

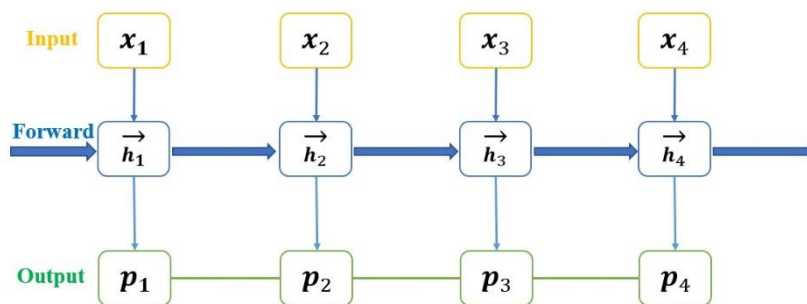


圖二、BI-LSTM Network

CRF networks(Conditional Random Fields) 邏輯回歸、線性 CRF 在數學上是相同的，訓練資料中的每個詞都有一個標註，對句子的第某個位置的詞抽取高維度特徵，透過學習特徵到標註的映射，能夠獲得特徵到任意標

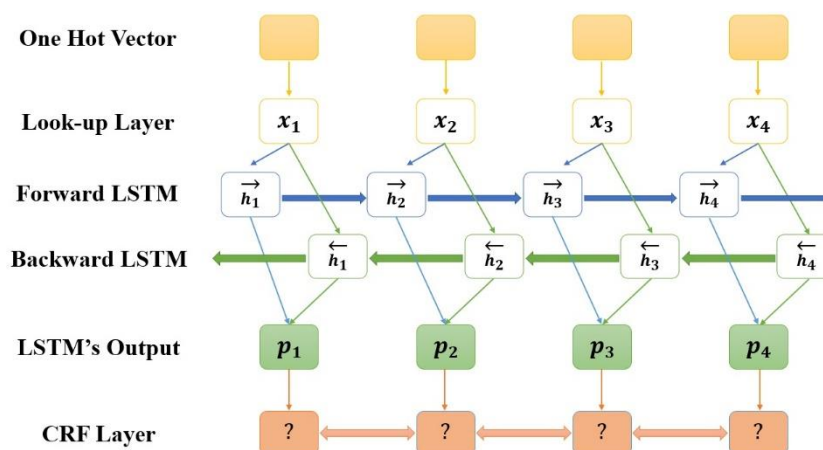
註的機率，測試時從句子開頭起抽取特徵，預測標註機率，並帶入下一個特徵，再預測下一輪標註的機率，使用維特比(Viterbi)演算法得到最佳路徑。

LSTM-CRF Networks 將LSTM、CRF結合，形成LSTM-CRF模型，可以透過LSTM層有效地使用過去的輸入功能，並通過CRF層有效地使用句子級別的標籤信息，如圖三。



圖三、LSTM-CRF Model

BI-LSTM-CRF Networks 與 LSTM-CRF 網路相似，將雙向 LSTM、CRF 結合起來，形成了 BI-LSTM-CRF 網路。除了 LSTM-CRF 模型中使用的過去輸入功能和句子級別標籤信息之外，BI-LSTM-CRF 模型還可以使用將來的輸入功能，這些額外的功能可以提高標記的準確性。CRF 層由連接連續輸出層的線表示，且以狀態轉移矩陣作為參數。通過這樣的層，有效地使用過去和未來的標籤來預測當前標籤，使用過去和將來的輸入功能，如圖四。



圖四、BI-LSTM-CRF Model

參、工具說明

- 工作站的硬體:

Intel Core i7-7740X CPU

Geforce GTX 1080 Ti

- 工作站的軟體:

Ubuntu Server 16.04 LTS server

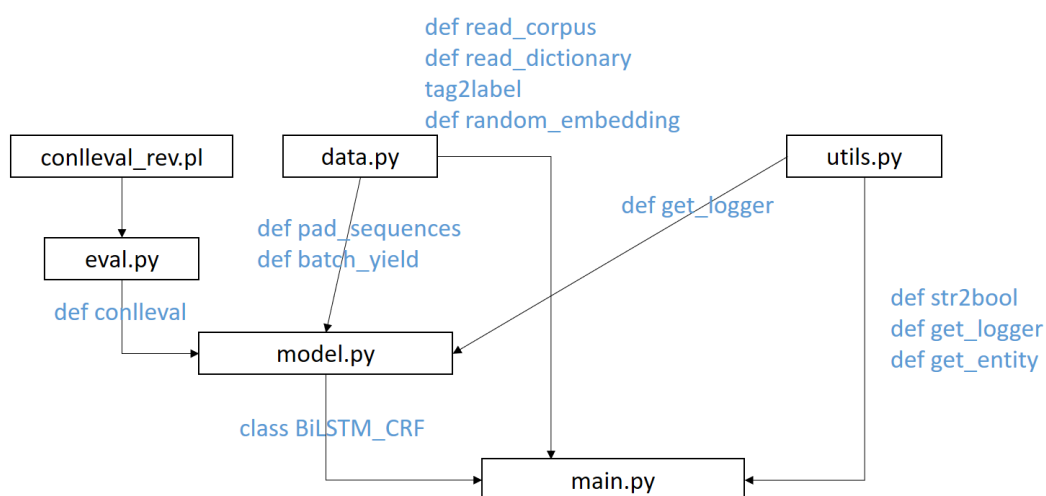
GNOME Desktop

NVIDIA Driver 390.25

CUDA 9.0

肆、流程說明

我們使用的 BI-LSTM-CRF 模型程式結構如圖五。



圖五、BI-LSTM-CRF 模型程式結構

步驟一：數據處理

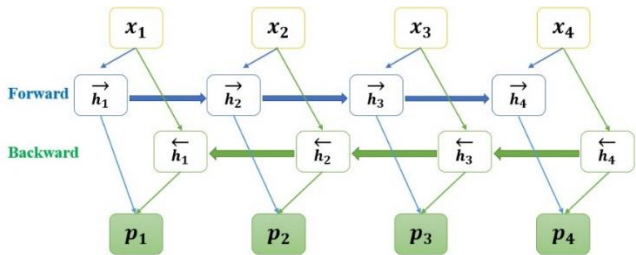
在 data.py 中

函式	輸入	輸出	功用
tag2label			建立資料中出現的 label 的字典，格式為 B-XXX, I-XXX
read_corpus	corpus_path	data(type:list , 含 sent_, tag_)	用來處理輸入的訓練集或驗證集
read_dictionary	vocab_path (=訓練集路徑)	word2id	用 pickle.load 輸出 word2id，pickle 套件可以創建 python 用的二進位制檔案
pad_sequences	sequences、pad_mark	seq_list、seq_len_list	提取出長度最長的語句，將全部樣本 padding 成相同長度

batch_yield	data、 batch_size、 vocab、 tag2label、 shuffle	seqs、labels	用來生成 batch
-------------	---	-------------	------------

步驟二：建立 BI-LSTM-CRF 模型

在 model.py 中的 class BiLSTM_CRF

函式	功用
<code>__init__</code>	讀取 main 所設定的超參數
<code>build_graph</code>	架構訓練網路
<code>add_placeholders</code>	添加 wordid、label、seq 長度、dropout 及 learning rate
<code>lookup_layer_op</code>	搜尋 word_embeddings 內句子 id 所對應的詞，並做 dropout
<code>biLSTM_layer_op</code>	<p>從 tf.contrib.rnn 匯入 LSTMCell，用以定義要輸入 tf.nn.bidirectional_dynamic_rnn 中的前項 rnn 與後項 rnn</p> <p>tf.nn.bidirectional_dynamic_rnn: 輸出前向和後向 rnn 輸出的張量，最後再以 tf.concat 串接結果，並 dropout</p>  <p>這裡的 output(形狀為[batch_size,steps,cell_num])為一層隱藏層的輸出，到下一層會經由 tf.matmul 將前一層與下一層中間的 Weight 連接加上 bias 輸入下一層，在這裡 h 就是 hidden_dim</p>
<code>loss_op</code>	<p>#crf_log_likelihood 為損失函式</p> <p>#inputs: unary potentials，即每個標籤的預測概率值</p> <p>#tag_indices 為真實的標籤序列了</p> <p>#sequence_lengths 為一個樣本真實的序列長度，為了對齊長度會做些 padding，但是可以把真實的長度放到這個引數裡</p> <p>#transition_params 為轉移概率，可以沒有沒有的話</p>

	<p>這個函式也會算出來</p> <p>#輸出：log_likelihood 標量、transition_params 轉移概率，如果輸入沒輸，它就自己算個給返回</p> <p>crf_log_likelihood: 輸出 log_likelihood 及 transition_params，目的是以 CRF 來計算 loss，Loss function 為 <code>tf.reduce_mean(log_likelihood)</code>，若不是 CRF 就以交叉熵做損失函式</p>
softmax_pred_op	如果資料形式不是 CRF，使用 softmax
trainstep_op	選擇使用的 optimizer，Adam\ Adadelata\ Adagrad\ RMSProp\ Momentum\ SGD
init_op	初始化參數
add_summary	將所有的 summary 記錄在 summary 檔案中
train	<p>add_summary</p> <p>做 epoch 次的 run_one_epoch</p>
test	利用 dev_one_epoch 讀出 label_list、seq_len_list，再用 evaluate
demo_one	利用 data 中的 batch_yield 輸出 seqs、labels，再用 predict_one_batch 得到 label_list、seq_len_list 存入 label_list、tag2label 為標籤的 bi_label，對於要 demo 的資料做 label
get_feed_dict	<p>從 data 匯入 pad_sequences</p> <p>#seq_len_list 用來統計每個樣本的真實長度</p> <p>#word_ids 即 seq_list，padding 後的樣本序列</p> <p>#labels 經過 padding 後，餵給 feed_dict</p>
run_one_epoch	訓練一次 epoch，並 dev_one_epoch 輸出至 evaluate 計算 validation
dev_one_epoch	<p>batch_yield 輸出 seqs, labels 輸入 predict_one_batch，</p> <p>將結果存入 label_list, seq_len_list 並輸出</p>
predict_one_batch	<p>使用 get_feed_dict 得到 feed_dict、seq_len_list、seq_len_list 用來統計每個樣本的真實長度，</p> <p>若是 CRF，輸出 label_list、seq_len_list、transition_params 代表轉移概率，由 crf_log_likelihood 方法計算出</p>
evaluate	<p>計算準確率，使用 conlleva1</p> <p>for _ in conlleva1(model_predict, label_path, metric_path):</p> <p>self.logger.info(_)</p>

	註: conllevl 在 eval 中
--	----------------------

步驟三:

在 utils.py 中

函式	輸入	輸出	功用
str2bool	v	Boolean	將字串轉換為布林值 ex. Yes->True, t -> True, No->False, false->False
get_entity	tag_seq、char_seq	all_start_position、 all_end_position、 all_entity_text、 all_entity_type	將預測的數據處理成想要的格式， 有所有的 start_position、 end_position、 entity_text、 entity_type
get_logger	filename (log_path= checkpoint/results/log.txt)	logger	取得 log 資料，並寫入 log.txt

步驟四:

在 eval.py 中

函式	輸入	輸出	功用
conllevl	label_predict、 label_path、 metric_path		在 data_path_save 中的 checkpoint 資料夾的 results 寫入 label_xx (由 label_predict 生成) 寫入 metrics_xx (由 conllevl_rev.pl 生成)

利用 conllevl_rev.pl 得到各 label 的 precision、recall 和 F1-score

步驟五：執行模型

在 main.py 中

呼叫 model 中的 BiLSTM_CRF

呼叫 utils 中的 str2bool, get_logger, get_entity

呼叫 data 中的 read_corpus, read_dictionary, tag2label, random_embedding

設定超參數

參數	功用
Train_data	更改訓練的資料集
Test_data	更改測試的資料集
Batch_size	訓練速度，取決於硬體設備
Epoch	訓練次數，過多會導致 OVERFITTING
Hidden_dim	隱藏層的神經元個數
Optimizer	優化器
CRF	使用 CRF 或 SOFTMAX
Lr	每次更新的速度
Clip	閾值不過度訓練
Dropout	隨機丟失值
update_embedding	update embedding during training
pretrain_embedding	use pretrained char embedding or init it randomly
embedding_dim	輸出層神經元個數，需與 HIDDEN_DIM 相等
shuffle	shuffle training data before each epoch
mode	TRAIN\TEST\DEMO
demo_model	選擇 CHECKPOINT
meta	選擇第幾個 GLOBAL_STEP

get char embeddings 讀取字典 (data 中的 read_dictionary(將自己的 train data) wordid 由 data 中的 vocab_build 生成)，然後以 pretrain_embedding 決定要使用 train data 還是預先訓練好的。

如果不是 demo 的形式，設定好 train 與 test 的資料(read_corpus 讀取 data 將句子及 label 分開)。

建立儲存訓練的 summaries checkpoint model log

決定要 train\test\demo 的其中一種模式

demo 可決定我們使用哪一個 checkpoint 檔來 demo 儲存 tsv 檔

新增資料集方法

- Addlabel.py (對於官方釋出的 wordnet 進行新增標籤的動作)

讀取官網釋出的 wordnet.xlsx，將 columns 分別對應的下方關鍵字加入字典，並搜尋原本訓練集中的句子，若沒有標註到，進行標註

- 第二版

發現關鍵字中有些特殊格式，如禮拜 X、X 月 X 號、XX 大學、X 醫師、電話號碼、email 等等的格式，所以使用正規化的處理，在遇到此格式時能正確做出 label

- 第三版

發現有一些病名屬於病患的隱私，如梅毒、B 肝、C 肝等等，加入 others

- Add_development.py

發現先前釋出的 development 與 test 測試及不進相同，所以利用了以前訓練的 weight 來做第一層標註，在經過上面的正規化程序，以增加我們的 training_data

使用 train 模式的執行畫面如下:

```
processed 314 tokens with 12 phrases; found: 12 phrases; correct: 12.
accuracy: 100.00%; precision: 100.00%; recall: 100.00%; FB1: 100.00
location: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
name: precision: 100.00%; recall: 100.00%; FB1: 100.00 2
time: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
2020-12-28 00:34:23 epoch 398, step 1, loss: 5.46, global_step: 9132
2020-12-28 00:34:23 epoch 398, step 23, loss: -1.737, global_step: 9154
=====validation=====
processed 314 tokens with 12 phrases; found: 12 phrases; correct: 12.
accuracy: 100.00%; precision: 100.00%; recall: 100.00%; FB1: 100.00
location: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
name: precision: 100.00%; recall: 100.00%; FB1: 100.00 2
time: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
2020-12-28 00:36:09 epoch 399, step 1, loss: 6.858, global_step: 9155
2020-12-28 00:36:09 epoch 399, step 23, loss: 5.744, global_step: 9177
=====validation=====
processed 314 tokens with 12 phrases; found: 12 phrases; correct: 12.
accuracy: 100.00%; precision: 100.00%; recall: 100.00%; FB1: 100.00
location: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
name: precision: 100.00%; recall: 100.00%; FB1: 100.00 2
time: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
2020-12-28 00:38:01 epoch 400, step 1, loss: 5.788, global_step: 9178
2020-12-28 00:38:01 epoch 400, step 23, loss: 0.6812, global_step: 9200
=====validation=====
processed 314 tokens with 12 phrases; found: 12 phrases; correct: 12.
accuracy: 100.00%; precision: 100.00%; recall: 100.00%; FB1: 100.00
location: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
name: precision: 100.00%; recall: 100.00%; FB1: 100.00 2
time: precision: 100.00%; recall: 100.00%; FB1: 100.00 5
(tensorflow) p16081295_1@a1-7:~/Desktop/1214-2033$
```

使用 demo 模式的執行畫面如下: (預測 test.txt)

```
432] Found device 0 with properties:
name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.683
pciBusID: 0000:01:00.0
totalMemory: 10.91GiB freeMemory: 10.28GiB
2020-12-28 22:47:53.877184: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
511] Adding visible gpu devices: 0
2020-12-28 22:47:54.062621: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
82] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-12-28 22:47:54.062656: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
88]      0
2020-12-28 22:47:54.062662: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
001] 0:  N
2020-12-28 22:47:54.062812: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 wit
h 9935 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1080 Ti, pci bus
id: 0000:01:00.0, compute capability: 6.1)
===== demo =====
INFO:tensorflow:Restoring parameters from ./data_path_save/2020-12-28-1005/check
points/model-8510
Restoring parameters from ./data_path_save/2020-12-28-1005/checkpoints/model-851
0
159
100%|████████████████████████████████████████| 159/159 [01:18<00:00, 2.02it/s]
(tensorflow) p16081295_1@a1-7:~/Desktop/1214-2033$
```

伍、組態說明

- 軟體環境:

python version: 3.6.5

tensorflow-gpu: 1.10.1

keras: 2.2.0

- 結果紀錄

在 BI-LSTM 中有 8 個參數可以調整，分別是:batch_size、epoch、hidden_dim、embedding_dim、optimizer、learning rate、clip、dropout，我們經過反覆的實驗以及推論，找出了一組較好的參數如下所示:

1. batch_size = 12

batch_size 為批次訓練的量，取決於本身的運算資源，我們這次使用 RTX-1080Ti 進行訓練，最大值可調至 12，不影響準確率，但可以改變訓練速度

2. epoch = 400

在經過觀察 loss 的趨勢，在 350 左右會開始收斂，所以之後的訓練會訂在 400，避免 overfitting 的情況

3. hidden_dim = 300

在模型中間有包含一層 hidden_layer，會決定訓練參數的多寡，注意 embedding_dim 需與 hidden_layer 數量相同，才不會造成 tensorflow 運算時的張量不同的錯誤

4. optimizer = Adam

決定在尋找 minimum 時所使用的方法

5. learning rate = 0.0005

決定參數在進行迭代時所前進的速度

6. clip = 5.0

限制參數一次可改變的幅度，避免極端情況

7. dropout = 0.5

隨機丟失部分參數，避免 overfitting

本組本次使用 Bi-LSTM-CRF 神經網路在 public leaderboard 的評估結果為 **0.68**，precision:0.63255、recall:0.73743

陸、外部資源與參考文獻

- **Paper**

Zhiheng Huang & Wei Xu & Kai Yu (2015) *Bidirectional LSTM-CRF Models for Sequence Tagging*

Guillaume Lample & Miguel Ballesteros & Sandeep Subramanian & Kazuya Kawakami & Chris Dyer (2016) *Neural Architectures for Named Entity Recognition*

- **GitHub**

https://github.com/huoliangyu/zh-NER-TF_Chinese-bi-LSTm-CRF