**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**

**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**

**目錄**

[壹、 摘要 2](#_Toc60144712)

[貳、 演算法說明 3](#_Toc60144713)

[參、 工具說明 5](#_Toc60144714)

[肆、 流程說明 6](#_Toc60144715)

[伍、 組態說明 13](#_Toc60144716)

[陸、 外部資源與參考文獻 14](#_Toc60144717)

1. **摘要**

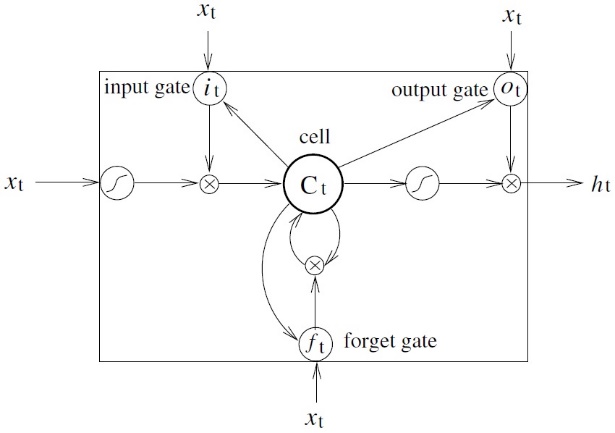
近年來本實驗室的研究跟人工智慧有相當大的關連，特別是深度學習的部分。除了研究上的應用，我們在研究所期間也修了幾門深度學習相關的課程，剛好在本學期開始時看到醫病訊息決策與對話語料分析競賽，而且又是成功大學主辦的，這激發了我們挑戰的動力，雖然我們之前沒做過自然語言處裡的應用，但相信透過此競賽可以獲得寶貴的經驗以及知識，也可以了解自己和其他參賽隊伍的實力差距。

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。

1. **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。

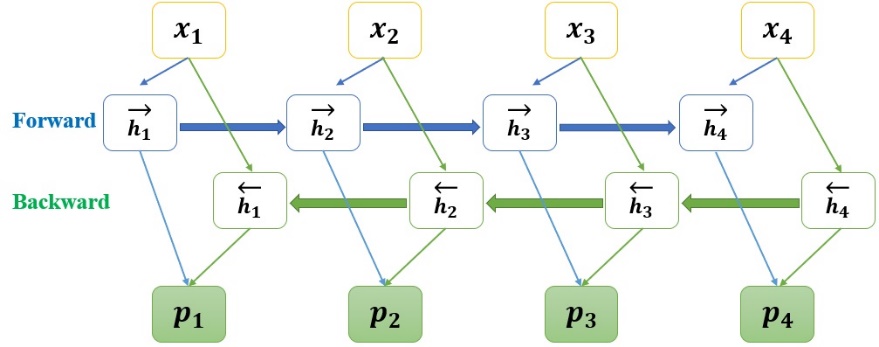
RNN(Recurrent Neural Network)能夠在輸入、輸出序列之間的映射過程中利用上下文有關的訊息，但能夠存取的訊息有限，使得隱藏層的輸入對於輸出的影響隨著神經網路不斷遞歸而衰減，為了解決此問題，創造了LSTM (Long Short-Term Memory)， 將RNN隱藏層中的神經元替代為LSTM元件，如圖一，更擅長查找和利用數據中的遠程依賴關係。



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。

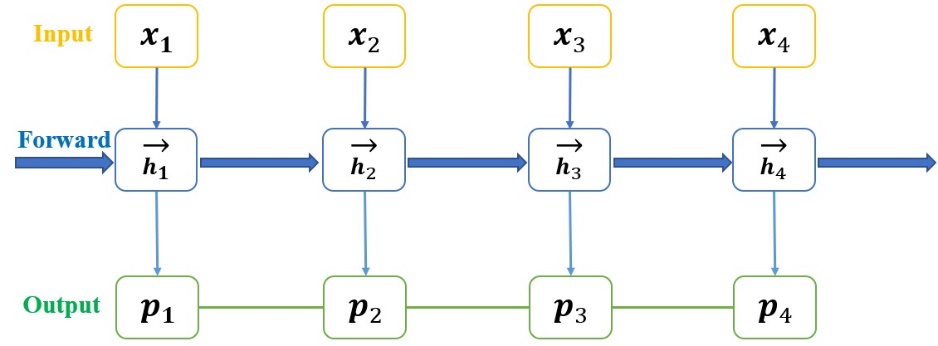
使用反向傳播(backpropagation through time)訓練雙向LSTM網絡，先在數據的開頭和結尾處進行特殊處理，對整個句子進行前進(Forward)和後退(Backward)操作，只需要在每個句子開始時將隱藏狀態重置為0，批次處理實現，可以同時處理多個句子。



圖二、BI-LSTM Network

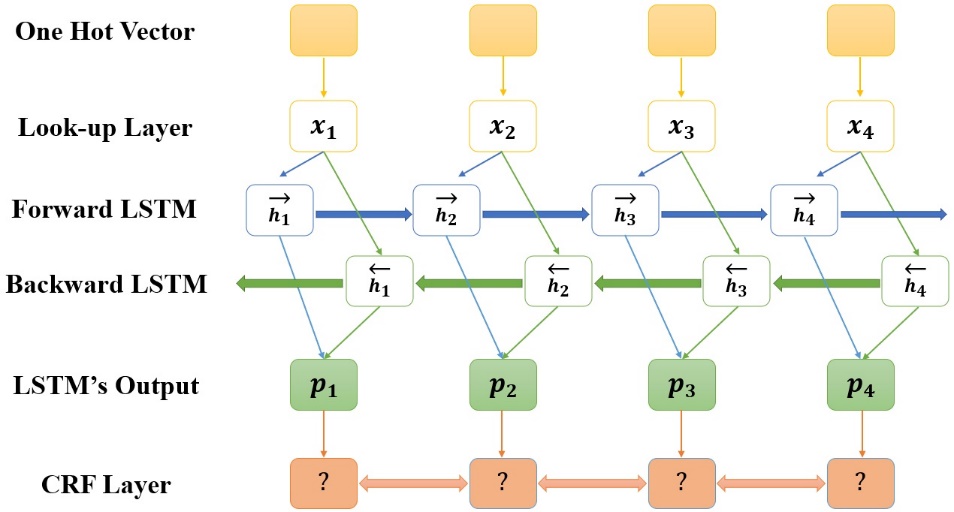
CRF networks(Conditional Random Fields) 邏輯回歸、線性CRF在數學上是相同的，訓練資料中的每個詞都有一個標註，對句子的第某個位置的詞抽取高維度特徵，透過學習特徵到標註的映射，能夠獲得特徵到任意標註的機率，測試時從句子開頭起抽取特徵，預測標註機率，並帶入下一個特徵，再預測下一輪標註的機率，使用維特比(Viterbi)演算法得到最佳路徑。

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

1. **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

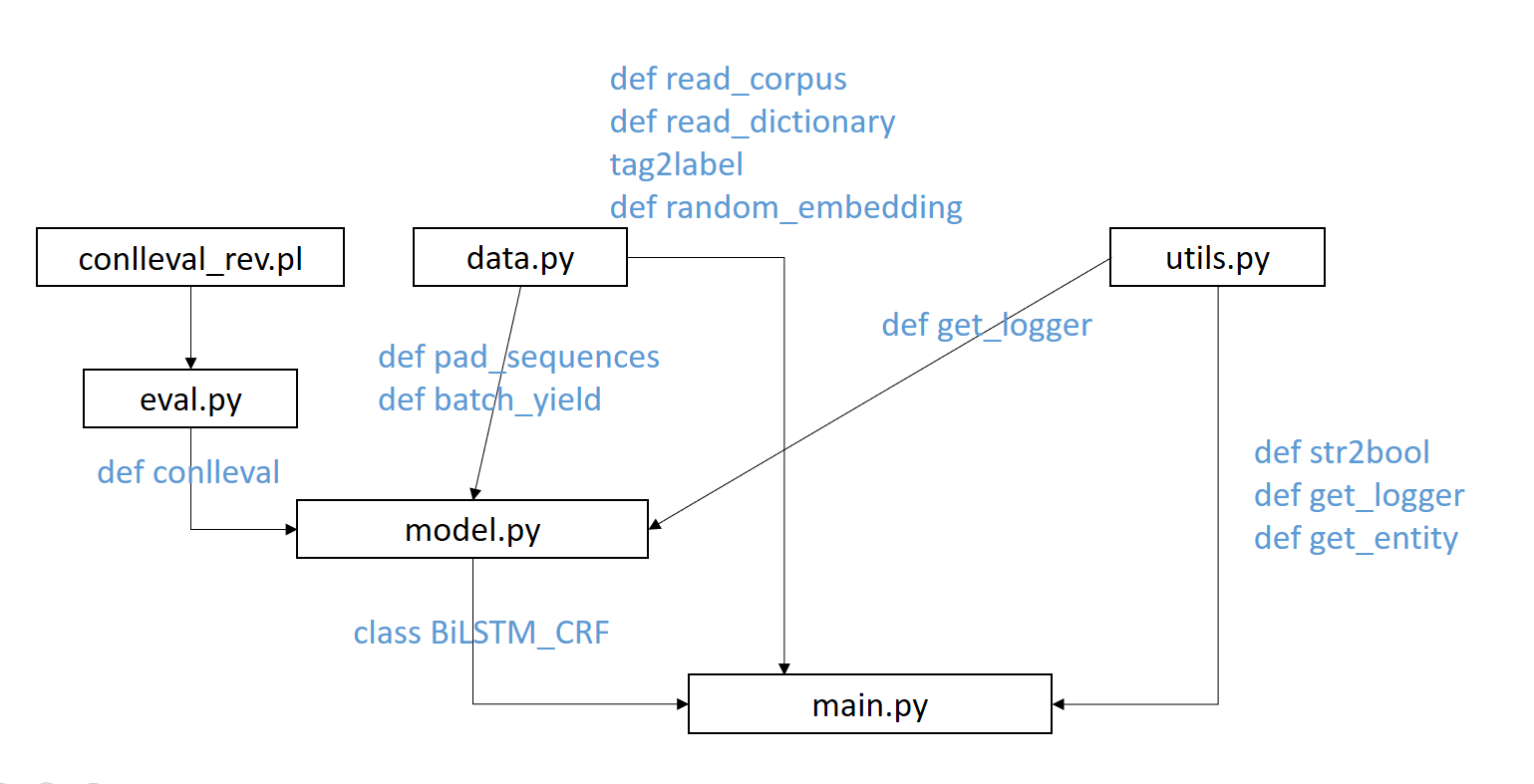
GNOME Desktop

NVIDIA Driver 390.25

CUDA 9.0

1. **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 |
| \_\_init\_\_ | 讀取main所設定的超參數 |
| build\_graph | 架構訓練網路 |
| add\_placeholders | 添加wordid 、label、 seq長度、dropout及learning rate |
| lookup\_layer\_op | 搜尋word\_embeddings內句子id所對應的詞，並做dropout |
| biLSTM\_layer\_op | 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  tf.nn.bidirectional\_dynamic\_rnn:  輸出前向和後向rnn输出的張量，最後再以tf.concat串接結果，並dropout    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 |
| loss\_op | #crf\_log\_likelihood為損失函式  #inputs：unary potentials，即每個標籤的預測概率值  #tag\_indices為真實的標籤序列了  #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  #transition\_params為轉移概率，可以沒有沒有的話這個函式也會算出來  #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  crf\_log\_likelihood: 輸出log\_likelihood及transition\_params，目的是以CRF來計算loss，Loss function為tf.reduce\_mean(log\_likelihood)，若不是CRF就以交叉熵做損失函式 |
| softmax\_pred\_op | 如果資料形式不是CRF，使用softmax |
| trainstep\_op | 選擇使用的optimizer，Adam\ Adadelta\ 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 | add\_summary  做epoch次的run\_one\_epoch |
| 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 | 從data匯入pad\_sequences  #seq\_len\_list用來統計每個樣本的真實長度  #word\_ids即seq\_list，padding後的樣本序列  #labels經過padding後，餵給feed\_dict |
| run\_one\_epoch | 訓練一次epoch，並dev\_one\_epoch輸出至evaluate計算validation |
| dev\_one\_epoch | batch\_yield輸出seqs, labels輸入predict\_one\_batch，  將結果存入label\_list, seq\_len\_list並輸出 |
| predict\_one\_batch | 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，  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 |
| evaluate | 計算準確率，使用conlleval  for \_ in conlleval(model\_predict, label\_path, metric\_path):  self.logger.info(\_)  註: conlleval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\_postion、  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 |
| conlleval | 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conlleval\_rev.pl生成) |

利用conlleval\_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 | 閥值不過度訓練 |
| 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可決定我們要使用哪一個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

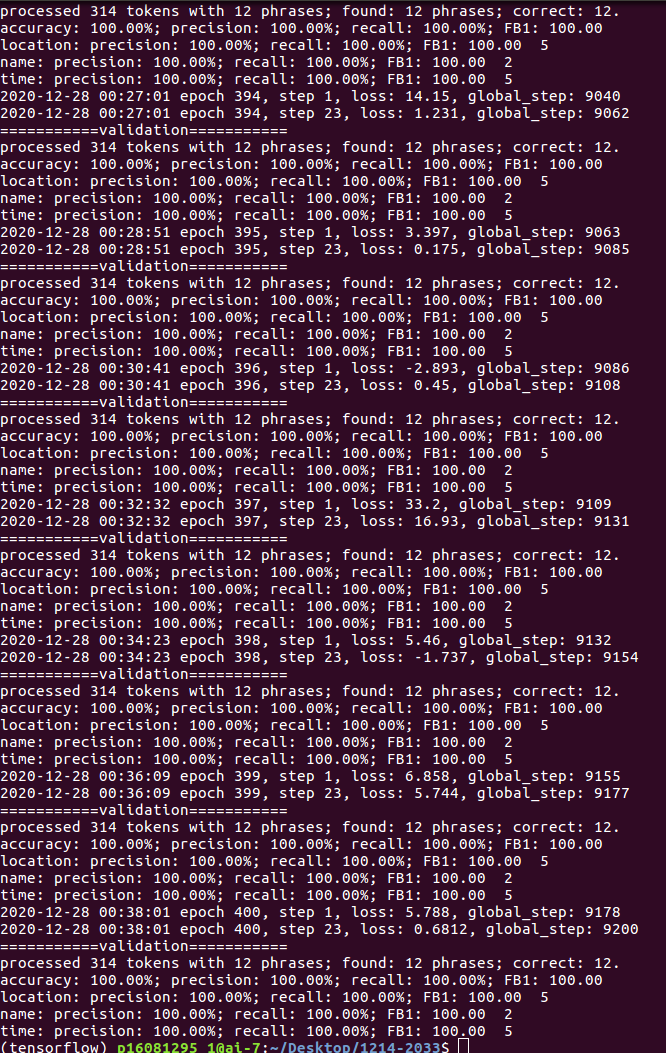
* 第三版

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:



使用demo模式的執行畫面如下: (預測test.txt)



1. **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

* 1. 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

* 1. 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

* 1. optimizer = Adam

決定在尋找minimum時所使用的方法

* 1. learning rate = 0.0005

決定參數在進行迭代時所前進的速度

* 1. clip = 5.0

限制參數一次可改變的幅度，避免極端情況

* 1. 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**

1. **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>