How I tokenize data and the pre-trained embedding I used

執行 preprcess_intent.py 生成
intent2idx.json：把 intent 轉為 id
vocab.pkl：把詞彙轉為 id
embedding.pt：把詞彙轉為 vector

執行 preprocessed_slot 則生成
token2idx: 把 token 轉為 id
vocab.pkl：把詞彙轉為 id
embedding.pt：把詞彙轉為 vector

1. intent2idx.json / token2idx

(1) 用途：

在 intent 分類的任務當中，我們會使用 intent2idx.json 把 eval.json, train.json 中的 intent 欄位轉換成數字 id，並且在 test_intent.py 也會讀取這個檔案把預測出來的數字 id 轉換成文字的 intent.

而在 slot 中則是用 tag2idx 把 tags 欄位中的 tag 轉換成相對應的數字編號

(2) 生成過程：

1. 讀取 train.json, eval.json 檔案
2. Intent
   A. 遍歷所有 intent 欄位，記下所有出現過的 intent 類型。
   B. 賦予每一個 intent 不重複的數字編號
   C. 轉換成一個儲存 intent 以及 intent 相對應的 id 的 dictionary。

3. Slot
    A. 遍歷 tags 欄位，紀錄出現過的 tag
    B. 賦予每一個 tag 不重複的數字編號
    C. 轉換成一個儲存 tag 以及其相對應的 id 的 dictionary。

(3) 內容：

dictionary，intent2idx 是儲存 intent 以及 intent 相對應的 id，tag2idx 則是儲存 tag 以及該 tag 相對應的 id

2. vocab.pkl

(1) 用途：

在 train_intent, test_intent 當中我們會用空白把 text 欄位的句子斷開得到詞彙，使用 vocab.pkl 把詞彙轉換成數字 id

(2) 生成過程

1. 讀取 train.json, eval.json 檔案
2. 建立一個名為 word 的 counter，
3. 遍歷所有 train.json, eval.json 當中的 text 欄位，用空白斷開字串得到詞彙
4. 詞彙儲存進去 word 當中，同時更新該詞彙目前出現的次數
5. 取出現頻率前 10,000 的詞彙
6. 輸出 vocab.pkl

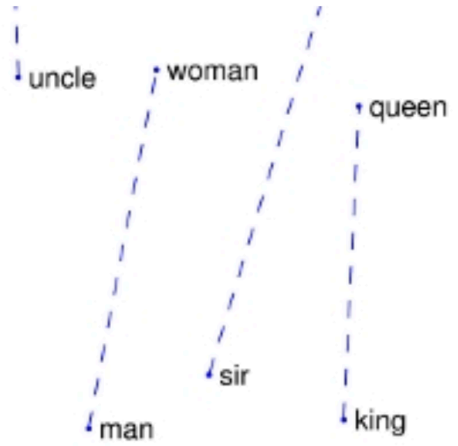
(3) 內容：

詞彙以及其相對應的數字 id

3. embedding.pt

我們使用 GloVe 作為 pretrain embedding，GloVe 的原理是計算詞彙在文章當中出現的頻率，歸納詞彙共同出現的機率，決定詞彙彼此之間的相似性。我們使用的檔案是 glove.840B.300d.txt，代表原始用

來訓練的語料庫包含 840B 個 token，每一個詞彙使用 300 維的向量儲存，用來代表每個詞彙的意思。向量之間的距離則代表詞彙跟詞彙之間的差別。



圖一[註 1]

(1) 用途：

把文字轉成對應的向量

(2) 生成過程：

讀取 train.json, eval.json 檔案
讀取 glove.840B.300d.txt，裡面包含詞彙以及其相對應的 vector
利用前述流程建立 train.json, eval.json 檔案的 word set，遍歷所有 glove 當中的詞彙，如果該詞彙也包含在 word set 當中就記錄下來
確認所有的向量長度都一樣
輸出成 embedding.pt

(3) 內容：

詞彙以及其相對應的 vector

Intent Classification Model Structure

1. Describe your model

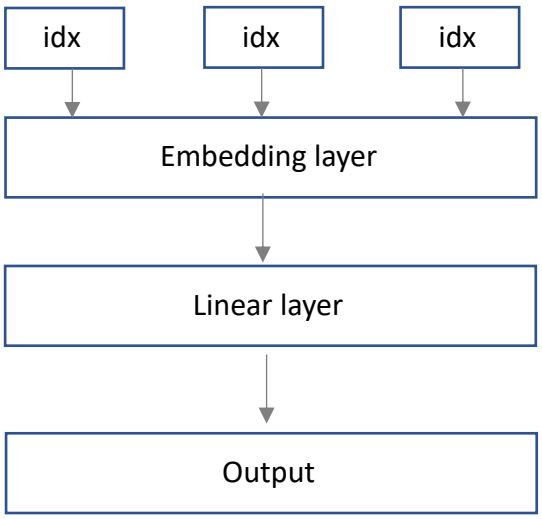


圖 2. Intent model

Model 包含兩層，第一層是 embedding layer，第二層是 linear layer，第一層的 embedding layer 的功能是把輸入的文字轉為 GloVe 當中相對應的向量傳入第二層的 linear layer，linear layer 則會對輸入的資料進行線性變換（transformer），輸出預測結果，以數學公式表示如下：

$$embed = embedding(T_i, O_i),$$

T_i is the i-th batch of text, the dimension of $T_i = 1 * \sum_{n=1}^{N} t_i$, where N is batch size and t_i is the number of tokens in i-th text.

O_i is the offsets of i-th batch of text, the dimension of $O_i = 1 * N$
The dimension of embed is batch size * embedding dimension

$$output = Linear(embed)$$

$$Linear(x) = x * A^T + b, where\ b\ is\ bias$$

The dimension of output is N * C, where C is number of class. In our case, number of class is 150, embedding dimension is 300

2. Performance:

   public score 0.67667

3. The loss function:

   The cross entropy loss

$$\text{Loss}(\text{x}, \text{y}) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right)$$

   因使用了 cross entropy loss，模型並未加上 Log Softmax layer

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

   learning rate = 1e-3

   batch size = 128

   optimization algorithm = Adam,
   with learning rate 1e-3, betas = (0.9, 0.999), epos=1e-08, weight decay=0, amsgrad=False, foreach=None, maximize=False, capturable=False)

Slot tagging model

1. Describe your model

   模型包含 5 層，embedding, LSTM, hidden layer, 以及兩層 linear layer。Embedding 把詞彙轉成 GloVe 當中詞彙對應的向量，接著將向量傳入 LSTM 當中，最後是 linear layer 把 lstm 輸出的結果則會最後輸出預測的結果進行線性變換。

   Model structure

   h_0 = Embeddings(T)

   where T is a batch of text, each text contains a sequence of tokens
   Mathematically we represent T as

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ \vdots & & \ddots & \vdots \\ w_{i1} & & \cdots & w_{ij} \end{bmatrix}$$

, where $w_{i\,j}$ means a vocabulary in i-th text and j-th positsion.

h_0 is a matrix which dimension is $N \times L_i \times E$, where N is batch size, L is max text length in i-th batch and E is embedded dimension

$$h_1 = \text{LSTM}(h_0)$$

h_1 is a matrix which dimension is $N \times L_i \times H$, where H is hidden dimentsion

$$h_2 = \text{reshape}(h_1)$$

h_2 is a matrix which dimension is $N * L_i \times H$

$$h_3 = \text{linear}(h_2)$$

$$linear(x) = xA^T + bias$$

h_3 is a matrix which dimension is $N * L_i \times C$, where C is number of classes

$$output = \text{softmax}(h_3)$$

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

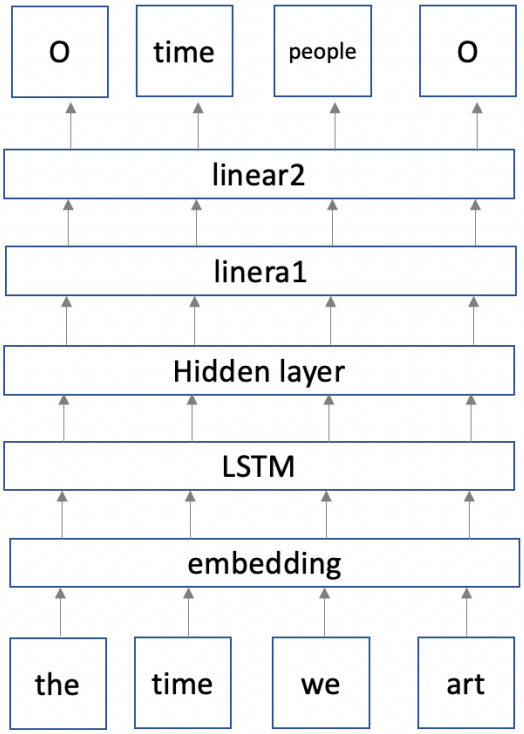


圖 3. slot model

2. performance of your model: public score 0.67667

   seqeval classification_report(scheme=IOB2, mode='strict').

	precision	recall	f1-score	support
date	0.55	0.60	0.58	205
first_name	0.82	0.71	0.76	100
last_name	0.81	0.51	0.62	77
people	0.53	0.52	0.53	237
time	0.73	0.79	0.76	217
micro avg	0.64	0.63	0.64	836
macro avg	0.69	0.63	0.65	836
weighted avg	0.65	0.63	0.64	836

Token accuracy 認定比較寬鬆，是計算分類正確 token 比例，模型的 token accuracy 約有 0.9，但如果是計算整句話都標記正確的 joint accuracy 則只有 0.67。按照 tag 類別分析，模型在預測 first name 以及 last name 的時候相對其他類別精準，precise 皆在 0.8 以上，其中 first

name 的 recall 也高，不過 last name 就相對 recall 比較低，代表雖然標記出來的 last name 準確率高，但是相對常有 last name 沒有標記到。

此外計算各個指標的 micro average、macro average，weight average 都差不多落在 0.64，代表模型不會因為特定的 token 類別數量比較多影響到判斷結果，也跟 joint accuracy0.67 非常接近。

3. The loss function you used.

   The negative log likelihood loss

$$l_{(x,y)} = L = \{l_1, \dots l_N\}^T,$$

$$l_n = -w_n x_{n,y_n}$$

$$w_c = weight[c] \times 1$$

$$l_{(x,y)} = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n$$

   x = input, y = target, w = weight, N = batch size,
   c = number of classes

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

   The optimization algorithm:

   Adam，使用跟 intent classification 一樣的設定

   Learning rate = 1e-3

   Batch size = 128

Sequence Tagging Evaluation

1. Please use seqeval to evaluate your model in Q3 on validation set and report *classification_report(scheme=IOB2, mode='strict')*.

2. Explain the differences between the evaluation method in seqeval, token accuracy, and joint accuracy.

   Joint accuracy 是測量有多少比例的句子（text）被正確的標記，需要整句話內的單詞（token）都標記正確，Token accuracy 則是計算單詞（token）被正確預測的比例。

   Token accuracy:

$$\frac{正確預測的token數量}{token總數}$$

   Joint accuracy:

$$\frac{正確預測的text數量}{text總數}$$

   Joint accuracy 還有 token accuracy 沒有特別區分 token 的類別，但是 precise, recall, F1 score 這三個分數是按照 token 的類別計算，並且區分成：

- 預測成是該 token，實際上不是的數量（TN）
- 預測該 token，實際上是的數量（TP）
- 預測不是該 token，實際上不是的數量（FN）
- 預測不是該 token，實際上是的數量（FP）

	實際 Positive	實際 Negative
預測 True	TP	TN
預測 False	FP	FN

第 i 個類別的 precise：

$$P_i = \frac{TP}{TP + FP}$$

第 i 個類別的 recall：

$$R_i = \frac{TP}{TP + FN}$$

第 i 個類別的 F1 score：

$$F_i = \frac{2 \times precise \times recall}{precise + recall}$$

如果 Precise 高，recall 低代表模型預測比較謹慎，有標記出來 token 高比例是對的，但是經常有抓不出來的狀況，反之則代表應該標記而有標記出來的比例高，但是標記出來的錯誤比例高。模型在預測 first name 以及 last name 的時候相對其他類別精準，precise 都有 0.8 以上，其中 first name 的 recall 也高，代表該抓的也有抓出來，不過 last name 就相對 recall 比較低，代表雖然抓出來的是正確的，但是經常有 last name 沒有標記到。

Micro Average, Macro Average, Weight Average 則是不分類別整體模型在 Precise, Recall, 去衡量模型在上述指標當中，整體而言的表現 Macro Average 是直接上述指標的算數平均數，每個類別權重都一樣；Weight Average 是根據該類別樣本所佔的比例賦予權重算平均， Micro 則是不分類別統計 TP, TN, FP, FN 的數量，再去計算指標。

以 Precise 為例

$$Macro\ Average\ precise = \sum_{i=1}^{n} \frac{1}{n} \times P_i$$

$$Mircro\ Average\ Precise = \sum_{i=1}^{n} \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} FP_i + \sum_{i=1}^{n} TP_i}$$

$$Macro\ Average\ precise = \sum_{i=1}^{n} \frac{1}{W_i} \times P_,$$

Q5: Compare with different configurations (1% + Bonus 1%)

1. Please try to improve your baseline method (in Q2 or Q3) with different configuration (includes but not limited to different number of layers, hidden dimension, GRU/LSTM/RNN) and EXPLAIN how does this affects your performance / speed of convergence / ...
2. Some possible BONUS tricks that you can try: multi-tasking, few-shot learning, zero-shot learning, CRF, CNN-BiLSTM

(1). 調整 laerning rate

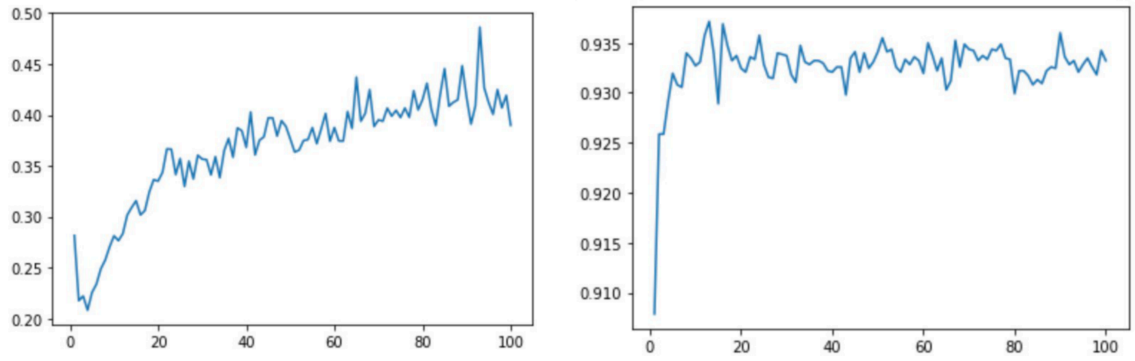


圖 3.slot 模型的 token accuracy（右）, loss, Learning rete（左）設為 1e-3

我們可以發現在訓練到約第 5 個 epoch 以後模型的 token accuracy 一直維持在約 0.9 的水平，但是 loss 卻持續上升，代表模型 over fitting

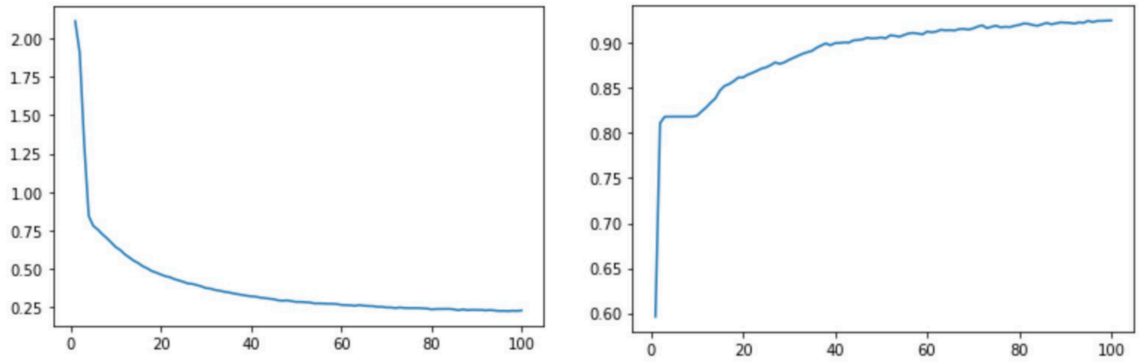


圖 4.slot 模型的 token accuracy（右）, loss, Learning rete（左）設為 1e-5

為了提升 accuracy，我們嘗試調低 learning rate，當我們把 learning rate 調為原先的百分之一以後一直訓練到第 100 epoch，loss 都不會升高，並且曲線也相對平緩許多。我們想像模型是一個在曲面上面持續滾動尋找低點的小球，當跨出的步伐大儒容易錯過低點，而當學習的步伐變小，模型比較容易貼合曲面的弧度下降。
但相對應的，學習的時間拉的更長，速度變得更慢，一直到約第 50 個 epoch 以後才到高點，而且最終的 accuracy 停留在 0.9 左右可能是落在 local minium 錯失了其他低點，反而比原本表現得更差。

(2) 疊加兩層 linear layer

因此我們把 laerning rate 調回原本的水平，在原本的模型上多疊加兩層的 linear layer 結果發現學習的曲線相較於原先變得更平滑，在前 100 個 epoch 當中也沒有越訓練 loss 越高的狀況。這可能是因為當神經網絡變得更深，模型變得更複雜，預測的效果也同時變得更好，原先的模型會在約 0.93 的水平陡升陡降，但是疊加了 linear layer 以後更穩定的維持在 0.94 的水平。

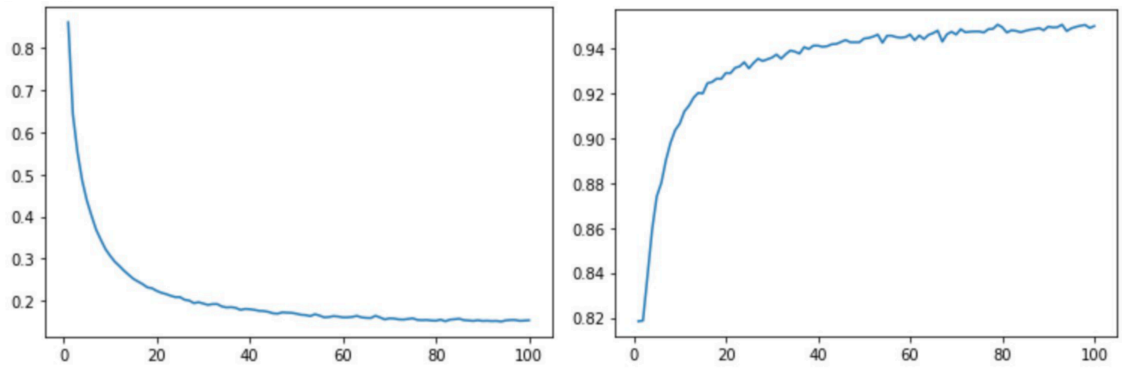


圖 5.slot 模型的 token accuracy（右）, loss, Learning rete（左）設為 1e-3, 疊加兩層 linear layer

(4) 疊加 CRF 層

最後我們嘗試疊加 CRF 層，首先最明顯的發現是訓練時間變長了。接著是 accuracy 並沒有明顯的提升，甚至還下降了，推測有可能是因為我們沒有進行 padding 而是直接把一整個 batch 的 text 放入模型當中的緣故，我們進一步嘗試加入 bidirectional=True，結果同樣也發現模型的表現下降了，這可能是因為 bidirectional 同樣也注重輸入 featrue 的順序，單句話之間彼此是不連貫的，我們卻把它拼接在一起，這可能是為什麼加入 crf 層以及 bidirectional 模型表現都無法提升的原因。

參考資料：

https://camo.githubusercontent.com/1f978d2a4a58b5c91426dc5166cbe4c629527ee902991ea49f41a59096813c7c/68747470733a2f2f6e6c702e7374616e666f72642e6564752f70726f6a656374732f676c6f76652f696d616765732f6d616e5f776f6d616e5f736d616c6c2e6a7067

https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html?highlight=negative%20log%20likelihood%20loss