Hw1 report

1. Data processing

我是使用 sample code

使用 GloVe 的 glove.840B.300B 當作 embedding

預處理 data 的部分，intents 和 slot 的部分都一樣，用 counter 型態的 words 變數和 set 型態的 intents/tags 變數取得單字和類別後，再把類別列舉出來，存成 tag2idx 和 intent2idx 的 json 檔。然後用 build_vocab 把 counter 中最常出現的 vocab_size 個字用 Vocab 的 class 存成 vocab.pkl 檔。然後把 glove 檔案開起來，拿出資料，如果有單字不在 glove 裡面，就 random 一個 300 維的向量。最後存成 embeddings.pt 當作 embedding 來用。

2. Describe your intent classification model

a.

我的 model 是先把 input 的 batch 經過

```
embed = self.embeddings(batch)
```

轉換成(batch_size, seq_len, 300]形狀的向量 embed，然後使用雙向 LSTM，

```
x,y = self.rnn(embed,None)
```

(這裡的 self.rnn 是設定成雙向 LSTM)並設定 batch_first 為

true，所以 input 的型態是(batch_size,max_len,300)，根據教學，輸出的向量會轉變成 x: (batch_size,max_len,hidden_size * 2)。因為在 train_intent 裡面需要做比對的 labels 變數的型態是(batch_size)，所以 outputs 的型態需要是(batch_size,outputs_dim)，所以我選擇先

$$x = x.\text{reshape}(\text{batch\_size}, -1)$$

這樣資訊量不會有多少流失。再 dropout，使用 linear

$$\text{nn.Dropout}(\text{dropout})$$
$$\text{nn.Linear}(\text{self.hidden\_size} * 256, \text{self.output\_dim})$$

完成維度上的轉換。就可以輸出 outputs 了。

b.

kaggle 上 public 是 0.89422，private 是 0.89688。

c.

loss function 是用 CrossEntropyLoss。

d.

optimization algorithm 是用 adam，learning rate 我沒改，是用 1e-3，batch_size 調為 100，其實是看 15000/128 沒整除不舒服所以改的。

3. Describe your slot tagging model

a.

基本上跟 intent 的 model 一樣，但有個優化。我在 intent 和 slot 部分處理輸入的句子時為了能變成 tensor 所以都會先統一成 max_len 的長度。但這樣後面多出來的資料就都會是雜訊。為了避免這樣，我在

$$\text{embed} = \text{self.embeddings(batch)}$$

後使用了

$$\text{x\_packed} = \text{pack\_padded\_sequence(embed, x\_lens, batch\_first} = \text{True, enforce\_sorted} = \text{False}$$

進行資料壓縮，無視因為統一長度而多出來的資料，然後

$$\text{output\_packed, y} = \text{self.rnn(x\_packed)}$$

$$\text{outputs, output\_lengths} = \text{pad\_packed\_sequence (output\_packed , batch\_first} = \text{True, total\_length} = 128)$$

放進雙向 LSTM，解除壓縮，並統一長度至 128。最後一樣經過

$$\text{nn.Dropout(dropout)}$$
$$\text{nn.Linear(self.hidden\_size} * 2, \text{self.output\_dim)}$$

就可以輸出 outputs 了。

b.

kaggle 上 public 是 0.77533，private 是 0.77438。

cd.

loss function 和 optimization algorithm 和其他都跟上面一樣，是 CrossEntropyLoss 跟 adam，learning rate 是 1e-3，

batch_size 是 100

4.

	precision	recall	f1-score	support
date	0.75	0.76	0.75	203
first_name	0.91	0.92	0.92	101
last_name	0.59	0.79	0.68	58
people	0.71	0.68	0.70	247
time	0.84	0.85	0.85	216
micro avg	0.77	0.78	0.78	825
macro avg	0.76	0.80	0.78	825
weighted avg	0.77	0.78	0.78	825

這個是我 train 到第 11 個 epoch 的狀況。

首先 joint accuracy 跟 token accuracy 的差別在於一個看整句一個只看對應的單字。所以通常 joint accuracy 是小於 token accuracy 的，像是我自己 train 是看 token accuracy，可以到 97%以上，但這次作業的評分，也就是 joint accuracy 是 77%，比較低。

Precision = True positive/(true positive + false negative)，是比較嚴格的計算，recall= True positive/(true positive + false positive)是比較寬鬆的估計，可以看出來我的狀況 date、last_name 和 people 是造成預測錯誤的主要原因。

5.

我曾經用 rnn 去 train intent model，我忘了結果是多少，但

記得是比 lstm 低一點。我覺得是因為畢竟是用整個句子去 predict 出結果，所以對前面內容記憶性比較好的 lstm 會比 rnn 高。我有試著對 model 加些奇怪的調整，像是用 x=x[:,-1,:]、torch.mean、logsoftmax 等等，但結果都沒有比較好。大概是因為加了前面那些東西會導致資訊量有所流失吧。