**Q1. Data processing**

使用 sample code

a. How do you tokenize the data.
以空格分隔文字，並計算每個詞在 dataset 出現的次數，選出出現頻率前 4500 名的詞（for intent classification）、前 1500 名（for slot tagging）建字典，並加上 unknown 以及 padding。

b. The pre-trained embedding you used.
使用 glove (840B tokens, 2.2M vocab, cased, 300d vectors)

**Q2. Intent classification model**

**a. Model**

分為 Encoder 以及 Decoder，Encoder 會將抽取句子的特徵向量，Decoder 會將此特徵向量轉換為每個 intent 的機率。

Given input sequence $X = \{x_0, x_1, \dots, x_t\}$

Encoder：

為 2 layers bi-LSTM，hidden state 為 256 維，dropout 為 0.2，先將 input sequence token 透過 glove embedding 轉換為 300 維的詞向量，當成 Encoder 的輸入，

$$\text{output}, (h_n, c_n) = Encoder(X, (h_0, c_0))$$

input 為 sequence glove embedding word vector，output 為每個時間點的 hidden state，$h_n$為 t = n 的 hidden state，將其最後兩層串接為 512 維當作句子的特徵向量$\hat{h}$。

$$\hat{h} = concat(h_n[-1], h_n[-2])$$

Decoder：

為 1 layer Linear layer with 0.2 dropout rate，會將 512 維特徵向量 $\hat{h}$ 轉換到 11 維 $\hat{y}$。

**b. Performance**

Dev accuracy: 0.9253

Public accuracy: 0.92311

Private accuracy: 0.91911

**c. The loss function**

使用 Cross Entropy Loss，$\text{Loss} = \text{CrossEntropyLoss}(\hat{y}, gt)$，$\hat{y}$為 Decoder output，gt 為 ground truth。

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

使用 Adam，learning rate 為 1e-3，batch size 為 128，lr scheduler 使用 OneCycleLR，在前 10% epochs 先做 warm up 避免因為一開始 learning rate 過大而提早收斂。

Q3. **Slot tagging model**

**a. Model**

Given input sequence $X = \{x_0, x_1, \dots, x_t\}$

RNN part：

為 2 layers bi-LSTM，hidden state 為 256 維，dropout 為 0.2，將 input sequence $X$ 當成 input，透過 bi-LSTM 找出詞之間的相互關係，將每個詞都會 512 維的 hidden state，將其當作新的詞向量 $V = \{v_0, v_1, \dots, v_t\}$

$$V, (h_n, c_n) = RNN(X, (h_0, c_0))$$

Tagging classifier:

為 1 layer Linear layer with 0.2 dropout rate，會將每個詞向量 $v_i$ 轉換為每個 tag 的機率。

$$\hat{y}_i = classifier(v_i)$$

預測的 tag 為 $\text{argmax}_i\{\hat{y}_i\}$

**b. Performance**

Dev joint accuracy: 0.826

Public joint accuracy: 0.82466

Private joint accuracy: 0.83065

**c. The loss function**

使用 Cross Entropy Loss，$\text{Loss} = \text{CrossEntropyLoss}(\hat{y}, gt)$，$\hat{y}$為 tagging classifier output，gt 為 ground truth。

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

使用 Adam，learning rate 為 5e-4，batch size 為 128，lr scheduler 使用 OneCycleLR，在前 10% epochs 先做 warm up 避免因為一開始 learning rate 過大而提早收斂。

**Q4. Sequence Tagging Evaluation**

```
Joint Acc: 0.8270 (827/1000)
Token Acc: 0.9692 (7648/7891)

seqeval classification report
              precision    recall  f1-score   support

        date       0.79      0.78      0.79       206
  first_name       0.98      0.91      0.94       102
   last_name       0.96      0.87      0.91        78
      people       0.76      0.73      0.75       238
        time       0.86      0.83      0.85       218

   micro avg       0.84      0.81      0.82       842
   macro avg       0.87      0.83      0.85       842
weighted avg       0.84      0.81      0.82       842
```

Joint accuracy = $\frac{\text{correct sequence predicted}}{\text{number of all sequence predicted}}$

Token accuracy = $\frac{\text{correct token predicted}}{\text{number of all token predicted}}$

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

for example:

y_true = [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

y_pred = [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

⇨ y_true = [[(MISC, 3, 6)], [(PER, 0, 2)]]

⇨ y_pred = [[(MISC, 2, 6)], [(PER. 0, 2)]]

再對每個 tag 計算 true positive(TP)，也就是每個 squence 同樣(tag, begin, end)數量，

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

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

Precision 代表預測出為正樣本的準確度，recall 代表預測能命中多少真實的正樣本，

F1 score $= \frac{2}{\frac{1}{precision}+\frac{1}{recall}}$ 為 precision 和 recall 的調和，

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

Micro avg $= \frac{\text{所有 tag 的 TP}}{\text{所有 tag 的 TP+FP(或 FN)}}$，

Macro avg 是對所有的 precision、recall 做平均，

Weighted avg 是根據 tag 數量做 weighted sum。

**Q5. Compare with different configurations**

**Compare with different Vocab size (Use the same model configurations a Q2 and Q3)**

1. **Intent classification**

| Vocab size | 3k | 3.5k | 4k | 4.5k | 5k |
|---|---|---|---|---|---|
| Dev Acc | 0.924 | 0.926 | 0.9283 | 0.9253 | 0.9227 |
| Public Acc | 0.91066 | 0.91688 | 0.91822 | 0.92311 | 0.91822 |
| Private Acc | 0.90311 | 0.92311 | 0.92622 | 0.91911 | 0.92177 |

把一些 count 為 1 的字刪掉，可以幫助 performance，因為這些不常出現的字會讓 model 比較難去訓練，如果在 testing set 很多這種單詞，反而會讓 performance 下降；但如果減少太多字彙量，會使得每個句子判斷 intent 的關鍵字消失，造成 confuse。

2. **Slot tagging**

| Vocab size | 1k | 1.5k | 2k | 3k | 4k |
|---|---|---|---|---|---|
| Dev joint acc | 0.836 | 0.826 | 0.825 | 0.812 | 0.787 |
| Public joint acc | 0.82627 | 0.82466 | 0.81554 | 0.82359 | 0.72761 |
| Private joint acc | 0.83011 | 0.83065 | 0.82100 | 0.81725 | 0.73043 |

Vocab size 如果太大會容易 overfitting，認為可能是因為字數差距太大造成一直在訓練常見的單詞，而少見的單詞難以訓練，因此把少見的單詞設為 unknown 表現會比較好。

**Others**

1. **Intent classification**

**Self-attentive Sentence Embedding**

Implement “A Structured Self-attentive Sentence Embedding”. If we use only the last hidden state to represent a sentence, we may lose some information of the sentence. To improve it, we can use the attention mechanism to extract different aspects of the sentence into multiple vector representations. In my approach, I average all the sentence representations, and feed it into decoder as Q2 part.

Detail:

Given hidden state $H = \{h_1, h_2, \dots, h_n\} \in R^{n \times 2u}$, u is hidden state dimension, and we need to train the weight $W_{S1} \in R^{d_a \times 2u}$ and $W_{S2} \in R^{r \times d_a}$. $d_a$ is a hyperparameter we can set arbitrarily, $r$ represents how many different parts to be extracted from the sentence.

The attention mechanism output $r$ vectors of weight.

$$\mathrm{A} = \mathrm{softmax}(W_{S2} \tanh(W_{S1} H^T)) \in R^{r \times n}$$

And we can get the sentence embedding by

$$\widehat{H} = mean(AH, axis = -1)$$

To avoid the attention mechanism providing the same weight for all the r hops, we need a penalization term to encourage the diversity of the weight vectors across different hops.

$$\mathrm{P} = \|AA^T - I\|_F^2$$

The diagonal element of $AA^T$ is forced to approximate to 1, which is encourages each summation vector to focus on as few number of words as possible, and all other elements to 0, which punish redundancy between two summation vector.

Performance:

We set $d_a$ to 128 dim and use the same hyperparameters as Q2

| Model | Q2 | Q2 + 4 hops | Q2 + 8 hops | Q2 + 16 hops | Q2 + 32 hops |
|---|---|---|---|---|---|
| Dev Acc | 0.9253 | 0.9307 | 0.935 | 0.9303 | 0.93 |
| Public Acc | 0.92311 | 0.92088 | 0.92488 | 0.92844 | 0.92266 |
| Private Acc | 0.91911 | 0.92755 | 0.92533 | 0.93244 | 0.92400 |

Because of the attention mechanism, the model can focus on the important part which can represent the intent of the sentence. The performance is about 0.013 higher than original model.

## 2. Slot tagging

### a. CNN-BiLSTM

Use Conv1D (kernel=5, padding=2) before feeding the sequence word vectors into biLSTM.

| Model | Bi-LSTM x2 | Conv1D x1 + Bi-LSTM x2 | Conv1D x2 + Bi-LSTM x2 |
|---|---|---|---|
| Dev Acc | 0.826 | 0.827 | 0.814 |
| Public Acc | 0.82466 | 0.82788 | 0.82520 |
| Private Acc | 0.83065 | 0.82529 | 0.82047 |

The conv can remove the noise in the sentence because it does something like weighted sum with the neighbor. However, in my experiment, it doesn't improve the performance. I think it is because my vocab size is relatively small, the noise has already been eliminated.

### b. BiLSTM-CRF

The CRF computes a conditional probability. Let y be a tag sequence and x be an input sequence of word. We compute

$$P(y|x) = \frac{exp(score(x,y))}{\sum_{y'} exp(score(x,y'))}$$

The score is determined by some log potential, and in the bi-LSTM CRF, we define two kind of potentials: emission and transition. The emission potential come from the hidden state of the Bi-LSTM at timestep $i$. And the transition score are stored as $|T| \times |T|$ matrix $\mathbf{P}$, where T is the tag set. $\mathbf{P}_{j,k}$ is the score of transitioning to tag j from tag k So we have

$$score(x,y) = \sum_i log\Psi_{EMIT}(y_i \to x_i) + log\Psi_{Tran}(y_{i-1} \to y_i)$$

$$= \sum_i h_i[y_i] + \mathbf{P}_{y_i,\ y_{i-1}}$$

The optimal function is

$$\max \ \log P(y|x) = \log \frac{\exp(score(x,y))}{\sum_{y'} \exp(score(x,y'))}$$

$$= \log \exp(score(x,y)) - \log \sum_{y'} \exp(score(x,y'))$$

$$\rightarrow \min -\log P(y|x) = -\log \exp(score(x,y)) + \log \sum_{y'} \exp(score(x,y'))$$

Performance:

Vocab size = 1.5 k

| Model | Bi-LSTM x2 | Bi-LSTM x2 + CRF |
|---|---|---|
| Dev Acc | 0.826 | 0.827 |
| Public Acc | 0.82466 | 0.81126 |
| Private Acc | 0.83065 | 0.82422 |

Vocab size = 4 k

| Model | Bi-LSTM x2 | Bi-LSTM x2 + CRF |
|---|---|---|
| Dev Acc | 0.787 | 0.783 |
| Public Acc | 0.72761 | 0.73029 |
| Private Acc | 0.73043 | 0.73311 |

BiLSTM-CRF doesn't improve the performance while vocab size is small. I think it is because the unknown token will rely more on the neighbor hidden state. However, if vocab size is large, the token will rely more on the emission score because it is hard to train the token which only appear once an epoch. Hence, the performance will be increased.

**Final leaderboard score**

Intent classification

Model: vocab size=4.5k, Bi-LSTM x2 + self attention($d_a = 128, r = 16$)

Dev score: 0.9303

Public score: 0.92844

Private score: 0.93244

Slot tagging

Model: vocab size=1.5k, Conv1D x1 + Bi-LSTM x2, without CRF

Dev score: 0.827

Public score: 0.82788

Private score: 0.82529

**Reference**


1. **A Structured Self-attentive Sentence Embedding (https://arxiv.org/abs/1703.03130)**
2. **BiLSTM-CRF**
   a. **https://pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html**
   b. **https://github.com/jidasheng/bi-lstm-crf**