Q1: Data processing

intent跟slot tagging的預處理部分我使用了助教的sample code，其步驟如下：

Intent classification

1. 從train.json跟eval.json load資料下來，以空白切割將每一個句子中出現過的字存起來
2. 建一個字典使每一個種類都對應到一個整數，將字典存到 intent2idx.json
3. 讓句子中出現的每一個字都有對應的一個整數，存到Vocab裡之後寫入vocab.pkl
4. 利用網路上的glove來建立一個tensor，第整數個元素對應到那個字的字向量，存到 embeddings.pt

Slot tagging

1. 從train.json跟eval.json load資料下來，將每一個句子中出現過的字存起來
2. 建一個字典使每一個種類都對應到一個整數，將字典存到 tag2idx.json
3. 讓句子中出現的每一個字都有對應的一個整數，存到Vocab裡之後寫入vocab.pkl
4. 利用網路上的glove來建立一個tensor，第整數個元素對應到那個字的字向量，存到 embeddings.pt

Q2: intent classification model

我的intent model是由一層RNN跟一層Linear所組成的

Overall Model

$model(I, H, N, D, O)$

- **Parameters**
  - I: input size (300)
  - H: hidden size (default = 512)
  - N: number of layers (default = 2)
  - D: dropout(default = 0.1)
  - O: output size (150)

$o = model(x, L)$

- **Input**
  - x: a padded tensor of shape (batch size(default = 128), max sequence length, features = 300)
  - L: a list where list[i] stores the sequence length(before padding) of the i_th batch
- **Output**
  - o: a tensor of shape (batch size, number of classes(150))

RNN Module

這邊我使用了雙向的GRU來當作RNN的架構

$GRU(I, H, N, D)$

- **Parameters**
    - I: input size (300)
    - H: hidden size (default = 512)
    - N: number of layers (default = 2)
    - D: dropout(default = 0.1)

$o_{gru}, h_t = GRU(x_{pack})$

- **Input**
    - x_{pack}: a pack sequence transformed from previous x and L

      (batch size, sequence length = $L[i]$, features = 300)
- **Output**
    - o_{gru}: a pack sequence
    - h_t: a tensor storing final states of each hidden layer(number of layers * 2, batch size, hidden size)

Linear Layer

由於GRU的output為一個pack sequence，無法餵進pytorch的linear model，所以先將pack sequence pad過，並且將各個batch最後一個output拿出來丟給linear層將300維打到種類數150維，最後再對每一個batch的150維取argmax，當作我預測的種類

$Linear(H, O)$

- **Parameters**
    - H: hidden size (512 * 2)
    - O: output size (150)

$o_{linear} = Linear(x_{pack\_pad})$

- **Input**
    - x_{pack_pad}: a tensor of shape (batch size, hidden size)
- **Output**
    - o_{linear}: a tensor of shape (batch size, number of classes(150))

Training

- **Loss**: CrossEntropyLoss
- **Optimizer**: Adam(learning rate = 1e-3)
- **Batch size**: 128
- **Epoch**: 100
- **hidden size**: 512
- **dropout**: 0.1

- **number of layers**: 2

Performance

Public score: 0.92133 Private score: 0.92044

Q3: slot tagging model

我的slot model一樣是由一層RNN跟一層Linear所組成的

Overall Model

$model(I, H, N, D, O)$

- **Parameters**
  - I: input size (300)
  - H: hidden size (default = 512)
  - N: number of layers (default = 2)
  - D: dropout(default = 0.1)
  - O: output size (150)

$o = model(x, L)$

- **Input**
  - x: a padded tensor of shape (batch size(default = 128), max sequence length, features = 300)
  - L: a list where list[i] stores the sequence length(before padding) of the i_th batch
- **Output**
  - o: a padded tensor of shape (batch size, max sequence length, number of tags(9))

RNN Module

這邊我使用了雙向的GRU來當作RNN的架構

$GRU(I, H, N, D)$

- **Parameters**
  - I: input size (300)
  - H: hidden size (default = 512)
  - N: number of layers (default = 2)
  - D: dropout(default = 0.1)

$o_{gru}, h_t = GRU(x_{pack})$

- **Input**
  - x_{pack}: a pack sequence transformed from previous x and L

    (batch size, sequence length, features = 300)
- **Output**
  - o_{gru}: a pack sequence
  - h_t: a tensor storing final states of each hidden layer(number of layers * 2, batch size, hidden size)

Linear Layer

為了不要讓o_{gru}用來padding的0向量影響Linear model的更新方向，我選擇一個batch一個batch的丟進linear層，再padding起來得到一形狀為(batch size, max sequence, number of tags)的tensor當做output，最後一樣將每一個batch中的最後一維取argmax當作我對每一個字所預測的tag。

$$Linear(H, O)$$

- **Parameters**
    - H: hidden size (512 * 2)
    - O: output size (9)

$$o_{linear} = Linear(x_{pack\_pad})$$

- **Input**
    - x_{pack_pad}: a tensor of shape (batch size, hidden size)
- **Output**
    - o_{linear}: a tensor of shape (batch size, sequence length, number of tags(9))

Training

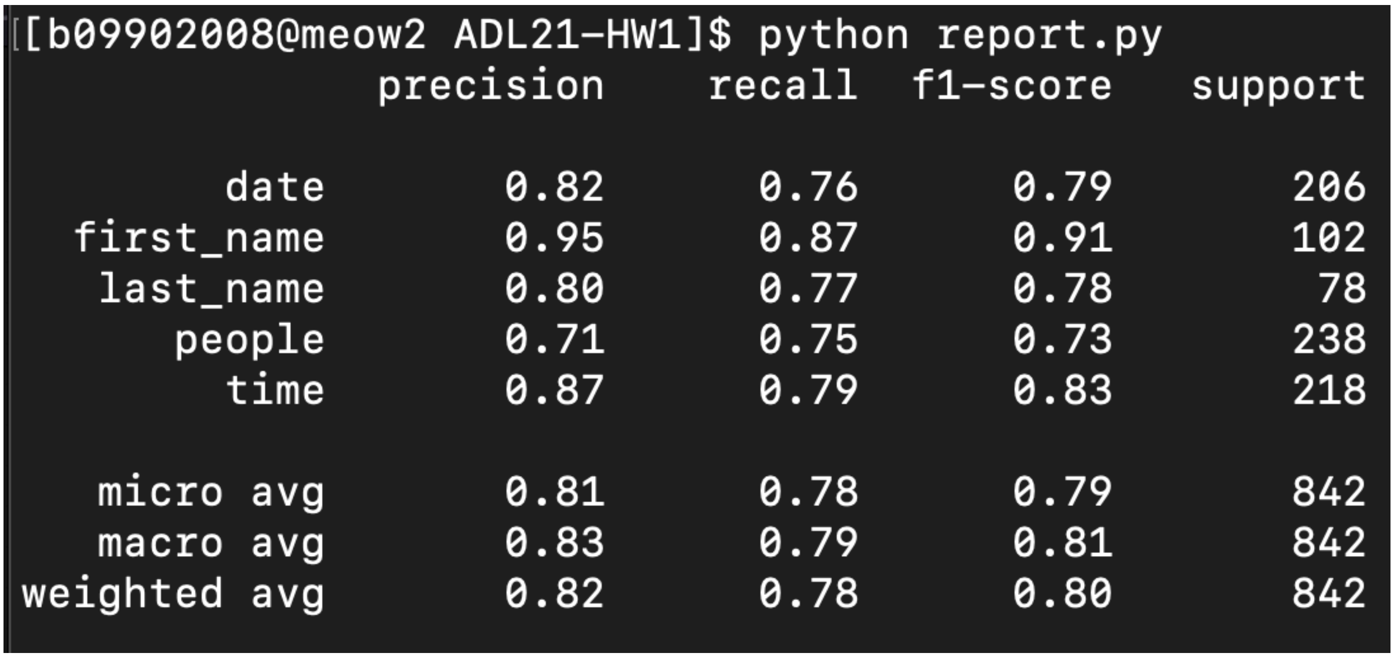
- **Loss**: CrossEntropyLoss
- **Optimizer**: Adam(learning rate = 1e-3)
- **Batch size**: 128
- **Epoch**: 100
- **hidden size**: 512
- **dropout**: 0.2
- **number of layers**: 2

Performance

Public score: 0.78498 Private score: 0.78188

Q4 Sequence Tagging Evaluation

Output of seqeval



```
[b09902008@meow2 ADL21-HW1]$ python report.py
              precision    recall  f1-score   support

        date       0.82      0.76      0.79       206
  first_name       0.95      0.87      0.91       102
   last_name       0.80      0.77      0.78        78
      people       0.71      0.75      0.73       238
        time       0.87      0.79      0.83       218

   micro avg       0.81      0.78      0.79       842
   macro avg       0.83      0.79      0.81       842
weighted avg       0.82      0.78      0.80       842
```

Differences between evaluation method

- $\text{Recall} = \frac{TP}{TP+FN}$
- $\text{Precision} = \frac{TP}{TP+FP}$
- $\text{F1-score} = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}}$
- Token Accuracy : 每個字獨立算正確率 $\frac{\text{number of wrong words}}{\text{number of words}}$
- Joint Accuracy : 只有當句子全對時才算對 $\frac{\text{number of wrong sentences}}{\text{number of sentences}}$

可以看得出來模型的precision比recall好，代表我的模型猜得很準但不太常猜

Q5 Different configurations

1. Different RNN

在intent的時候，我先測試過了各種RNN的架構(BiRNN, BiLSTM, BiGRU)，發現BiGRU在score board上的效果最好，因此之後的RNN模型都用雙向GRU(模型的參數都為預設)

Intent classification

RNN	Public score
BiRNN	0.88000
BiLSTM	0.90044
BiGRU	0.91555

2. Better Initialization

為了避免Trainging多次但還是逃不出local minimum和梯度消失或爆炸的情況，我試了兩種hidden layer的初始化方法，一個是隨機生成hidden layer的參數，一個是正交化hidden layer的參數(模型的參數都為預設)

Intent classification

Models with different intialization	Train Loss	Train Acc	Val Loss	Val Acc
original model	7.91673e-06	1	0.46116	0.9277
randomized model	9.88e-06	1	0.39439	0.931667
orthogonalized model	2.7087e-05	1	0.40716	0.9307

Slot tagging

Models with different intialization	Train Loss	Train Joint Acc	Val Loss	Val Joint Acc
original model	1.494771	0.9001	1.631906	0.791
randomized model	1.494697	0.9369	1.659072	0.761
orthogonalized model	1.494885	0.9420	1.646730	0.771

可以發現Intent比較適合random的初始化，slot則比較適合使用pytorch預設的初始化

3. Attention Layer

在處理intent model的時候，我嘗試在RNN跟Linear layer之間多加一層Attention Layer來提高model的表現

Intent classification

Models	Train Loss	Train Acc	Val Loss	Val Acc
original model	3.122747362810468e-06	1	0.453627	0.9313
model with attention	1.538460061687979e-05	1	0.405342	0.9267

但可以發現加入Attention Layer後不一定會提高模型的表現

4. Different parameters

以下是組合過不同的hidden size, number of layers, dropout的結果

調參的順序是先從dropout跟hidden size做組合，再加深RNN的深度並依overfitting的程度來提高dropout(因為有事先試過加深到4層，但ACC表現都不佳，所以只加深到第三層)

Intent classification

normal

(hidden size, number of layers, dropout)	Train Loss	Train Acc	Val Loss	Val Acc
512, 2, 0.1	4.108875e-06	1	0.438470	0.9303
512, 2, 0.2	3.875268e-06	1	0.438183	0.9303
512, 2, 0.4	3.264969e-03	0.9993	0.400597	0.9283
1024, 2, 0.1	1.866838e-06	1	0.455507	0.9343
1024, 2, 0.2	1.364709e-06	1	0.488672	0.9310
1024, 2, 0.4	2.793065e-06	1	0.566375	0.9253
1024, 3, 0.1	1.143829e-02	0.9971	0.442984	0.9243
1024, 3, 0.4	1.474726e-02	0.9961	0.477564	0.9183

with Attention

(hidden size, number of layers, dropout)	Train Loss	Train Acc	Val Loss	Val Acc
512, 2, 0.1	3.82496e-04	0.9999	0.348215	0.9307
512, 2, 0.2	1.66296e-04	1	0.380701	0.9290
512, 2, 0.4	8.903035e-04	0.9997	0.459864	0.9283
1024, 2, 0.1	3.022740e-06	1	0.427086	0.9263
1024, 2, 0.2	6.553047e-05	1	0.410432	0.9207
1024, 2, 0.4	9.128111e-04	0.9998	0.432570	0.9263
512, 3, 0.1	5.305901e-06	1	0.532175	0.9173
512, 3, 0.4	1.379173e-03	0.9997	0.637830	0.9167
512. 3. 0.6	1.823423e-02	0.9939	0.644460	0.9150
512, 3, 0.8	5.618098e-02	0.9817	0.670772	0.9073

Slot tagging

(hidden size, number of layers, dropout)	Train Loss	Train Joint Acc	Val Loss	Val Joint Acc
512, 2, 0.1	1.495781	0.951546	1.660643	0.778
512, 2, 0.2	1.494027	0.949199	1.648562	0.791
512, 2, 0.4	1.495809	0.935946	1.640444	0.783
1024, 2, 0.1	1.493250	0.855743	1.680521	0.685
1024, 2, 0.2	1.495025	0.657096	1.670477	0.605
1024, 2, 0.4	1.492191	0.778990	1.655778	0.677
512, 3, 0.2	1.494864	0.880315	1.667004	0.756
512, 3, 0.5	1.494854	0.853948	1.668078	0.747

總結來看可以得知，單純地讓RNN加深加廣是沒有用的，反而會降低模型的表現，且就算相對應地提高dropout避免overfitting，只會使模型表現更糟，因此最後模型的參數為intent: 不加Attention(512, 2, 0.1), slot: (512, 2, 0.2)