ADL HW1 @NTU, 2021 spring

B06902135 資工三 蔡宜倫

1. Data processing (2%)

- **Describe how do you use the data for intent_cls.sh, slot_tag.sh:**
  - **How do you tokenize the data.**

    我將intent和slot的train.json和eval.json讀入，並特別處理屬於['intent']和['text']的部分，得到一個table是intent2idx，用於將150種intent label好，另外text則建一個Vocab的class（在utils.py），一樣也是將每一個text裡出現過的vocabulary標號存起來，用於之後的查表，也就是在處理intent和slot的tasks時，從token轉成id，並且從2開始，0為`PAD`1為`UNK`，用於處理padding和若testing時查表不存在的情況。考慮到效率的問題我的想法是只存10000個比較常見的vocabulary，因為較少見的字在slot tagging task比較有機會被分到O，而在intent classification通常intent可以由較常見的字得到好的representation，但後來發現在兩個task的data中都沒有到10000個字，所以這個考慮是多餘的。
  - **The pre-trained embedding you used.**

    我使用GloVe作為word Embedding(Glove.840B.300d)，這是一種利用co-occurrence probability的ratio得到word embedding的方法。

2. Describe your intent classification model. (2%)

- **Your model**

```
gClassifier(
(embed): Embedding(6491, 300)
(gru): GRU(300, 512, num_layers=2, batch_first=True,
directional=True)
(classifier): Sequential(
  (0): Dropout(p=0.5, inplace=False)
  (1): Linear(in_features=2048, out_features=150, bias=True)
  (2): Sigmoid()
)

```

  - $e_i = Embedding(s_i)$ 代表將每一個token代表的word餵進GloVe的embedding得到 e_i。

- $o_t, h_t = GRU(e_t, h_{t-1}, o_{t-1})$ 代表將e_t餵進去得到o_t output features，h代表hidden layer。
- 在GRU之後，Linear layer是由Dropout 50%、一層全連接層和Sigmoid組成，詳細的input dimension和output dimension寫在上表。
- $result = Classifier(o)$，而result為一個$[batchsize \times 150]$的矩陣，判斷每一個sequence屬於哪一個intent。

- **Performance of your model.(public score on kaggle)**
  - 我的best public score是`0.91377`。
- **The loss function you used.**
  - 我使用`nn.CrossEntropyLoss()`作為loss function。
- **The optimization algorithm (e.g. Adam), learning rate and batch size.**
  - Optimizer： `optim.Adam()`。
  - Learning rate： 起始是`1e-3`，並且加上schedular，使用`torch.optim.lr_scheduler.ReduceLROnPlateau()`調整learning rate。
  - Batch size：試過64, 128, 256，最後performance最好為`64`。

3. Describe your slot tagging model. (2%)

- **Your model**

```
tClassifier(
embed): Embedding(4117, 300)
gru): GRU(300, 512, num_layers=2, batch_first=True,
irectional=True)
classifier): Sequential(
 (0): Dropout(p=0.5, inplace=False)
 (1): Linear(in_features=1024, out_features=1024, bias=True)
 (2): Sigmoid()
 (3): Dropout(p=0.5, inplace=False)
 (4): Linear(in_features=1024, out_features=9, bias=True)


```

  - $e_i = Embedding(s_i)$ 代表將每一個token代表的word餵進GloVe的embedding得到e_i。
  - $o_t, h_t = GRU(e_t, h_{t-1}, o_{t-1})$ 代表將e_t餵進去得到o_t output features，h代表hidden layer。
  - 在GRU之後，Linear layer是由Dropout 50%、一層全連接層和Sigmoid，再接到Dropout 50%並由fully connected layer輸出，詳細的input dimension和output dimension寫在上表。

- $result = Classifier(o)$，而result為一個$[batchsize \times seqLen \times 9]$的矩陣，判斷每一個sequence的token屬於哪一個slot。

- **Performance of your model.(public score on kaggle)**
  - 我的best public score是`0.78552`。
- **The loss function you used.**
  - 我使用`nn.CrossEntropyLoss()`作為loss function，有試過用CRF但效果不彰，因此最後沒有使用，詳細比較在第五題。
- **The optimization algorithm (e.g. Adam), learning rate and batch size.**
  - Optimizer： `optim.Adam()`。
  - Learning rate： 起始是`1e-3`，並且加上schedular，使用`torch.optim.lr_scheduler.ReduceLROnPlateau()`調整learning rate。
  - Batch size：從256, 128...16, 8，後來發現最好performance為`16`。

4. Sequence Tagging Evaluation (2%)

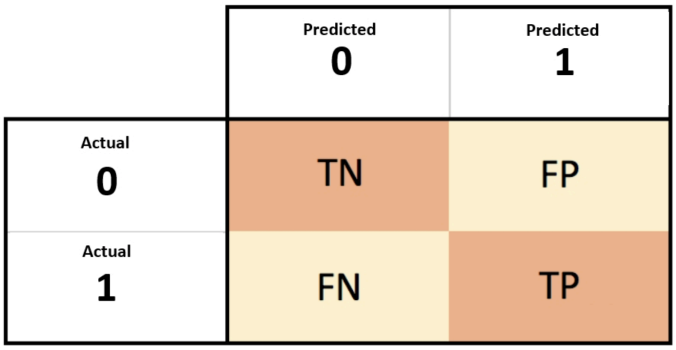
- **Please use seqeval to evaluate your model in Q3 on validation set and report .** `*classification_report(schema=IOB2, mode='strict')*`

```
           precision    recall  f1-score   support

     date       0.88      0.94      0.91       206
irst_name       0.82      0.97      0.89       102
last_name       1.00      0.82      0.90        78
   people       0.91      0.89      0.90       238
     time       0.94      0.97      0.95       218

nicro avg       0.90      0.93      0.92       842
nacro avg       0.91      0.92      0.91       842
ghted avg       0.91      0.93      0.92       842
```

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



- 以上的每一個entity都會根據True Positive(TP), False Negative(FN), TN, FP建構出自己的一個confusion matrix。並且根據以下公式算出precision, recall和 f1-score：
  - $Precision = \frac{TP}{TP+FP}$，代表在所有預測為positive中，有多少實際為positive。
  - $Recall = \frac{TP}{TP+FN}$，代表在所有實際為positive的樣本當中，能夠預測positive的比例。
  - $f_1\,score = \frac{2 \times Precision \times Recall}{Precision + Recall}$，以上兩者的調和平均數。
- 假設model的Precision高但是Recall低，代表這個模型比較謹慎，儘管沒辦法召回太多的entity，但只要抓出來就幾乎都是正確的；相反的，若model的Recall高而Precision低，代表可以抓出幾乎所有的entity，但篩選較寬鬆。但是若只看這兩個指標，有可能會有太極端的結果，前者可能認為slot不屬於任何entity，後者錯誤率太高，因此使用f-1 score作為更general評估model表現的標準。
- 至於token accuracy和 joint accuracy，前者是對於每一個sequence裡的每一個word對ground truth做比對，後者則是每一個sequence要完全符合ground truth才算正確，我在`seqVal.py`算了這些分數：
  - $Token\ acc = \frac{correct\ tokens}{total\ tokens}$，我的model在validation set上取得了`0.986`。
  - $Joint\ acc = \frac{correct\ sequence}{total\ sequence}$，我的model在validation set上取得了`0.825`。
- Seqeval 會對每一個entity的performance做評分，相對的，token accuracy和joint accuracy只會對model的overall performance 做評分，因此若想對model有更好的理解，可以使用seqeval 的metric。

5. Compare with different configurations (1% + Bonus 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 / ...**
  - **Intent Classification:**
    - 在決定一開始的參數時我做了這些嘗試，其中dropout=0.2，lr=1e-3，最後發現在batch size = 64時最好。

Batch size	**256**	**128**	**64**	**32**
Kaggle	0.87688	0.88355	0.88977	0.86666

    - 接下來我調整lr和dropout rate，發現在dropout 50%時有最好的結果。

lr	**0.0015**	**0.001**	**0.0005**	**0.001, DO=0.2**	**0.001, DO=0.5**
Kaggle	0.86666	0.88977	0.88711	0.88977	0.90488

        - 另外，我嘗試過更改embedding需不需要隨著我們的data去fine tune，發現可能是因為data量太少，fine tune的結果不如預期，甚至造成score降低，因此最後決定將embedding fix住。
    - **Slot Tagging:**
        - 在決定一開始的參數時我做了這些嘗試，其中dropout=0.5，lr=1e-3，最後發現在batch size = 16時最好。

Batch size	**128**	**64**	**32**	**16**
Kaggle	0.65147	0.67453	0.69865	0.74304

        - 接下來我改了hidden layer的維度，在512時有最好的結果。

Hidden layer	**256**	**512**	**1024**
Kaggle	0.73149	0.74304	0.72255

        - 調整了classifier FC層，我嘗試從一層改成兩層，dropout用0.5，score從0.74304.進步到77801。
        - 最一開始我是用LSTM，後來嘗試用了GRU，每個epoch速度上加快了許多，並且在kaggle上進步了1%。
        - GRU的layer設定2層，改成一層和三層performance都下降許多，尤其三層的訓練時間多很多但反而成績很低就很幹。
- **Some possible BONUS tricks that you can try: multi-tasking, few-shot learning, zero-shot learning, CRF, CNN-BiLSTM**
    - **Intent Classification:**
        - 我有做過Semi-supervised，我把所有的testing data都在每個epoch去做預測，並設定threshold=0.8，只要通過softmax有某一個intent高於閾值，我就加進去一起訓練，但可能是因為unlabeld data並沒有很多，或是本來就已經太fit在training data上了，並沒有取得比較好的結果，從0.88977退步到0.88355。
    - **Slot Tagging:**
        - 在`train_slotCRF.py`我用了CRF層作為我最後輸出層，並取代CrossEntropyLoss以之來算model的loss，得到的結果是在一開始valid acc會上升地比原本快很多，但最後反而開始下降，並且由於多通過一個model訓練時間變長許多，不斷震盪，我並沒有用CRF取得較好的成果。在嘗試CRF的過程中，我調整過batch size，並在發現16有最好的表現時調整了兩層FC層中間的activation function，我嘗試了sigmoid和PReLU。

Batch size	**8**	**16**	**32**	**16, Sigmoid**	**16, PReLU**
Kaggle	0.74959	0.77319	0.76782	0.77801	0.75978