ADL Hw1

R11944014 戴靖婷

tags: ADL Python Report

Q1: Data Processing

- Describe how do you use the data:
  a. How do you tokenize the data.
  - I made use of the sample code.
  - I tokenized “text” in the intent data via split function by the space. And I converted them into index and saved them in the file intent2idx.json / tag2idx.json.
  - Then, I used the function, build_vocab, to build the most common words as Vocab and saved in vocab.pkl.
    b. The pre-trained embedding you used.
  - I used **GloVe**(840B tokens, 2.2M vocab, cased, 300d vectors).
    - 840B: 840 billion tokens.
    - 300d: a word is represented as a 300 dim vector.
  - Then I did word embedding by converting the vocabulary words into 300d vectors via GloVe and saved in embeddings.pt.

Q2: Intent classification

- Describe your intent classification model.
  a. your model
  - I used GRU as the main model and followed with a Linear layer for classification.
    - out,h=GRU(inputs)
      - hidden size = 512
      - num layers = 2
      - dropout = 0.1
      - bidirectional = True
    - out=Linear(GRU_out)
      - input size:
        - if bidirectional = True: encoder_output_size = 2 * hidden_size
        - else: encoder_output_size = hidden_size
      - output size: num class = 150

b. performance of your model.

- public score = 0.89600
  - did not save the model with public score = 0.90177

c. the loss function you used.

- loss=CrossEntropyLoss(y_pred,y_true)
  - y_pred: predictions from the model
    - size: (batch_size,num_class)
  - y_true: real target y
    - size: (batch_size), turn into one-hot encoding

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

- optimizer=Adam(lr=1e−3,batch_size=32)

Q3: Slot Tagging

- Describe your slot tagging model.
  a. your model

- I used GRU as the main model and followed with a Linear layer for classification.
  - out,h=GRU(inputs)
    - hidden size = 512
    - num layers = 2
    - dropout = 0.1
    - bidirectional = True
  - out=Linear(GRU_out)
    - input size:
      - if bidirectional = True: encoder_output_size = 2 * hidden_size
      - else: encoder_output_size = hidden_size
    - output size: max_len = 128, num class = 9

b. performance of your model.

- public score = 0.75603

c. the loss function you used.

- loss=CrossEntropyLoss(y_pred_resize,y_true_resize)
  - y_pred_resize: predictions from the model
    - size: (batch_size∗max_len,num_class), flatten the first (batch_size) and the second (max_len) dimensions to tag for each word

- y_true_resize: real target y
  - size: (batch_size∗max_len), flatten the first (batch_size) and the second (max_len) dimensions to tag for each word, and turn into one-hot encoding

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

- optimizer=Adam(lr=1e−3,batch_size=16)

Q4: Sequence Tagging Evaluation

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

```
Joint accuracy: 0.787
Token accuracy: 0.964263084526676
Classification report:
              precision    recall  f1-score   support

        date       0.76      0.70      0.73       206
  first_name       0.90      0.88      0.89       102
   last_name       0.85      0.68      0.76        78
      people       0.72      0.73      0.72       238
        time       0.84      0.87      0.86       218

   micro avg       0.79      0.77      0.78       842
   macro avg       0.81      0.77      0.79       842
weighted avg       0.79      0.77      0.78       842
```

- Explain the differences between the evaluation method in seqeval, token accuracy, and joint accuracy.
  - $Joint\ Accuracy = \frac{Num\ Predicted\ Correctly\ Samples}{Num\ Total\ Samples}$:
    - The accuracy of a sample with all tokens predicted correctly.
  - $Token\ Accuracy = \frac{Num\ Predicted\ Correctly\ Tokens}{Num\ Total\ Tokens}$
    - The accuracy of every predicted token.
  - $Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$
    - The ratio of number of predicted true correctly tags over the number of all predicted true tags.
  - $Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$
    - The ratio of number of all true positive tags over the number of predicted true tags.
  - $F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$
  - $Support = \frac{True\ Positives}{True\ Positives + False\ Negatives}$
    - The number of actual occurrences tags of the class.

Q5: Compare with different configurations

- Please try to improve your baseline method 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 / ...

- The results are the validation accuracy with the default settings same as the model descriptions within 20 epochs.

- I have tried to add one or two more Linear layers and use ReLU before passing the inputs.
  - The accuracy with ReLU is slightly higher, but there were no significant difference between them in Intent Classification.
  - However, the results in test data (kaggle) were worse than the one without ReLU. It was because of overfitting by the awarness of high validation accuracy but low test accuracy.
Public Test	**w/o ReLU**	**w/ ReLU**
Intent	0.9050	0.9043
Tagging	0.7960	0.7980

- Besides, I changed the number of layers. However, as the num_layers increase, it converges more slowly compared to num_layers = 2.
  - In intent task with num_layers = 3, it took 20 epochs to reach validation accuracy = 0.87, which might sometimes be the first epoch result of num_layers = 2.
  - In tagging task with num_layers = 3, though the accuracy is higher, it might be overfitting. Therefore, I still chose the model with num_layers = 2 for efficiency.
Validation	**num_layers = 2**	**num_layers = 3**	**num_layers = 1**
Intent	0.9050	0.8677	0.9057
Tagging	0.7960	0.8090	0.7790

- Moreover, I tried many different learning rate.
  - When the learning rate is small, it converges super slow because it updates the parameters only a little each epoch. Within 20 epochs, the accuracy is still improving, which would reach a better result if trained with more epochs.
  - When the learning rate is large, it may updates too much which leads to missing the optimize. In my trial, the accuracy does not improve within 20 epochs.

Validation	lr=1e-3	lr=1e-4	lr=1e-2
Intent	0.9050	0.2740	0.4683
Tagging	0.7960	0.7660	0.5830

- Also, I adjusted the number of hidden size.
  - The running time increases when the hidden size is larger, and vice versa.
  - The accuracy of those smaller number of hidden size is lower. In my opinion, if we want to maintain the same complexity of the model, when the number of hidden size reduces, the model layers should be increased.

Validation	hidden_size = 512	hidden_size = 256	hidden_size = 128
Intent	0.9050	0.8793	0.8480
Tagging	0.7960	0.7680	0.7910