ADL2022-Fall Homework #1 report

電信碩二 R09942171 黃繼綸

Q1: Data processing (2%)

I used sample code to process the data

A. How do you tokenize the data.

First, compute how often each word appears in the dataset. In the intent detection and slot tagging tasks, I select the most common words within 10,000 vocab to create the dictionary and add <UNK> and <PAD> tokens to it.

B. The pre-trained embedding you used.

I used the pretrained glove with 840B tokens, 2.2M vocab, and 300-dim vectors.

Q2: Describe your intent classification model. (2%)

A. Your model

The model consists of two parts, an encoder and a classifier, where the encoder will extract the feature from the input sentence, and the classifier will take this extracted representation as input and transform it into a probability of each intent.

Encoder:

The input token sequence (x) will first pass through the pretrained glove embedding layer, transforming the input token sequence into a 300-dim vector (x_{vec}). Such representation (x_{vec}) will be input to the GRU as follows:

$$x_{vec} = embedding(x)$$

$$x_{encode}, (h_n, c_n) = GRU(x_{vec}, (h_0, c_0)),$$

Notice that the embedding layer will be fixed during training, which can prevent overfitting.

Parameters for GRU are as follows:

- Bidirectional
- Hidden_size: 512-dim
- Dropout: 0.6
- Num_layers: 2

Classifier:

The output of the encoder is with the shape of (batch, seq_len, hidden_size), I took the last layer of seq_len as the classifier's input. The classifier is a 1-layer linear layer with a dropout rate of 0.6, transforming the extracted representation (x_{encode}) into 150-dim intent categories.

$$\boldsymbol{y = classifier(x_{encode}[:, -1, :])}$$

B. Performance of your model

The public score of my model is 0.89866.

C. The loss function you used.

I used the CrossEntropyLoss to conduct loss computation.

$$Loss = CrossEntropyLoss(\hat{y}, y),$$

where $\hat{y}$ denotes the predicted results, and y denotes the ground-truth intent category.

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

I used AdamW as optimizer with learning 0.001, batch size 64. And I used LinearWarmupCosineAnnealing as learning rate scheduler, warmup within the first 5 epochs to prevent the learning rate from being too large.

Q3: Describe your slot tagging model. (2%)

A. Your model

The model consists of two parts, an encoder and a decoder, where the encoder will extract the feature from the input sentence, and the decoder will take this extracted representation as input and transform the sequence of token representation into a probability of each tagging category.

Encoder:

The input token sequence (x) will first pass through the pretrained glove embedding layer, transforming the input token sequence into a 300-dim vector (x_{vec}). Such representation (x_{vec}) will be input to the GRU as follows:

$$x_{vec} = embedding(x)$$
$$x_{encode}, (h_n, c_n) = GRU(x_{vec}, (h_0, c_0)),$$

Notice that the embedding layer will be fixed during training, which can prevent overfitting.

Parameters for GRU are as follows:

- Bidirectional
- Hidden_size: 512-dim
- Dropout: 0.6
- Num_layers: 2

Decoder:

The output of the encoder is with the shape of (batch, seq_len, hidden_size), I took the entire seq_len as the decoder's input. The classifier is a 1-layer linear layer with a dropout rate of 0.6, transforming the extracted representation (x_{encode}) into 9-dim intent categories.

$$\boldsymbol{y = decoder(x_{encode})}$$

B. Performance of your model

The public score of my model is 0.80160.

C. The loss function you used.

I used the CrossEntropyLoss to conduct loss computation.

$$Loss = CrossEntropyLoss(\hat{y}, y),$$

where $\hat{y}$ denotes the predicted results, and y denotes the ground-truth intent category.

Notice that the loss computed only on the same length of input token sequences instead of the entire sequence length (after padding).

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

I used AdamW as optimizer with learning 0.0007, batch size 64. And I used LinearWarmupCosineAnnealing as learning rate scheduler, warmup within the first 5 epochs to prevent the learning rate from being too large.

Q4: Sequence Tagging Evaluation (2%)

Evaluation method in seqeval:

Seqeval will separate each sequence's predicted result and ground-truth sequence into (tag, begin, end).

Precision: the accuracy of predicting the positive samples

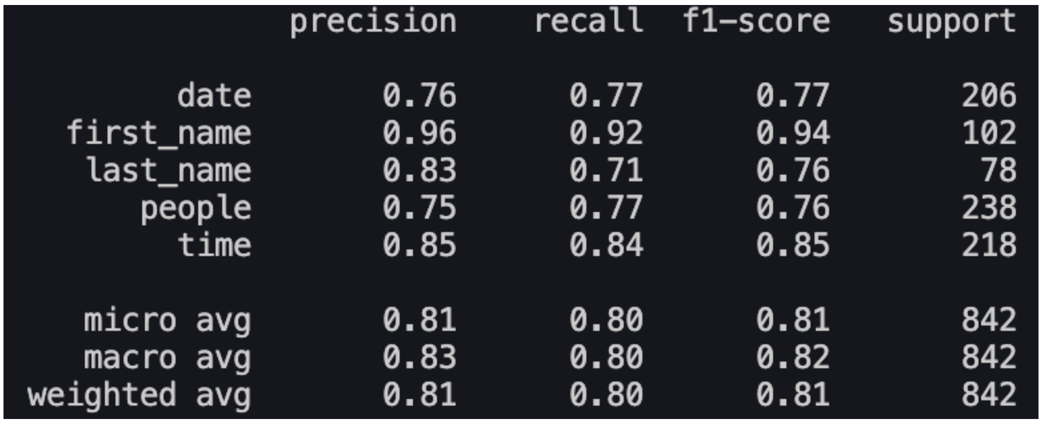
Recall: how many actual samples the model can hit

F1-score: Harmonic mean of precision and recall

Macro avg: Average over precision and recall

Micro avg: $\frac{TP\ forall\ tags}{All\ tags}$

Weighted avg: weighted sum according to the number of supports



```
              precision    recall  f1-score   support

        date       0.76      0.77      0.77       206
  first_name       0.96      0.92      0.94       102
   last_name       0.83      0.71      0.76        78
      people       0.75      0.77      0.76       238
        time       0.85      0.84      0.85       218

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

Token accuracy: 96.718% (7632 / 7891)

$$TokenACC = \frac{Number\ of\ correct\ predicted\ tokens}{Number\ of\ tokens}$$

Joint accuracy: 81.200% (812 / 1000)

$$JoinACC = \frac{Number\ of\ correct\ predicted\ sequences}{Number\ of\ sequences}$$

Q5: Compare with different configurations (1% + Bonus 1%)

Intent classification

In this task, I try some methods as follows:

- Different hidden size of GRU (others are the same):

Hidden size	Validation score	Public score
512	0.91200	0.89866
384	0.90967	---
256	0.88700	---

- Different maximum sequence length (others are the same):

Seq_len	Validation score	Public score
48	0.91200	0.89866
64	0.89433	---
32	0.91067	---

- Fix embedding or not (others are the same):

Fix embedding	Validation score	Public score
True	0.91200	0.89866
False	0.90000	---

From these experiments, we can find that the fix embedding or not is a significant action. If not fixed, the model will be easier to overfit.

Slot tagging

In this task, I try some methods as follows:

- Different hidden size of GRU (others are the same):

Hidden size	Validation score	Public score
512	0.81200	0.80160
384	0.82600	0.79195
256	0.80700	---

- Different maximum sequence lengths (others are the same):

Seq_len	Validation score	Public score
48	0.81200	0.80160

64	0.80300	---

- Fix embedding or not (others are the same):

Fix embedding	Validation score	Public score
True	0.81200	0.80160
False	0.79500	---

- Different model architectures (others are the same)

Model architecture	Validation score	Public score
GRU	0.81200	0.80160
LSTM	0.81600	0.77479

In this section, the experiments are almost the same in the intent classification. I did an additional experiment with the model architecture (GRU v.s. LSTM). From this experiment, although LSTM could get better performance in the validation set, but the public score is lower than GRU. Therefore, in this dataset, fewer parameters are better to prevent overfitting.

Others

I implemented the transformer model to compare it with the RNN-based model. The experimental results are as follows:

Intent classification

architecture	Validation score	Public score
GRU	0.91200	0.89866
Transformer	0.90633	0.89644

Slot Tagging

architecture	Validation score	Public score
GRU	0.81200	0.80160
Transformer	0.80000	0.77962

Although the transformer-based model has an attention mechanism that can weight the entire sequence and select which words in the sequence are the most critical to the output. In this experiment, however, the RNN-based method has better performance than the transformer-based method. I guessed that the reason is that due to the small dataset for this homework, the transformer-based method cannot effectively learn very well.