2022 fall ADL HW1 Report

R11922A15 張仲喆

Q1: Data processing

1. Describe how do you use the data.

a. Tokenize data.

I tokenize the intent classification data with the following method. First, I remove all punctuation marks. Next, I split the data using ".split()" like how we split a string. Finally, I got the dataset with data as lists of words.

b. Pre-trained embedding

The pretrained-embedding that I use on both problems is the same as the sample code, Glove.

2. Explain what it does.

Tokenization:

In NLP problems we consider the relation between words. In this case, we cannot process the words then it's still present as sentences. So we split the sentence into single words which is more reasonable.

Embedding:

The words which are in text format couldn't show characteristics between themselves and other words. Embed word to word-vector which is much more reasonable for ML and NLP problems and presents much more information than original textual data.

Q2: Describe intent cls model

a. Model

(Ignore batch_size)
128, 300 = Glove_embedding(128)
2048, 300 = LSTM(300, 1024, dropout=0.3, layer=1, bidirectional=True)
2048 = Max(dim=1)
1024 = dropout(ReLU(Linear(2048, 1024)), p=0.3)
512 = dropout(ReLU(Linear(1024, 512)), p=0.3)
256 = dropout(ReLU(Linear(512, 256)), p=0.3)
class_num = Linear(256, class_num)

b. Performance

Acc.: 0.91111

c. Loss function

CrossEntropyLoss

d. Optimizer, lr, batch size

Adam
Lr = 5e-3
Batch size = 16

Q3:Describe slot tagging model

a. Model
(Ignore batch_size)
128, 300 = Glove_embedding(128)
2048, 300 = GRU(300, 1024, dropout=0.3, layer=1, bidirectional=True)
512 = dropout(ReLU(Linear(2048, 512)), p=0.3)
256 = dropout(ReLU(Linear(512, 256)), p=0.3)
class_num = Linear(256, class_num)
b.Performance
Acc.: 0.72010

c. Loss function
CrossEntropyLoss

d. Optimizer, lr, batch size
Adam
Lr = 1e-4
Batch size = 128

Q4:Sequence tagging evaluation

```
f1_score:              0.793
word-wise acc_scoer:   0.966
joint acc_scoer:       0.800
              precision    recall  f1-score   support

        date       0.81      0.77      0.79       206
  first_name       0.91      0.87      0.89       102
   last_name       0.81      0.73      0.77        78
      people       0.73      0.73      0.73       238
        time       0.85      0.82      0.84       218

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

- Token accuracy, Joint accuracy
It calculates the token/word-wise accuracy. It can show the model's accuracy performance in each token. But in some cases, we expect the model could predict the whole sentence properly. Joint accuracy is only judged as correct when all predictions in one sample are correct. Which is also an important target when training and choosing a model.

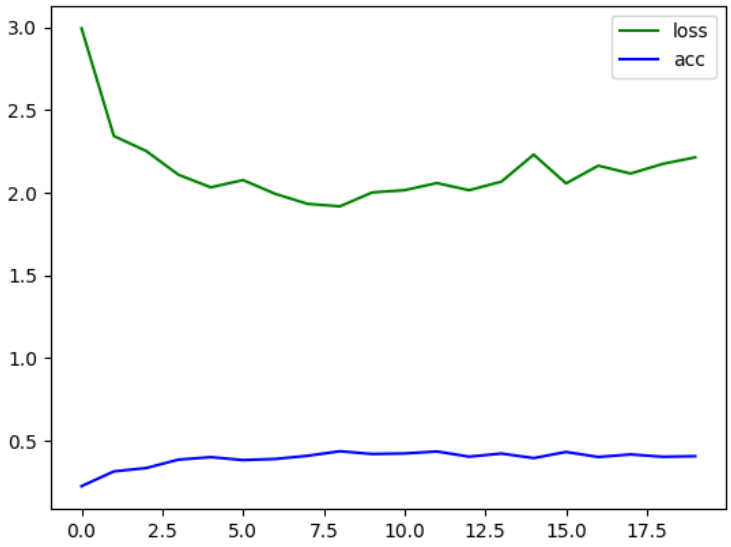
- Precision= TP / (TP+FP)
- Recall= TP / (TP+FN)
- F1-score = 2 * Precision * Recall / (Precision + Recall)

High precision shows that most of the model's positive predicts are correct. It's more careful but has higher accuracy.
On the other hand, a high recall value model won't let negative samples slip through its fingers. It can stop most of the positive samples although maybe make some mistakes.
F1-score is here to balance two evaluation criteria.

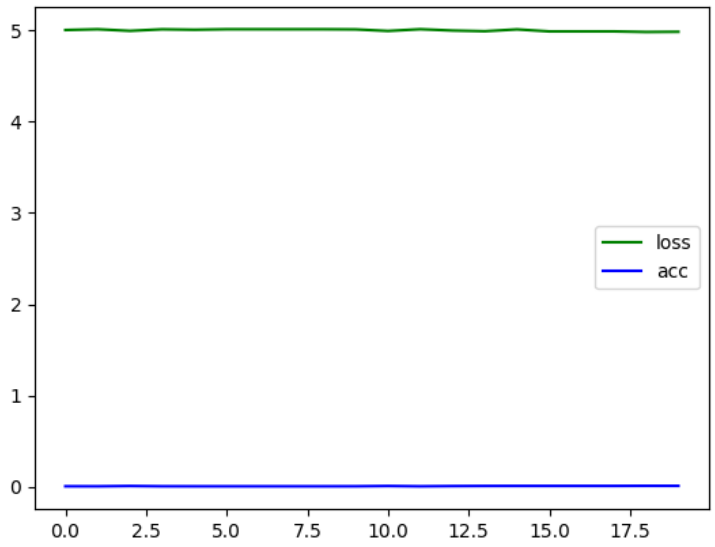
Q5: Compare different configurations

I tried different hidden layers in LSTM layers at the beginning.
With hidden layers of more than 1, the model took a lot of times and was untrainable. I did some experiments with 20 epochs comparing different LSTM hidden layers models. Only the one-hidden-layer model was trainable and got better performance.
In the slot-tagging task, I replace LSTM with GRU for a faster speed of convergence.

1-hidden-layer LSTM:



2-hidden-layer LSTM:



3-hidden-layer LSTM: