ADL

Homework 1

B07502166 魏子翔

Q1: Data processing.

a. How do you tokenize the data.

I use preprocess_intent.py and preprocess_slot.py provided in the sample code. It build a vocab set and count the appearing times of every words.

b. The pre-trained embedding you used.

The pre-trained embedding is GloVe(840B tokens, 2.2M vocab, cased, 300d vectors), an unsupervised learning algorithm for obtaining vector representations for words. Here's a table showing the numbers of tokens in datasets that covered in glove.

	intent	slot
token count	6491	4117
token match	5435	3000
match/count	0.837	0.729

Q2: Describe your intent classification model.

a. your model

The input sequence pass through three layers, which are Embedding(preprocess), LSTM, and linear layer with dropout. The process is as follows.

$$W = Embedding(S)$$
$$output, (h_t, c_t) = LSTM(w_t, h_{t-1}, c_{t-1})$$
$$Y_{pred} = Linear(output)$$

S: the input sequence

W: $\{w_0, w_1, w_2, ..., w_t\}$

w_t: the word embedding of the t-th token.

Y_{pred}: probability of each intent, so the final result is the intent with maximum probability

b. performance of your model.

local acc	public acc	private acc
0.924	0.917	0.908

c. the loss function you used.

cross entropy loss, where,

$$Loss = CrossEntropyLoss(Y_{pred}, groundtruth)$$

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

learning rate: 1e-3

batch size: 128

Q3: Describe your slot tagging model.

a. your model

The model of slot tagging is basically same as the intent model. The input sequence pass through three layers, which are Embedding(preprocess), LSTM, and linear layer with dropout. The process is as follows.

$$W = Embedding(S)$$
$$output, (h_t, c_t) = LSTM(w_t, h_{t-1}, c_{t-1})$$
$$Y_{pred} = Linear(output)$$

S: the input sequence

W: $\{w_0, w_1, w_2, ..., w_t\}$

w_t: the word embedding of the t-th token.

Y_{pred}: probability of each tag, so the final result is the tag with maximum probability

b. performance of your model

local acc	public acc	private acc
0.836	0.831	0.834

c. the loss function you used.

cross entropy loss, where,

$$Loss = CrossEntropyLoss(Y_{pred}, groundtruth)$$

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

learning rate: 1e-3

batch size: 128

Q4: Sequence Tagging Evaluation.

	precision	recall	f1-score	support
date	0.78	0.76	0.77	206
first_name	0.97	0.97	0.97	102
last_name	0.94	0.94	0.94	78
people	0.76	0.75	0.75	238
time	0.84	0.83	0.84	218
micro avg	0.83	0.82	0.82	842
macro avg	0.86	0.85	0.85	842
weighted avg	0.83	0.82	0.82	842

True Positive(TP): Predicted Positive and was Positive

True Negative(TN): Predicted Negative and was Negative

False Positive(FP): Predicted Positive but was Negative

False Negative(FN): Predicted Negative but was Positive

Precision $= \frac{\text{TP}}{\text{TP+FP}}$

Precision is a measure of how many of the positive predictions made are correct.

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

Recall is a measure of how many of the positive cases the classifier correctly predicted, over all the positive cases in the data.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-Score weights precision and recall in a balanced way, which is much more sensitive to one of the two inputs having a low value.

Support is the number of occurrences of each label in groundtruth.

$$\text{Micro Avg} = \frac{\text{TP of all tags}}{\text{TP+FP of all tags}}$$

Macro Avg: the average of all precision and recall

Weighted Avg: the weighted average according to support.

$$\text{Token Accuracy} = \frac{\text{correctly predicted tokens}}{\text{number of all tokens}}$$

$$\text{Joint Accuracy} = \frac{\text{correctly predicted sequences}}{\text{number of all sequences}}$$

Q5: Compare with different configurations.

a. intent classification

hidden size	dropout	local acc	public acc	private acc
512	0.2	0.924	0.917	0.908
512	0.4	0.916	0.909	0.897
256	0.2	0.921	0.905	0.903
256	0.1	0.923	0.910	0.906

Larger hidden size seems to be a little bit more accurate. I think it's because the larger hidden size can prevent the model from the local minimum, although the effect is really small.

b. tagging

hidden size	dropout	local acc	public acc	private acc
512	0.2	0.826	0.829	0.843
512	0.4	0.832	0.822	0.835
256	0.2	0.836	0.831	0.834
256	0.4	0.834	0.827	0.827

In this case, the model seems to be overfitting, so the simpler model has a better performance. Therefore, I set (256, 0.2) model to be my best model and take it to CNN-BiLSTM test below.

bonus. I tried CNN-BiLSTM for my tagging model. With 2 layers of biLSTM, 256 hidden size and 0.2 dropout, I focus on the impacts of different number of layers of CNN.

	no CNN	1 CNN	2 CNN
local acc	0.836	0.845	0.822
public acc	0.831	0.826	0.822
private acc	0.834	0.822	0.812

Obviously, adding CNN layers does not improve the performance. Since the most usage of CNN layers is to eliminate noise, so I think it's because that the data is relatively small, so the noise is easily eliminated by the other parts of my model.