ADL21-HW1

B08303028 Hsiang-An Chuang

October 2022

1 Data processing

1. I use package `word_tokenize` from `nltk.tokenize` to do word tokenization. As for word embedding, I use `glove.840B.300d.txt`.

2. After executing proprocess.sh, we can get `glove.840B.300d.txt`. I create a function called `get_embedding_dic()` to transfer `glove.840B.300d.txt` to a dictionary where the key is a token and the value is its corresponding word vector. Moreover, I create a function call `get_alter_token()` by observing unknown word for glove in dataset and implementing some simple parsing. For example, If we put 12:30pm into `get_alter_token()`, it returns 12:30, which is in glove dictionary. Though this function, I hope more word can be transfer to word vector but not just a random or zero vector.
   Namely, I didn't use `Vocab()` provided by sample code. Instead, I do all pre-processing in `collate_fn()` by myself. In `collate_fn()`, after getting tokens in a sentence, I try to get their corresponding vector in glove dictionary, if failed, we try to use `get_alter_token()` to get an alter token and try to find it in glove dictionary again. If still failed, it will be a zero vector representing an unknown token. After getting all word vectors, I simply stack all vectors, labels, and ids and return them as a dictionary.

2 Describe your intent classification model

1. I apply multi-layer GRU with fully-connected layers as my intent classification model. For each word embedding vector at time t, each layer in GRU do:

$$r_t = \sigma(W_{ir} + b_{ir} + W_{hr}h_{t-1} + b_{hr})$$
$$z_t = \sigma(W_{iz} + b_{iz} + W_{hz}h_{t-1} + b_{hz})$$
$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{t-1}) + b_{hn})$$
$$h_t = (1 - z_t) * n_t + z_t * h_{t-1}$$

where h_t is the hidden state at time t, x_t is the input at time t, and r_t, z_t, n_t are reset, update and new gates. σ is the sigmoid function, and * is the Hadamard product.
We can get the final output of word vector x_{final}, and we put this vector into fully connected layer FC to dense it and get the logits L we want. Since there are 150 intent classes, the size of logits should be $(N \times 150)$ where N is the batch size:

$$L = FC(x_{final})$$

Final, we take $\arg\max$ to L to get the highest possibility class index P to be our prediction:

$$P = \arg\max(L)$$

2. performance: 0.91822
3. loss function: BCEWithLogitsLoss
4. optimization algorithm: AdamW
5. learning rate: 0.001
6. batch size: 64

3 Describe your slot tagging model

1. model structure is same as intent classifier model, the only difference is the output dimension of fully connected layer. Since there are only 9 classes in this task, the shape of final output of fully connected layer should be modified to $(N \times 9)$ where N is the batch size.

2. performance: 0.81286
3. loss function: BCEWithLogitsLoss
4. optimization algorithm: AdamW
5. learning rate: 0.001
6. batch size: 256

4 Sequence Tagging Evaluation

1. evaluation of my model using `classification_report()` from `seqeval`:

	precision	recall	f1-score	support
date	0.81	0.79	0.80	206
first_name	0.95	0.91	0.93	102
last_name	0.84	0.88	0.86	78
people	0.70	0.70	0.70	238
time	0.87	0.86	0.86	218
micro avg	0.81	0.81	0.81	842
macro avg	0.83	0.83	0.83	842
weighted avg	0.81	0.81	0.81	842

2. The difference between token accuracy and joint accuracy is the units they evaluate correctness. For joint accuracy, only all tagging in a sentence are all correct will be seem as right prediction, that is, if a sentence contains 100 tokens and only one tag are predicted wrong, joint accuracy seem it as wrong prediction. On the other hand, token accuracy evaluate accuracy by the correctness of tokens, in the above example, token accuracy will be 99 out of 100 but not 0. It is quit obvious that token accuracy is stricter. It consider more about entire correctness for each sentence in a paragraph.

5 Compare with different configurations

In terms of model, I've tried LSTM and GRU.
With other parameters fixed(including hidden layer and other parameters in model structure), Two of them got similar score but the LSTM take about 1.5 to 2 times time to converge and so does the whole training process. So, I decided to pick GRU as my backbone model.
In terms of batch size, I've tried 512, 216, 128 and 64.
The larger the batch size I set, the less time it need to convergence. However, in all trails, I found when I decrease batch size, the score getting better, so I keep decrease it until I found 128 and 64 is then optimize choice.
I also try to adjust hidden layers. I've tried 512, 256 and 128.
The time to converge decrease when I cut down the number of hidden layers.
In my trails, 512 and 128 hidden layers got similar score, but 216 improve the score with 0.03, witch is a huge improvement.