School of Data and Computer Science, Sun Yat-sen University

Chinese Segmentation Using BiLSTM-CRF

18308045 Zhengyang Gu

November 30, 2020

Contents

1 README!!!

My vscode crashed when I was editing the tex file causing my losing the tex file, but the pdf of the first part is retained. Therefore, I splited the whole report into two parts.

The first part of the report.

Codes and README.md are open-sourced on github.

2 My Implementation

2.1 Tagging Using CRF

2.2 Training

We can address this issue by making a few changes. Let $smat(Y_j, Y_{j+1})$ be $\alpha(Y_j) + f(X_j, Y_j) + t(Y_j, Y_{j+1})$.

$$\begin{aligned}
\alpha(Y_{j+1}) &= \log(\sum_{Y_j} \exp(\alpha(Y_j) + f(X_j, Y_j) + t(Y_j, Y_{j+1}))) \\
&= \log(\sum_{Y_j} \exp(smat(Y_j, Y_{j+1}))) \\
&= \log(\sum_{Y_j} \exp(smat(Y_j, Y_{j+1}) - \max_{Y_j} smat(Y_j, Y_{j+1}))) + \max_{Y_j} smat(Y_j, Y_{j+1})
\end{aligned}$$

Then when after we calculate the **exp**, the max number we get is 1, so there won't be any overflow errors in python. In addition, I add the batch to the first dimension. The implementation of the **log_sum_exp** is as follows.

```
def log_sum_exp(smat_batch):
    vmax_batch = smat_batch.max(dim=1, keepdim=True).values
    return (smat_batch - vmax_batch).exp().sum(axis=1, keepdim=True).log() + vmax_batch
```

2.3 Estimating The $f(X_i, Y_i)$ Using BiLSTM

I use the BiLSTM implemented in pytorch, so there isn't much to say in this section.

2.4 Batch

I add the batch to the first dimension, so I have to make the **batch_first** to be **True**.

```
self.lstm = nn.LSTM(embedding_dim, hidden_dim // 2, num_layers=1, bidirectional=True, batch_first = True)
```

However, the **batch_first** only affects the input and output but not the hidden state, so when initializing the hidden state I have to add the batch to the second dimension.

```
hidden_batch = torch.randn(2, self.batch_size, self.hidden_dim // 2).to(self.device),\
        torch.randn(2, self.batch_size, self.hidden_dim // 2).to(self.device)
```

In addition, the loss of a batch here is defined as the average value of each **neg_log_likelihood** in the batch.

```
return (forward_score_batch - gold_score_batch).sum(0) / self.batch_size
```

2.5 Hyperparameters

- GPU: If **True**, GPU will be used to improve the speed of training.
- LR: Learning rate of the optimizer.
- MAX_EPOCH: The max number of epoches when training.
- BATCH_SIZE: The size of each batch.
- EMBEDDING_DIM: The dimension of the word embedding.
- HIDDEN_DIM: The dimension of BiLSTM' s hidden state.
- SHUFFLE: If **True**, every epoch the training data will be shuffled.

3 Result

3.1 Hyperparameters

I tested different values of some of the hyperparameters ahead of the learning process to choose a fairly good combination of hyperparameters.

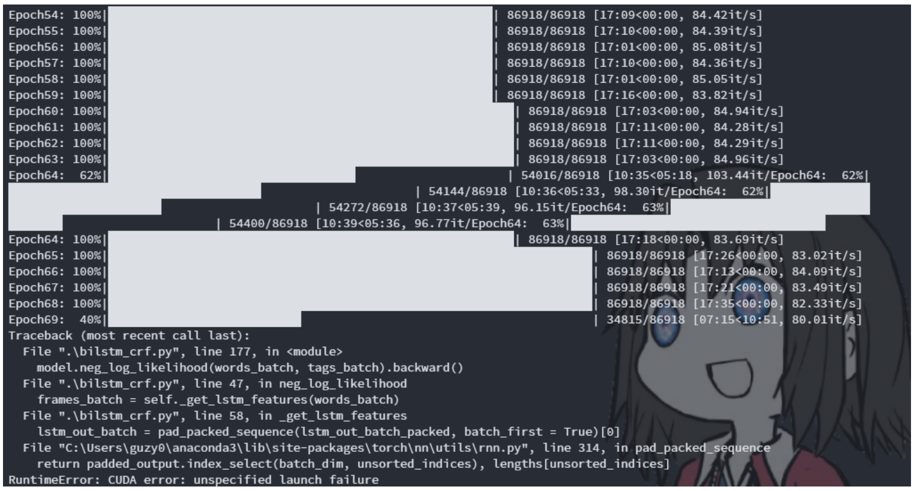
MAX_EPOCH	BATCH_SIZE	EMBEDDING_DIM	HIDDEN_DIM	Time	F1
1	128	16	16	17:04	0.789882198385716
1	128	128	16	17:47	0.7985092541747879
1	128	128	128	17:35	0.8670882556323042
1	256	128	128	19:34	0.8519033872880736

3.2 Final Result

The hyperparameters I choose are shown as below.

```
GPU = True
LR = 0.01
MAX_EPOCH = 128
BATCH_SIZE = 128
EMBEDDING_DIM = 128
HIDDEN_DIM = 128
SHUFFLE = True
```

Here came an error when I was training the model.



```
Epoch54: 100%|                                        | 86918/86918 [17:09<00:00, 84.42it/s]
Epoch55: 100%|                                        | 86918/86918 [17:10<00:00, 84.39it/s]
Epoch56: 100%|                                        | 86918/86918 [17:01<00:00, 85.08it/s]
Epoch57: 100%|                                        | 86918/86918 [17:10<00:00, 84.36it/s]
Epoch58: 100%|                                        | 86918/86918 [17:01<00:00, 85.05it/s]
Epoch59: 100%|                                        | 86918/86918 [17:16<00:00, 83.82it/s]
Epoch60: 100%|                                          | 86918/86918 [17:03<00:00, 84.94it/s]
Epoch61: 100%|                                          | 86918/86918 [17:11<00:00, 84.28it/s]
Epoch62: 100%|                                          | 86918/86918 [17:11<00:00, 84.29it/s]
Epoch63: 100%|                                          | 86918/86918 [17:03<00:00, 84.96it/s]
Epoch64:  62%|                                         | 54016/86918 [10:35<05:18, 103.44it/Epoch64:  62%|
                                              | 54144/86918 [10:36<05:33, 98.30it/Epoch64:  62%|
                                    | 54272/86918 [10:37<05:39, 96.15it/Epoch64:  63%|
                         | 54400/86918 [10:39<05:36, 96.77it/Epoch64:  63%|
Epoch64: 100%|                                          | 86918/86918 [17:18<00:00, 83.69it/s]
Epoch65: 100%|                                                  | 86918/86918 [17:26<00:00, 83.02it/s]
Epoch66: 100%|                                                  | 86918/86918 [17:13<00:00, 84.09it/s]
Epoch67: 100%|                                                  | 86918/86918 [17:21<00:00, 83.49it/s]
Epoch68: 100%|                                                  | 86918/86918 [17:35<00:00, 82.33it/s]
Epoch69:  40%|                                                  | 34815/86918 [07:15<10:51, 80.01it/s]
Traceback (most recent call last):
  File ".\bilstm_crf.py", line 177, in <module>
    model.neg_log_likelihood(words_batch, tags_batch).backward()
  File ".\bilstm_crf.py", line 47, in neg_log_likelihood
    frames_batch = self._get_lstm_features(words_batch)
  File ".\bilstm_crf.py", line 58, in _get_lstm_features
    lstm_out_batch = pad_packed_sequence(lstm_out_batch_packed, batch_first = True)[0]
  File "C:\Users\guzy0\anaconda3\lib\site-packages\torch\nn\utils\rnn.py", line 314, in pad_packed_sequence
    return padded_output.index_select(batch_dim, unsorted_indices), lengths[unsorted_indices]
RuntimeError: CUDA error: unspecified launch failure
```

Figure 1: Error I met

Hence, I just trained the model for 69 epoches. The bug is just related to the CUDA configuration, but has nothing to do with my codes. [1]

The average F1 score tested on the **msr_test_gold.utf8** is shown as below.

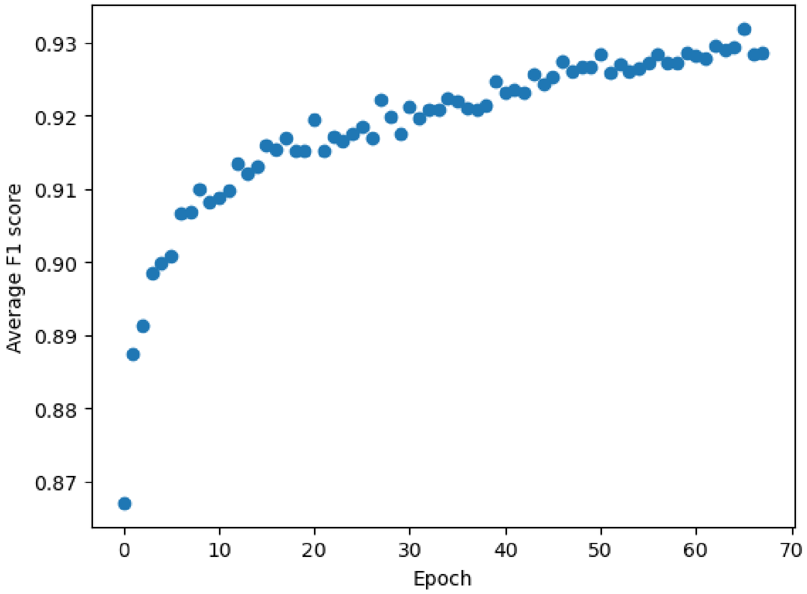


Figure 2: Average F1 score

We can find that the F1 score is still increasing.

The segmentation result of **msr_test.utf8** is result.utf8.

4 Conclusions And Bonuses I May Get

1. I implemented the BiLSTM-CRF model to accomplish the Chinese segmentation task.
2. I use batch and GPU to speed up the training process.
3. I tested different combination of hyperparameters ahead of training.
4. I compose my report in English, although it may be not fluent.

Future work may include the experiment with more running epochs.

References

[1] https://www.cnblogs.com/dgwblog/p/12868068.html