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

DQN Breakout

18340047 Xiaolong Guo, 18308045 Zhengyang Gu

November 29, 2020

Contents

1 Abstract

I implemented a BiLSTM-CRF to accomplish the Chinese Words Segmentation task. To improve the speed of convergence, I adopted the batch learning method. And I tested different values of some of the hyperparameters ahead of the learning process to choose a fairly good combination of hyperparameters.

2 Related Works

2.1 BiLSTM

While traditional neural network doesn' t support present thought based on previous thoughts like humans do, RNN addresses this issue by allowing information to be passed from one step of the network to the next.

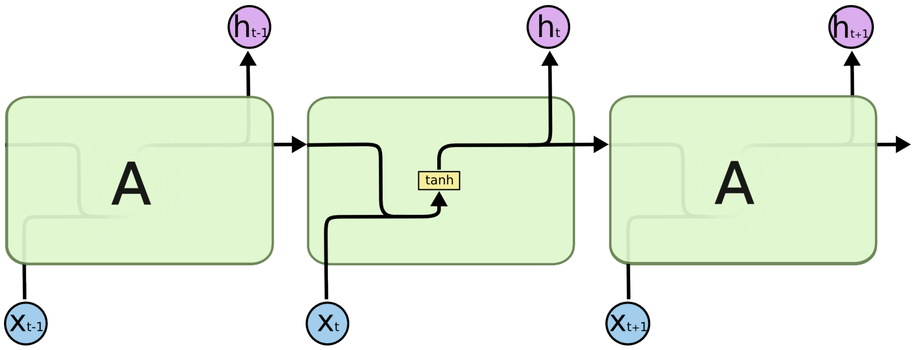


Figure 1: Simple RNN[2]

Simple RNN' s always looking at all of the information in the whole histry, however, can be useless in some cases. For instance, when we are processing a sentence, usually, only the most recent words should be looked at. The LSTM exploits a special structure with four interacting layers to acheive forgetting like humans do to avoid the long term dependency.

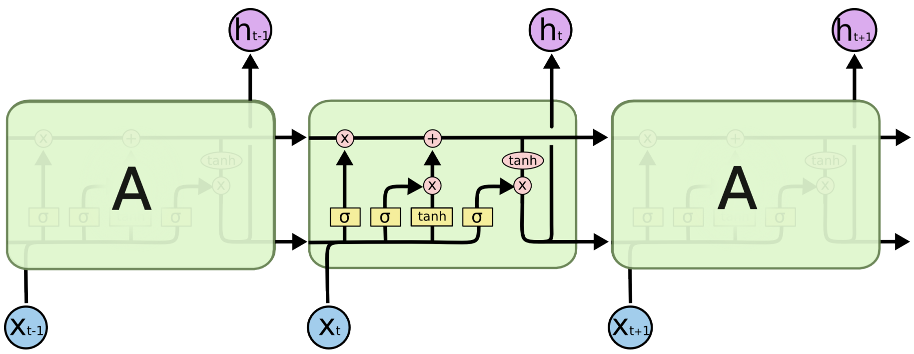
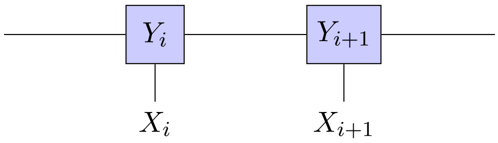


Figure 2: LSTM chain[2]

BiLSTM is a variation of LSTM, which is a LSTM that is fed with the data once from beginning to the end and once from end to beginning. This method is usually faster than LSTM.[1]

2.2 CRF

CRF is a special case of Markov random field. It assumes that there are only two kindes of variables X and Y in Markov random field. And there are two kinds of feature function in CRF, one of which is defined between adjacent Xs and Ys that has the form of $s(X_i, Y_i)$, the other of which is defined between two adjacent Ys that has the from of $t(Y_i, Y_{i+1})$.



They look like emission probability and transition probability in HMM. Informally, we can define the emission probability as $P(X_i|Y_i) = \exp(f(X_i, Y_i))$, and define the transition probability as $P(Y_i|Y_{i-1}) = \exp(t(Y_i, Y_{i+1}))$. Then given X as evidence, we can compute the probability distribution of Y.

$$P(Y|X) = \frac{\exp(\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1}))}{\sum_Y \exp(\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1}))}$$

3 My Implementation

3.1 Chinese Segmentation Based On Tagging

Those Chinese characters that make up a single-character words are tagged as **S**. Interms of the characters in, multi-characters words, the characters at the beginning, the characters in the middle and the characters in the end are respectively tagged as **B**, **M** and **E**. Every character in a sentence, therefore, can be tagged, so we can segment the sentence basing on these tags.

3.2 Tagging Using CRF

We can use X_i to represent the ith character in a sentence, and use Y_i to represent the tag of the character. Hence, the $f(X_i, Y_i)$ here is related to the posibility of the character X_i given the tag Y_i, and the $t(Y_i, Y_{i+1})$ is related to the posibility of the next tag Y_{i+1} given the tag Y_i.

3.2.1 Decoding

Finding the tags sequence of given sentence, therefore, is to find a tag sequence Y that have the largest $P(Y|X)$. We can exploit Viterbi decoding to solve this problem. Viterbi decoding is actually a kind of dynamic programming method. The initial state is defined as below.

$$\alpha(X_0, Y_0) = f(X_0, Y_0) = \begin{cases} 0, & \text{if } Y_0 = \text{START\_TAG} \\ -\infty, & \text{else} \end{cases}$$

And the transition equation is defined as below.

$$\alpha(X_{i+1}, Y_{i+1}) = \max_{Y_i}\{\alpha(X_i, Y_i) + t(Y_i, Y_{i+1})\} + f(X_{i+1}, Y_{i+1})$$

Assuming that there are n characters in a sentence, we get the equation below.

$$\max_Y\{\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1})\} = \max_{Y_n}\{\alpha(X_n, Y_n) + t(Y_n, \text{END\_TAG})\}$$

If we put these best Y_n into an array **best_path**, the best tag sequence Y, therefore, can be calculated as below.

$$\begin{aligned} \arg\max_Y P(Y|X) &= \arg\max_Y\{\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1})\} \\ &= \textbf{best\_path} + \arg\max_{Y_n}\{\alpha(X_n, Y_n) + t(Y_n, \text{END\_TAG})\} \end{aligned}$$

These functions α, f, t can be written in the form of matrix. Therefore, the initial state can be implemented as below.

```
alpha = torch.full((1, self.n_tags), -10000).to(self.device)
alpha[0, self.tag2ix[START_TAG]] = 0
```

The transition equation can be implemented as follows.

```
for frame in frames:
    smat = alpha.T + frame.unsqueeze(0) + self.transitions
    backtrace.append(smat.argmax(0))
    alpha = smat.max(dim = 0)[0].unsqueeze(0)
```

And the last best tag has the following form.

```
smat = alpha.T + 0 + self.transitions[:, [self.tag2ix[END_TAG]]]
best_tag_id = smat.squeeze().argmax().item()
```

Hence, we can exploit these to calculate the best sequence.

3.2.2 Traning

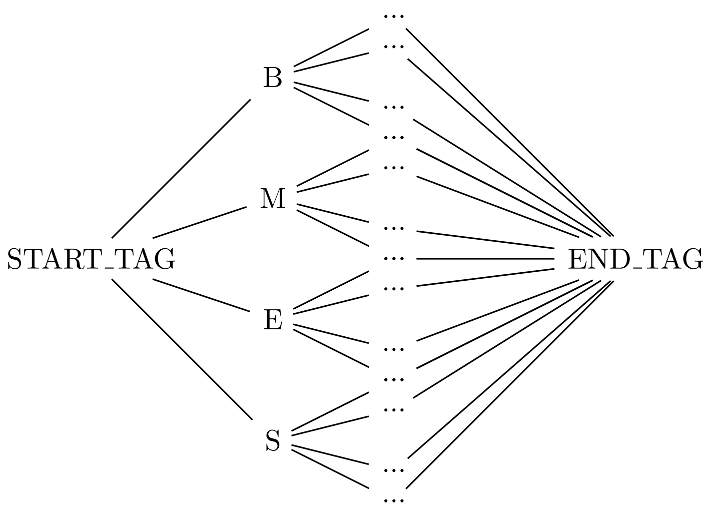
Given a sentence X and its tags Y, our goal is to find an f and a t to maximize the $P(Y|X)$, which is actually maximum likelihood estimation. We can apply the log function to it to simplify the calculation.

$$
\begin{aligned}
\arg\max_{f,t} P(Y|X) =& \arg\max_{f,t} \log(P(Y|X)) \\
=& \arg\max_{f,t} \{\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1}) \\
& - \log(\sum_Y \exp(\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1})))\} \\
=& \arg\min_{f,t} \{\log(\sum_Y \exp(\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1}))) \\
& - (\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1}))\}
\end{aligned}
$$

The second term on the right side of the equation can be simply calculated as follows

```
score_batch = torch.zeros(self.batch_size, 1).to(self.device)
for i in range(frames_batch.shape[1]):
    score_batch += self.transitions[tags_batch_tensor[:, i], tags_batch_tensor[:, i + 1]].unsqueeze(1) + frames_batch[range(self.batch_size), i, [tags_batch_tensor[j, i + 1] for j in range(self.batch_size)]].unsqueeze(1)
return score_batch + self.transitions[tags_batch_tensor[:, -1], self.tag2ix[END_TAG]].unsqueeze(1)
```

, while the calculation of the first term is tricky resulting from that it's extremely hard to enumerate all of the possible sequence Y.



However, it can simplified by dynamic programming.

We define the state $\alpha(Y_j)$ as the **log_sum_exp** of all possible paths terminated with tag Y_j. Let Y terminated with Y_j be defined as Y^j.

$$\alpha(Y_j) = \log(\sum_{Y^j} \exp(\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1})))$$

Therefore, that tricky term of a sentence of length n is $\alpha(X_{n+1}, Y_{n+1})$. In addition, the initial state has the following form.

$$\alpha(Y_0) = f(X_0, Y_0) = \begin{cases} 0, & \text{if } Y_0 = \text{START\_TAG} \\ -\infty, & \text{else} \end{cases}$$

And we can get the transition equation as follows.

$$\begin{aligned} \alpha(Y_{j+1}) &= \log(\sum_{Y^{j+1}} \exp(\sum_i f(X_i, Y_i) + \sum_i t(Y_i, Y_{i+1}))) \\ &= \log(\sum_{Y_j} \exp(\alpha(Y_j) + f(X_j, Y_j) + t(Y_j, Y_{j+1}))) \end{aligned}$$

These functions α, f, t can be written in the form of matrix. Additionally, I added the batch on the first dimention. Therefore, the initial state can be implemented as below.

```
alpha_batch = torch.full((self.batch_size, 1, self.n_tags), -10000).to(self.device)
alpha_batch[:, 0, self.tag2ix[START_TAG]] = 0
```

The transition equation can be implemented as follows.

```
for i in range(frames_batch.shape[1]):
    alpha_batch = BiLSTM_CRF.log_sum_exp(alpha_batch.transpose(1, 2) + frames_batch[:, i].unsqueeze(1) + self.transitions)
```

And the final score can be implemented as below.

```
return BiLSTM_CRF.log_sum_exp(alpha_batch.transpose(1, 2) + 0 + self.transitions[:, [self.tag2ix[END_TAG]]]).squeeze(1)
```

Hence, we can exploit these to calculate the score.

When calculating the **log_sum_exp**, chances are that we may get extremely large numbers after calculating the **exp**, which may cause errors in python.

References

[1] https://datascience.stackexchange.com/questions/25650/what-is-lstm-bilstm-and-when-to-use-them

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/