BiLSTM-CRF on POS: The Final DL Course Report

Guozhen She
Department of Computer Science
Fudan University
github.com/hazelnutsgz
hippo@cs.cranberry-lemon.edu

Abstract

In this report, we reproduced the work at the paper "Bidirectional LSTM-CRF Models for Sequence Tagging" on EMNLP'16. This model applied a bidirectional LSTM CRF (denoted as BI-LSTM-CRF) model to NLP benchmark sequence tagging data sets. The report can be divided into the following, first we gave a thorough description of the task, and the motivation to finish the task. In the second part, we introduced some state-of-art models and proposed our own Bi-LSTM CRF model based on them, and some background information to help understanding it. In the third part, we introduced a new deep learning framework named fastNLP, and how we use it to build and train our new model. Then we will give some results on Conll2003 dataset, and analyzed the problems we have met during the building and tuning the model. At last part, we compared our model with some state-of-art models about inference performance and training speed efficiency, and give a blueprint about how to improve our work.

1 Introduction

1.1 Background

Sequence tagging including part of speech tagging(POS) has always been a classic NLP task. It has drawn the research attention for decades of years because of its wide applications in different areas. For example, a named entity recognizer trained on user search queries can be utilized to identify which spans of tex are products, thus triggering certain products ads. Another example is that such tag information can be used by a search engine to find relevant webpages.

1.2 Dataset

The datasets in POS task are various, and the most acknowledged dataset must be the conll, so in this project, we introduce the Conll2003 data[1], the dataset can be divided into 3 parts, train.txt, test.txt, valid.txt, and each file is of the same structure—-Each line in the txt file is either blank(means the end of the sentence) or 4-token like structure(means the word, POS, NER1, NER2 seperately). To parse and analyze the dataset, I contributed the dataloader tools to the fastNLP, [2]

2 Model

In this part, I will introduce the our new model. First we will recap the state-of-art model for the POS task, such as CRF and LSTM, then we will bring out the new model, BiLSTM-CRF.

[1]https://www.clips.uantwerpen.be/conll2003/ner/
[2]https://github.com/fastnlp/fastNLP/pull/115

2.1 LSTM

Recurrent neural networks (RNN) have been em- ployed to produce promising results on a variety of tasks including language model (Mikolov et al., 2010; Mikolov et al., 2011) and speech recognition (Graves et al., 2005). A RNN maintains a memory based on history information, which enables the model to predict the current output conditioned on long distance features. Figure 1 shows the RNN structure (Elman, 1990) which has an input layer x, hidden layer h and output layer y. In named entity tagging context, x represents input features and y represents tags. Figure 1 illustrates a named entity recognition system in which each word is tagged with other (O) or one of four entity types: Person (PER), Location (LOC), Organization (ORG), and Miscellaneous (MISC). The sentence of EU rejects German call to boycott British lamb . is tagged as B-ORG O B-MISC O O O B-MISC O O, where B-, I- tags indicate beginning and intermediate positions of entities.

An input layer represents features at time t. They could be one-hot-encoding for word feature, dense vector features, or sparse features. An input layer has the same dimensionality as feature size. An output layer represents a probability distribution over labels at time t. It has the same dimen sionality as size of labels. Compared to feedforward network, a RNN introduces the connection between the previous hidden state and current hid- den state (and thus the recurrent layer weight parameters). This recurrent layer is designed to store history information. The values in the hidden and output layers are computed as follows:

$$h(t) = f(\mathbf{U}x(x) + \mathbf{W}h(t-1)) \qquad (1)$$

$$y(t) = g(\mathbf{V}h(t)) \qquad (2)$$

where **U**, **W**, and **V** are the connection weights to be computed in training time, and f(z) and g(z) are sigmoid and softmax activation functions as follows:

$$f(z) = \frac{1}{1+e^{-z}} \qquad (3)$$

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \qquad (4)$$

In this paper, we will introduce a efficient version of RNN, Long Short-Term Memory (Hochreiter and Schmidhuber, 1997; Graves et al., 2005) to sequence tagging. Long Short- Term Memory networks are the same as RNNs, except that the hidden layer updates are replaced by purpose-built memory cells. As a result, they may be better at finding and exploiting long range dependencies in the data. Fig. 2 illustrates a single LSTM memory cell (Graves et al., 2005). The LSTM memory cell is implemented as the following:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o$$

$$h_t = o_t tanh(c_t)$$

where σ is the logistic sigmoid function, and i, f, o and c are the input gate, forget gate, output gate and cell vectors, all of which are the same size as the hidden vector h. The weight matrix subscripts have the meaning as the name suggests. For ex- ample, Whi is the hidden-input gate matrix, Wxo is the input-output gate matrix etc. The weight matrices from the cell to gate vectors (e.g. Wci) are diagonal, so element m in each gate vector only receives input from element m of the cell vector.

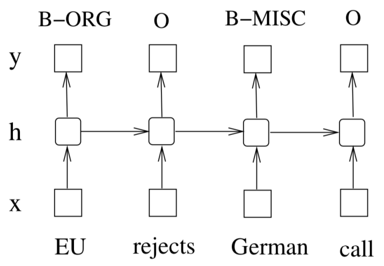

Figure 1: A simple RNN model.

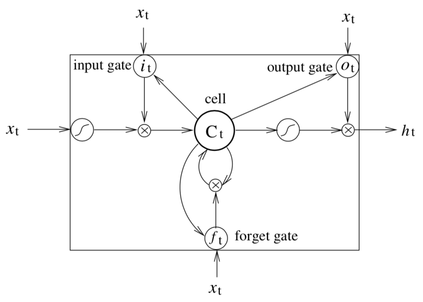

Figure 2: A Long Short-Term Memory Cell.

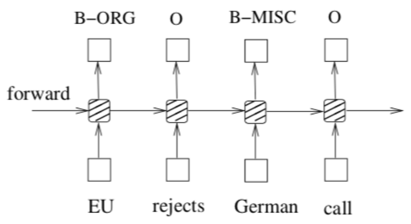

Figure 1: LSTM Network

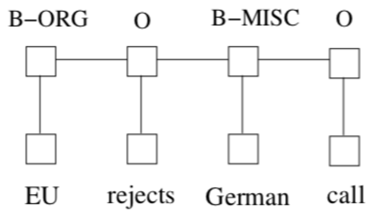

Figure 2: CRF Network

2.2 Bidirectional LSTM Networks

In sequence tagging task, we have access to both past and future input features for a given time, we can thus utilize a bidirectional LSTM network (Figure 4) as proposed in (Graves et al., 2013). In doing so, we can efficiently make use of past features (via forward states) and future features (via backward states) for a specific time frame. We train bidirectional LSTM networks using back- propagation through time (BPTT)(Boden., 2002). The forward and backward passes over the unfolded network over time are carried out in a similar way to regular network forward and backward passes, except that we need to unfold the hidden states for all time steps. We also need a special treatment at the beginning and the end of the data points. In our implementation, we do forward and backward for whole sentences and we only need to reset the hidden states to 0 at the begging of each sentence. We have batch implementation which enables multiple sentences to be processed at the same time.

2.3 CRF

There are two different ways to make use of neighbor tag information in predicting current tags. The first is to predict a distribution of tags for each time step and then use beam-like decoding to find optimal tag sequences. The work of maximum entropy classifier (Ratnaparkhi, 1996) and Maximum entropy Markov models (MEMMs) (McCallum et al., 2000) fall in this category. The second one is to focus on sentence level instead of individual positions, thus leading to Conditional Random Fields (CRF) models (Lafferty et al., 2001) (Fig. 5). Note that the inputs and outputs are directly connected, as opposed to LSTM and bidirectional LSTM networks where memory cells/recurrent components are employed. It has been shown that CRFs can produce higher tagging accuracy in general. It is interesting that the relation between these two ways of using tag information bears resemblance to two ways of using input features (see aforementioned LSTM and BI-LSTM networks), and the results in this paper confirms the superiority of BI-LSTM compared to LSTM.

2.4 LSTM-CRF Network

We combine a LSTM network and a CRF network to form a LSTM-CRF model, which is shown in Fig. 6. This network can efficiently use past input features via a LSTM layer and sentence level tag information via a CRF layer. A CRF layer is represented by lines which connect consecutive output layers. A CRF layer has a state transition matrix as parameters. With such a layer, we can efficiently use past and future tags to predict the current tag, which is similar to the use of past and future input

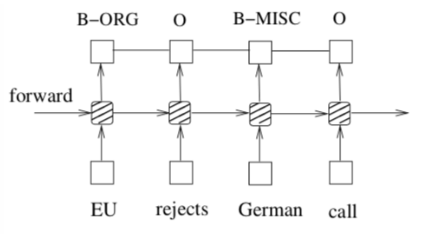

Figure 3: LSTM Network

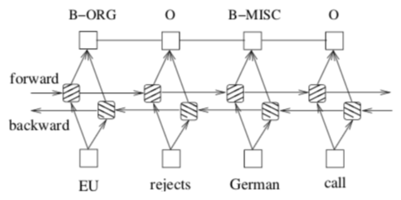

Figure 4: BiLSTM-CRF Model

features via a bidirectional LSTM network. We consider the matrix of scores $f([x]_{T1})$ are output by the network. We drop the input $[x]_{T1}$ for notation simplification. The element $[f]_{i,t}$ of the ma- trix is the score output by the network with param- eters , for the sentence $[x]_{T1}$ and for the i-th tag, at the t-th word. We introduce a transition score $[A]_{i,j}$ to model the transition from i-th state to j-th for a pair of consecutive time steps. Note that this transition matrix is position independent. We now denote the new parameters for our network as $= [A]_{i,j}i, j$. The score of a sentence $[x]_{T1}$ along with a path of tags [i]T1 is then given by the sum of transition scores and network scores:

$$\text{s}([\text{x}]_1^T, [i]_1^T, \theta) = \Sigma_{t=1}^T([A]_{[i]_{t-1},[i]_t} + [f_\theta]_{[i]_t,t})$$

The dynamic programming (Rabiner, 1989) can be used efficiently to compute $[A]_{i,j}$ and optimal tag sequences for inference. See (Lafferty et al., 2001) for details.

3 Experiment: Building, Training

We conducted our model based on fastNLP framework, which is a software layer based on Pytorh framework, can facilitate the development of deep learning model in NLP area. In this part, we will introduce our experience of the pipeline(build, train, test) of the POS task using Bi-LSTM.

3.1 Building Model

We builded our model based on fastNLP, which is one of our requirements, so we investigated the library by three ways: read the tutorial, dive into the source code, communicate with the community members(by Wechat or Issue on GitHub. During the building procedure, we found some minor bugs of the fastNLP and help the maintainers of the project to improve the library. Finally, we organized our runnable and testable codebase, then donated the project to the fastNLP under the reproduction directory by making a pull request on Github page.

3.1.1 Codebase Layout

Our codebase is composed of main.py, metric.py, process.py, model.py, save/, data/. The metric.py defined a metric to measure the performance of the model. The process.py provided some utility function for users to process the data. The model.py provided the Bi-LSTM model based on the fastNLP API with adjustable parameters. The main.py is a terminal tool to train or test model on given data with different parameters. The save/ directory stored the recent training log by tensorboard and the model paramters (by torch.save) which produce the best validation result. The data/ directory store the data for trainvalidtest.

3.2 Training Procedure

All models used in this paper share a generic SGD forward and backward training procedure. We choose the most complicated model, BI-LSTM- CRF, to illustrate the training algorithm as shown in Algorithm 1. In each epoch, we divide the whole training data to batches and process one batch at a time. Each batch contains a list of sentences which is determined by the parameter of batch size. In our experiments, we use batch size of 100 which means to include sentences whose total length is no greater than 100. For each batch, we first run bidirectional LSTM-CRF model forward pass

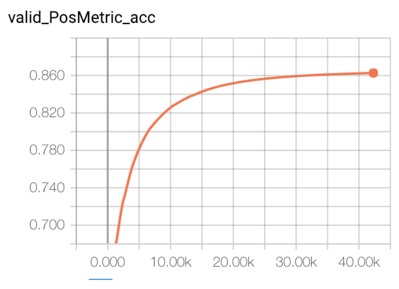

Figure 5: Partial Metric of valid data(not training)

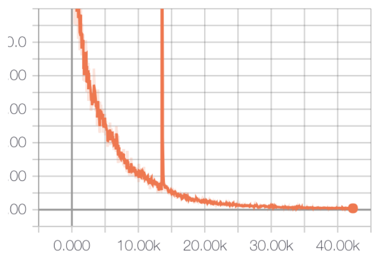

Figure 6: The loss of training data

which includes the forward pass for both forward state and backward state of LSTM. As a result, we get the the output score f([x]T1) for all tags at all positions. We then run CRF layer forward and backward pass to compute gradients for network output and state transition edges. After that, we can back propagate the er- rors from the output to the input, which includes the backward pass for both forward and backward states of LSTM. Finally we update the network parameters which include the state transition matrix [A]i,ji,j, and the original bidirectional LSTM parameters .

Algorithm 1 Training Algorithm of Bi-LSTM CRF

```
Input: latent dimension $K$, $G$, target predicate $p$
Output: $U^p$, $V^p$, $b^p$
 1: for each epoch do
 2:     for each batch do
 3:         1) bidirectional LSTM-CRF model forward pass:
 4:            forward pass for forward state LSTM
 5:            pass for backward state LSTM
 6:         2) CRF layer forward and backward pass
 7:         3) bidirectional LSTM-CRF model backward pass:
 8:            backward pass for forward state LSTM
 9:            backward pass for backward state LSTM
10:         4) update parameters
11:     end for
12: end for
```

4 Evaluation

4.1 Setup

We conducted our experiments on Ubuntu Linux Server equipped with Nvidia GPU 1080, We conduct the different experiments in three following methods: 1.We adjusted the batch size of training epochs, and model paramters to utilize various range of GPU memory from 600MB to 7200MB. 2. We adjust the learning rate and optimizer of the Trainer at different training stage(The fastNLP will save the best model for us, so we can load the extant model with different learning config) The figure provided a visulized representation of training stage(supported by tensorboard).

4.2 Comparison

We compared our model with some state-of-art models, LSTM, CRF, and so on, We compared both the training efficiency and training accuracy of the different methods on the same dataset. The table 1 has shown some results of the comparison. Though our result didn't catch up the reported performance of the original paper due to the lack of machine resources and fine tuning trick, we can still prove the superiority of our new model by comparing with the other STOA model with the same training condition. We can see our model, say, Bi-LSTM+CRF reached the best accuracy. 92.3% on test datasets. On the other hand, the training time of our model is not as satisfied as we expected. So it is our hope to trim the model parameter in our future work.

5 Future Work

The work in the next step can be divided into two main aspects. On the one hand, we will add the data downloader utility to the fastNLP proeject, which has been put high on agenda by the maintainer of the project. One the other hand, it is expected that we will improve the existing Bi-LSTM model on POS task, for example, we intended to add the char embedding features to the current work, which may capture some more detailed infromation about our datasets.

References

[Xipeng Qiu, Shaojing Wang, Ziyuan Feng 2018] fastNLP framework(https://fastnlp.readthedocs.io)

[Ando and Zhang.2005] R. K. Ando and T. Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. Jour- nal of Machine Learning Research (JMLR).

[Boden.2002] M. Boden. 2002. A Guide to Recurrent Neural Networks and Back-propagation. In the Dal- las project.

[Chieu.2003] H. L. Chieu. 2003. Named entity recog- nition with a maximum entropy approach. Proceed- ings of CoNLL. [Collobert et al.2011] R. Collobert, J. Weston, L. Bot- tou, M. Karlen, K. Kavukcuoglu and P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research (JMLR). [Elman1990] J. L. Elman. 1990. Finding structure in time. Cognitive Science.

[Finkel et al.2005] J. R. Finkel, T. Grenager, and C. Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of ACL.

[Florian et al.2003] R. Florian, A. Ittycheriah, H. Jing, and T. Zhang. 2003. Named entity recognition through classifier combination. Proceedings of NAACL-HLT.

[Gimenez and Marquez2004] J. Gimenez and L. Mar- quez. 2004. SVMTool: A general POS tagger gen- erator based on support vector machines. Proceed- ings of LREC.

[Graves et al.2005] A. Graves and J. Schmidhuber. 2005. Framewise Phoneme Classification with Bidi- rectional LSTM and Other Neural Network Archi- tectures. Neural Networks.

[Graves et al.2013] A. Graves, A. Mohamed, and G. Hinton. 2013. Speech Recognition with Deep Re- current Neural Networks. arxiv.

[Hammerton2003] J. Hammerton. 2003. Named Entity Recognition with Long Short-Term Memory. Pro- ceedings of HLT-NAACL.

[Hochreiter and Schmidhuber1997] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. Neural Computation, 9(8):1735-1780.

[Kudo and Matsumoto2000] T. Kudo and Y. Mat- sumoto. 2000. Use of support vector learning for chunk identification. Proceedings of CoNLL.

[Kudo and Matsumoto2001] T. Kudo and Y. Mat- sumoto. 2001. Chunking with support vector machines. Proceedings of NAACL-HLT.

[Lafferty et al.2001] J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling se- quence data. Proceedings of ICML.

[McCallum et al.2000] A. McCallum, D. Freitag, and F. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. Pro- ceedings of ICML.

[Mcdonald et al.2005] R. Mcdonald , K. Crammer , and F. Pereira. 2005. Flexible text segmentation with structured multilabel classification. Proceedings of HLT-EMNLP.

[Mesnil et al.2013] G. Mesnil, X. He, L. Deng, and Y. Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for language understanding. Proceedings of INTERSPEECH.