Sentence-State LSTM

Anonymous submission

Abstract

This is our project report for the course "Natural Network and Deep Learning, Autumn 2018". We reproduce the ACL 2018 paper "Sentence-State LSTM for Text Representation" based on the structure of FastNLP[1]. Our model shows comparable results to the original one on all their datasets, except the Cornell movie review dataset which lacks a standard split. Besides, we also apply the famous modular BERT on this task to explore if the pretrained language model can bring about additional promotions.

1 Introduction

Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] has been a dominant method in Natural Language Process to deal with the dependencies between words and sentences. Bidirectional LSTM (BiLSTM) is further used to combine information from past and future states simultaneously. However, both of them suffer several limitations due to their sequential nature. For example, hidden states can not be computed parallel within the same time serial compared to self-attention [Vaswani et al., 2017] and local n-grams information has been ignored compared to window-based Convolutional Neural Network (CNN). In addition, BiLSTM also shows deficiencies on modeling very long-range dependencies such as the encoding of a long document [Hermann et al., 2015].

Zhang et al. [2018] investigated a new LSTM cell called Sentence-State LSTM (S-LSTM) to deal with these problems. They intended to model all words at the same time by separating recurrent steps and words. For traditional LSTM, the recurrent process goes along the word sequence and each word is a individual recurrent step. Thus, the word must wait for its predecessor or successor to get the context information. Instead, S-LSTM assumes individual words have different sub-states at different recurrent steps. The word current sub-states depends on its neighbor sub-states of previous recurrent steps and then the whole word sequences can be modeled simultaneously. N-grams information is captured through the usage of context window at previous recurrent steps. A sentence state node is added in order to integrate the global state. Both word sub-states and sentence state are updated recurrently by exchanging information between each other. Like CNN, the more recurrent steps are, the larger receptive field a word can reach. Therefore, long-range dependencies problem will be solved by more recurrent steps.

For evaluation, Zhang et al. verified S-LSTM performance on two NLP task: text classification and sequence labeling. They first chose Cornell movie review dataset [Pang et al., 2008] to investigate different configurations for S-LSTM and compare it with some common encoding structure such as CNN, BiLSTM and Transformer. They also further compared their S-LSTM with BiLSTM on Liu et al. [2017] 16 datasets of various topics. For Sequence labeling they investigated Part-of-Speech (POS) tagging task and Name Entity Recognition (NER) task on Penn Treebank [Manning, 2011] and CoNLL 2003 [Tjong Kim Sang and De Meulder, 2003] datasets. Results show that S-LSTM outperforms BiLSTM using the same number of parameters with less computation time.

[1] https://github.com/fastnlp/fastNLP

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

In this project, we reproduce the S-LSTM based on FastNLP, which is a modularized and extensible toolkit for Natural Language Processing developed by Fudan NLP group [2]. The original model is based on tensorflow and can be found at https://github.com/leuchine/S-LSTM. Our rebuilt model shows comparable results on Liu et al. 16 datasets, Penn Treebank POS tagging dataset and CoNLL 2003 dataset. We give up the movie review dataset because it lacks a standard split for train, development and test set, which prevents the performance comparison. We also try to apply the popular BERT model [Devlin et al., 2018] on one of these datasets to explore the performance improvement for this task.

The report is organized as follows: Section 2 introduces the background knowledge of the project, such as LSTM and BiLSTM, and the task involved. Section 3 describes S-LSTM in detail and the usages for the two task. Section 4 puts forward the experiment environment and settings for reproduction and Section 5 shows the results and analysis the difference between our results and the original ones. We also apply BERT to this task and show results to explore possible improvement. At Section 6 we conclude our work.

2 Background

LSTM is a variant for Recurrent Natural Network (RNN) to deal with the gradient vanishing and gradient exploding. It is composed of a cell with an input gate, an output gate and a forget gate to decide which information to remember and which to forget.

Given a sequence $\boldsymbol{s} = w_1, w_2, \ldots, w_n$, where w_i is the ith word and n is the sequence length. Our goal is to get a high-level representation $\boldsymbol{h}_i$ for each word and aggregate them into a sentence vector $\boldsymbol{s}$.

The base LSTM uses cell to keep track of the dependencies between the elements in the input sequence. The input gate $\boldsymbol{i}_t$ controls how much new information to remember, the forget gate $\boldsymbol{f}_t$ decides which a value remains in the cell and the output gate $\boldsymbol{o}_t$ controls the extent of the cell to compute the hidden state. The update gate $\boldsymbol{u}_t$ is the activation function to compute hidden state from the cell context. The calculating process of $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n$ can defined as follows:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}; \boldsymbol{x}_t] + \boldsymbol{b}_i)$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}; \boldsymbol{x}_t] + \boldsymbol{b}_f)$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}; \boldsymbol{x}_t] + \boldsymbol{b}_o)$$

$$\boldsymbol{u}_t = \tanh(\boldsymbol{W}_u[\boldsymbol{h}_{t-1}; \boldsymbol{x}_t] + \boldsymbol{b}_u)$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{u}_t$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

where $\boldsymbol{x}_t$ is the word embedding of w_t. $\boldsymbol{W}_i$, $\boldsymbol{W}_f$, $\boldsymbol{W}_o$, $\boldsymbol{W}_u$ are the weights for LSTM cell gates and $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, $\boldsymbol{b}_o$, $\boldsymbol{b}_u$ are the biases respectively. σ is the sigmoid function to map the gate value into $[0, 1]$.

BiLSTM The basic LSTM only remember the forward left-to-right information while BiLSTM can be more powerful by considering both sides. Each BiLSTM is composed of two different direction LSTMs, one for forward flow and the other for backward flow. The initial states of forward LSTM and backward LSTM are $\overleftarrow{\boldsymbol{h}}_0$ and $\overrightarrow{\boldsymbol{h}}_{t+1}$ respectively. Different weights and biases are used for backward process. Finally the hidden state of w_t is the concatenation of $\overleftarrow{\boldsymbol{h}}_t$ and $\overrightarrow{\boldsymbol{h}}_t$:

$$\boldsymbol{h}_t = [\overleftarrow{\boldsymbol{h}}_t; \overrightarrow{\boldsymbol{h}}_t]$$

Then the word representation $\boldsymbol{h}_t$ combine information of both sides, which models its syntactic and semantic context better.

[2] http://nlp.fudan.edu.cn/xpqiu/

[width=0.8]SLSTM.jpg

Figure 1: Sentence-State LSTM shown by [Zhang et al., 2018]

2.1 Classification Task

Text classification is a fundamental task in NLP, which classifies sentences or text documents into one or more defined categories according to its content. Common classification tasks include spam detection, sentiment analysis and tag prediction.

The goal of supervised classification task is to train a classifier using labeled data. Natural network is usually applied to find a high-level document representation. For a document $\boldsymbol{s} = w_1, w_2, \ldots, w_n$ with n words, it is first feed into the word embedding layer to get dense word vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$. After several nonlinear layers we will finally get hidden states for those words $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n$. Then different aggregate methods are applied to get a global representation $\boldsymbol{g}$ for the whole document.

The document representation is passed to a $softmax$ function to calculate the probability distribution of output class labels. The classifier will classify the document into the class with highest probability. The objective function is defined as standard log-likelihood loss between predictions and ground truths.

2.2 Sequence Labeling Task

Sequence labeling assigns a categorical label to each member of a sequence of observed values. Common examples include POS, NER and word segmentation. POS seeks to assign a part of speech tag to each word in a input sentence and NER tries to local name entity mentions in the text by classifying words into predefined categories such as person names, organizations, locations and so on.

Sequence labeling can be viewed as several independent classification tasks. However, in most of the time the prediction accuracy will be improved if nearby labels are taken into consideration during the classification. It is because that the current label is often closely related to its neighbors. For example, in POS task, a word may have different meanings in different context, which makes it to have different part-of-speech tags.

Due to complex dependencies between words, the most common hypothesis in use is the Markov assumption, i.e. that the choice of label for a particular word is directly dependent only on the immediately adjacent labels; hence the set of labels forms a Markov chain. Common models in use are Hidden Markov Model (HMM) and Conditional Random Field (CRF).

3 Model

Different from BiLSTM, S-LSTM assigns various sub-states for each word at different recurrent steps. Formally, the hidden state of the input sequence $\boldsymbol{s} = w_1, w_2, \ldots, w_n$ with special start w_0 and end w_{n+1} can be indicated as:

$$\boldsymbol{H}^t = < \boldsymbol{h}_0^t, \boldsymbol{h}_1^t, \ldots, \boldsymbol{h}_{n+1}^t, \boldsymbol{g}^t >$$

where $\boldsymbol{h}_i^t$ is the sub-state of w_i at time step t and $\boldsymbol{g}^t$ is current sentence state. The $\boldsymbol{h}^t$ represents the whole state at current recurrent step.

Thus, the recurrent process is no long along the input sequence but vertical to it, as 1 shows. At each recurrent step, information exchange is conducted between consecutive words in the sequence. And then the word sub-states are iteratively updated by the recurrent state transition process. For the initial state $\boldsymbol{H}^0$, $\boldsymbol{h}_i^0$ are $\boldsymbol{g}^0$ both set as zero. The state transition from $\boldsymbol{H}^{t-1}$ to $\boldsymbol{H}^t$ consists of sub-state transition from $\boldsymbol{h}_i^{t-1}$ to $\boldsymbol{h}_i^t$ for all word i and from sentence state $\boldsymbol{g}^{t-1}$ to $\boldsymbol{g}^t$.

S-LSTM use similar cell gate for the transition: cell $\boldsymbol{c}_i^t$ remembers the dependencies between words and gates $\boldsymbol{i}_i^t$, $\boldsymbol{f}_i^t$, $\boldsymbol{o}_i^t$ control information flow. It adds an extra sentence-controlled gate $\boldsymbol{s}_i^t$ to deal with the update of sentence state $\boldsymbol{g}^t$. In order to model ngrams information, $\boldsymbol{l}_i^t$ and $\boldsymbol{r}_i^t$ are used to control the content of previous cell $\boldsymbol{c}_{i-1}^{t-1}$ and next cell $\boldsymbol{c}_{i+1}^{t-1}$ used in current update for 1-word windows.

Imitating LSTM formula, S-LSTM can be defined as:

$$\boldsymbol{\xi}_i^t = [\boldsymbol{h}_{i-1}^{t-1}, \boldsymbol{h}_i^{t-1}, \boldsymbol{h}_{i+1}^{t-1}]$$
$$\boldsymbol{i}_i^t = \sigma(\boldsymbol{W}_i[\boldsymbol{\xi}_i^t; \boldsymbol{x}_i; \boldsymbol{g}^{t-1}] + \boldsymbol{b}_i)$$
$$\boldsymbol{l}_i^t = \sigma(\boldsymbol{W}_l[\boldsymbol{\xi}_i^t; \boldsymbol{x}_i; \boldsymbol{g}^{t-1}] + \boldsymbol{b}_l)$$
$$\boldsymbol{f}_i^t = \sigma(\boldsymbol{W}_f[\boldsymbol{\xi}_i^t; \boldsymbol{x}_i; \boldsymbol{g}^{t-1}] + \boldsymbol{b}_f)$$
$$\boldsymbol{r}_i^t = \sigma(\boldsymbol{W}_r[\boldsymbol{\xi}_i^t; \boldsymbol{x}_i; \boldsymbol{g}^{t-1}] + \boldsymbol{b}_r)$$
$$\boldsymbol{s}_i^t = \sigma(\boldsymbol{W}_s[\boldsymbol{\xi}_i^t; \boldsymbol{x}_i; \boldsymbol{g}^{t-1}] + \boldsymbol{b}_s)$$
$$\boldsymbol{o}_i^t = \sigma(\boldsymbol{W}_o[\boldsymbol{\xi}_i^t; \boldsymbol{x}_i; \boldsymbol{g}^{t-1}] + \boldsymbol{b}_o)$$
$$\boldsymbol{u}_i^t = \tanh(\boldsymbol{W}_u[\boldsymbol{\xi}_i^t; \boldsymbol{x}_i; \boldsymbol{g}^{t-1}] + \boldsymbol{b}_u)$$
$$\boldsymbol{c}_i^t = \boldsymbol{l}_i^t \odot \boldsymbol{c}_{i-1}^{t-1} + \boldsymbol{f}_i^t \odot \boldsymbol{c}_i^{t-1} + \boldsymbol{r}_i^t \odot \boldsymbol{c}_{i+1}^{t-1} + \boldsymbol{s}_i^t \odot \boldsymbol{c}_g^{t-1} + \boldsymbol{i}_i^t \odot \boldsymbol{u}_i^t$$
$$\boldsymbol{h}_i^t = \boldsymbol{o}_i^t \odot \tanh(\boldsymbol{c}_i^t)$$

where $\boldsymbol{\xi}^t$ is the concatenation of hidden vectors of a context window, and all of $\boldsymbol{W}$ and $\boldsymbol{b}$ are model parameters. The value of $\boldsymbol{i}_i^t$, $\boldsymbol{l}_i^t$, $\boldsymbol{f}_i^t$, $\boldsymbol{r}_i^t$, $\boldsymbol{s}_i^t$ are normalized by softmax and sum to 1.

The sentence state $\boldsymbol{g}^t$ is computed based on the sub-states of all words $i \in [0..n+1]$. It treats the average of all sub-states as input and assigns forget gates for each of them. Then the update is similar to basic LSTM. The process is defined as:

$$\overline{\boldsymbol{h}^t} = avg(\boldsymbol{h}_0^{t-1}, \boldsymbol{h}_1^{t-1}, \ldots, \boldsymbol{h}_{n+1}^{t-1})$$
$$\boldsymbol{f}_g^t = \sigma(\boldsymbol{W}_g[\boldsymbol{g}^{t-1}; \overline{\boldsymbol{h}^t}] + \boldsymbol{b}_g)$$
$$\boldsymbol{f}_i^t = \sigma(\boldsymbol{W}_f[\boldsymbol{g}^{t-1}; \boldsymbol{h}_i^{t-1}] + \boldsymbol{b}_f), i \in [0..n+1]$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{g}^{t-1}; \overline{\boldsymbol{h}^t}] + \boldsymbol{b}_o)$$
$$\boldsymbol{c}_g^t = \boldsymbol{f}_g^t \odot \boldsymbol{c}_g^{t-1} + \sum_i \boldsymbol{f}_i^t \odot \boldsymbol{c}_i^{t-1}$$
$$\boldsymbol{g}^t = \boldsymbol{o}^t \odot \tanh(\boldsymbol{c}_g^t)$$

where $\boldsymbol{f}_i^t$ represents the forget gate for each word cell $\boldsymbol{c}_i^{t-1}$, $i \in [0..n+1]$ and $\boldsymbol{f}_g^t$ is the forget gate for sentence cell $\boldsymbol{c}_g^t$. All of these gates are normalized. $\boldsymbol{W}$ and $\boldsymbol{b}$ is a separate set of parameters from that of word sub-states.

Compared with BiLSTM, S-LSTM assigns several sub-states $\boldsymbol{h}_i^t$ for each word w_i while BiLSTM only has two different directional hidden states $\overleftarrow{\boldsymbol{h}}_i$ and $\overrightarrow{\boldsymbol{h}}_i$. This means that S-LSTM has more powerful representative ability to capture semantic and syntactic features. Besides, BiLSTM passes information from one end of the sequence to the other and holds the global representation while ignores the local context for each word. S-LSTM uses windows to capture the ngrams context which may be important to the modelling of single word. With the recurrent step progressing, the receptive field of windows enlarges and finally can obtain 3-gram, 5-gram, 7-gram information and so on. In addition, the number of time steps scales with the size of the input in BiLSTM while empirically 3-6 steps is enough for S-LSTM. All sub-states can be computed simultaneously in S-LSTM compared with non-parallel BiLSTM. Both of them show that S-LSTM is more efficient than BiLSTM.

Window size The default S-LSTM only uses one word before and after as the context. The context window size is 1. This size can be easily extended to 2, 3 for larger context need. For this, the concatenation of hidden vectors $\boldsymbol{\xi}^t$ and forget gates for those words should be modified according the window size. For example, with the window size of 2, $\boldsymbol{\xi}^t = [\boldsymbol{h}_{i-2}^{t-1}; \boldsymbol{h}_{i-1}^{t-1}; \boldsymbol{h}_i^{t-1}; \boldsymbol{h}_{i+1}^{t-1}; \boldsymbol{h}_{i+2}^{t-1}]$ and two extra forget gates $\boldsymbol{ll}_i^t$, $\boldsymbol{rr}_i^t$ are used.

Sentence-level nodes The idea of S-LSTM is inspired by message passing over graphs [Murphy et al., 1999, Scarselli et al., 2009]. It is a simplified graph-structure neural models with linear dependencies. The sentence state is actually the super node in graph. Thus more sentence states can be added to enrich the parameter space, and the communication between those states will provide more global information. When one sentence-level node is used for classification outputs, the other sentence level node can serve as hidden memory units, or latent features.

3.1 Classification Model

The classification task uses the sentence state as the document representation and passed it through a $softmax$ layer:

$$\boldsymbol{y} = softmax(\boldsymbol{W}_c\boldsymbol{g} + \boldsymbol{b}_c)$$

where $\boldsymbol{y}$ is the probability distribution of output class labels and $\boldsymbol{W}_c$ and $\boldsymbol{b}_c$ are the parameters for the linear projection.

It has shown that attention mechanism is useful on most NLP tasks and it has been an effective method to aggregate hidden states in text classification [Yang et al., 2016]. The basic additive attention Bahdanau et al. [2014] is applied in our classification model:

$$\boldsymbol{g} = \sum_t \alpha_t \boldsymbol{h}_t$$

where

$$\alpha_t = \frac{\exp \boldsymbol{u}^T \boldsymbol{\epsilon}_t}{\sum_i \exp \boldsymbol{u}^T \boldsymbol{\epsilon}_i}$$

$$\boldsymbol{\epsilon}_t = \tanh(\boldsymbol{W}_\alpha \boldsymbol{h}_t + \boldsymbol{b}_\alpha)$$

Here $\boldsymbol{W}_\alpha$, $\boldsymbol{u}$ and $\boldsymbol{b}_\alpha$ are model parameters.

3.2 Sequence Labeling Model

Sequence labeling use the hidden state $\boldsymbol{h}_i$ to classify each word w_i into a label. Similar to text classification, the hidden state of each word is also passed through a $softmax$ layer to map it into the label distribution space. Besides, the label dependencies are also important for word prediction. Thus there is often a CRF layer on the top of the last hidden state layer [Huang et al., 2015, Ma and Hovy, 2016]:

$$P(\boldsymbol{Y}_1^n|\boldsymbol{h}, \boldsymbol{W}_s, \boldsymbol{b}_s) = \frac{\prod_{i=1}^n \psi_i(y_{i-1}, y_i, \boldsymbol{h})}{\sum_{\boldsymbol{Y}_1^{n'}} \prod_{i=1}^n \psi_i(y'_{i-1}, y'_i, \boldsymbol{h})}$$

$$\psi_i(y_{i-1}, y_i, \boldsymbol{h}) = \exp(\boldsymbol{W}_s^{y_{i-1}, y_i} h_i + b_s^{y_{i-1}, y_i})$$

where $\boldsymbol{W}_s^{y_{i-1}, y_i}$ and $b_s^{y_{i-1}, y_i}$ are parameters specific to two consecutive labels y_{i-1} and y_i.

4 Experiment

In this section, we describe our experiment model and our experiment settings in detail. Our code is based on FastNLP structure on Pytorch and our experiments are conducted on GeForce GTX 1080 Ti with 11GB memory.

4.1 Experiment Environment

We implement our model based on the structure of FastNLP. It is a NLP toolkit based on Pytorch which is more friendly to newbies of deep learning. The class $Trainer$ and $Tester$ wrap the training and testing process and users only need to feed hyperparameters. It also implements common NLP evaluation metrics such as accuracy and f1.

We first use the origin $read_csv$ function to read the csv fomart classification dataset and redefine a $Dataset$ to load our conll format files for NER and POS. Then we implement our S-LSTM as a Pytorch $nn.modular$. Finally we pass the model to the $Trainer$ with training hyperparameters and designate the optimizer and loss. The code will be released at https://github.com/brxx122/SLSTM later.

Pay attention that there are some small bugs in $SpanFPreRecMetric$ (in fastNLP.core.metrics Line 443) that the $number\_classes$ should be calculated before the $argmax$ on the last dimension of Tensor $pred$. This will trigger a value error during evaluation.

4.2 Experiment Details

We use the 16 datasets provided by Liu et al. [2017] as classification datasets. We follow the original split of Liu et al. to split train, development and test section with 1400, 200 and 400 examples respectively. The document labeled with 1 indicate it belongs to the category.

Penn Treebank POS tagging task uses [Manning, 2011] with 39831 for training, 1699 for development and 2415 for test. There are total 46 part-of-speech tags in this dataset and accuracy is used as evaluation metrics.

The standard CoNLL 2003 [Tjong Kim Sang and De Meulder, 2003] is the most widely used datasets for NER. It is labeled with BIOES tagging scheme [Ratinov and Roth, 2009] with 17 tags in our experiment. The train, development and test sets include 14987, 3466 and 3684 examples. Finally the micro-F1 is used to evaluate the model performance which first counts the total TF, FN and FP among all categories and then calculates the precision, recall and F.

Hyperparameter Our hyperparameters are a litter different from the original paper. We also initialize word embeddings using 300 dimensional uncased GloVe [Pennington et al., 2014]. The embedding is fixed during the training phrase because of the small size dataset. Dropout is applied separately to the word embedding and the hidden state after the recurrent process with a probability of 0.2. All model are optimized by Adam with $1e-3$ learning. We stop training after 20 epochs and choose the model with best performance on development set.

Table 1: The results of 16 sentiment classification tasks in Liu et al. [2017]. The bold is the best result without BERT, and the + indicates the improvement over the bold ones.

Dataset	**BiLSTM**	**2 BiLSTM**	**S-LSTM**	**Ours**	**BERT**
Camera	87.05	88.07	90.02	**90.23**	92.35(+2.12)
Video	84.73	85.23	86.75	**87.02**	91.06(+4.04)
Health	85.52	85.89	**86.50**	86.10	93.10(+6.60)
Music	78.74	80.45	**82.04**	81.76	88.70(+6.66)
Kitchen	82.22	83.77	84.54	**86.20**	90.95(+4.75)
DVD	83.71	84.77	85.52	**87.13**	87.87(+0.74)
Toys	85.72	85.82	85.25	**86.56**	91.83(+5.27)
Baby	84.51	85.45	86.25	**87.03**	93.58(+6.55)
Books	82.12	82.77	83.44	**86.22**	91.27(+5.05)
IMDB	86.02	86.55	87.15	**87.76**	90.85(+3.09)
MR	75.73	75.98	**76.20**	74.75	81.42(+5.22)
Appeal	86.05	86.35	85.75	**90.17**	90.32(+0.15)
Magazines	92.52	92.89	**93.75**	92.58	94.98(+1.23)
Electronics	82.51	82.33	83.25	**84.15**	91.45(+7.30)
Sports	84.04	84.78	**85.75**	83.26	91.25(+5.50)
Software	86.73	86.97	87.75	**88.77**	93.43(+4.66)
Average	84.01	84.64	85.38	**86.23**	90.90(+4.67)

5 Result

We compare our model with the original one on above three datasets. We also list results of BiLSTM of different layers and other available models. Most of them are provided by [Zhang et al., 2018] in their paper.

5.1 Classification

To simplify, we use the same hyperparameters during the whole classification. Our results show large variance on the single sub category so we train and test for ten times on all sub datasets and use the average accuracy for the final result, which is shown in 1. Our rebuilt model shows competitive results compared with the original S-LSTM with better performance on 11 of 16 datasets. The largest advantage appears on Appeal with 4.42% improvement. However, it also shows some shortage, such

as 2.49% decline on Sports. One of the possible reason is that we do not change our hyperparameters according to different datasets.

Table 2: The results of POS task at Penn Treebank POS tagging dataset

Model	**Accuracy**
Manning [2011]	97.28
Collobert et al. [2011]	97.29
Sun [2014]	97.36
Søgaard [2011]	97.50
Huang et al. [2015]	**97.55**
Ma and Hovy [2016]	**97.55**
Yang et al. [2017]	**97.55**
BiLSTM	97.35
2 BiLSTM	97.41
3 stacked BiLSTM	97.40
S-LSTM	**97.55**
Ours	97.12

Table 3: The results of NER task at CoNLL 2003 dataset

Model	**F1**
Collobert et al. [2011]	89.59
Passos et al. [2014]	90.90
Luo et al. [2015]	91.20
Huang et al. [2015]	90.10
Lample et al. [2016]	90.94
Ma and Hovy [2016]	91.21
Yang et al. [2017]	91.26
Rei [2017]	86.26
Peters et al. [2017]	**91.93**
BiLSTM	90.96
2 stacked BiLSTM	91.02
3 stacked BiLSTM	91.06
S-LSTM	91.57
Ours	90.35

5.2 Sequence labeling

We follow the hyperparameters provided by Zhang et al. [2018] and set the maximum recurrent number 7 for POS and 9 for NER. The results are shown in 2 and 3. Unfortunately, our model does not perform well on both of two task. We only get 90.12% on POS and 90.35% on NER, worse than most of the existing work. We analyze the results and find that it is because that we only use word and word embeddings as input while most of other work takes characters and their embeddings into consideration to deal with the Out-of-Vocabulary(OOV) problem. It will cost more time to train and test but lead to better results on these tasks.

5.3 Improvement

Apart from the comparison between our rebuilt S-LSTM and the original one, we also apply the most popular model BERT [Devlin et al., 2018] in our experiment. BERT is a pre-training language model of deep bidirectional transformers and has shown excellent performance on plenty of NLP tasks, such as question answering and language inference. Devlin et al. [2018] trained a multi-layer bidirectional Transformer encoder [Vaswani et al., 2017] on BooksCorpus and Wikipedia. They defined two novel pre-training tasks: masked language model and next sentence prediction to help training. The pre-trained model can be used in two ways: feature-based for word embedding and

fine-tuning to transfer. Google released two types of BERT with small and large set of parameters, called base BERT and large BERT respectively.

With the help of PyTorch reimplementation of BERT [3], we use the base BERT in a feature-based approach by generating contextual representations for words as embeddings. After BERT we pass the embeddings to a MLP layer and then classify the output with $softmax$. We do not add complex modular after it because empirically BERT itself is powerful enough for classification. The results on 16 datasets are also shown in 1.

In the last row we can see the huge improvement BERT brings. However, it still has a lot of room for improvement since none of the accuracy is more than 95%. The performance can be further improve by fine-tuning BERT instead of fixing the weight or change the base BERT model into large, but we give up for the limited GPU memory.

6 Conclusion

In this project, we rebuild Sentence-State LSTM model in Pytorch and apply it on two tasks. The results on three datasets show that our S-LSTM has comparable ability with the original one. Besides we also take a look at the popular model BERT to see how it performs. The usage of FastNLP accelerates our coding process, but the shortage of API documents and some small unrevealed bugs also confuse us and cost our more time.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12 (Aug):2493–2537, 2011.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. URL `http://arxiv.org/abs/1810.04805`.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. *arXiv preprint arXiv:1704.05742*, 2017.

Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. Joint entity recognition and disambiguation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 879–888, 2015.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International conference on intelligent text processing and computational linguistics*, pages 171–189. Springer, 2011.

[3] https://github.com/huggingface/pytorch-pretrained-BERT

Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

Alexandre Passos, Vineet Kumar, and Andrew McCallum. Lexicon infused phrase embeddings for named entity resolution. *arXiv preprint arXiv:1404.5367*, 2014.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*, 2017.

Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

Marek Rei. Semi-supervised multitask learning for sequence labeling. *arXiv preprint arXiv:1704.07156*, 2017.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Anders Søgaard. Semisupervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 48–52. Association for Computational Linguistics, 2011.

Xu Sun. Structure regularization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 2402–2410, 2014.

Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. 2017. ISSN 0140-525X. doi: 10.1017/S0140525X16001837. URL `http://arxiv.org/abs/1706.03762`.

Zhilin Yang, Ruslan Salakhutdinov, and William W Cohen. Transfer learning for sequence tagging with hierarchical recurrent networks. *arXiv preprint arXiv:1703.06345*, 2017.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

Yue Zhang, Qi Liu, and Linfeng Song. Sentence-State LSTM for Text Representation. 2018. ISSN 03610128. doi: 10.2113/gsecongeo.94.3.423. URL `http://arxiv.org/abs/1805.02474`.