A Neural Network Parser

Wei Lu, luwei@brandeis.edu

December 19, 2018

In this project, I used the encoder-decoder model to train a neural network parser using the Penn TreeBank data. The implementation of the model is based on the NMT: TensorFlow Neural Machine Translation (`https://github.com/tensorflow/nmt`). The output is evaluated by EVALB.

1 Code Structure

The code is composed mainly of three parts: preprocessing, training, and postprocessing.

1.1 Preprocessing

This step is to prepare the training data for the encoder-decoder model. Since we only care the tags and parentheses in the parsing tree, we drop those terminal words. For example, if we have a raw training sentence:

(S (INTJ (RB No)) (, ,) (NP-SBJ (PRP it)) (VP (VBD was) (RB n't) (NP-PRD (NNP Black) (NNP Monday))) (. .))

We drop the terminal words and corresponding parentheses, meanwhile we add a corresponding tag after each closing parenthesis. Then we will have:

(S (INTJ RB)INTJ , (NP-SBJ PRP)NP-SBJ (VP VBD RB (NP-PRD NNP NNP)NP-PRD)VP .)S

This is implemented by the following functions:

```
def count_right_parentheses(tag):
```

This functions counts the the number of closing parentheses, which determines how many corresponding close tags we should add.

```
def preprosessing(sent):
```

This is the function to implement the desired output above.

In the training tags, there are tags with attached numbers, for example, ")WHNP-66, (WHNP-6, (WHNP-95", which is used to distinguish the same nested tags. We drop those numbers during training, which will not only decrease the size of the target vocabulary, but also increase the accuracy of our prediction. We can then use these processed data to do the training.

1.2 Training

I use the TensorFlow Neural Machine Translation to do the training, the calling function and parameters are as follows:

```
python -m nmt.nmt \

    --attention=scaled_luong \

    --encoder_type=bi

    --src=vi --tgt=en \

    --vocab_prefix=/home/f/guest/luwei/tmp/nmt_data/vocab  \

    --train_prefix=/home/f/guest/luwei/tmp/nmt_data/train \

    --dev_prefix=/home/f/guest/luwei/tmp/nmt_data/tst2012  \

    --test_prefix=/home/f/guest/luwei/tmp/nmt_data/tst2013 \

    --out_dir=/home/f/guest/luwei/tmp/nmt_attention_model \

    --num_train_steps=12000 \

    --steps_per_stats=100 \

    --num_layers=2 \

    --num_units=128 \

    --dropout=0.2 \

    --metrics=bleu
```

1.3 Postprocessing

After we obtain the output from the training, we do some post-processing to make the results readable by EVALB, that is, put no more than two items within a pair of parentheses. This is implemented by the following two functions:

```
def parentheses_match(sent):
```

This function makes the open and close parentheses match. If there are more open parentheses than close ones, we add more corresponding close parentheses. Otherwise, we drop the extra close parentheses.

```
def post_process(sent, terminal_words):
```

This function put the terminal words back into our output parsing tree, and add parentheses with the POS tags and terminal words, making sure that there are no more than two items within a pair of parentheses.

2 Model Description

In this section we describe the model on input feature design, model architecture, and hyperparameters we experimented with.

2.1 Feature Design

We use the input words as the feature. Given the categorical nature of words and tags, the model must first look up the source and target embeddings to retrieve the corresponding word representations. For this embedding layer to work, a vocabulary is first chosen for each language. We look through all the words and tags to build our source and target vocabulary. Moreover, we add three more words in the vocabulary: ⟨unk⟩ for the unknown words, ⟨s⟩ for the start of the decoding sequence, and ⟨/s⟩ for the end of the decoding sequence. The embedding weights, one set per language, are usually learned during training.

2.2 Model Architecture

we consider as examples a deep multi-layer RNN which is unidirectional and uses LSTM as a recurrent unit. We show an example of such a model in the figure below (from nmt: `https://github.com/tensorflow/nmt/blob/master/nmt/g3doc/img/seq2seq.jpg?raw=true`). In this example, we build a model to translate a source sentence "I am a student" into a target sentence "Je suis tudiant". At a high level, the NMT model consists of two recurrent neural networks: the encoder RNN simply consumes the input source words without making any prediction; the decoder, on the other hand, processes the target sentence while predicting the next words.

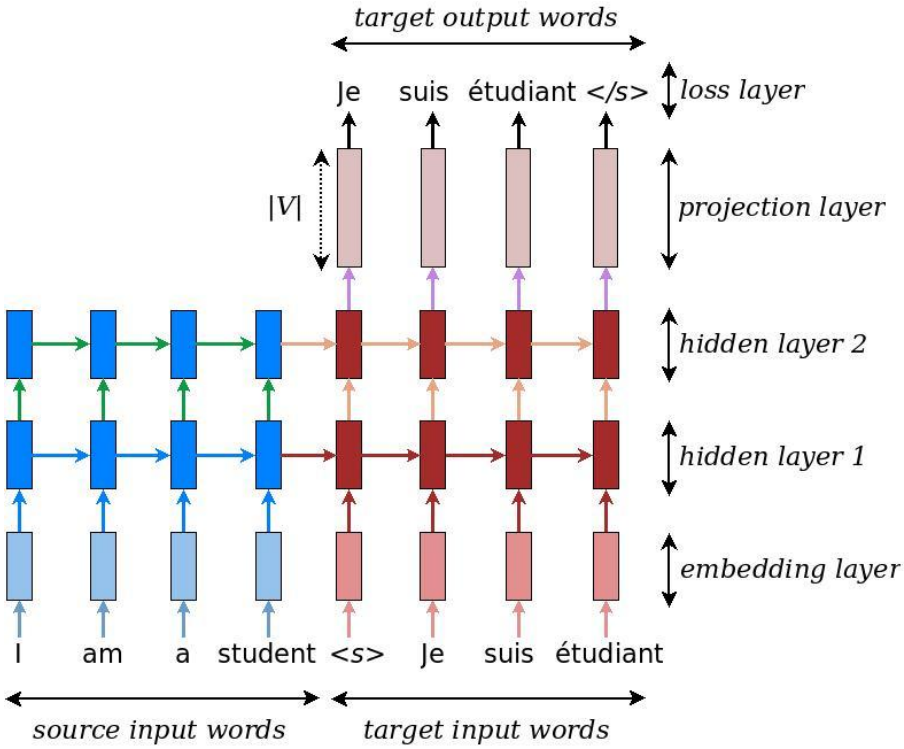

Figure 1: Neural machine translation

Besides the unidirectional RNN, we can use Bidirectional RNNs on the encoder side, which generally gives better performance (with some degradation in speed as more layers are used).

2.3 Hyperparameters

I did experiments about attention mechanism, biLSTM and beam search. The hyperparameters and results are as follows. (The detailed evaluation results are attached in the appendix at the end of the report)

2.3.1 No Attention VS Attention

This is implemented by adding the parameter: –attention=scaled_luong

```
No attention vs attention results (both with unidirectional LSTM):
Tagging accuracy without attention: 59.49
Tagging accuracy with attention (with LSTM): 88.46
```

2.3.2 Unidirectional LSTM VS BiLSTM

This is implemented by adding the parameter: –encoder_type=bi

```
LSTM VS biLSTM results (both with attention):
Tagging accuracy with LSTM: 88.46
Tagging accuracy with biLSTM: 91.29
```

2.3.3 Greedy Search VS Beam Search

The idea of beam search is to better explore the search space of all possible translations by keeping around a small set of top candidates as we translate. This is implemented by adding the parameter: –infer_mode=beam_search, –beam_width=10. Generally it can further boost performance. However, in my experiment (with beam width 10), to my surprise, the result is not as good as greedy search.

```
Greedy search VS beam search (both with attention and biLSTM):
Tagging accuracy with greedy search: 91.29
Tagging accuracy with beam width 10 search: 89.49
```

3 Experiments And Results With Different Training Data Sizes

Under the setting of attention mechanism, biLSTM and greedy search, I experimented with training data size of 5000, 15000, and 39832 (all sentences). The results, as shown below, follows the general belief that the larger the training size, the better the performance.

```
Experiments with different training data sizes:
Tagging accuracy of size 5000: 38.61
Tagging accuracy of size 15000: 41.54
Tagging accuracy of size 39832: 91.29
```

Since EVALB doesn't regard ',' , '-NONE-' , etc as valid tags, many sentences suffer the problem of "Length unmatch" and "Words unmatch", so they are not counted in the final evaluation.

4 Best Experimental Performances

As shown in the previous sections, the best performance is 91.29 of tagging accuracy, which is obtained under the setting of attention mechanism, biLSTM and greedy search, with 39832 (all) training sentences. As we can see, attention mechanism improves the performance a lot (from 59.49 to 88.46), and biLSTM boost the result still a bit more (from 88.46 to 91.29).

5 Appendix: Experimental Results (all experiments are done with all training sentences)

Unidirectional LSTM without attention mechanism and greedy search (Tagging accuracy: 59.49):

```
=== Summary ===

-- All --
Number of sentence        =   2416
Number of Error sentence  =   1888
Number of Skip  sentence  =      0
Number of Valid sentence  =    528
Bracketing Recall         =  51.22
Bracketing Precision      =  49.47
Bracketing FMeasure       =  50.33
Complete match            =  28.41
Average crossing          =   2.88
No crossing               =  46.21
2 or less crossing        =  63.26
Tagging accuracy          =  59.49

-- len<=40 --
Number of sentence        =   2245
Number of Error sentence  =   1717
Number of Skip  sentence  =      0
Number of Valid sentence  =    528
Bracketing Recall         =  51.22
Bracketing Precision      =  49.47
Bracketing FMeasure       =  50.33
Complete match            =  28.41
Average crossing          =   2.88
No crossing               =  46.21
2 or less crossing        =  63.26
Tagging accuracy          =  59.49
```

Unidirectional LSTM with attention mechanism and greedy search (Tagging accuracy: 88.46):

```
=== Summary ===

-- All --
Number of sentence        =   2416
Number of Error sentence  =   1081
Number of Skip  sentence  =      0
Number of Valid sentence  =   1335
Bracketing Recall         =  70.03
Bracketing Precision      =  71.86
Bracketing FMeasure       =  70.93
Complete match            =  31.39
Average crossing          =   2.32
No crossing               =  55.21
2 or less crossing        =  71.16
Tagging accuracy          =  88.46
```

```
-- len<=40 --
Number of sentence        =   2245
Number of Error sentence  =    913
Number of Skip  sentence  =      0
Number of Valid sentence  =   1332
Bracketing Recall         =  70.23
Bracketing Precision      =  71.94
Bracketing FMeasure       =  71.07
Complete match            =  31.46
Average crossing          =   2.31
No crossing               =  55.33
2 or less crossing        =  71.32
Tagging accuracy          =  88.59
```

BiLSTM with attention mechanism and greedy search (Tagging accuracy: 91.29, best performance):

```
=== Summary ===

-- All --
Number of sentence        =   2416
Number of Error sentence  =    840
Number of Skip  sentence  =      0
Number of Valid sentence  =   1576
Bracketing Recall         =  75.63
Bracketing Precision      =  76.45
Bracketing FMeasure       =  76.04
Complete match            =  29.25
Average crossing          =   2.05
No crossing               =  57.74
2 or less crossing        =  76.33
Tagging accuracy          =  91.29

-- len<=40 --
Number of sentence        =   2245
Number of Error sentence  =    678
Number of Skip  sentence  =      0
Number of Valid sentence  =   1567
Bracketing Recall         =  75.93
Bracketing Precision      =  76.63
Bracketing FMeasure       =  76.28
Complete match            =  29.42
Average crossing          =   2.01
No crossing               =  58.01
2 or less crossing        =  76.71
Tagging accuracy          =  91.44
```

BiLSTM with attention mechanism and width 10 beam search (Tagging accuracy: 89.49):

```
=== Summary ===

-- All --
Number of sentence        =   2416
Number of Error sentence  =    939
```

```
Number of Skip  sentence  =      0
Number of Valid sentence  =   1477
Bracketing Recall         =  72.54
Bracketing Precision      =  76.10
Bracketing FMeasure       =  74.28
Complete match            =  34.87
Average crossing          =   2.04
No crossing               =  63.44
2 or less crossing        =  75.83
Tagging accuracy          =  89.49

-- len<=40 --
Number of sentence        =   2245
Number of Error sentence  =    769
Number of Skip  sentence  =      0
Number of Valid sentence  =   1476
Bracketing Recall         =  72.60
Bracketing Precision      =  76.10
Bracketing FMeasure       =  74.31
Complete match            =  34.89
Average crossing          =   2.04
No crossing               =  63.48
2 or less crossing        =  75.88
Tagging accuracy          =  89.57
```