Assignment 3: Transition Parsing with Neural Networks

Niranjan Balsubramanian and Jun Kang

CSE 628 Spring 2018

1 Due Date and Collaboration

- The assignment is due on Apr 7(Sat), 2018 at 11:59 pm. We will accept late submissions until Apr 9(Mon), 2018 11:59 pm with a penalty of 10 points per day. (Out of 100 points)
- You can collaborate to discuss ideas and to help each other for better understanding of concepts and math.
- You should NOT collaborate on the code level. This includes all implementation activities: design, coding, and debugging.
- You should NOT not use any code that you did not write to complete the assignment.

2 Transition Parsing System

In this assignment you will implement the Neural Network based Transition Parsing system described in [Chen and Manning, 2014]. This assignment consists of following three components:

2.1 Parsing Algorithm

In ParsingSystem.py, there is an overall wrapper that begins with the initial state, and asks the classifier to provide a distribution over the available actions, selects the next available legal action, and then changes the state by applying the action.

You will be implementing the ARC standard system rather than the eager version. This is a slight modification with only three types of actions (shift, left, right) instead of the four. The paper[Nivre, 2004] describes in detail.

2.2 Neural Network Classifier using TensorFlow

You will implement the training and test functionality for the transition classifier. You will generate the features given the (sentence, tree) pairs and write the loss function to use for training the network. (in DependencyParser.py)

2.3 Experiments

Refer to section 3 Experiments for details.

3 Experiments

You will investigate the impact of different parts of this system. Note this is the empirical part of NLP. You are asked here to try a few different configurations that are likely to affect performance. This should also give you a sense for how important each of the following aspects are to the overall performance.

1. **Number of hidden layers** – The original paper proposed only one hidden layer. Experiment with two and three hidden layers.
2. **Capturing interactions** – The paper used a cube non-linearity to account for interactions between different combinations of token features.
   (a) Try sigmoid, tanh, and ReLU.
   (b) Try using the cube non-linearity but avoid connections that go across (words, pos, deps). Have three separate parallel hidden layers, one for combining word embeddings, one for POS, and one for deps.
   (c) Effect of fixing Word, POS and Dep Embeddings – The paper allowed POS and Dep embeddings to be learnt and the Word embeddings to be modified via backprop. You can fix these (use a bit vector to uniquely index each POS and Dep category) and use the pre-trained word embeddings without change, and see how your performance varies.
   (d) Best Configuration – Use the dev set to find the best model. Note that you may have to run many random seed experiments.
   (e) Gradient clipping – In the training phase, a technique called "gradient clipping" is introduced. Explain in your own words, what gradient clipping is and why it is useful in general. Remove the gradient clipping section of the code and report the performance of the best configuration without gradient clipping. What did you observe?

4 Implementation Details and Tips

- You will use the embeddings you learned from the assignment 1 as pre-trained embeddings. For those of you who don't have the embeddings, consult with TA.

- Most of the system is already written and you are asked to write some key parts. So, don't panic!
- Even so, it would take quite time to 1) understand the system design (from the cited papers and codes), 2) implement the system, and 3) do experiments.
- **Computation Overhead and Seawulf**
  - Running this system will consume large amount of memory and computation power.
  - With one GPU, it took about 11 mins to finish training and testing. (15 mins without GPU).
  - I stopped running this on my laptop when I saw it was taking more than 2GB of memory. (And the fans were at their full speed.)
  - You might develop this on your laptop, but when you begin to experiment, you might want to use Seawulf. If you haven't tested your account or haven't tried to run anything on it, you'd better to do so as early as possible.
- More detailed information can be found in README file in the package. You can consult the paper for additional details.

5 What should you turn in?

Prepare and submit a single zip file with following items:
filename: ⟨SBU_ID⟩.zip (ex. 10111111.zip)

- Report – Include a table that contains results of each of the experiment above. Present an analysis to explain your findings/observations. Please include a section that mentions the settings of the best configuration.
- Predictions on the test data - One prediction file (results_test.conll) from the best model on the test data.
- Codes - Leave comments in your code that explains what you are trying to do. Include following files:
  - DependencyParser.py
  - ParsingSystem.py
  - Configuration.py

6 Grading Rubric

- 50 points – Implementation
- 30 points – Experiments
- 10 points – Analysis of the results
- 10 points – Evaluation of the test predictions. (Relative grading)

* 1 point will be deducted for each case of not following the specified file names.

References

[Chen and Manning, 2014] Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

[Nivre, 2004] Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.