Learning PyTorch Through A Neural Dependency Parsing Exercise

David Jurgens
School of Information
University of Michigan
jurgens@umich.edu

Abstract

Dependency parsing is increasingly the popular parsing formalism in practice. This assignment provides a practice exercise in implementing the shift-reduce dependency parser of Chen and Manning (2014). This parser is a two-layer feed-forward neural network, which students implement in PyTorch, providing practice in developing deep learning models and exposure to developing parser models.

1 Introduction

Deep learning methods are ubiquitous in nearly all areas of NLP. However, some applications for these models require extensive training or data that make implementing the model in a classroom infeasible without additional computational support. This homework introduces a simple-yet-powerful network for performing dependency parsing using the shift-reduce parser of Chen and Manning (2014). This three-layer network is efficient to train, making it suitable for development by all students, exposes students to dependency parsing, and helps connect how a neural network can be used in practice. Through the assignment, students implement the network, the training procedure, and explore how the parser operates.

2 Design and Learning Goals

This exercise is designed for an advanced undergraduate or graduate-level course in NLP. The material is appropriate for students who have previously covered simple neural networks and are learning about dependency parsing. The exercise contains extensive scaffolding code that simplifies the training process by turning the CoNLL treebank data (Nivre et al., 2007) into training examples for the students and computing the unlabeled attachment score (UAS) for evaluating the model. The technical depth of the material is likely too involved for an Applied NLP or Linguistics-focus setting; the material could be re-purposed for an early exercise in a Machine-learning focused course with the addition of more content on network design. Students typically have three weeks to complete the assignment, with the majority finishing the main tasks within a week. Through doing the exercise, students practice building neural models and understanding how the core training procedure is implemented in PyTorch (Paszke et al., 2019, e.g., what is a loss function and an optimizer), which enables them to design, extend, or modify a variety of PyTorch implementations for later assignments and course projects.

The assignment has the following three broad learning objectives. First, the exercise is an introduction to developing neural models using the PyTorch library. The relatively simple nature of the network reduces the scope of design (e.g., compared with implementing an RNN or LSTM), which allows students to understand how to construct a basic network, use layers, embeddings, and loss functions. Further, because the model can be trained efficiently, this allows students to complete the entire assignment on a laptop that is several years old, which reduces the overhead of needing students to gain access to GPUs or more advanced computing. Through the exercise, students experiment with changing different network hyperparameters and designs and measuring their effect on performance. These simple modifications allow students to build confidence and gain intuition on which kinds of modifications may improve performance—or dramatically slow training time.

Second, students gain familiarity with how to use pre-trained embeddings in downstream applications. The dependency parser makes use of these embeddings for its initial word representations and the assignment provides an optional exercise to have students try embeddings from different sources (e.g., Twitter-based embeddings) or no pre-training at all in order to see how these affect performance and convergence time. This learning objective helps bridge the conceptual material to later pre-trained language models like BERT if they have not been introduced earlier.

Third, students should gain a basic familiarity with dependency parsing and how a shift-reduce parser works. Shift-reduce parsing is a classic technique (Aho and Ullman, 1973) and has been widely adopted for multiple parsing tasks beyond syntax, such as semantic parsing (Misra and Artzi, 2016) or discourse parsing (Ji and Eisenstein, 2014). This assignment helps students understand the basic structures for parsing (e.g., the stack and buffer) to see how neural approaches can be used in practice. The concepts in the homework are connected to textbook material in Chapter 14 of *Speech & Language Processing* (Jurafsky and Martin, 2021, 3rd ed.), which provides additional examples and definitions.

3 Homework Description

This homework has students implement two key components of the Chen and Manning (2014) parser in PyTorch, using extensive scaffolding code to handle the parsing preparation. First, students implement the feed-forward neural network, which requires using three common elements of neural networks in NLP: multiple layers, embeddings, and a custom activation function. These elements provide conceptual scaffolding for later implementations on attention and recurrent layers in networks. Second, students implement the main training loop, which requires students to understand how the loss is computed and gradient descent is applied. This second step is found in nearly all code using networks built from scratch (or built upon pre-trained parameters) and enables students to learn this process in a simplified setting for use later. These two components are broken into multiple discrete steps to help students figure out where to start in the code.

The second part of the homework has students explore the parser in two ways. First, students examine the parsing data structures and report the full parse of a sentence of their choice; this exploration has students consider the steps required for a successful parse. As a part of this exploration, students are required to report a parse that is incorrect; this latter task requires students to understand what is a correct dependency parse and diagnose what steps the parser has taken to introduce the error. In some iterations, we have asked the students to report on an error introduced from a non-projective parse of their choice, though some students found it difficult to come up with an example of their own. Second, students are asked to extend the network in some way of their choice (e.g., add a layer or add dropout). This extension helps introduce additional design components and build intuition.

4 Potential Extensions

Prior parts of the course examine word vectors and this parsing exercise includes an optional extension to allow students to see their effect in practice. Here, we provide students with multiple pre-trained vectors from different domains (e.g., Twitter and Wikipedia) and ask them to report on convergence times and accuracy. Students may also optionally freeze these embeddings to test how well their information can generalize. Since training data are known to contain biases that affect downstream performance (e.g., Garimella et al., 2019), one additional extension could be to test how particular embeddings perform better or worse on parsing text from specific groups.

After implementing the model, the assignment has students explore the parsing outputs and intermediate state, which provides some grounding for how shift-reduce works in practice. However, due to the focus on learning PyTorch, the parsing component of the exercise is less in-depth; a more parsing-focus variant of this assignment could have students perform the CoNLL-to-training-data conversion in order to see how dependency trees can be turned into a sequence of shift-reduce operations and how such a process introduces errors for non-projective parses.

5 Reflection on Student Experiences

In two iterations of the homework, students have reported the exercise helped demystify deep learning and make the concepts accessible. Once the model was implemented and working, some students were surprisingly active in trying different model designs; these students reported feeling excited that making these simple extensions could be so easy, which encouraged them to try implementing deep learning models in their course projects.

The initial version of this homework did not include any parsing introspection. Students expressed feeling like the homework was more about building a network than learning about parsing. As a result, the second iteration added more diagnostics for students to see what the parser is doing and the exploration component. This addition was intentionally simple to avoid substantially expanding the scope of the assignment. However, adding more parsing-oriented tasks remains an active area of development for this homework.

References

Alfred V Aho and Jeffrey D Ullman. 1973. *The theory of parsing, translation, and compiling*, volume 1. Prentice-Hall Englewood Cliffs, NJ.

Danqi Chen and Christopher D Manning. 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.

Aparna Garimella, Carmen Banea, Dirk Hovy, and Rada Mihalcea. 2019. Women's syntactic resilience and men's grammatical luck: Gender-bias in part-of-speech tagging and dependency parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3493–3498.

Yangfeng Ji and Jacob Eisenstein. 2014. Representation learning for text-level discourse parsing. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 13–24.

Dan Jurafsky and James H. Martin. 2021. *Speech & Language Processing*, 3rd edition. Prentice Hall.

Dipendra Misra and Yoav Artzi. 2016. Neural shift-reduce ccg semantic parsing. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 1775–1786.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The conll 2007 shared task on dependency parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 915–932.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.