A Dependency Parser Using Neural Networks for NLTK

Tsolak Ghukasyan
Faculty of Computer Science, Department of Software Engineering, Higher School of Economics
Moscow, Russia
Email: tsggukasyan@edu.hse.ru

***Abstract*— The Natural Language Toolkit (NLTK) is a leading suite of libraries for natural language processing, and with the rise of artificial intelligence, its popularity has only increased in the recent years. Since existing parsers in NLTK are either outdated or not implemented in Python, its primary programming language, this project proposes an implementation of a fast and accurate dependency parser to include in the toolkit. The parsing algorithm created by Danqi Chen and Christopher D. Manning introduces several groundbreaking concepts, and therefore is of great scientific value. The parser employs an efficient new way of feature representation, particularly for part-of-speech tags and arc labels. Another noteworthy innovation is the use of an artificial neural network classifier with cube activation function instead of the commonly used tanh and sigmoid functions.**

***Index Terms*— dependency parsing, neural networks, NLTK.**

I. Introduction

Artificial intelligence has seen resurgence in the past few years. We encounter its applications constantly in social networks, search engines, and recommender systems. This success can be attributed to the advances in artificial neural networks and natural language processing (NLP), a cornerstone of artificial intelligence. NLP enables computers to derive meaning from natural language input, providing techniques for machine translation, speech and optical character recognition.

Dependency parsing is an essential part of working with human language data. Machine translators use parsing algorithms to determine the dependencies in sentences and extract information from texts. Considering its significant role in information extraction, computer scientists and linguists have been actively working on the development of efficient parsers, with faster and more accurate algorithms being introduced every year.

NLTK is a leading open-source platform in the Python programming language, providing instruments for processing natural language. Despite many similar projects, NLTK stands out because of its reputation for combining high-quality algorithms with educational material and tutorials. The platform is widely used in NLP studies and the development of machine learning algorithms by researchers around the world. Since the code of the toolkit is open and fastidiously documented, the platform serves an important educational purpose and is the ultimate guide for novices in the field.

As a result of the rapidity with which artificial intelligence is developing, state-of-the-art algorithms are significantly faster and more accurate than the current transition parser of NLTK. In spite of the presence of a wide range of parsers in NLTK, namely Malt and Stanford parsers, and phrase chunkers, those parsers are either interfaces to implementations outside the platform or their function is primarily educational, hence cannot be employed in serious NLP studies.

To address the absence of an up-to-date parser in NLTK, this project implements a dependency parser introduced by Chen and Manning [1], and incorporates it into the platform. Taking into account the recent success of artificial neural networks, the parser predicts transitions using a network trained on a large dataset of manually tagged dependency trees. During the experiments carried out on English treebank datasets, the algorithm demonstrated an unlabeled attachment score of 92% and a labeled attachment score of 90%. The biggest breakthrough is in feature extraction speed. This parser processes texts up to 100 times faster than existing established solutions.

In this paper, the theoretical basis for the parsing algorithm is provided in the second section. The neural network classifier's architecture and the innovative feature representation methods are presented in the third section. The fourth section discusses relevant research in the field, describing the distinguishing differences of the proposed algorithm.

II. Dependency Parsing

The goal of dependency parsing is to find the dependencies between individual words in a sentence. Conventional approaches employ transition systems to derive those dependencies. Starting from an initial

configuration, transition-based parsers one by one process the words, predicting the next transition after each step. The dependency tree of the sentence is considered complete when the parser reaches a terminal configuration.

This parser is based on the arc-standard transition system introduced by Joakim Nivre [2]. The initial configuration includes a stack with the single element ROOT, a buffer containing the words of the sentence, and an empty dependency tree. To make a transition to the next state, there are three possible parse actions:

- RIGHT-ARC action: adds a dependency s2 → s1 to the parse tree and removes s2 from the stack provided that the stack contains at least two elements.
- LEFT-ARC action: adds a dependency s1 → s2 to the parse tree and removes s1 from the stack provided that the stack contains at least two elements.
- SHIFT action: shifts the top of the buffer to the stack provided that the buffer is not empty.

Above, s1 and s2 are respectively the first and second elements on the stack's top. At each step, a classifier is used to predict the correct transition, which is then applied and the configuration is updated. The parsing of the sentence completes when the buffer and the stack are empty and the dependencies between all words are stored in the tree.

For classification, this parser uses an artificial neural network. The network classifies configuration steps based on a wide range of features. From the configuration, the following information is obtained:

- all the tokens and their part of speech (e.g., goes / VBZ);
- the dependency label between the token and its head (e.g., NSUBJ);
- the token's position on the buffer or stack, or whether it has already been added to the tree.

Some common approaches retrieve the combination of words, tags, and labels of up to three stack and buffer elements as indicator features. However, these features have serious drawbacks. Specifically, they suffer from sparsity, incompleteness, and are expensive to compute. In practice, generation of such features consumes over 95% of the parsing time (Chen and Manning, 2014). To overcome this problem, the words, part-of-speech tags and dependency labels in this project are represented in the form of embedding vectors.

III. Neural Network Classifier

This parser uses an artificial neural network to predict transitions. The network consists of three layers: input, hidden, and output. The input layer accepts the vector containing lexical and part-of-speech related features derived from the configuration. Then, the hidden layer is computed from the input via a cube activation function. In the network's final layer, the softmax function is applied to the values from the hidden layer.

Each word is represented as a d-dimensional vector $e_i^w \in \mathbb{R}^d$ stored in an embedding matrix $E^w \in \mathbb{R}^{d \times Nw}$ (N_w is the size of the dictionary). Similarly, part-of-speech tags and dependency labels are mapped to a d-dimensional vector space, with respective embedding matrices $E^t \in \mathbb{R}^{d \times Nt}$ and $E^l \in \mathbb{R}^{d \times Nl}$ (N_t and N_l are the number of distinct part-of-speech tags and dependency labels).

For each type of information (word, tag, or label), a set of elements is selected based on their stack and buffer positions, denoted S^w, S^t, S^l respectively. During classification, the embeddings of the elements from S^w, S^t, S^l are added to the input layer of the network. Denoting n_w, n_t, n_l as the number of chosen elements of each type, we add $x^w = [e^w_{w_1}; e^w_{w_2}; \ldots e^w_{w_n}]$ to the input layer, where $S^w = \{w_1, \ldots w_{n_w}\}$. Likewise, the part-of-speech tag features x^t and dependency label features x^l are added to the input layer.

The hidden layer with d_h nodes is computed from the input layer through a cube activation function:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

where $W_1^w \in \mathbb{R}^{dh \times (d \cdot nw)}$, $W_1^t \in \mathbb{R}^{dh \times (d \cdot nt)}$, $W_1^l \in \mathbb{R}^{dh \times (d \cdot nl)}$. $b_1 \in \mathbb{R}^{dh}$ is the bias.

The third layer is a softmax function that computes class probabilities:

$$p = \text{softmax}(W_2 h)$$

where $W_2 \in \mathbb{R}^{|T| \times dh}$.

Following (Zhang and Nivre, [3]), the following 18 elements are selected for S^w:

(1) The top 3 tokens on the stack and buffer;
(2) The first and second leftmost, rightmost children of the stack's top two tokens;
(3) The leftmost of leftmost and rightmost of rightmost children of the stack's top two tokens.

Correspondingly, S^t also contains 18 elements, the parts of speech of the aforementioned words. Meanwhile, S^l stores the dependency labels, excluding the labels of the 6 tokens on the stack and buffer. One of the biggest advantages of this parser is its ability to add a rich set of features efficiently (instead of computationally expensive indicators).

Even though part-of-speech tags and dependency labels constitute relatively small sets, replacing the commonly used one-hot encoding with distributed representations leads to substantial improvements in parsing accuracy. This can be attributed to the fact that similar to words, labels and part-of-speech tags exhibit semantical similarities, which are captured by their embeddings.

In addition to dense representations, a novel cube activation function is introduced in order to model the interaction of input units better. Unlike the popular tanh and sigmoid functions, cube activation directly captures the product terms of word, part-of-speech and label vectors.

During parsing, greedy decoding is performed. Word, part-of-speech and label embeddings are extracted at every step from the current configuration. Then, the neural network's hidden layer is calculated, after which the transition with the highest score at the softmax layer is applied to the configuration.

To accelerate feature extraction, a precomputation technique is applied (Devlin et al., [4]). Matrix multiplications for the most frequent 10,000 tokens are precomputed and stored in a table. Thus, instead of performing matrix multiplication operations to compute the hidden layer during the parsing, the precomputed values from the table are used. According to the experiments on the parser's speed, this precomputation technique speeds up the parsing 8 ~ 10 times (Chen and Manning, 2014).

IV. Related Work

Although this project's algorithm draws from previous work in the field of parsing, it introduces a few distinctive novel ideas. Earlier studies of parsing algorithms generally employed one-hot encoding for tokens (Mayberry III and Miikkulainen, [5]). The earliest attempt to predict transitions using an artificial neural network was by James Henderson [7]. However, these parsers' architectures are significantly less scalable, thus limiting their practical application. Lately, with the advent of deep learning techniques, there have been attempts to utilize them in dependency parsers (Stenetorp, [6]), albeit without significant success.

Several dependency parsers are already implemented in NLTK for dependency parsing, namely the probabilistic projective parser (Eisner 1996) and the non-projective parser following the MST approach. The platform also provides interfaces to Stanford CoreNLP and Malt parsers. The performance of the latter algorithms is satisfactory, but their implementations are not in Python.

V. Conclusion

NLTK is a widely used platform that provides instruments for text processing, including parsing. Aiming to address the absence of a fast and accurate parser in the toolkit, this project implements a modern dependency parsing algorithm and incorporates it into NLTK. Several innovative ideas are displayed in this work, namely the dense representations of part-of-speech tags and arc labels, and the cube activation function.

The proposed project is a significant advance for NLTK and contributes to the dissemination of contemporary approaches to dependency parsing. Considering the rapid growth of artificial intelligence, this algorithm's inclusion in NLTK is of great importance, since it will serve as a guide to newcomers and facilitate the study of natural language processing.

References

[1] Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In Proc. EMNLP, pages 740–750.

[2] Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In Proc. ACL Workshop on Incremental Parsing, pages 50–57.

[3] Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In Proc. ACL-HLT, pages 188–193.

[4] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In ACL.

[5] Mayberry, M. and Miikkulainen, R. (2005). Broad-Coverage Parsing with Neural Networks. Neural Processing Letters, 21(2), pp.121-132.

[6] Pontus Stenetorp. 2013. Transition-based dependency parsing using recursive neural networks. In NIPS Workshop on Deep Learning.

[7] James Henderson. 2004. Discriminative training of a neural network statistical parser. In Proc. ACL, Main Volume, pages 95–102.

[8] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-ofspeech tagging with a cyclic dependency network. In NAACL.

[9] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, pages 62–69. Somerset, NJ: Association for Computational Linguistics. http://arXiv.org/abs/cs/0205028.

[10] Ivan Titov and James Henderson. 2007. Fast and robust multilingual dependency parsing with a generative latent variable model. In Proc. EMNLP, pages 947–951.

1695 words