SyntaxNet Models for the CoNLL 2017 Shared Task

Chris Alberti, Daniel Andor, Ivan Bogatyy, Michael Collins, Dan Gillick,
Lingpeng Kong[*], Terry Koo, Ji Ma, Mark Omernick, Slav Petrov,
Chayut Thanapirom, Zora Tung, David Weiss

Google Inc
New York, NY

Abstract

We describe a baseline dependency parsing system for the CoNLL2017 Shared Task. This system, which we call "ParseySaurus," uses the DRAGNN framework [Kong et al., 2017] to combine transition-based recurrent parsing and tagging with character-based word representations. On the v1.3 Universal Dependencies Treebanks, the new system outpeforms the publicly available, state-of-the-art "Parsey's Cousins" models by 3.47% absolute Labeled Accuracy Score (LAS) across 52 treebanks.

1 Introduction

Universal Dependencies[1] are growing in popularity due to the cross-lingual consistency and large language coverage of the provided data. The initiative has been able to connect researchers across the globe and now includes 64 treebanks in 45 languages. It is therefore not surprising that the Conference on Computational Natural Language Learning (CoNLL) in 2017 will feature a shared task on "Multilingual Parsing from Raw Text to Universal Dependencies."

To facilitate further research on multilingual parsing and to enable even small teams to participate in the shared task, we are releasing baseline implementations corresponding to our best models. This short paper describes (1) the model structure employed in these models, (2) how the models were trained and (3) an empirical evaluation comparing these models to those in Andor et al. [2016]. Our model builds on the DRAGNN framework [Kong et al., 2017] to improve upon Andor et al. [2016] with dynamically constructed, recurrent transition-based models. The code as well as the pretrained models is available at the SyntaxNet github repository.[2]

We note that this paper describes the parsing model used in the baseline. So far, there have not been any changes to the segmentation model compared to SyntaxNet.

2 Character-based representation

Recent work has shown that learned sub-word representations can improve over both static word embeddings and manually extracted feature sets for describing word morphology. Jozefowicz et al. [2016] use a convolutional model over the characters in each word for language modeling. Similarly, Ling et al. [2015a,b]

[*] Carnegie Mellon University, Pittsburgh, PA.

[1] http://universaldependencies.org/

[2] https://github.com/tensorflow/models/tree/master/syntaxnet

use a bidirectional Long Short-Term Memory network (LSTM) over characters in each word for parsing and machine translation.

Chung et al. [2016] take a more general approach. Instead of modeling each word explicitly, they allow the model to learn a hierarchical "multi-timescale" representation of the input, where each layer corresponds to a (learned) larger timescale.

Our modeling approach is inspired by this multi-timescale architecture in that we generate our computation graph dynamically, but we define the timescales explicitly. The input layer operates on characters and the subsequent layer operates on words, where the word representation is simply the hidden state computed by the first layer at each word boundary. In principle, this structure permits fully dynamic word representations based on left context (unlike previous work) and simplifies recurrent computation at the word level (unlike previous work with standard stacked LSTMs [Gillick et al., 2015]).

More related work [Kim et al., 2015, Miyamoto and Cho, 2016, Lankinen et al., 2016, Ling et al., 2015a] describes alternate neural network architectures for combining character- and word-level modeling for various tasks. Like Ballesteros et al. [2015], we use character-based LSTMs to improve the Stack-LSTM Dyer et al. [2015] model for dependency parsing, but we share a single LSTM run over the entire sentence.

3 Model

Our model combines the recurrent multi-task parsing model of Kong et al. [2017] with character-based representations learned by an LSTM. Given a tokenized text input, the model processes as follows:

- A single LSTM processes the entire character string (including whitespace)[3] left-to-right. The last hidden state in a given token (as given by the word boundaries) is used to represent that word in subsequent parts of the model.
- A single LSTM processes the word representations (from the first step) in right-to-left order. We call this the "lookahead" model.
- A single LSTM processes the *lookahead* representations right-to-left. This LSTM has a softmax layer which is trained to predict POS tags, and we refer to it as the "tagger" model.
- The recurrent compositional parsing model [Kong et al., 2017] predicts the parse tree left-to-right using the *arc-standard* transition system. Given a stack s and a input pointer to the buffer i, the parser dynamically links and concatenates the following input representations:
  - Recurrently, the two steps that last modified the s_o and s_1 (either SHIFT or REDUCE operations).
  - From the *tagger* layer, the hidden representations for s_0, s_1, and i.
  - From the *lookahead* layer, the hidden representation for i.
  - All are projected to 64 dimensions before concatenating.
  - The parser also extracts 12 discrete features for previously predicted parse labels, the same as in Kong et al. [2017].

  At inference time, we use beam decoding in the parser with a beam size of 8. We do not use local normalization, and instead train the models with "self-normalization" (see below).

This model is implemented using the DRAGNN framework in TensorFlow. All code is publicly available at the SyntaxNet repository. The code provides tools to visualize the unrolled structure of the graph at runtime.

[3] In our UD v1.3 experiments, the raw text string is not available. Since we use gold segmentations, the whitespace is artificially induced, and functions as a "new word" signal for languages with no naturally occurring whitespace.

3.1 Training

We train using the multi-task, maximum-likelihood "stack-propagation" method described in Kong et al. [2017] and Zhang and Weiss [2016]. Specifically, we use the gold labels to alternate between two updates:

1. TAGGER: We unroll the first three LSTMs and backpropagate gradients computed from the POS tags.
2. PARSER: We unroll the entire model, and backpropagate gradients computed from the oracle parse sequence.

We use the following schedule: pretrain TAGGER for 10,000 iterations. Then alternate TAGGER and PARSER updates at a ratio of 1:8 until convergence.

To optimize for beam decoding, we regularize the softmax objective to be "self-normalized."Vaswani et al. [2013], Andreas and Klein [2015] With this modification to the softmax, the log scores of the model are encouraged (but not constrained) to sum to one. We find that this helps mitigate some of the bias induced by local normalization Andor et al. [2016], while being fast and efficient to train.

3.2 Hyperparameters

Like the ratio above, many hyperparameters, including design decisions, were tuned to find reasonable values before training all 64 baseline models. While the full recipe can be deciphered from the code, here are some key points for practitioners:

- We use Layer Normalization [Ba et al., 2016] in all of our networks, both LSTM and the recurrent parser's Relu network cell.
- We always project the LSTM hidden representations down from 256→64 when we pass from one component to another.
- We use moving averages of parameters at inference time.
- We use the following ADAM recipe: $\beta_1 = \beta_2 = 0.9$, and set ϵ to be one of $10^{-3}, 10^{-4}, 10^{-5}$ (typically 10^{-4}).
- We normalize all gradients to have unit norm *before* applying the ADAM updates.
- We use dropout both recurrently and on the inputs, at the same rate (typically 0.7 or 0.8).
- We use a minibatch size of 4, with 4 asynchronous training threads doing asynchronous SGD.

4 Comparison to Parsey's Cousins

Since the v2 test set are not available for the contest, we use v1.3 of the Universal Dependencies treebanks to compare to prior state-of-the-art on 52 languages. Our results are in Table 1. We observe that the new model outperforms the original SyntaxNet baselines, sometimes quite dramatically (e.g. on Latvian, by close to 12% absolute LAS). We note that this is not an exhaustive experiment, and further study is warranted in the future. Nonetheless, these results show that the new baselines compare very favorably to at least one publicly available state-of-the-art baseline.

	Parsey		ParseySaurus			Parsey		ParseySaurus	
Language	UAS	LAS	UAS	LAS	Language	UAS	LAS	UAS	LAS
Ancient Greek-PROIEL	78.74	73.15	81.14	75.81	Indonesian	80.03	72.99	82.55	76.31
Ancient Greek	68.98	62.07	73.85	68.1	Irish	74.51	66.29	75.71	67.13
Arabic	81.49	75.82	85.01	79.8	Italian	89.81	87.13	91.14	88.78
Basque	78.00	73.36	82.05	78.72	Kazakh	58.09	43.95	65.93	52.98
Bulgarian	89.35	85.01	90.87	86.87	Latin-ITTB	84.22	81.17	88.3	85.87
Catalan	90.47	87.64	91.87	89.7	Latin-PROIEL	77.60	70.98	80.27	74.29
Chinese	76.71	71.24	81.04	76.56	Latin	56.00	45.80	63.49	52.52
Croatian	80.65	74.06	82.84	76.78	Latvian	58.92	51.47	69.96	63.29
Czech-CAC	87.28	83.44	89.26	85.26	Norwegian	88.61	86.22	90.69	88.53
Czech-CLTT	77.34	73.40	79.9	75.79	Old Church Slavonic	84.86	78.85	87.1	81.47
Czech	89.47	85.93	89.09	84.99	Persian	84.42	80.28	86.66	82.84
Danish	79.84	76.34	81.93	78.69	Polish	88.30	82.71	91.86	87.49
Dutch-LassySmall	81.63	78.08	84.02	80.53	Portuguese-BR	87.91	85.44	90.52	88.55
Dutch	77.70	71.21	79.89	74.29	Portuguese	85.12	81.28	88.33	85.07
English-LinES	81.50	77.37	83.74	80.13	Romanian	83.64	75.36	87.41	79.89
English	84.79	80.38	87.86	84.45	Russian-SynTagRus	91.68	87.44	92.67	88.68
Estonian	83.10	78.83	86.93	83.69	Russian	81.75	77.71	84.27	80.65
Finnish-FTB	84.97	80.48	88.17	84.5	Slovenian-SST	65.06	56.96	68.72	61.61
Finnish	83.65	79.60	86.96	83.96	Slovenian	87.71	84.60	89.76	87.69
French	84.68	81.05	86.61	83.1	Spanish-AnCora	89.26	86.50	91.06	88.89
Galician	84.48	81.35	86.22	83.65	Spanish	85.06	81.53	87.16	84.03
German	79.73	74.07	84.12	79.05	Swedish-LinES	81.38	77.21	83.97	79.93
Gothic	79.33	71.69	82.12	74.72	Swedish	83.84	80.28	86.58	83.48
Greek	83.68	79.99	86.2	82.66	Tamil	64.45	55.35	69.53	60.78
Hebrew	84.61	78.71	86.8	81.85	Turkish	82.00	71.37	83.96	74.06
Hindi	93.04	89.32	93.82	90.23					
Hungarian	78.75	71.83	81.82	76.22	**Average**	**81.12**	**75.85**	**84.07**	**79.33**

Table 1: Comparison to prior SyntaxNet models on UD v1.3. On average, we observe a 14.4% relative reduction in error (RRIE), or 3.47% absolute increase in LAS.

4.1 CoNLL2017 Shared Task

We provide pre-trained models for all 64 treebanks in the CoNLL2017 Shared Task on the SyntaxNet website. All source code and data is publicly available. Please see the task website for any updates.

Acknowledgements

We thank our collaborators at the Universal Dependencies and Conll2017 Shared task, and especially Milan Straka of UD-Pipe. We'd also like to thank all members of the Google Parsing Team (current and former) who made this release possible.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2442–2452, 2016.

Jacob Andreas and Dan Klein. When and why are log-linear models self-normalizing? In *HLT-NAACL*, pages 244–249, 2015.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, 2015.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 334–343, 2015.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. Multilingual language processing from bytes. *arXiv preprint arXiv:1512.00103*, 2015.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

Lingpeng Kong, Chris Alberti, Daniel Andor, Ivan Bogatyy, and David Weiss. Dragnn: A transition-based framework for dynamically connected neural networks. *ArXiV*, 2017.

Matti Lankinen, Hannes Heikinheimo, Pyry Takala, Tapani Raiko, and Juha Karhunen. A character-word compositional neural language model for finnish. *arXiv preprint arXiv:1508.06615*, 2016.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, 2015a.

Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W Black. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*, 2015b.

Yasumasa Miyamoto and Kyunghyun Cho. Gated word-character recurrent language model. *arXiv preprint arXiv:1508.06615*, 2016.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. Decoding with large-scale neural language models improves translation. In *EMNLP*, pages 1387–1392. Citeseer, 2013.

Yuan Zhang and David Weiss. Stack-propagation: Improved representation learning for syntax. In *Proc. ACL*, 2016.