Parsing with Context Embeddings

Ömer Kırnap **Berkay Furkan Önder** **Deniz Yuret**

Koç University
Artificial Intelligence Laboratory
İstanbul, Turkey
okirnap,bonder17,dyuret@ku.edu.tr

Abstract

We introduce *context embeddings*, dense vectors derived from a language model that represent the left/right context of a word instance, and demonstrate that context embeddings significantly improve the accuracy of our transition based parser. Our model consists of a bidirectional LSTM (BiLSTM) based language model that is pre-trained to predict words in plain text, and a multi-layer perceptron (MLP) decision model that uses features from the language model to predict the correct actions for an ArcHybrid transition based parser. We participated in the CoNLL 2017 UD Shared Task as the "Koç University" team and our system was ranked 7th out of 33 systems that parsed 81 treebanks in 49 languages.

1 Introduction

Recent studies in parsing natural language has seen a shift from shallow models that use high dimensional, sparse, hand engineered features, e.g. (Zhang and Nivre, 2011), to deeper models with dense feature vectors, e.g. (Chen and Manning, 2014). Shallow linear models cannot represent feature conjunctions that may be useful for parsing decisions, therefore designers of such models have to add specific combinations to the feature list by hand: for example Zhang and Nivre (2011) define 72 hand designed conjunctive combinations of 39 primitive features. Deep models can represent and automatically learn feature combinations that are useful for a given task, so the designer only has to come up with a list of primitive features. Two questions about feature representation still remain critical: what parts of the parser state to represent, and how to represent these (typically discrete) features with continuous embedding vectors.

In this work we derive features for the parser from a bidirectional LSTM language model trained with pre-tokenized text to predict words in a sentence using both the left and the right context. In particular we derive *word embeddings* and *context embeddings* from the language model. Word embeddings represent the general features of a word type averaged over all its occurrences. Taking advantage of word embeddings derived from language models in other applications is common practice, however, using the same embedding for every occurrence of an ambiguous word ignores polysemy and meaning shifts. To mitigate this problem, we also construct and use *context embeddings* that represent the immediate context of a word *instance*. Context embeddings were previously shown to improve tasks such as part-of-speech induction (Yatbaz et al., 2012) and word sense induction (Başkaya et al., 2013). In this study, we derive context embeddings from the hidden states of the forward and backward LSTMs of the language model that are generated while predicting a word. These hidden states summarize the information from the left context and the right context of a word that was useful in predicting it. Our main contribution is to demonstrate that using context embeddings as features leads to a significant improvement in parsing performance.

The rest of the paper is organized as follows: Section 2 introduces basic components of a transition based neural network parser and describes related work based on their design choices. Section 3 describes the details of our model and training method. Section 4 discusses our results and Section 5 summarizes our contributions.

Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 80–87,
Vancouver, Canada, August 3-4, 2017.

2 Related work

In this section, we describe related work in transition based neural network parsers in terms of their design decisions regarding common components.

2.1 Embedding words and features

In neural network parsers, words, part of speech tags, and other discrete features are represented with numeric vectors. These vectors can be initialized and optimized in a number of ways. The first choice is between binary (one-hot) vectors vs dense continuous vectors. If dense vectors are to be used, they can be initialized randomly or transferred from a model for a related task such as language modeling. Finally, once initialized, these vectors can be fixed or fine-tuned during the training of the dependency parser.

Chen and Manning (2014) initialize with pre-trained word vectors from (Collobert et al., 2011) in English and (Mikolov et al., 2013) in Chinese, and dense, randomly initialized vectors for POS tags. Similarly, Dyer et al. (2015) get pre-trained word embeddings from Bansal et al. (2014) and use POS tag vectors that are randomly initialized. Both studies fine-tune the vectors during parser training.

Kiperwasser and Goldberg (2016) start with random POS embeddings and fine-tuned word embeddings from (Dyer et al., 2015) and further optimize all embeddings during parser training. They also report that initialization with random word vectors give inferior performance.

In (Alberti et al., 2017), a character-level LSTM reads each word character by character and the last hidden state creates a *word representation*. The word representation is used as input to a word-level LSTM whose hidden states constitute the *lookahead representation* of each word. Finally, the lookahead representation is used by a *tagger* LSTM trained to predict POS tags. Concatenation of the *lookahead* and *tagger* representations of a word, together with additional features are used to represent the word in the parser model.

2.2 Feature extraction

A neural network parser uses a feature extractor that represents the state of the parser using continuous embeddings of its various elements. Chen and Manning (2014); Kiperwasser and Goldberg (2016); Andor et al. (2016) use POS tag and word embedding features of the stack's first and second words, their right and leftmost children, and the buffer's first word. They have done experiments with different subsets of those features, but they report their best performance using all of them. Alberti et al. (2017) extract the *tagger* features that are explained in 2.1 for the first and second words of the stack and the first word of the buffer plus the *lookahead* feature of buffer's first word. They also use the last two transitions executed by the parser (including shift and reduce operations) as binary encoded features in their parser model.

2.3 Decision module

A transition based parser composes the parse of a sentence by taking a number of parser actions. We name the component that picks a parser action using the extracted features the *decision module*. Chen and Manning (2014) use an MLP decision module with a hidden size of 200 whose input is a concatenation of word, POS tag, and dependency embeddings. Kuncoro et al. (2016) use an LSTM as the decision module instead, carrying internal state between actions. Dyer et al. (2015) introduce stack-LSTMs, which have the ability to recover earlier hidden states. They construct the parser state using three stack-LSTMs, representing the buffer, the stack, and the action history. Kiperwasser and Goldberg (2016) train a BiLSTM whose input is word and POS embeddings and whose hidden states are fed to an MLP that decides parsing actions.

2.4 Training

Parsing is a structured prediction problem and a number of training objectives and optimization methods have been proposed beyond simple likelihood maximization of correct parser actions.

Kiperwasser and Goldberg (2016) use dynamic oracle training proposed in (Goldberg and Nivre, 2012). In dynamic oracle training, the parser takes predicted actions rather than gold actions which lets it explore states otherwise not visited. Andor et al. (2016), use beam training based on (Collins and Roark, 2004). The objective in beam training is to maximize the probability of the whole sequence rather than a single action. Andor et al. (2016) use global normalization with beam search (Collins and Roark, 2004) which normalizes the total score of the action sequence instead of turning the score of each action into a probability. This allows the model to represent a richer set of probability distributions. They report that their MLP

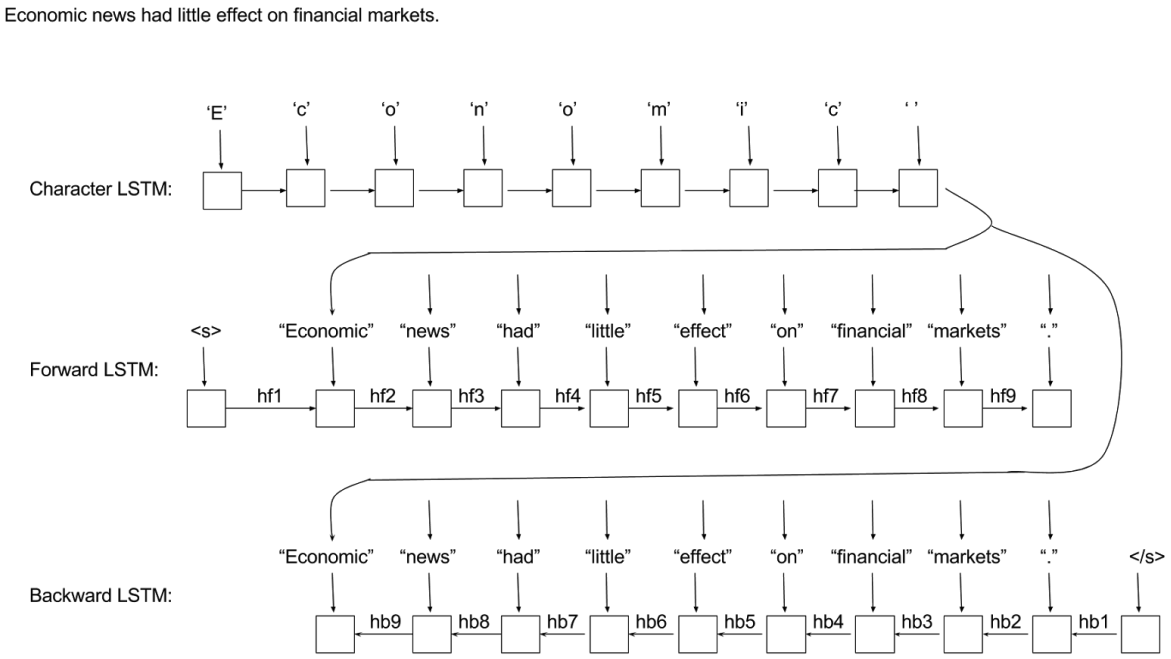

Figure 1: Processing of the sentence "Economic news had little effect on financial markets" by the bidirectional LSTM language model. Word embeddings are generated by the character LSTM. Each word is predicted (e.g. "news") by feeding the adjacent hidden states (e.g. "hf2" and "hb8") to a softmax layer.

based globally normalized parser performs better than locally normalized recurrent models.

3 Model

Our parser uses a bidirectional language model to generate word and context embeddings, an ArcHybrid transition system (Kuhlmann et al., 2011) to construct a parse tree, and a simple MLP decision module to pick the right parser actions. These components are detailed below. The model was implemented and trained using the Knet deep learning package in Julia (Yuret, 2016) and the source code is publicly available at `https://github.com/CoNLL-UD-2017`.

3.1 Language Model

We trained bidirectional language models to extract *word* and *context embeddings* using the Wikipedia data sets provided by task organizers (Ginter et al., 2017) and tokenized with UDPipe (Straka et al., 2016). Our language models consist of two parts: a character based unidirectional LSTM to produce word embeddings, and a word based bidirectional LSTM to predict words and produce context embeddings. First, each word of a sentence is padded in the beginning and the end by a start character and an end character respectively. Next, the character based LSTM reads each word left to right and the final hidden layer is used as the *word embedding*. This step is repeated until all the words of an input sentence is mapped to dense embedding vectors. Next, those word embeddings become inputs to the BiLSTM, which tries to predict each word based on its left and right contexts. A *context embedding* for a word is created by concatenating the hidden vectors of the forward and backward LSTMs used in predicting that word.

Figure 1 depicts the language model processing an example sentence. The unidirectional character LSTM produces the word embeddings (shown for the word "Economic" in the Figure) which are fed as input to the bidirectional word LSTM. The bidirectional LSTM predicts a given word using the adjacent forward and backward hidden states at that position (e.g. the word "news" is predicted using "hf2" and "hb8").

The parser uses both word embeddings produced by the character LSTM (350 dimensions) and the context embeddings produced by the word LSTM (300+300 dimensions) as features. We did not fine-tune the LM weights during parser training.

The character and word LSTMs were trained end-to-end using backpropagation through time (Werbos, 1990) using Adam (Kingma and Ba, 2014) with default parameters and with gradients clipped at 5.0. Sentences that are longer than 28 words were skipped during LM training. In addition, if a word is longer than 65 characters, only the first 65 characters were used and the rest was ignored. The output vocabulary was restricted to the most frequent 20K words of each language. The training was stopped if there was no significant improvement in out-of-sample perplexity during the last 1M words. Table 3 includes the perplexity of each bidirectional language model we used.

3.2 The ArcHybrid transition system

We used the ArcHybrid transition system (Kuhlmann et al., 2011) in our model where the state of the parser $c = (\sigma, \beta, A)$, consists of a stack of tree fragments σ, a buffer of unused words β and a set A of dependency arcs. The initial state has an empty stack and dependency set and all words start in the buffer. The system has 3 types of transitions:

- shift$(\sigma, b|\beta, A) = (\sigma|b, \beta, A)$
- left$_d(\sigma|s, b|\beta, A) = (\sigma, b|\beta, A \cup \{(b, d, s)\})$
- right$_d(\sigma|s|t, \beta, A) = (\sigma|s, \beta, A \cup \{(s, d, t)\})$

where $|$ denotes concatenation and (b, d, s) is a dependency arc between b (head) and s (modifier) with label d. The parser stops when the buffer is empty and there is a single word in the stack, which is assumed to be the root.

3.3 Features

Abbrev	Feature
c	context embedding
v	word embedding
p	universal POS tag
d	distance to the next word
a	number of left children
b	number of right children
A	set of left dependency labels
B	set of right dependency labels
L	dependency label of current word

Table 1: Possible features for each word

Table 1 lists the potential features our model is able to extract for each word. Context and word embeddings come from the language model. The 17 universal POS tags are mapped to 128 dimensional embedding vectors and the 37 universal dependency labels are mapped to 32 dimensional embedding vectors. These are initialized randomly and trained with the parser. To represent sets of dependency labels we simply add the embeddings of each element in the set. Each distinct left/right child count and distance is represented using a randomly initialized 16 dimensional embedding vector trained with the parser. Counts and distances larger than 10 were truncated to 10.

This leaves the question of which words to use and which of their features to extract. The transition system informs feature selection: ArcHybrid transitions directly effect the top word in the buffer and the top two words in the stack. Figure 2 lists the features that are actually extracted by our model to represent each parser state. $s0, s1, \ldots$ are stack words, $n0, n1, \ldots$ are buffer words, $s1r$ and $s0r$ are the rightmost children of the top two stack words, $n0l$ is the leftmost child of the top buffer word. The letters below each word are the features extracted for that word (using the notation in Table 1). Nonexistent features (e.g. the dependency label of $n0l$ when $n0$ does not have any left children) are represented with vectors of zeros.

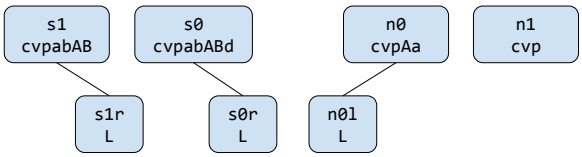

Figure 2: Features used by our model. See the text and Table 1 for an explanation of the notations.

3.4 Decision module

We use a simple MLP with a single hidden layer of 2048 units to choose parser actions. The embeddings of each feature are concatenated to provide the input to the decision module, which results in a 4664 dimensional input vector. Note that word and context embeddings come from the language model and are fixed, whereas the other embeddings are randomly initialized and trained with the MLP.

The output of the MLP is a 73 dimensional softmax layer. These represent the shift, 36 left and 36 right (labeled) actions of the parser: there are no actions for the "root" label.

To train the MLP we used Adam with a dropout rate of 0.5. We train 5 to 30 epochs, quitting with the best model when the dev score does not improve for 5 epochs.

3.5 Training

We followed different procedures for training languages that had training and development data, languages that did not have development data, and surprise languages that only had a small amount of sample data. We detail our methodology below.

3.5.1 Languages with training and development data

For most languages, a substantial amount of training data with gold parses along with development data were supplied. In this case we first trained our language models using the additional raw data (Ginter et al., 2017) provided by *CoNLL 2017 UD Shared Task Organizers* as described in 3.1. Next, the decision module (MLP part) is trained as described in 3.5 using the *context* and *word embeddings* from the language model as fixed inputs. The development data was used to determine when to stop training.

3.5.2 Languages without development data

For languages with no development data, we used 5 fold cross validation on the training data to determine the number of epochs for training. The MLP model is trained on each fold for up to 30 epochs during the 5 fold cross validation. If the LAS score on the test split does not improve for 5 epochs, training is stopped and the number of epochs to reach the best score is recorded. In the final step, the MLP model is trained using the whole training data for a number of epochs determined by the average of the 5 splits.

3.5.3 Surprise languages

The surprise languages did not come with raw data to train a language model, so we decided to use unlexicalized parsers for them. An unlexicalized model in our case is simply one that does not use the "c" and "v" features in Figure 2, i.e. no word and context embeddings. The surprise languages also did not have enough training data to train a parser. We decided that an unlexicalized parser trained on a related language may perform better than one trained on the small amount of sample data we had for each surprise language. We trained unlexicalized parsers for most of the languages provided in the task and tried them as "parent" languages for each surprise language. An unlexicalized model trained on the parent language was finetuned for the surprise language with its small amount of sample data. Table 2 lists the parent language used for each surprise language and the LAS score achieved on the sample data provided using 5-fold cross validation.[1]

Language	Parent Language	LAS
North Sami	Estonian	60.48
Buryat	Turkish	47.68
Kurmanji	Bulgarian	46.87
Upper Sorbian	Croatian	65.98

Table 2: Parent models used for parsing surprise languages and LAS scores obtained after pre-train and finetuning.

4 Results and Discussion

We submitted our system to *CoNLL 2017 UD Shared Task* as the "Koç University" team and our scoring can be found under official *CoNLL 2017 UD Shared Task* website[2] replicated here in Table 3. All our experiments are done with UD version 2.0 datasets (Nivre et al., 2017). In this section we discuss our best/worst results relative to other task participants, and analyze the benefit of using context vectors.

4.1 Best and worst results

Looking at our best/worst results may give insights into the strengths and weaknesses of our approach. Relative to other participants, Finnish, Hungarian, and Turkish are among our best languages: all agglutinative languages with complex morphology. This may be due to our character based language model which can capture morphological features when constructing word vectors. Our worst results are in ancient languages: Ancient Greek, Gothic, Old Church Slavonic. We believe this is due to lack of raw text to construct high quality language models. Finally, our results for languages with large treebanks (Syntagrus and Czech) are also relatively worse than languages with smaller treebanks. A large treebank may offset the advantage

[1] Note that for two ancient languages, Gothic and Old Church Slavonic, our LM training was not successful, and we used unlexicalized models for them like the surprise languages.

[2] http://universaldependencies.org/conll17/results.html

Language	LM Perp.	Rank	LAS	Language	LM Perp.	Rank	LAS
ar	99.21	13	66.14	hsb	Not used	17	50.25
ar_pud	99.21	12	44.97	hu	27.83	4	69.55
bg	25.60	9	84.95	id	52.64	9	75.54
bxr	Not used	14	24.96	it	27.97	10	86.45
ca	18.49	10	86.09	it_pud	27.97	10	84.52
cs	37.65	20	81.55	ja	29.14	18	72.67
cs_cac	44.87	15	82.91	ja_pud	29.14	15	76.27
cs_cltt	52.64	10	73.88	kk	715.23	17	22.34
cs_pud	37.65	20	78.57	kmr	Not used	4	42.11
cu	Not used	26	58.63	ko	34.60	8	71.70
da	30.28	7	76.39	la	111.51	10	47.08
de	33.98	11	72.44	la_ittb	59.28	16	76.15
de_pud	33.98	6	70.96	la_proiel	130.01	13	59.36
el	20.14	7	81.35	lv	37.81	6	63.63
en	44.50	15	75.96	nl	32.43	11	70.24
en_lines	40.79	10	74.39	nl_lassysmall	35.62	8	80.85
en_ParTUT	51.57	11	75.71	no_bokmaal	34.38	12	83.73
en_pud	44.50	11	79.51	no_nynorsk	31.03	9	82.72
es	26.33	7	83.34	pl	27.97	9	80.84
es_ancora	26.33	9	85.63	pt	24.11	9	82.92
es_pud	26.33	8	78.74	pt_br	33.6	10	86.7
et	45.77	6	62.04	pt_pud	24.11	6	75.02
eu	39.92	8	71.47	ro	21.02	7	81.48
fa	63.29	12	79.56	ru	26.99	7	77.11
fi	29.36	5	77.72	ru_pud	26.99	3	71.2
fi_ftb	41.03	11	75.37	ru_syntagrus	29.36	20	85.24
fi_pud	29.36	4	82.37	sk	21.99	7	76.46
fr	18.76	9	81.30	sl_sst	194.75	8	49.56
fr_ParTUT	14.60	7	80.22	sme	Not used	4	37.93
fr_pud	18.76	6	76.04	sv	40.42	9	78.31
fr_sequoia	16.75	7	81.97	sv_lines	34.21	7	75.71
ga	56.32	8	63.22	sv_pud	40.42	6	72.36
gl	28.70	5	80.27	tr	57.31	6	56.8
gl_treegal	32.32	4	69.13	tr_pud	57.31	6	34.65
got	Not used	24	56.81	ug	866.74	21	31.59
grc	116.72	23	49.31	uk	36.16	6	63.76
grc_proiel	227.78	22	61.70	ur	105.38	11	77.64
he	78.75	10	58.98	vi	91.67	13	38.3
hi	37.36	10	87.23	zh	92.01	19	57.15
hi_pud	37.36	9	51.49	hr	33.29	7	79.22

Table 3: Our official results in *CoNLL 2017 UD Shared Task*

of extra information we capture from a language model trained on raw text. Our simple MLP model trained with a static oracle is probably not competitive on large datasets. Whether our pre-trained language model and context embeddings would boost the scores of more sophisticated approaches (e.g. stack-LSTMs or global normalization) is an open question.

4.2 Impact of context vectors

Feats	Hungarian	En-ParTUT	Latvian
p	63.6	76.6	55.9
v	73.5	75.9	63
c	72.2	76	63.5
v-c	76	79	67.6
p-c	78	82.5	70.6
p-v	76.6	80.8	67.7
p-fb	74.7	79.7	66.3
p-v-c	79.3	83.2	74.2

Table 4: Feature comparison results on three languages. p=postag, v=word-vector, c=context-vector, fb=Facebook-vector.

To analyze the impact of context vectors and other embeddings on parsing performance, we performed experiments on three corpora (Hungarian, English-ParTUT, Latvian) with different feature combinations. These corpora were chosen for their relatively small sizes to allow quick experimentation. We tried eight different feature combinations on each language. In each setting, we used a different subset of context, word, and postag embeddings. The "p-fb" setting uses postag embeddings and Facebook's pre-trained word embeddings (Bojanowski et al., 2016) instead of the ones from our language model. We can make some observations consistent across all three languages based on the results in Table 4:

- Word vectors from our BiLSTM language model perform slightly better than Facebook vectors (p-v vs p-fb).
- Both part-of-speech tags and context vectors have significant contributions (comparing v with p-v or v-c).
- Context vectors seem to provide independent information on top of part-of-speech tags that significantly boosts parser accuracy (p-v vs p-v-c).

5 Contributions

We introduced a transition based neural network parser that uses word and context embeddings derived from a bidirectional language model as features. Our experiments suggest that context embeddings can have a significant positive impact on parsing accuracy. Our source code is publicly available at `https://github.com/CoNLL-UD-2017`.

Acknowledgments

This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) grants 114E628 and 215E201.

References

Chris Alberti, Daniel Andor, Ivan Bogatyy, Michael Collins, Dan Gillick, Lingpeng Kong, Terry Koo, Ji Ma, Mark Omernick, Slav Petrov, Chayut Thanapirom, Zora Tung, and David Weiss. 2017. Syntaxnet models for the conll 2017 shared task. *CoRR* abs/1703.04929. http://arxiv.org/abs/1703.04929.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *CoRR* abs/1603.06042. http://arxiv.org/abs/1603.06042.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *ACL (2)*. pages 809–815.

Osman Başkaya, Enis Sert, Volkan Cirik, and Deniz Yuret. 2013. Ai-ku: Using substitute vectors and co-occurrence modeling for word sense induction and disambiguation. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*. Association for Computational Linguistics, Atlanta, Georgia, USA, pages 300–306. http://www.aclweb.org/anthology/S13-2050.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* .

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*. pages 740–750.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, page 111.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12(Aug):2493–2537.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *CoRR* abs/1505.08075. http://arxiv.org/abs/1505.08075.

Filip Ginter, Jan Hajič, Juhani Luotolahti, Milan Straka, and Daniel Zeman. 2017. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. http://hdl.handle.net/11234/1-1989.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *COLING*. pages 959–976.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *CoRR* abs/1603.04351. http://arxiv.org/abs/1603.04351.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, pages 673–682.

Adhiguna Kuncoro, Yuichiro Sawai, Kevin Duh, and Yuji Matsumoto. 2016. Dependency parsing with lstms: An empirical evaluation. *CoRR* abs/1604.06529. http://arxiv.org/abs/1604.06529.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. pages 3111–3119.

Joakim Nivre et al. 2017. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, `http://hdl.handle.net/11234/1-1983`. http://hdl.handle.net/11234/1-1983.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portoro, Slovenia.

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78(10):1550–1560.

Mehmet Ali Yatbaz, Enis Sert, and Deniz Yuret. 2012. Learning syntactic categories using paradigmatic representations of word context. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP-CONLL 2012)*. Association for Computational Linguistics, Jeju, Korea. http://denizyuret.blogspot.com/2012/05/learning-syntactic-categories-using.html.

Deniz Yuret. 2016. Knet: beginning deep learning with 100 lines of julia. In *Machine Learning Systems Workshop at NIPS 2016*.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*. Association for Computational Linguistics, pages 188–193.