Tree-stack LSTM in Transition Based Dependency Parsing

Ömer Kırnap **Erenay Dayanık** **Deniz Yuret**

Koç University
Artificial Intelligence Laboratory
İstanbul, Turkey
okirnap,edayanik16,dyuret@ku.edu.tr

Abstract

We introduce tree-stack LSTM to model state of a transition based parser with recurrent neural networks. Tree-stack LSTM does not use any parse tree based or hand-crafted features, yet performs better than models with these features. We also develop new set of embeddings from raw features to enhance the performance. There are 4 main components of this model: stack's σ-LSTM, buffer's β-LSTM, actions' LSTM and tree-RNN. All LSTMs use continuous dense feature vectors (embeddings) as an input. Tree-RNN updates these embeddings based on transitions. We show that our model improves performance with low resource languages compared with its predecessors. We participate in *CoNLL 2018 UD Shared Task* as the "KParse" team and ranked 16th in LAS, 15th in BLAS and BLEX metrics, of 27 participants parsing 82 test sets from 57 languages.

1 Introduction

Recent studies in neural dependency parsing creates an opportunity to learn feature conjunctions only from primitive features.(Chen and Manning, 2014) A designer only needs to extract primitive features which may be useful to take parsing actions. However, extracting primitive features from state of a parser still remains critical. On the other hand, representational power of recurrent neural networks should allow a model both to summarize every action taken from the beginning to the current state and tree-fragments obtained until a current state.

We propose a method to concretely summarize previous actions and tree fragments within current word embeddings. We employ word and context embeddings from (Kırnap et al., 2017) as an initial representer. Our model modifies these embeddings based on parsing actions. These embeddings are able to summarize, children-parent relationship. Finally, we test our system in *CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

Rest of the paper is organized as follows: Section 2 summarizes related work done in neural transition based dependency parsing. Section 3 describes the models that we implement for tagging, lemmatization and dependency parsing. Section 4 discusses our results and section 5 presents our contributions.

2 Related Work

In this section we describe the related work done in neural transition based dependency parsing and morphological analysis.

2.1 Morphological Analysis and Tagging

Finite-state transducers (FST) have an important role in previous morphological analyzers. (Koskenniemi, 1983) Unlike modern neural systems, these type of analyzers are language dependent rule based systems. Morphological tagging, on the other hand, tries to solve tagging and analysis problem at the same stage. Koskenniemi proposed conditional random fields (CRFs) based model and Heigold et al. proposed neural network architectures to solve tagging and analysis problem immediately. Modern systems heavily based on word and context based features that we explain in the following paragraph.

2.2 Embedding Features

Chen and Manning, Kiperwasser and Goldberg, use pre-trained word and random part-of-speech (POS) embeddings. Ballesteros et al. use

Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 124–132
Brussels, Belgium, October 31 – November 1, 2018.
https://doi.org/10.18653/v1/K18-2012

character-based word representation for the stack-LSTM parser. In Alberti et al., end-to-end approach is taken for both word and POS embeddings. In other words, one component of their model has responsibility to generate POS embeddings and the other to generate word embeddings.

2.3 Decision Module

We name a part of our model, which provides transitions from features, as decision module. Decision module is a neural architecture designed to find best feature conjunctions. Chen and Manning uses MLP, Dozat et al. applies BiLSTM stacked with MLP as a decision module. We inspire from Dyer et al.'s stack-LSTM which basically represents each component of a state (buffer, stack and actions) with an LSTM. We found new inputs to tree-RNN, and modify this model to obtain better results.

3 Model

In this section, we describe MorphNet (Dayanık et al., 2018) used for tagging and lemmatization; and Tree-stack LSTM used for dependency parsing. We train these models separately. MorphNet employs UDPipe (Straka et al., 2016) for tokenization to generate conll-u formatted file with missing head and dependency relation columns. Tree-stack LSTM takes that for dependency parsing. We detail these models in the remaining part of this section.

3.1 Lemmatization and Part of Speech Tagging

We implement MorphNet (Dayanık et al., 2018) for lemmatization and Part of Speech tagging. It is trained on (Nivre et al., 2018). MorphNet is a sequence-to-sequence recurrent neural network model used to produce a morphological analysis for each word in the input sentence. The model operates with a unidirectional Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) encoder to create a character-based word embeddings and a bidirectional LSTM encoder to obtain context embeddings. The decoder consists of two layers LSTM.

The input to the MorphNet consists of an N word sentence $S = [w_1, \ldots, w_N]$, where w_i is the i'th word in the sentence. Each word is input as a sequence of characters $w_i = [w_{i1}, \ldots, w_{iL_i}], w_{ij} \in \mathcal{A}$ where $\mathcal{A}$ is the set of alphanumeric characters and L_i is the number of characters in word w_i.

The output for each word consists of a stem, a part-of-speech tag and a set of morphological features, e.g. "earn+Upos=verb+Mood=indicative+Tense=past" for "earned". The stem is produced one character at a time, and the morphological information is produced one feature at a time. A sample output for a word looks like $[s_{i1}, \ldots, s_{iR_i}, f_{i1}, \ldots, f_{iM_i}]$ where $s_{ij} \in \mathcal{A}$ is an alphanumeric character in the stem, R_i is the length of the stem, M_i is the number of features, $f_{ij} \in \mathcal{T}$ is a morphological feature from a feature set such as $\mathcal{T} = \{$Verb,Adjective,Mood=Imperative,Tense=Past,$\ldots\}$.

In Word Encoder we map each character w_{ij} to an A dimensional character embedding vector $a_{ij} \in \mathbb{R}^A$.The word encoder takes each word and processes the character embeddings from left to right producing hidden states $[h_{i1}, \ldots, h_{iL_i}]$ where $h_{ij} \in \mathbb{R}^H$. The final hidden state $e_i = h_{iL_i}$ is used as the word embedding for word w_i.

$$h_{ij} = \text{LSTM}(a_{ij}, h_{ij-1}) \quad (1)$$
$$h_{i0} = 0 \quad (2)$$
$$e_i = h_{iL_i} \quad (3)$$

We model context encoder by using a bidirectional LSTM. The inputs are the word embeddings $e_1, \cdots, e_N$ produced by the word encoder. The context encoder processes them in both directions and constructs a unique context embedding for each target word in the sentence. For a word w_i I define its corresponding context embedding $c_i \in \mathbb{R}^{2H}$ as the concatenation of the forward $\overrightarrow{c}_i \in \mathbb{R}^H$ and the backward $\overleftarrow{c}_i \in \mathbb{R}^H$ hidden states that are produced after the forward and backward LSTMs process the word embedding e_i. Figure illustrates the creation of the context vector for the target word earned.

$$\overrightarrow{c}_i = \text{LSTM}_f(e_i, \overrightarrow{c}_{i-1}) \quad (4)$$
$$\overleftarrow{c}_i = \text{LSTM}_b(e_i, \overleftarrow{c}_{i+1}) \quad (5)$$
$$\overrightarrow{c}_0 = \overleftarrow{c}_{N+1} = 0 \quad (6)$$
$$c_i = [\overrightarrow{c}_i; \overleftarrow{c}_i] \quad (7)$$

The decoder is implemented as a 2-Layer LSTM network that outputs the correct tag for a single target word. By conditioning on the input embeddings and its own hidden state, the decoder

learns to generate $y_i = [y_{i1}, \ldots, y_{iK_i}]$ where y_i is the correct tag of the target word w_i in sentence S, $y_{ij} \in \mathcal{A} \cup \mathcal{T}$ represents both stem characters and morphological feature tokens, and K_i is the total number of output tokens (stem + features) for word w_i. The first layer of the decoder is initialized with the context embedding c_i and the second layer is initialized with the word embedding e_i.

$$d^1_{i0} = \text{relu}(W_d \times c_i \oplus W_{db}) \quad (8)$$
$$d^2_{i0} = e_i \quad (9)$$
$$(10)$$

We parameterize the distribution over possible morphological features and characters at each time step as

$$p(y_{ij}|d^2_{ij}) = \text{softmax}(W_s \times d^2_{ij} \oplus W_{sb}) \quad (11)$$

where $W_s \in \mathbb{R}^{|\mathcal{Y}| \times H}$ and $W_{sb} \in \mathbb{R}^{|\mathcal{Y}|}$ where $\mathcal{Y} = \mathcal{A} \cup \mathcal{T}$ is the set of characters and morphological features in output vocabulary.

3.2 Word and Context Embeddings

We benefit pre-trained word embeddings from (Kırnap et al., 2017) in our parser. Both word and context embeddings are extracted from the language model described in section 3.1 of (Kırnap et al., 2017).

3.3 Features

We use limited number of continuous embeddings in parser model. These are POS, word, context, and morphological feature embeddings. Word and context embeddings are pre-trained and not fine-tuned during training. POS and morphological feature embeddings are randomly initialized and learned during training.

Abbrev	Feature
c	context embedding
v	word embedding
p	universal POS tag
f	morphological features

Table 1: Possible features for each word

3.4 Morphological Feature Embeddings

We introduce morphological feature embeddings, which differs from (Dyer et al., 2015), as an additional input to our model. Each feature is represented with 128 dimensional continuous vector. We experiment that vector sizes lower than 128 reduces the performance of a parser, and higher than 128 does not bring further enhancements. We formulate morphological feature embeddings by adding feature vectors of a word. For example, suppose we are given a word *it* with following morphological features: Case=Nom and Gender=Neut and Number=Sing and Person=3 and PronType=Prs. We basically sum corresponding 5 unique vectors to provide morphological feature embedding. However, our experiments suggest that not all languages benefit from morphological feature embeddings. (See section 4 for details)

3.5 Dependency Label Embeddings

Each distinct dependency label defined in *CoNLL 2018 UD Shared Task* represented with a 128 dimensional continuous vector. These vectors combined to construct hidden states in tree-RNN part of our model. We randomly initialize these vectors and learned during training.

3.6 ArcHybrid Transition System

We implement the ArcHybrid Transition System which has three components, namely a stack of tree fragments σ, a buffer of unused words β and a set A of dependency arcs, $c = (\sigma, \beta, A)$. Stack is empty, there is no any arcs and, all the words of a sentence are in buffer initially. This system has 3 type of transitions:

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

where $|$ denotes concatenation and (b, d, s) is a dependency arc between b (head) and s (modifier) with label d. The system terminates parsing when the buffer is empty and the stack has only one word assumed to be the root.

3.7 Tree-stack LSTM

Tree-stack LSTM has 4 main components: buffer's β-LSTM, stack's σ-LSTM, actions'-LSTM and tree's tree-RNN or t-RNN in short. We aim to represent each component of the transition system, $c = (\sigma, \beta, A)$, with a distinct LSTM similar to (Dyer et al., 2015). Initial inputs to these LSTMs are embeddings obtained by concatenating the features explained in section 3.3.

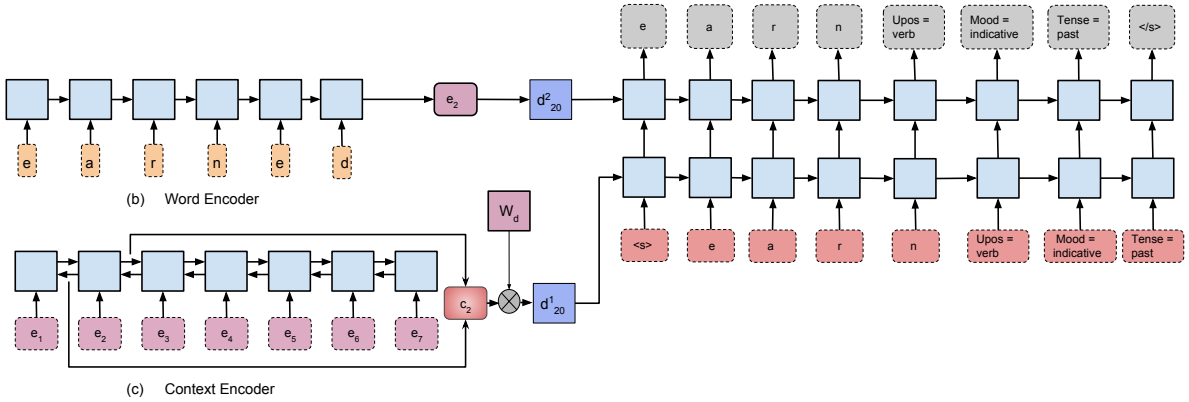

Figure 1: MorphNet illustration for the sentence *"Bush earned 340 points in 1969"* and target word ***"earned"***.

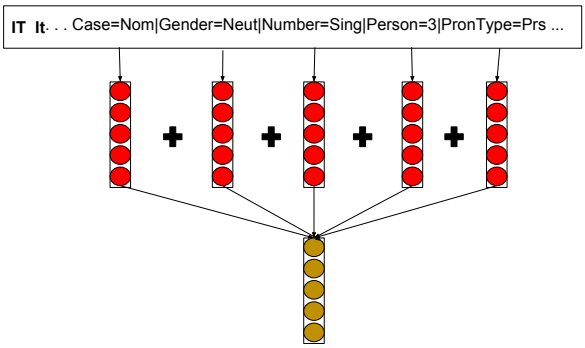

Figure 2: Morphological feature embeddings obtained by adding individual feature embeddings

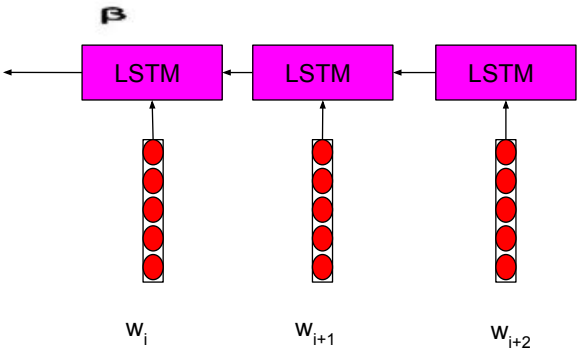

Figure 3: $\beta - LSTM$ processing a sentence. It starts to read from right to left. Each vector $((w_i)$ represents the concatenation of POS, language and morph-feat embeddings.

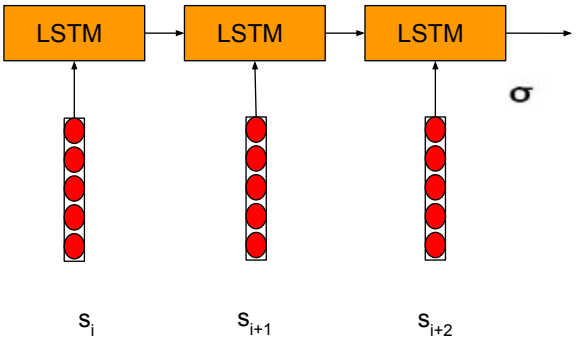

Figure 4: $\sigma - LSTM$ processing a sentence. It starts to read from left to right. Each input (s_i) is transformed version of initial feature vector. Transformations are based on local transitions see 3.6 for details.

Our model differs from (Dyer et al., 2015) by representing actions and dependency relations separately and including morphological feature embeddings. The transition system (see section 3.6 for details) is also different from theirs.

Buffer's β-LSTM is initialized with zero hidden state, and fed with input features from last word to the beginning. Similarly, stack's σ-LSTM is also initialized with zero hidden state and fed with input features from the beginning word to the last word of a stack. Actions' LSTM is also started with zero hidden state, and updated after each action. Inputs to σ-LSTM and β-LSTM are updated via tree-RNN.

We update either buffer's or stack's input embeddings based on parsing actions. For instance, suppose we are given β_i a top word in buffer and σ_i a final word in stack. The $left_d$ transition taken in current state. tree-RNN uses concatenation of previous embedding, σ_i, and dependency relation embedding (explained in 3.5) as a hidden h_{t-1}.Input of a tree-RNN is a previous word embedding, β_i. Output h_t becomes a new word embedding for buffer's top word β_{i-new}. Figure 5

depicts this flow. Similarly to left transition, right transition updates the stack's second top word. Hidden state of an RNN is calculated by concatenating stack's top word and dependency relation.

There are 73 distinct actions for shift, labeled left and labeled right actions. We randomly initialize 128 dimensional vector for each labeled action and shift. These vectors become an input for action-LSTM shown in Figure 6.

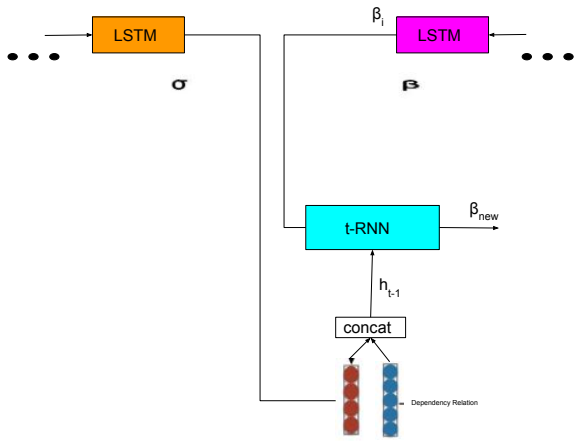

Figure 5: Buffer word's embedding update based on left move. Inputs are old embeddings obtained from Table 3.3

.

Concatenation of stack's LSTM, buffer's LSTM and actions' LSTM's final hidden layer becomes an input to MLP which outputs the probabilities for each transition in the next step.

3.8 Training

Our training strategy varies based on training data sizes. We divide datasets into 4 parts: 100k tokens or more, tokens in between 50k and 100k, and more than 20k less than 50k tokens.

For languages having more than 50k tokens in training data, we employ morphological feature-embeddings as an additional input dimension (see Figure 2). However, for languages having tokens less than 50k we do not use this feature dimension. Finally we realize that the languages with more than 100k tokens, using morphological feature embeddings does not improve parsing performance but we use that additional feature dimension.

We use 5-fold cross validation for languages without development data. We do not change the LSTMs' hidden dimensions, but record the number of epochs took for convergence. The average of these epochs is used to train a model with whole training set.

3.8.1 Optimization and Hyper-Parameters

We conduct experiments to find best set of hyper-parameters. We start with a dimension of 32 and increase the dimension by powers of two until 512 for LSTM hiddens, 1024 for LM matrix (explained in below). We report the best hyper-parameters in this paper. Although the performance does not decrease after the best setting, we choose the minimum-best size not to sacrifice from training speed.

All the LSTMs and tree-RNN have hidden dimension of 256. The vectors extracted from LM having dimension of 950, but we reduce that to 512 by a matrix-vector multiplication. This matrix is also learned. We use Adam optimizer with default parameters. (Kingma and Ba, 2014).Training is terminated if the performance does not improve for 9 epochs.

4 Results

In this section we inspect our best/worst results and the conclusions we obtain during *CoNLL 2018 UD Shared Task* experiments.

We submit our system to *CoNLL 2018 UD Shared Task* as "KParse" team. Our scoring is provided under the official *CoNLL 2018 UD Shared Task* website.[1] as well as in Table 4.1. All experiments are done with UD version 2.2 datasets (Nivre et al., 2018) and (Nivre et al., 2017) for training and testing respectively. The model improves performance by reducing hand-crafted feature selection. In order to analyze our tree-stack LSTM, we compare that model with Kırnap et al. sharing similar feature interests and transition system with our model. The difference between these two models is that Kırnap et al. based on hand-crafted feature selection from state, e.g. number of left children of buffer's first word. However, tree-stack LSTM only needs raw features and previous parsing actions.

Our model comparatively performs better with languages less than 50k training tokens, e.g. sv_lines and hu_szeged and tr_imst. However, when the number of training examples increases the performance improve slightly saturates, e.g. ar_padt, en_ewt. This may be due to convergence problems of our model. This conclusion

[1] http://universaldependencies.org/conll18/results.html

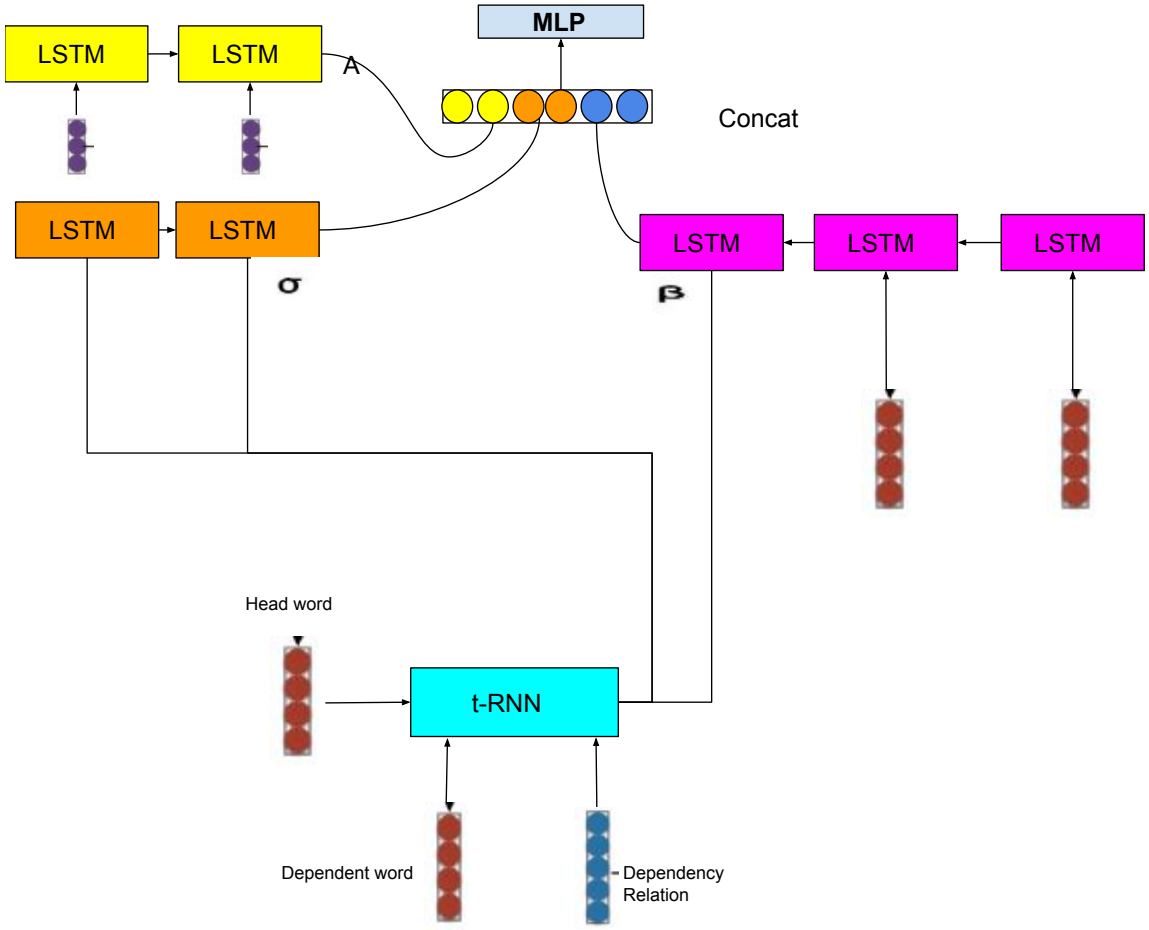

Figure 6: End-to-end tree-stack model composed of 4 main components, namely, β-LSTM, σ-LSTM and actions' LSTMs and the tree-RNN.

Lang code	Kırnap et al.	New Model
tr_imst	56.78	58.75
hu_szeged	66.21	68.18
en_ewt	74.87	75.77
ar_padt	67.83	68.02
cs_cac	83.39	83.57
sv_lines	71.12	74.81

Table 2: Comparison of two models

Lang code	Morp-Feats	no Morp-Feats
ko_gsd	73.74	72.54
got_proiel	54.33	53.24
id_gsd	75.76	73.97

Table 3: Morphological feature embeddings in some languages having tokens more than 50k and less than 100k in training data

is also agrees with our official ranking in *CoNLL 2018 UD Shared Task* because our ranking in low-resource languages is 10, but general ranking is 16.

We next analyze the performance gain by including morphological features with languages training token in between 50k and 100k. As we deduce from Table 3, tree-stack LSTM benefits from morphological information with mid-resource languages. However, we could not gain the similar performance enhancement with languages more than 100k training tokens.

4.1 Languages without Training Data

We have three criteria to choose a trained model for languages without training data. If there is a training corpus with the same language we use that as a parent. If there is no data from the same language, we pick a parent language from the same family. If there are more than one parent for a language, we select a parent with more training data.

We list our selections in Table 4.

Language	Parent Language
en_pud	en_ewt
ja_modern	ja_gsd
cs_pud	cs_pdt
sv_pud	sv_talbanken
fi_pud	fi_tdt
th_pud	id_gsd
pcm_nsc	en_ewt
br_keb	en_ewt

Table 4: Our parent choices in languages without train data

5 Discussion

We use tree-stack LSTM model in transition based dependency parsing framework. Our main motivation for this work is to reduce the human designed features extracted from state components. Our results prove that the model is able to learn better than its predecessors. Moreover, we examine that the model performs better in languages with low resources compared in *CoNLL 2018 UD Shared Task*. We also constitute morphological feature embeddings which become useful for dependency parsing. All of our work is done in transition based dependency parsing, which sacrifices performance due to locality and non projectiveness. This study opens a question on adapting the tree-stack LSTM in graph based dependency parsing. Our code is publicly available at `https://github.com/kirnap/ku-dependency-parser2`.

Acknowlegments

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.

Miguel Ballesteros, Chris Dyer, and Noah A Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. *arXiv preprint arXiv:1508.00657* .

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

Erenay Dayanık, Ekin Akyürek, and Deniz Yuret. 2018. Morphnet: A sequence-to-sequence model that combines morphological analysis and disambiguation. *arXiv preprint arXiv:1805.07946* .

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies* pages 20–30.

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.

Georg Heigold, Guenter Neumann, and Josef van Genabith. 2017. An extensive empirical evaluation of character-based morphological tagging for 14 languages. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. volume 1, pages 505–513.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

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.

Ömer Kırnap, Berkay Furkan Önder, and Deniz Yuret. 2017. Parsing with context embeddings. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies* pages 80–87.

Kimmo Koskenniemi. 1983. Two-level model for morphological analysis. In *IJCAI*. volume 83, pages 683–685.

Joakim Nivre et al. 2017. Universal Dependencies 2.0 CoNLL 2017 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, `http://hdl.handle.net/11234/1-2184`. http://hdl.handle.net/11234/1-2184.

Joakim Nivre et al. 2018. Universal Dependencies 2.2. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, `http://hdl.handle.net/11234/1-1983xxx`. http://hdl.handle.net/11234/1-1983xxx.

Language	LAS	MLAS	BLEX	Ranks	Language	LAS	MLAS	BLEX	Ranks
af_afribooms	78.12	65.12	63.93	17-15-19	ar_padt	68.02	58.05	61.21	16-11-13
bg_btb	84.53	75.58	77.63	20-16-9	br_keb	8.91	0.35	1.77	19-15-19
bxr_bdt	9.93	0.49	0.69	18-22-23	ca_ancora	85.89	77.22	77.22	18-17-17
cs_cac	83.57	75.3	77.24	19-12-18	cs_fictree	82.67	71.93	75.31	18-15-16
cs_pdt	81.43	73.77	71.44	21-21-22	cs_pud	78.69	64.14	84.52	18-19-17
cu_proiel	59.42	46.96	81.55	23-22-21	da_ddt	76.4	67.05	63.09	17-15-19
en_ewt	75.77	66.78	68.76	20-20-18	en_gum	76.44	65.19	64.32	16-12-15
de_gsd	71.59	46.87	35.41	17-8-23	el_gdt	83.34	65.74	66.6	16-14-18
en_lines	73.96	64.91	62.76	17-15-19	en_pud	78.41	66.16	69.25	19-19-17
es_ancora	84.99	76.71	77.06	18-17-16	et_edt	74.52	65.7	63.5	20-19-17
eu_bdt	74.55	63.11	61.25	17-13-18	fa_seraji	81.18	74.84	71.65	17-16-14
fi_ftb	75.84	65.53	67.91	17-16-11	fi_pud	81.55	74.18	66.29	13-12-10
fi_tdt	78.42	70.4	65.89	16-15-12	fo_oft	22.5	0.29	5.44	20-19-20
fr_gsd	81.07	72.07	73.96	19-19-17	fr_sequoia	84.36	76.56	71.33	14-10-20
fr_spoken	57.32	15	57.76	10-10-10	fro_srcmf	76.92	67.85	71.35	20-19-20
ga_idt	63.13	34.36	40.76	13-19-16	gl_ctg	79.02	66.13	71.12	14-14-9
gl_treegal	70.45	52.15	56.38	10-9-9	got_proiel	54.33	40.58	40.51	24-24-22
grc_perseus	55.03	34.19	37.0	20-15-16	grc_proiel	62.11	43.92	37.00	22-21-20
he_htb	58.28	45.06	48.09	18-16-16	hi_hdtb	86.86	70.44	79.98	20-17-15
hr_set	81.6	65.23	68.74	17-8-18	hsb_ufal	30.81	6.22	15.39	8-9-7
hu_szeged	68.18	51.13	50.53	14-20-21	hy_armtdp	24.58	7.24	6.96	12-9-21
id_gsd	75.51	65.02	63.92	17-13-17	it_isdt	85.80	75.5	70.16	22-21-22
it_postwita	70.03	56.04	47.53	13-12-20	ja_gsd	73.30	59.46	59.89	14-14-20
ja_modern	23.35	8.94	10.13	5-4-5	kk_ktb	23.86	5.98	8.02	11-11-13
kmr_mg	23.39	3.97	7.56	15-15-16	ko_gsd	73.74	67.31	60.52	19-18-16
ko_kaist	78.81	71.49	65.29	19-19-16	la_ittb	75.79	71.49	71.66	20-19-19
la_perseus	51.6	33.65	38.04	11-8-8	la_proiel	59.35	46.36	51.13	20-19-19
lv_lvtb	72.33	57.08	60.54	16-13-14	nl_alpino	78.83	64.22	65.79	17-16-15
nl_lassysmall	76.70	63.97	64.58	16-15-15	no_bokmaal	82.32	37.93	73.52	20-19-18
no_nynorsk	80.57	70.78	72.27	20-19-17	no_nynorsklia	53.33	41.01	44.46	13-10-11
pcm_nsc	15.84	5.3	13.61	10-1-11	pl_lfg	86.12	71.96	76.71	21-21-20
pl_sz	82.83	63.76	73.05	16-17-14	pt_bosque	82.71	68.01	73.07	18-17-15
ro_rrt	80.90	72.39	72.59	17-14-14	ru_syntagrus	82.89	56.8	75.48	20-19-18
ru_taiga	60.55	39.41	44.05	11-9-9	sk_snk	75.75	53.49	61.0	20-20-16
sl_ssj	78.18	62.94	69.49	16-18-14	sl_sst	48.77	34.66	39.82	10-11-9
sme_giella	53.39	39.13	41.75	19-19-15	sr_set	80.85	67.46	72.09	20-19-16
sv_lines	74.81	59.72	67.35	17-15-15	sv_pud	70.77	43.04	54.67	16-12-15
sv_talbanken	77.91	69.25	69.87	17-16-17	th_pud	0.74	0.04	0.44	7-8-8
tr_imst	58.75	48.28	49.84	15-12-13	ug_udt	57.04	36.63	44.44	16-16-14
uk_iu	76.5	36.63	65.5	16-16-13	ur_udtb	78.12	51.25	64.66	17-15-15
vi_vtb	40.48	33.88	36.03	16-14-15	zh_gsd	59.76	50.87	53.11	19-15-18

Table 5: Our official results in *CoNLL 2018 UD Shared Task*, ranks are given in LAS-MLAS-BLEX order

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.