Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task

Timothy Dozat
Stanford University
tdozat@stanford.edu

Peng Qi
Stanford University
pengqi@stanford.edu

Christopher D. Manning
Stanford University
manning@stanford.edu

Abstract

This paper describes the neural dependency parser submitted by Stanford to the CoNLL 2017 Shared Task on parsing Universal Dependencies. Our system uses relatively simple LSTM networks to produce part of speech tags and labeled dependency parses from segmented and tokenized sequences of words. In order to address the rare word problem that abounds in languages with complex morphology, we include a character-based word representation that uses an LSTM to produce embeddings from sequences of characters. Our system was ranked first according to all five relevant metrics for the system: UPOS tagging (93.09%), XPOS tagging (82.27%), unlabeled attachment score (81.30%), labeled attachment score (76.30%), and content word labeled attachment score (72.57%).

1 Introduction

In this paper, we describe Stanford's approach to tackling the CoNLL 2017 shared task on Universal Dependency parsing (Nivre et al., 2016; Zeman et al., 2017; Nivre et al., 2017b,a). Our system builds on the deep biaffine neural dependency parser presented by Dozat and Manning (2017), which uses a well-tuned LSTM network to produce vector representations for each word, then uses those vector representations in novel biaffine classifiers to predict the head token of each dependent and the class of the resulting edge. In order to adapt it to the wide variety of different treebanks in Universal Dependencies, we make two noteworthy extensions to the system: first, we incorporate a word representation built up from character sequences using an LSTM, theorizing that this should improve the model's ability to adapt to rare or unknown words in languages with rich morphology; second, we train our own taggers for the treebanks using nearly identical architecture to the one used for parsing, in order to capitalize on potential improvements in part of speech tag quality over baseline or off-the-shelf taggers. This approach gets state-of-the-art results on the macro average of the shared task datasets according to all five POS tagging and attachment accuracy metrics.

One noteworthy feature of our approach is its relative simplicity. It uses a single tagger/parser pair per language, trained on only words and tags; thus we refrain from taking advantage of ensembling, lemmas, or morphological features, any one of which could potentially push accuracy even higher.

2 Architecture

2.1 Deep biaffine parser

The basic architecture of our approach follows that of Dozat and Manning (2017), which is closely related to Kiperwasser and Goldberg (2016), the first neural graph-based (McDonald et al., 2005) parser.[1] In Dozat and Manning's 2017 parser, the input to the model is a sequence of tokens and their part of speech tags, which is then put through a multilayer bidirectional LSTM network. The output state of the final LSTM layer (which excludes the cell state) is then fed through four separate ReLU layers, producing four specialized vector representations: one for the word as a dependent seeking its head; one for the word as a head seeking all its dependents; another for the word as a dependent deciding on its label; and a fourth for the word as head deciding on the labels of its depen-

[1]For other neural graph-based parsers, cf. Cheng et al. (2016); Hashimoto et al. (2016); Zhang et al. (2016)

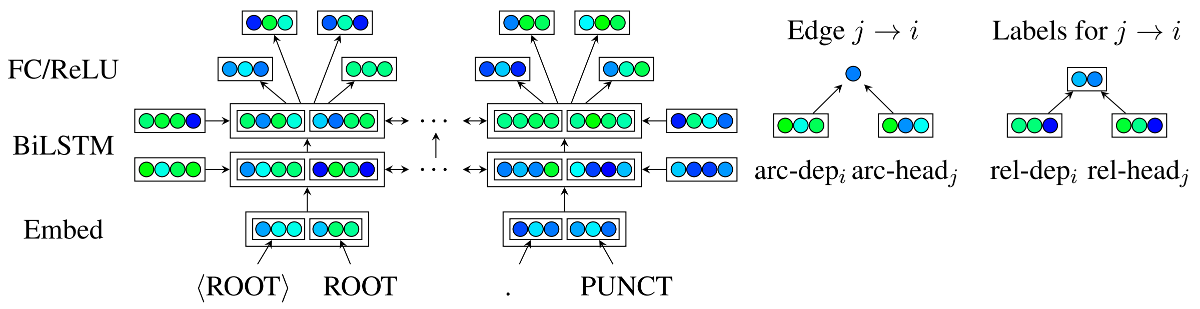


Figure 1: The architecture of our parser. Arrows indicate structural dependence, but not necessarily trainable parameters.

dents.[2] These vectors are then used in two biaffine classifiers: the first computes a score for each pair of tokens, with the highest score for a given token indicating that token's most probable head; the second computes a score for each label for a given token/head pair, with the highest score representing the most probable label for the arc from the head to the dependent. This is shown graphically in Figure 1.

Put formally, given a sequence of n word embeddings (to be described in more detail in Section 2.2) $(\mathbf{v}_1^{(word)}, \ldots, \mathbf{v}_n^{(word)})$ and n tag embeddings $(\mathbf{v}_1^{(tag)}, \ldots, \mathbf{v}_n^{(tag)})$, we concatenate each pair together and feed the result into a BiLSTM with initial state $\mathbf{r}_0$:[3]

$$\mathbf{x}_i = \mathbf{v}_i^{(word)} \oplus \mathbf{v}_i^{(tag)} \tag{1}$$

$$\mathbf{r}_i = \text{BiLSTM}\big(\mathbf{r}_0, (\mathbf{x}_1, \ldots, \mathbf{x}_n)\big)_i \tag{2}$$

$$\mathbf{h}_i, \mathbf{c}_i = \text{split}(\mathbf{r}_i) \tag{3}$$

We then produce four distinct vectors from each recurrent hidden state $\mathbf{h}_i$ (without the recurrent cell state $\mathbf{c}_i$) using ReLU perceptron layers:

$$\mathbf{h}_i^{(arc\text{-}dep)} = \text{MLP}^{(arc\text{-}dep)}(\mathbf{h}_i) \tag{4}$$

$$\mathbf{h}_i^{(arc\text{-}head)} = \text{MLP}^{(arc\text{-}head)}(\mathbf{h}_i) \tag{5}$$

$$\mathbf{h}_i^{(rel\text{-}dep)} = \text{MLP}^{(rel\text{-}dep)}(\mathbf{h}_i) \tag{6}$$

$$\mathbf{h}_i^{(rel\text{-}head)} = \text{MLP}^{(rel\text{-}head)}(\mathbf{h}_i) \tag{7}$$

In order to produce a prediction $y_i'^{(arc)}$ for token i, we use a biaffine classifier involving the (arc) hidden vectors:

$$\begin{aligned}\mathbf{s}_i^{(arc)} &= H^{(arc\text{-}head)} W^{(arc)} \mathbf{h}_i^{(arc\text{-}dep)} \\ &\quad + H^{(arc\text{-}head)} \mathbf{b}^{\top(arc)}\end{aligned} \tag{8}$$

$$y_i'^{(arc)} = \arg\max_j s_{ij}^{(arc)} \tag{9}$$

Note first the similarity between line 8 and a traditional affine classifier of the form $W\mathbf{h} + \mathbf{b}$, with each of W and $\mathbf{b}$ first being transformed by $H^{(arc\text{-}head)}$. Note also that both terms of the biaffine layer have intuitive interpretations: the first relates to the probability of word j being the head of word i given the information in both $\mathbf{h}^{(arc)}$ vectors (for example, the probability of word i depending on word j given that word i is *the* and word j is *cat*); the second relates to the probability of word j being the head of word i given only the information in the head's vector (for example, the probability of word i depending on word j given that word j is *the*, which should be very small no matter what word i is).

After deciding on a head y_i' for word i, we use another biaffine transformation—this time involving the (rel) hidden vectors—to produce a predicted label:

$$\begin{aligned}\mathbf{s}_i^{(rel)} &= \mathbf{h}_{y_i'^{(arc)}}^{\top(rel\text{-}head)} \mathbf{U}^{(rel)} \mathbf{h}_i^{(rel\text{-}dep)} \\ &\quad + W^{(rel)} \big(\mathbf{h}_i^{(rel\text{-}dep)} \oplus \mathbf{h}_{y_i'^{(arc)}}^{(rel\text{-}head)}\big) \\ &\quad + \mathbf{b}^{(rel)}\end{aligned} \tag{10}$$

$$y_i'^{(rel)} = \arg\max_j s_{ij}^{(rel)} \tag{11}$$

Again, each term in line 10 has an intutive interpretation: the first term relates to the probability of observing a label given the information in both $\mathbf{h}^{(rel)}$ vectors (e.g. the probability of the label *det* given word i is *the* with head *cat*); the second relates to the probability of observing a label given

[2]Interestingly, other researchers have found similar approaches to be beneficial for other tasks; cf. Reed and de Freitas (2016); Miller et al. (2016); Daniluk et al. (2017)

[3]We adopt the convention of using lowercase italics for scalars, lowercase bold for vectors, uppercase italics for matrices, and uppercase bold for tensors. We maintain this convention when indexing and stacking; so $\mathbf{a}_i$ is the ith vector of matrix A, and matrix A is the stack of all vectors $\mathbf{a}_i$.

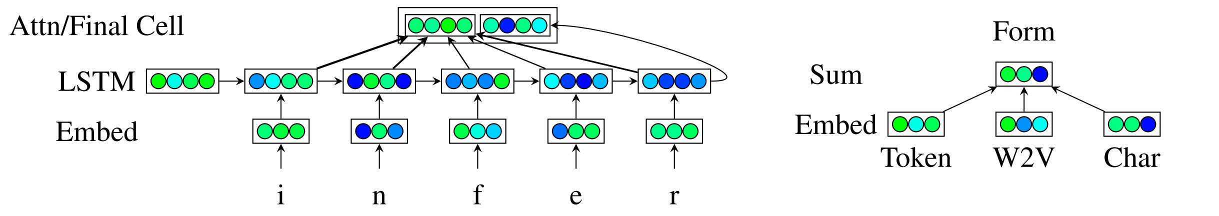


Figure 2: The architecture of our embedding model. Arrows indicate structural dependence, but not necessarily trainable parameters.

either $\mathbf{h}^{(rel)}$ vector (e.g. the probability of the label *det* given that word i is *the* or that word j is *cat*); the last relates to the prior probability of observing a label.

We jointly train these two biaffine classifiers by optimizing the sum of their softmax cross-entropy losses. At test time, we ensure the tree is well-formed by iteratively identifying and fixing cycles for each proposed root and selecting the one with the highest score, which is both simple and sufficient for our purposes. [4]

2.2 Character-level model

Dozat and Manning (2017) represented words as the sum of a pretrained vector[5] and a holistic word embedding for frequent words. However, that approach seems insufficient for languages with rich morphology; so we add a third representation built up from sequences of characters. Each character is given a trainable vector embedding, and each sequence of character embeddings is fed into a unidirectional LSTM. However, the LSTM produces a *sequence* of recurrent states $(\mathbf{r}_1, \ldots, \mathbf{r}_n)$, which we need to convert into a single vector. The simplest approach is to take the last one—which would represent a summary of all the information aggregated one character at a time—and linearly transform it to the desired dimensionality. Another approach, suggested by Cao and Rei (2016), is to use attention over the hidden states, and then trasform the resulting context vector to the desired size; in theory, this should both allow the model to learn morpheme information more easily by attending more closely to the LSTM output at morpheme boundaries. We choose to combine both approaches, using the hidden states for attention and the cell state for summarizing, shown in Figure 2.

That is, given a sequence of n character embeddings and an initial state $\mathbf{r}_0$ for the LSTM, we each embedding into an LSTM as before, extracting hidden and cell states:

$$\mathbf{r}_i = \text{LSTM}\big(\mathbf{r}_0, (\mathbf{v}_1^{(char)}, \ldots, \mathbf{v}_n^{(char)})\big)_i \tag{12}$$

$$\mathbf{h}_i, \mathbf{c}_i = \text{split}(\mathbf{r}_i) \tag{13}$$

We then compute linear attention over the stack of hidden vectors H and concatenate it to the final cell state:

$$\mathbf{a} = \text{softmax}\big(H\mathbf{w}^{(attn)}\big) \tag{14}$$

$$\tilde{\mathbf{h}} = H^\top \mathbf{a} \tag{15}$$

$$\hat{\mathbf{v}} = W\big(\tilde{\mathbf{h}} \oplus \mathbf{c}_n\big) \tag{16}$$

In this way we use the hidden states for attention and the cell state as a final summary vector.

After computing the character-level word embedding, we add together elementwise the pretrained embedding, the holistic frequent token embedding, and the newly generated character-level embedding. We also add together embeddings for the language's UPOS and XPOS tags. The resulting two vectors are used as input to the BiLSTM parser in Section 2.1.

2.3 POS tagger

The final piece of our system is a separately-trained part of speech tagger. The architecture for the tagger is almost identical to that of the parser (and shares fundamental properties with other neural taggers; cf. Ling et al. (2015); Plank et al. (2016))—it uses a BiLSTM over word vectors (using the tripartite representation from Section 2.2), then uses ReLU layers to produce one vector representation for each type of tag.

[4]Although in the future we intend to implement than the Chu-Liu/Edmonds algorithm for nonprojective MST parsing (Chu and Liu, 1965; Edmonds, 1967)

[5]We use the provided CoNLL vectors trained on word2vec (Mikolov et al., 2013); for Gothic, which had no provided vector embeddings, we used Facebook's FastText vectors (Bojanowski et al., 2016)

Thus we use a BiLSTM, as with the parser architecture:

$$\mathbf{r}_i = \text{BiLSTM}\big(\mathbf{r}_0, (\mathbf{v}_1^{(word)}, \ldots, \mathbf{v}_n^{(word)})\big)_i \tag{17}$$

$$\mathbf{h}_i, \mathbf{c}_i = \text{split}(\mathbf{r}_i) \tag{18}$$

And we use affine classifiers for each type of tag, which we add together for the parser:

$$\mathbf{h}_i^{(pos)} = \text{MLP}^{(pos)}(\mathbf{h}_i) \tag{19}$$

$$\mathbf{s}_i^{(pos)} = W\mathbf{h}_i^{(pos)} + \mathbf{b}^{(pos)} \tag{20}$$

$$y_i'^{(pos)} = \arg\max_j s_{ij}^{(pos)} \tag{21}$$

The tag classifiers are trained jointly using cross-entropy losses that are summed together during optimization, but the tagger is trained independently from the parser.

3 Training details

Our model largely adopts the same hyperparameter configuration laid out by Dozat and Manning (2017), with a few exceptions. The parser uses three BiLSTM layers with 100-dimensional word and tag embeddings and 200-dimensional recurrent states (in each direction); the arc classifier uses 400-dimensional head/dependent vector states and the label classifier uses 100-dimensional ones; we drop word and tag embeddings independently with 33% probability;[6] we use same-mask dropout (Gal and Ghahramani, 2015) in the LSTM, ReLU layers, and classifiers, dropping input and recurrent connections with 33% probability; and we optimize with Adam (Kingma and Ba, 2014), setting the learning rate to $2e^{-3}$ and $\beta_1 = \beta_2 = .9$. We train models for up to 30,000 training steps (where one step/iteration is a single minibatch with approximately 5,000 tokens), at first saving the model every 100 steps if fewer than 1,000 iterations have passed, and afterwards only saving if validation accuracy increases (or training accuracy for languages with no validation data). When 5,000 training steps pass without improving accuracy, we terminate training.

For the character model, we use 100-dimensional uncased character embeddings with 400-dimensional recurrent states. We don't drop characters but do include 33% dropout in the LSTM and attention connections.

[6]When only one is dropped, we scale the other by a factor of two

In the tagger we use nearly identical settings, with a few exceptions: the BiLSTM is only two layers deep, we increase the dropout between recurrent connections to 50%, and we use cased character embeddings.

Our approach for dealing with the surprise languages was to train delexicalized "language family" parsers with the same architecture detailed in Section 2.1 on UDPipe v1.1 (Straka et al., 2016)'s UPOS tags with no word-level information. For Buryat (Altaic), we used as input the training datasets for Turkish, Uyghur, Kazakh, Korean, and Japanese; for Kurmanji (Indo-Iranian), we used Persian, Urdu, and Hindi; for North Sámi (Uralic), we used Finnish, Finnish-FTB, Estonian, and Hungarian; and for Upper Sorbian (Slavic), we used Bulgarian, Czech, Old Church Slavonic, Polish, Russian, Russian-SynTagRus, Slovak, Slovenian, Slovenian-SST, and Ukrainian.

There's substantial variability in training and testing speed across treebanks, but on an NVidia Titan X GPU the models train at 100 to 1000 sentences/sec and test at 1000 to 5000 sentences/sec. Even without GPU acceleration a tagger or parser can be run on an entire test treebank in ten to twenty seconds. By far the greatest runtime overhead comes not from the model itself, but from reading in the large matrices of pretrained embeddings, which can take several minutes. A full run over the 81 test sets on the TIRA virtual machine (Potthast et al., 2014) takes about 16 hours, but when parallelized on faster machines it can be done in under an hour.

4 Results

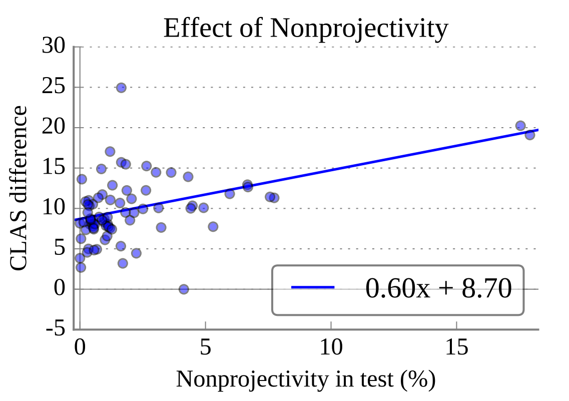
Our model uses a provided tokenization and segmentation and produces UPOS tags, XPOS tags, arcs, and labels. Thus the relevant metrics for the system are UPOS accuracy, XPOS accuracy, unlabeled attachment score, labeled attachment score, and content labeled attachment score. Our system achieves the highest aggregated score on all five of these metrics in the shared task. Below we explore where our model does particularly well, and where it can be improved. We choose to evaluate on CLAS performance because we feel it more accurately reflects model performance, being a principled extension of the common practice of removing punctuation from evalution. We also exclude surprise languages from the following analyses.

One small point to that end is that our sys-

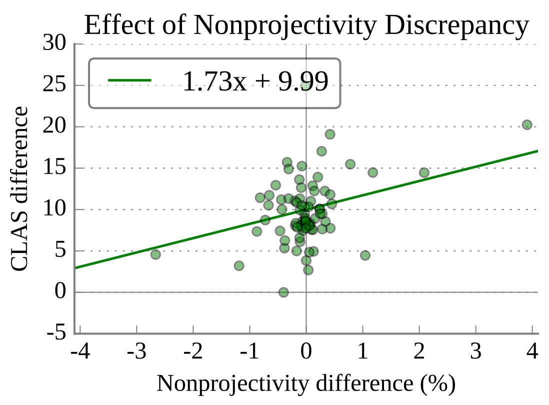
	UPOS	XPOS	UAS	LAS	CLAS
ar	89.36	**87.66**	76.59	71.97	68.17
ar_pud	71.17	0.00	58.87	49.50	46.06
bg	**98.75**	**96.71**	**92.89**	**89.81**	**86.53**
bxr	84.12	99.35	**51.19**	30.00	25.37
ca	**98.59**	**98.58**	**92.88**	**90.70**	**86.70**
cs	**98.83**	**95.86**	**92.62**	**90.17**	**88.44**
cs_cac	**99.05**	**95.16**	**93.14**	**90.43**	**88.31**
cs_cltt	**97.91**	**89.98**	86.02	82.56	79.62
cs_pud	96.42	**92.60**	**89.11**	**84.42**	**81.60**
cu	**95.90**	**96.20**	77.10	71.84	70.49
da	**97.40**	99.69	**85.33**	**82.97**	**80.03**
de	**94.41**	**97.29**	**84.10**	**80.71**	**76.97**
de_pud	**85.71**	20.89	**80.88**	**74.86**	**73.96**
el	**97.74**	**97.76**	**89.73**	**87.38**	**83.59**
en	**95.11**	**94.82**	**84.74**	**82.23**	**78.99**
en_lines	**96.64**	**95.41**	**85.16**	**82.09**	**78.71**
en_partut	**95.22**	**95.08**	86.10	82.54	77.40
en_pud	**95.40**	**94.29**	**88.22**	**85.51**	**82.63**
es	**96.59**	99.69	**90.01**	**87.29**	**82.08**
es_ancora	**98.72**	**98.73**	**92.11**	**89.99**	**86.15**
es_pud	88.39	1.76	**88.14**	**81.05**	**74.60**
et	**93.01**	**95.05**	**78.08**	**71.65**	**69.85**
eu	**95.89**	99.96	**85.28**	**81.44**	**79.71**
fa	**97.15**	**97.12**	**89.64**	**86.31**	**82.93**
fi	**96.62**	**97.37**	**87.97**	**85.64**	**84.25**
fi_ftb	**96.30**	**95.31**	**89.24**	**86.81**	**84.12**
fi_pud	**97.54**	0.00	**90.60**	**88.47**	**86.82**
fr	96.20	98.87	**88.57**	**85.51**	**82.14**
fr_partut	**96.16**	**95.88**	88.64	85.05	79.49
fr_pud	**89.32**	2.40	**83.45**	**78.81**	**77.37**
fr_sequoia	**97.41**	99.06	88.48	86.53	83.37
ga	**92.43**	**91.31**	78.50	**70.06**	**61.38**
gl	**97.72**	**97.50**	85.87	**83.23**	**78.05**
gl_treegal	**94.51**	**91.65**	78.28	73.39	66.02
got	**95.74**	**96.49**	73.10	66.82	63.87
grc	**92.64**	**84.47**	**78.42**	**73.19**	**67.59**
grc_proiel	**97.06**	**97.51**	78.30	74.25	68.83
he	82.42	82.45	67.70	63.94	56.78
hi	**97.50**	**97.01**	**94.70**	**91.59**	**87.92**
hi_pud	**85.48**	**34.82**	**67.24**	**54.49**	**48.87**
hr	**97.68**	99.93	**90.11**	**85.25**	**82.36**
hsb	90.30	99.84	67.83	60.01	**56.32**
hu	**95.34**	99.82	**82.35**	**77.56**	**76.08**
id	**94.09**	99.99	85.17	**79.19**	**77.15**
it	**98.04**	**97.93**	**92.51**	**90.68**	**86.18**
it_pud	**93.74**	2.48	**91.08**	**88.14**	**84.49**
ja	88.14	89.68	75.42	74.72	65.90
ja_pud	89.41	7.50	78.64	77.92	68.95
kk	57.36	55.72	43.51	25.13	19.32
kmr	90.04	89.84	47.71	35.05	28.72
ko	**96.14**	**93.02**	**85.90**	**82.49**	**80.85**
la	**90.67**	**76.69**	**72.56**	**63.37**	**58.96**
la_ittb	**98.36**	**94.79**	**89.44**	**87.02**	**84.94**
la_proiel	**96.72**	**96.93**	73.71	69.35	66.56
lv	**93.59**	**80.05**	**79.26**	**74.01**	**70.22**
nl	**93.24**	**90.61**	**85.17**	**80.48**	**75.19**
nl_lassysmall	**98.39**	99.93	**89.56**	**87.71**	**85.22**
no_bokmaal	**98.35**	99.75	**91.60**	**89.88**	**87.67**
no_nynorsk	**98.11**	99.85	**90.75**	**88.81**	**86.41**
pl	**98.15**	**91.97**	**93.98**	**90.32**	**87.94**
pt	**97.24**	**83.04**	**89.90**	**87.65**	**83.27**
pt_br	**98.22**	**98.22**	**92.76**	**91.36**	**87.48**
pt_pud	**88.99**	0.00	83.27	77.14	71.68
ro	**97.59**	**96.98**	**90.43**	**85.92**	**81.87**
ru	**96.99**	**96.73**	87.15	**83.65**	**81.80**
ru_pud	86.85	**80.17**	**82.31**	**75.71**	**73.13**
ru_syntagrus	**98.59**	99.57	**94.00**	**92.60**	**90.11**
sk	**96.87**	**85.00**	**89.58**	**86.04**	**83.86**
sl	**98.63**	**94.74**	**93.34**	**91.51**	**88.98**
sl_sst	**94.04**	**86.87**	61.71	56.02	51.04
sme	86.81	88.98	51.13	37.21	39.22
sv	**97.70**	**96.40**	**88.50**	**85.87**	**83.71**
sv_lines	**96.74**	**94.84**	**86.51**	**82.89**	**79.92**
sv_pud	**94.33**	**92.33**	**81.90**	**78.49**	**76.48**
tr	**93.86**	**93.11**	**69.62**	**62.79**	**60.01**
tr_pud	**72.73**	0.00	58.72	37.72	31.71
ug	76.65	78.69	56.86	39.79	30.11
uk	**94.31**	**79.42**	**81.44**	**75.33**	**71.72**
ur	**93.95**	**92.30**	**87.98**	**82.28**	**75.88**
vi	75.28	73.56	46.14	42.13	38.59
zh	85.26	85.07	68.95	65.88	62.03

	UPOS	XPOS	UAS	LAS	CLAS
All treebanks	**93.09**	**82.27**	**81.30**	**76.30**	**72.57**
Large treebanks	**95.58**	**94.56**	**85.16**	**81.77**	**78.40**
Parallell treebanks	**88.25**	30.66	**80.17**	**73.73**	**69.88**
Small treebanks	**87.02**	**82.03**	70.19	61.02	54.76
Surprise treebanks	–	–	54.47	40.57	37.41

Table 1: Results on each treebank in the shared task plus the macro average over all of them. State of the art performance by the system is in bold.



(a) Difference in CLAS between our parser and UDPipe v1.1 as a function of the nonprojectivity of the test set



(b) Difference in CLAS between our parser and UDPipe v1.1 as a function of the difference between the nonprojectivity of the test and training sets

Figure 3: How the percent of nonprojective arcs in the training and test set influence accuracy of our graph-based and a transition-based parser

tem assumes tokenization and segmentation has already been done; we therefore trained on gold segmentation and evaluated using the segmentation provided by UDPipe. For most treebanks this was easily sufficient, but for Vietnamese, Chinese, Japanese, and Arabic, UDPipe's lower performance at segmenting or tokenizing was correlated with a relatively large gap between CLAS and gold-aligned CLAS. Because our model reports comparable numbers for nearly all other treebanks, we take this to mean that alignment errors propagated through the system into parsing errors.

4.1 Nonprojectivity

In Universal Dependencies, unlike many other popular benchmarks, several treebanks have a large fraction of crossing dependencies, so any competitive system will need to be able to produce nonprojective arcs. One of the most frequently used approaches for producing fully nonprojective parsers in transition-based systems is to add the `swap` action (Nivre, 2009). This makes any arbitrary nonprojective arc possible, but increases the number of transition steps required to produce that arc. One valid concern is that this might bias the model toward producing projective arcs; in our graph-based system, by contrast, there's little reason to think nonprojective arcs should be harder to predict than projective ones. Here we aim to explore how the fraction of nonprojective arcs in a treebank affects the performance of the two types of systems.

To test the relative performance of a graph-based and a transition-based model, we compute the difference in per-treebank CLAS performance between our parser and the UDPipe v1.1 baseline (Straka et al., 2016), which uses a transition-based parser with the `swap` operation (Straka et al., 2015). We then plot this against the frequency of nonprojective arcs in the test set. To determine whether there is a significant relationship between the difference in performance, we fit the data to a generalized linear mixed effects regression model (Fisher, 1930), using Markov chain Monte Carlo sampling (Hadfield, 2010). We include log data size, morphological complexity (see Section 5.2), and training set projectivity as random effects. We plot the data with the learned regression lines in Figure 3a. What we find is that the margin between the performance of the graph-based and transition-based parsers increases with the nonprojectivity of the test set significantly ($p < 0.001$). This remains significant even when outliers[7] are excluded ($p < 0.05$). To the extent that UDPipe represents a typical nonprojective transition-based parser, our results suggest that a graph-based approach is better suited to parsing UD treebanks that have significant syntactic freedom or complexity than a transition-based one.

Predicting crossing arcs requires more operations (and therefore more long-term planning on behalf of the parser) when using the `swap` feature in a transition-based system, but in our graph-

[7]Korean (top); Ancient Greek, Latin (right)

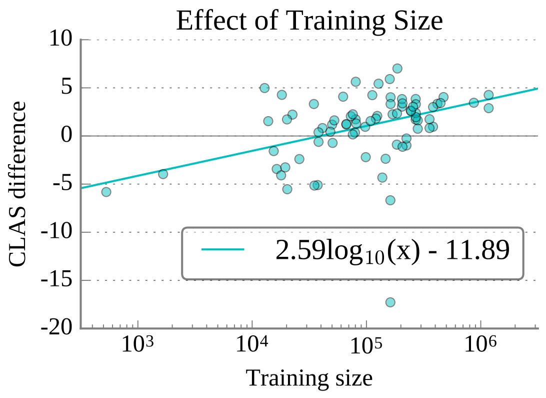


Figure 4: Performance difference between our model and the highest-performing model other than ours as a function of log training data size

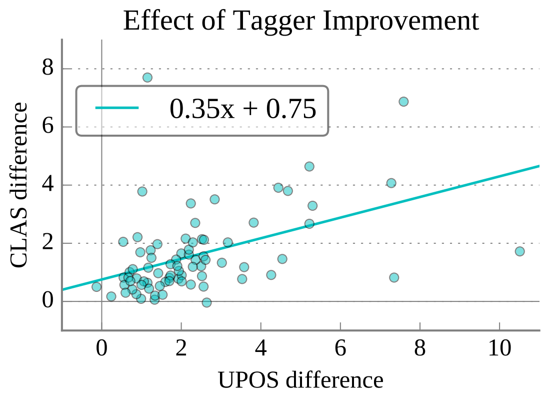


Figure 5: Performance difference between a version of our model trained on our own predicted tags and a version trained on UDPipe v1.1 tags as a function of the performance difference between our taggers and the UDPipe taggers

based system they can be predicted as easily as projective arcs. One might hypothesize that because of this, a transition-based `swapping` system would need to see more examples of crossing dependencies than a graph-based system in order to generalize well. The data shown in Figure 3b support this hypothesis: we computed the difference between the projectivity of each test and training set, and used this as the fixed effect in another mixed effects model with data size, morphological complexity, and train/test nonprojectivity as random effects. We find that when the training set has drastically fewer crossing dependencies than the test set, the graph-based model achieves relatively higher accuracy; but when the transition-based parser can train on many crossing arcs, the models are closer in performance ($p < 0.001$), even when excluding the same outliers ($p < 0.05$). This suggests that the graph-based approach learns and generalizes crossing dependencies more efficiently than the transition-based approach, although this again comes with the assumption that UDPipe's parser is representative of most transition-based `swapping` parsers when it comes to producing nonprojective parses.

4.2 Data size

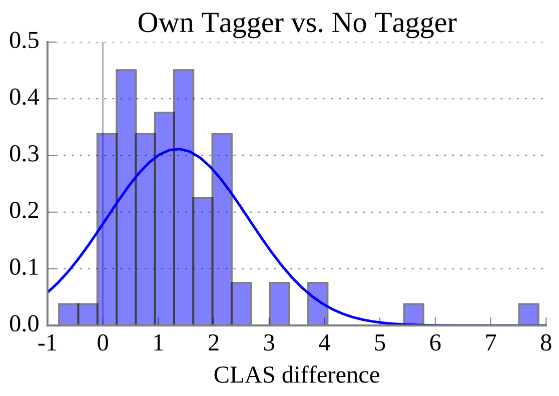
We use the same hyperparameter configuration for all datasets, regardless of how much training data there is for that treebank, which means we may have overfit to small training datasets or underfit to large ones. To test this, we computed the per-treebank difference between the test CLAS performance of our model and that of the highest-performing model other than ours, and plotted that ratio against the log training data size in Figure 4. We fit the differences to another mixed effects regression model with train/test projectivity and morphological complexity set as random effects, finding that our system on average tends to do relatively better on larger datasets compared to other approaches and worse on smaller ones ($p < 0.001$). When the outliers are excluded,[8] this tendency is still significant ($p < 0.001$). This suggests that our model is overfitting to smaller datasets, and that increasing regularization or decreasing model capacity may improve accuracy for lower-resource languages.

5 Ablation Studies

5.1 POS Tagger

We chose to train our parsers on our own predicted tags instead of using provided taggers; here we aim to justify that strategy empirically with an ablation study. We trained another set of parsers with otherwise identical hyperparameter settings using the baseline tags provided by UDPipe v1.1, and computed the difference in CLAS between our reported models and the new ones. We also computed the difference in UPOS accuracy between UDPipe v1.1's taggers and our own. In Figure 5, we plot how the difference in tagger quality affects the CLAS of the parser, making

[8]Kazakh, Uyghur (left); Japanese (bottom); Czech-CAC, Russian-SynTagRus, Czech (right)



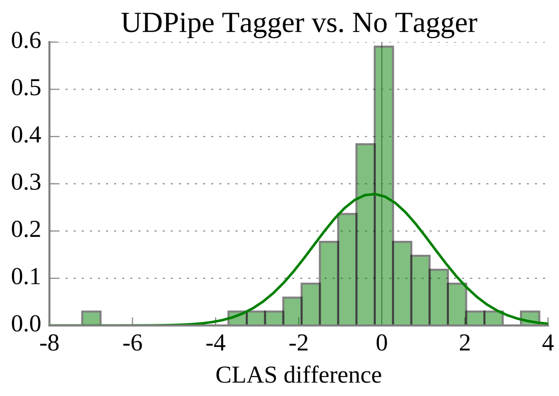


Figure 6: Performance difference between parsers using our taggers and parsers without tags (left) and between parsers using UDPipe v1.1's tags and parsers without tags (right), with both histograms fit to skew normal distributions

two noteworthy observations. The first is that the performance difference between the set of models trained on our own tags is statistically significantly better than the performance of the models trained on UDPipe tags according to a Wilcoxon test ($p < 0.001$). The second is that this can be explained by the improvement of our tagger over UDPipe v1.1, again accounting for dataset size, nonprojectivity, and morphology in a mixed effects model ($p < 0.001$). This suggests that improving upstream tagger performance is an effective way of improving downstream parser accuracy. We also examined the effect of training size on the difference in parser performance, finding no significant correlation ($p > 0.05$).

The approach laid out in this paper uses one neural network to tag the sequences of tokens, and a second neural network to produce a parse from the tokens and tags. One might ask to what extent the tagger network is actually necessary, for a number of reasons: presumably whatever predictive patterns it learns from the token sequences would also be learnable by the parser network; errors by the tagger are likely to be propagated by the parser; and Ballesteros et al. (2015) found that POS tags are drastically less important for character-based parsers. In order to examine how useful the POS tag information is to our character-based system, we trained an additional set of parsers without UPOS or XPOS input, comparing them to the other two, with the differences graphed in Figure 6. We find that the variant with no POS tag input is likewise significantly worse than our reported model according to a Wilcoxon test ($p < 0.001$), but not statistically different from the one trained with UDPipe tags ($p > 0.05$). This suggests that predicted POS tags are still useful for achieving maximal parsing accuracy in our system, provided the tagger's performance is sufficiently high.

5.2 Character model

One of the ways in which we build on Dozat and Manning's 2017 work is by adding a character-level word representation similar to that of Ballesteros et al. (2015), hypothesizing that it should allow the model to more effectively learn the relationships between words in languages with rich morphology and loose word order. We test this using another ablation study; we trained a second set of taggers and parsers on the dataset with only whole token and pretrained vectors, leaving out the vector composed from character sequences (for maximal comparability, we use the original character-based taggers for the token-based parsers). As morphological complexity increases, the difference between the models should increase as well.

The basis of our approach to quantifying morphological complexity will be the assumption that in a morphologically complex language, the ratio between the size of the vocabulary $|V^{(X)}|$ of a corpus to the size of the corpus $|X|$ will be relatively high, because the same lemma may occur with many different forms; but in a morphologically simplex language, that ratio will be smaller, because a given lemma will normally appear with only a few forms. Assuming both languages have

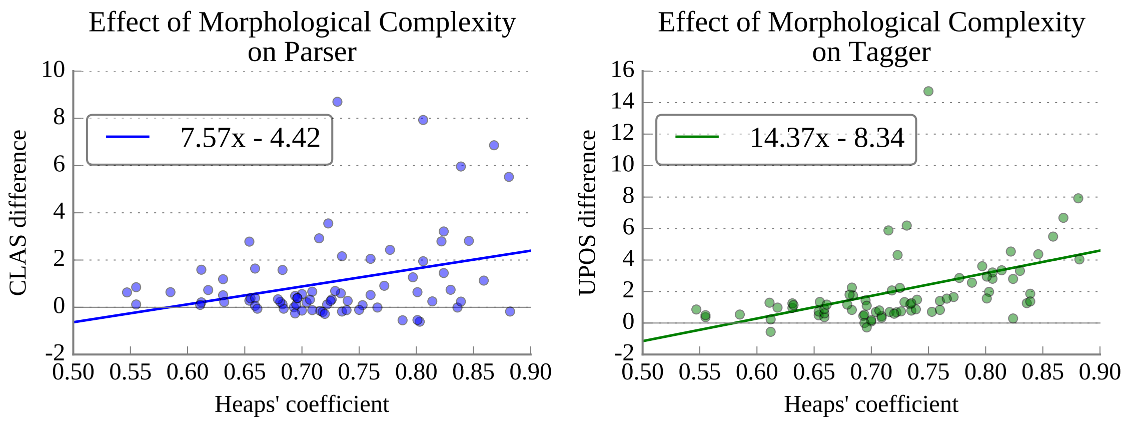


Figure 7: Performance difference between our character-based approach and a pure token-based approach for parsing (left) and tagging (right) as a function of approximated morphological complexity

the same number of lemmas, the vocabulary size of the complex language will then be larger. The most principled way of modeling this intuition is through Heaps' law (Herdan, 1960; Heaps, 1978) in Equation 22, which says that the log vocabulary size increases linearly in the log corpus size.

$$\log(|V^{(X)}|) = w \log(|X|) + b \tag{22}$$

We can take advantage of Heaps' law directly in approximating morphological complexity. Morphologically richer languages should increase the size of their vocabulary at a faster rate as the corpus size grows, because a new token being added to the corpus has a higher probability of having a previously observed lemma with a previously unobserved morphological form, thereby increasing the vocabulary size; in a morphologically simplex language, previously observed lemmas are unlikely to have many morphological forms that could increase $|V|$. Therefore, we would expect the parameter w of Equation 22 to be higher for languages with rich morphology. We computed this value for each treebank, and the results generally align with our intuition (although not without some variation, attributable to domain and dataset size): Hindi and Urdu—which have significant allomorphy—are among the lowest, having $w = .555$ and .585 respectively; English and Vietnamese have .631 and .661; Spanish and Portuguese have .7 and .704; and Finnish, Estonian, and Hungarian have some of the highest, at .806, .822, and .846.

Thus we use the coefficient w in Equation 22 as our metric for morphological richness, and plot the difference between models trained with character-level word embeddings and token-level word embeddings against this value in Figure 7. First we perform a Wilcoxon signed rank test, finding that the difference between the two approaches is statistically significant for the taggers ($p < 0.001$) and parsers ($p < 0.001$). Then we fit a mixed effects model to the data with treebank size and training/test projectivity as random effects, finding that the character-level approach tends to significantly improve performance more as complexity grows both for parsing ($p < 0.005$) and tagging ($p < 0.001$).[9] This indicates that incorporating subword information into UD parsing models is a promising way to improve performance on languages with significant morphology.

6 Conclusion

In this paper we describe our relatively simple neural system for parsing that achieved state-of-the-art performance on the 2017 CoNLL Shared Task on UD parsing without utilizing lemmas, morphological features, or ensembling. The system uses BiLSTM networks for tagging and parsing, and includes character-level word representations in addition to token-level ones. We also examined what can be learned more generally from our model's performance. We explore the relative performance of nonprojective graph-based and transition-based architectures on this task, finding evidence that modern graph-based parsers might be better at producing nonprojective arcs (with some caveats). Additionally, our network

[9]The assumption of linearity is clearly wrong, but the negative y-values preclude using a log-linear model on which we run significance tests

performs better when there's an abundance of data, suggesting that more regularization could improve accuracy on lower-resource languages.

We also sought to quantitatively justify the additional complexity of our system. We considered how important the POS tagger is to the system, comparing the downstream performance of parsers using our tagger, the baseline tagger, and no tagger at all. We find that our tagger beats both baselines significantly, whereas the two baselines don't statistically differ from each other, indicating that POS tags can help our system but must be sufficiently accurate. The character-based approach was found to significantly boost performance on languages that scored high on our metric for morphological complexity—both for parsing and tagging—suggesting that constructing token representation from subtoken information is effective for capturing the influence of morphology on syntax, and the naïve approach of using only holistic word embeddings is insufficient. Our success at the shared task demonstrates that a well-tuned, straightforward neural approach to parsing and tagging can get state-of-the-art performance for datasets with a wide variety of syntactic properties.

References

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

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *EMNLP* .

Kris Cao and Marek Rei. 2016. A joint model for word embedding and word morphology. *ACL 2016* page 18.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. *EMNLP 2016* .

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On shortest arborescence of a directed graph. *Scientia Sinica* 14(10):1396.

Michal Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. 2017. Frustratingly short attention spans in neural language modeling. *ICLR 2017* .

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. *ICLR 2017* .

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B* 71(4):233–240.

Ronald Aylmer Fisher. 1930. *The genetical theory of natural selection: a complete variorum edition*. Oxford University Press.

Yarin Gal and Zoubin Ghahramani. 2015. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning* .

Jarrod D Hadfield. 2010. Mcmc methods for multi-response generalized linear mixed models: The MCMCglmm R package. *Journal of Statistical Software* 33(2):1–22. http://www.jstatsoft.org/v33/i02/.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2016. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587* .

Harold Stanley Heaps. 1978. *Information retrieval: Computational and theoretical aspects*. Academic Press, Inc.

Gustav Herdan. 1960. *Type-token mathematics*, volume 4. Mouton.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations* .

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *NAACL* .

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 523–530.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *International Conference on Learning Representations* .

Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *ACL 2016*. pages 1400–1409.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*. Association for Computational Linguistics, pages 351–359.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017a. 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. http://hdl.handle.net/11234/1-2184.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portoro, Slovenia, pages 1659–1666.

Joakim Nivre et al. 2017b. 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.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. 2016. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *ACL* .

Martin Potthast, Tim Gollub, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. 2014. Improving the reproducibility of PAN's shared tasks: Plagiarism detection, author identification, and author profiling. In Evangelos Kanoulas, Mihai Lupu, Paul Clough, Mark Sanderson, Mark Hall, Allan Hanbury, and Elaine Toms, editors, *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14)*. Springer, Berlin Heidelberg New York, pages 268–299. https://doi.org/10.1007/978-3-319-11382-1_22.

Scott E. Reed and Nando de Freitas. 2016. Neural programmer-interpreters. *ICLR 2016* .

Milan Straka, Jan Hajic, Jana Straková, and Jan Hajic jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *International Workshop on Treebanks and Linguistic Theories (TLT14)*. page 208.

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.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gökırmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Hěctor Martínez Alonso, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadova, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2016. Dependency parsing as head selection. *EACL 2017* .