Universal Dependency Parsing from Scratch

Peng Qi,* Timothy Dozat,* Yuhao Zhang,* Christopher D. Manning
Stanford University
Stanford, CA 94305
{pengqi, tdozat, yuhaozhang, manning}@stanford.edu

Abstract

This paper describes Stanford's system at the *CoNLL 2018 UD Shared Task*. We introduce a complete neural pipeline system that takes raw text as input, and performs all tasks required by the shared task, ranging from tokenization and sentence segmentation, to POS tagging and dependency parsing. Our single system submission achieved very competitive performance on big treebanks. Moreover, after fixing an unfortunate bug, our corrected system would have placed the 2nd, 1st, and 3rd on the official evaluation metrics LAS, MLAS, and BLEX, and would have outperformed all submission systems on low-resource treebank categories on all metrics by a large margin. We further show the effectiveness of different model components through extensive ablation studies.

1 Introduction

Dependency parsing is an important component in various natural langauge processing (NLP) systems for semantic role labeling (Marcheggiani and Titov, 2017), relation extraction (Zhang et al., 2018), and machine translation (Chen et al., 2017). However, most research has treated dependency parsing in isolation, and largely ignored upstream NLP components that prepare relevant data for the parser, *e.g.*, tokenizers and lemmatizers (Zeman et al., 2017). In reality, however, these upstream systems are still far from perfect.

To this end, in our submission to the *CoNLL 2018 UD Shared Task*, we built a raw-text-to-CoNLL-U pipeline system that performs all tasks required by the Shared Task (Zeman et al., 2018).[1] Harnessing the power of neural systems, this pipeline achieves competitive performance in each of the inter-linked stages: tokenization, sentence and word segmentation, part-of-speech (POS)/morphological features (UFeats) tagging, lemmatization, and finally, dependency parsing. Our main contributions include:

- New methods for combining symbolic statistical knowledge with flexible, powerful neural systems to improve robustness;
- A biaffine classifier for joint POS/UFeats prediction that improves prediction consistency;
- A lemmatizer enhanced with an *edit* classifier that improves the robustness of a sequence-to-sequence model on rare sequences; and
- Extensions to our parser from (Dozat et al., 2017) to model linearization.

Our system achieves competitive performance on big treebanks. After fixing an unfortunate bug, the corrected system would have placed the 2nd, 1st, and 3rd on the official evaluation metrics LAS, MLAS, and BLEX, and would have outperformed all submission systems on low-resource treebank categories on all metrics by a large margin. We perform extensive ablation studies to demonstrate the effectiveness of our novel methods, and highlight future directions to improve the system.[2]

2 System Description

In this section, we present detailed descriptions for each component of our neural pipeline system, namely the tokenizer, the POS/UFeats tagger, the lemmatizer, and finally the dependency parser.

*These authors contributed roughly equally.

[1] We chose to develop a pipeline system mainly because it allows easier parallel development and faster model tuning in a shared task context.

[2] To facilitate future research, we make our implementation public at: https://github.com/stanfordnlp/stanfordnlp.

2.1 Tokenizer

To prepare sentences in the form of a list of words for downstream processing, the tokenizer component reads raw text and outputs sentences in the CoNLL-U format. This is achieved with two subsystems: one for joint tokenization and sentence segmentation, and the other for splitting multi-word tokens into syntactic words.

Tokenization and sentence segmentation. We treat joint tokenization and sentence segmentation as a unit-level sequence tagging problem. For most languages, a *unit* of text is a single character; however, in Vietnamese orthography, the most natural units of text are single *syllables*.[3] We assign one out of five tags to each of these units: end of token (EOT), end of sentence (EOS), multi-word token (MWT), multi-word end of sentence (MWS), and other (OTHER). We use bidirectional LSTMs (BiLSTMs) as the base model to make unit-level predictions. At each unit, the model predicts hierarchically: it first decides whether a given unit is at the end of a token with a score $s^{(\text{tok})}$, then classifies token endings into finer-grained categories with two independent binary classifiers: one for sentence ending $s^{(\text{sent})}$, and one for MWT $s^{(\text{MWT})}$.

Since sentence boundaries and MWTs usually require a larger context to determine (*e.g.*, periods following abbreviations or the ambiguous word "des" in French), we incorporate token-level information into a two-layer BiLSTM as follows (see also Figure 1). The first layer BiLSTM operates directly on raw units, and makes an initial prediction over the categories. To help capture local unit patterns more easily, we also combine the first-layer BiLSTM with 1-D convolutional networks, by using a one hidden layer convolutional network (CNN) with ReLU nolinearity at its first layer, giving an effect a little like a residual connection (He et al., 2016). The output of the CNN is simply added to the concatenated hidden states of the BiLSTM for downstream computation:

$$\mathbf{h}_1^{\text{RNN}} = [\overrightarrow{\mathbf{h}_1}, \overleftarrow{\mathbf{h}_1}] = \text{BiLSTM}_1(\mathbf{x}), \quad (1)$$

$$\mathbf{h}_1^{\text{CNN}} = \text{CNN}(\mathbf{x}), \quad (2)$$

$$\mathbf{h}_1 = \mathbf{h}_1^{\text{RNN}} + \mathbf{h}_1^{\text{CNN}}, \quad (3)$$

$$[\mathbf{s}_1^{(\text{tok})}, \mathbf{s}_1^{(\text{sent})}, \mathbf{s}_1^{(\text{MWT})}] = W_1\mathbf{h}_1, \quad (4)$$

where $\mathbf{x}$ is the input character representations, and W_1 contains the weights and biases for a linear classifier.[4] For each unit, we concatenate its trainable embedding with a four-dimensional binary feature vector as input, each dimension corresponding to one of the following feature functions: (1) does the unit start with whitespace; (2) does it start with a capitalized letter; (3) is the unit fully capitalized; and (4) is it purely numerical.

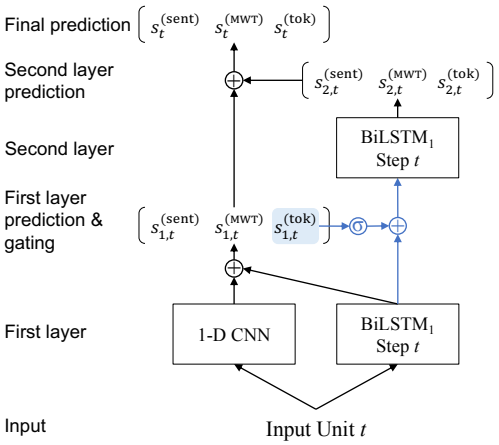

Figure 1: Illustration of the tokenizer/sentence segmenter model. Components in blue represent the gating mechanism between the two layers.

To incorporate token-level information at the second layer, we use a gating mechanism to suppress representations at non-token boundaries before propagating hidden states upward:

$$\mathbf{g}_1 = \mathbf{h}_1 \odot \sigma(\mathbf{s}_1^{(\text{tok})}) \quad (5)$$

$$\mathbf{h}_2 = [\overrightarrow{\mathbf{h}_2}, \overleftarrow{\mathbf{h}_2}] = \text{BiLSTM}_2(\mathbf{g}_1), \quad (6)$$

$$[\mathbf{s}_2^{(\text{tok})}, \mathbf{s}_2^{(\text{sent})}, \mathbf{s}_2^{(\text{MWT})}] = W_2\mathbf{h}_2, \quad (7)$$

where $\odot$ is an element-wise product broadcast over all dimensions of $\mathbf{h}_1$ for each unit. This can be viewed as a simpler alternative to multi-resolution RNNs (Serban et al., 2017), where the first-layer BiLSTM operates at the unit level, and the second layer operates at the token level. Unlike multi-resolution RNNs, this formulation is end-to-end differentiable, and can more easily leverage efficient off-the-shelf RNN implementations.

To combine predictions from both layers of the BiLSTM, we simply sum the scores to obtain $\mathbf{s}^{(X)} = \mathbf{s}_1^{(X)} + \mathbf{s}_2^{(X)}$, where $X \in \{\text{tok}, \text{sent}, \text{MWT}\}$. The final probability over the tags is then

$$p_{\text{EOT}} = p_{+--} \qquad p_{\text{EOS}} = p_{++-}, \quad (8)$$

$$p_{\text{MWT}} = p_{+-+} \qquad p_{\text{MWS}} = p_{+++}, \quad (9)$$

where $p_{\pm\pm\pm} = \sigma(\pm s^{(\text{tok})})\sigma(\pm s^{(\text{sent})})\sigma(\pm s^{(\text{MWT})})$,

[3] In this case, we define a syllable as a consecutive run of alphabetic characters, numbers, or individual symbols, together with any leading white spaces before them.

[4] We will omit bias terms in affine transforms for clarity.

and $\sigma(\cdot)$ is the logistic sigmoid function. p_{OTHER} is simply $\sigma(-s^{(\text{tok})})$. The model is trained to minimize the standard cross entropy loss.

Multi-word Token Expansion. The tokenizer/sentence segmenter produces a collection of sentences, each being a list of tokens, some of which are labeled as multi-word tokens (MWTs). We must expand these MWTs into the underlying syntactic words they correspond to (*e.g.*, "im" to "in dem" in German), in order for downstream systems to process them properly. To achieve this, we take a hybrid approach to combine symbolic statistical knowledge with the power of neural systems.

The symbolic statistical side is a frequency lexicon. Many languages, like German, have only a handful of rules for expanding a few MWTs. We leverage this information by simply counting the number of times a MWT is expanded into different sequences of words in the training set, and retaining the most frequent expansion in a dictionary to use at test time. When building this dictionary, we lowercase all words in the expansions to improve robustness. However, this approach would fail for languages with rich clitics, a large set of unique MWTs, and/or complex rules for MWT expansion, such as Arabic and Hebrew. We capture this by introducing a powerful neural system.

Specifically, we train a sequence-to-sequence model using a BiLSTM encoder with an attention mechanism (Bahdanau et al., 2015) in the form of a multi-layer perceptron (MLP). Formally, the input multi-word token is represented by a sequence of characters $x_1, \ldots, x_I$, and the output syntactic words are represented similarly as a sequence of characters $y_1, \ldots, y_J$, where the words are separated by space characters. Inputs to the RNNs are encoded by a shared matrix of character embeddings E. Once the encoder hidden states $\mathbf{h}^{\text{enc}}$ are obtained with a single-layer BiLSTM, each decoder step is unrolled as follows:

$$\mathbf{h}^{\text{dec}}_j = \text{LSTM}_{\text{dec}}(E_{y_{j-1}}, \mathbf{h}^{\text{dec}}_{j-1}), \quad (10)$$

$$\alpha_{ij} \propto \exp(\mathbf{u}_\alpha^\top \tanh(W_\alpha[\mathbf{h}^{\text{dec}}_j, \mathbf{h}^{\text{enc}}_i])), \quad (11)$$

$$\mathbf{c}_j = \sum_i \alpha_{ij} \mathbf{h}^{\text{enc}}_i, \quad (12)$$

$$P(y_j = w | y_{<j}) \propto \mathbf{u}_w^\top \tanh(W[\mathbf{h}^{\text{dec}}_j, \mathbf{c}_j]). \quad (13)$$

Here, w is a character index in the output vocabulary, y_0 a special start-of-sequence symbol in the vocabulary, and $\mathbf{h}^{\text{dec}}_0$ the concatenation of the last hidden states of each direction of the encoder.

To bring the symbolic and neural systems together, we train them separately and use the following protocol during evaluation: for each MWT, we first look it up in the dictionary, and return the expansion recorded there if one can be found. If this fails, we retry by lowercasing the incoming token. If that fails again, we resort to the neural system to predict the final expansion. This allows us to not only account for languages with flexible MWTs patterns (Arabic and Hebrew), but also leverage the training set statistics to cover both languages with simpler MWT rules, and MWTs in the flexible languages seen in the training set without fail. This results in a high-performance, robust system for multi-word token expansion.

2.2 POS/UFeats Tagger

Our tagger follows closely that of (Dozat et al., 2017), with a few extensions. As in that work, the core of the tagger is a highway BiLSTM (Srivastava et al., 2015) with inputs coming from the concatenation of three sources: (1) a pretrained word embedding, from the word2vec embeddings provided with the task when available (Mikolov et al., 2013), and from fastText embeddings otherwise (Bojanowski et al., 2017); (2) a trainable frequent word embedding, for all words that occurred at least seven times in the training set; and (3) a character-level embedding, generated from a unidirectional LSTM over characters in each word. UPOS is predicted by first transforming each word's BiLSTM state with a fully-connected (FC) layer, then applying an affine classifier:

$$\mathbf{h}_i = \text{BiLSTM}^{(\text{tag})}_i(\mathbf{x}_1, \ldots, \mathbf{x}_n), \quad (14)$$

$$\mathbf{v}^{(\text{u})}_i = \text{FC}^{(\text{u})}(\mathbf{h}_i), \quad (15)$$

$$P(y^{(\text{u})}_{ik} | X) = \text{softmax}_k(W^{(\text{u})} \mathbf{v}^{(\text{u})}_i). \quad (16)$$

To predict XPOS, we similarly start with transforming the BiLSTM states with an FC layer. In order to further ensure consistency between the different tagsets (*e.g.*, to avoid a VERB UPOS with an NN XPOS), we use a *biaffine* classifier, conditioned on a word's XPOS state as well as an embedding for its gold (at training time) or predicted (at inference time) UPOS tag $y^{(\text{u})}_{i*}$:

$$\mathbf{v}^{(\text{x})}_i = \text{FC}^{(\text{x})}(\mathbf{h}_i), \quad (17)$$

$$\mathbf{s}^{(\text{x})}_i = [E^{(\text{u})}_{y^{(\text{u})}_{i*}}, 1]^\top \mathbf{U}^{(\text{x})} [\mathbf{v}^{(\text{x})}_i, 1], \quad (18)$$

$$P(y^{(\text{x})}_{ik} | y^{(\text{u})}_{i*}, X) = \text{softmax}_k(\mathbf{s}^{(\text{x})}_i). \quad (19)$$

UFeats is predicted analogously with separate parameters for each individual UFeat tag. The tagger is also trained to minimize the cross entropy loss.

Some languages have composite XPOS tags, yielding a very large XPOS tag space (*e.g.*, Arabic and Czech). For these languages, the biaffine classifier requires a prohibitively large weight tensor $\mathbf{U}^{(\mathrm{x})}$. For languages that use XPOS tagsets with a fixed number of characters, we classify each character of the XPOS tag in the same way we classify each UFeat. For the rest, instead of taking the biaffine approach, we simply share the FC layer between all three affine classifiers, hoping that the learned features for one will be used by another.

2.3 Lemmatizer

For the lemmatizer, we take a very similar approach to that of the multi-word token expansion component introduced in Section 2.1 with two key distinctions customized to lemmatization.

First, we build two dictionaries from the training set, one from a (word, UPOS) pair to the lemma, and the other from the word itself to the lemma. During evaluation, the predicted UPOS is used. When the UPOS-augmented dictionary fails, we fall back to the word-only dictionary before resorting to the neural system. In looking up both dictionaries, the word is never lowercased, because case information is more relevant in lemmatization than in MWT expansion.

Second, we enhance the neural system with an *edit* classifier that shortcuts the prediction process to accommodate rare, long words, on which the decoder is more likely to flounder. The concatenated encoder final states are put through an FC layer with ReLU nonlinearity and fed into a 3-way classifier, which predicts whether the lemma is (1) exactly identical to the word (*e.g.*, URLs and emails), (2) the lowercased version of the word (*e.g.*, capitalized rare words in English that are not proper nouns), or (3) in need of the sequence-to-sequence model to make more complex edits to the character sequence. During training time, we assign the labels to each word-lemma pair greedily in the order of identical, lowercase, and sequence decoder, and train the classifier jointly with the sequence-to-sequence lemmatizer. At evaluation time, predictions are made sequentially, *i.e.*, the classifier first determines whether any shortcut can be taken, before the sequence decoder model is used if needed.

2.4 Dependency Parser

The dependency parser also follows that of (Dozat et al., 2017) with a few augmentations. The highway BiLSTM takes as input pretrained word embeddings, frequent word and lemma embeddings, character-level word embeddings, summed XPOS and UPOS embeddings, and summed UFeats embeddings. In (Dozat et al., 2017), unlabeled attachments are predicted by scoring each word i and its potential heads with a biaffine transformation

$$\mathbf{h}_t = \mathrm{BiLSTM}_t^{(\mathrm{parse})}(\mathbf{x}_1, \ldots, \mathbf{x}_n), \quad (20)$$

$$\mathbf{v}_i^{(\mathrm{ed})}, \mathbf{v}_j^{(\mathrm{eh})} = \mathrm{FC}^{(\mathrm{ed})}(\mathbf{h}_i), \mathrm{FC}^{(\mathrm{eh})}(\mathbf{h}_j), \quad (21)$$

$$s_{ij}^{(\mathrm{e})} = [\mathbf{v}_j^{(\mathrm{eh})}, 1]^\top U^{(\mathrm{e})} [\mathbf{v}_i^{(\mathrm{ed})}, 1], \quad (22)$$

$$= \text{Deep-Biaff}^{(\mathrm{e})}(\mathbf{h}_i, \mathbf{h}_j), \quad (23)$$

$$P(y_{ij}^{(\mathrm{e})}|X) = \mathrm{softmax}_j(\mathbf{s}_i^{(\mathrm{e})}), \quad (24)$$

where $\mathbf{v}_i^{(\mathrm{ed})}$ is word i's *edge-dependent* representation and $\mathbf{v}_i^{(\mathrm{eh})}$ its *edge-head* representation. This approach, however, does not explicitly take into consideration relative locations of heads and dependents during prediction; instead, such predictive location information must be implicitly learned by the BiLSTM. Ideally, we would like the model to explicitly condition on $(i-j)$, namely the dependent i and its potential head j's location relative to each other, in modeling $p(y_{ij})$.[5]

Here, we motivate one way to build this into the model. First we factorize the relative location of word i and head j into their *linear order* and the *distance* between them, *i.e.*, $P(y_{ij}|\mathrm{sgn}(i-j), \mathrm{abs}(i-j))$, where $\mathrm{sgn}(\cdot)$ is the sign function. Applying Bayes' rule and assuming conditional independence, we arrive at the following

$$P(y_{ij}|\mathrm{sgn}(i-j), \mathrm{abs}(i-j)) \propto \quad (25)$$
$$P(y_{ij}) P(\mathrm{sgn}(i-j)|y_{ij}) P(\mathrm{abs}(i-j)|y_{ij}).$$

In a language where heads always follow their dependents, $P(\mathrm{sgn}(i-j) = 1|y_{ij})$ would be extremely low, heavily penalizing rightward attachments. Similarly, in a language where dependencies are always short, $P(\mathrm{abs}(i-j) \gg 0|y_{ij})$ would be extremely low, penalizing longer edges.

$P(y_{ij})$ can remain the same as computed in Eq. (24). $P(\mathrm{sgn}(i-j)|y_{ij})$ can be computed similarly with a deep biaffine scorer (cf. Eqs. (20)–(23)) over the recurrent states. This results in the score of j preceding i; flipping the sign wherever i precedes j turns this into the log odds of the ob-

[5] Henceforth we omit the (e) superscript and X.

served linearization. Applying the sigmoid function then turns it into a probability:

$$s_{ij}^{(\mathrm{l})} = \text{Deep-Biaff}^{(\mathrm{l})}(\mathbf{h}_i, \mathbf{h}_j), \quad (26)$$

$$s_{ij}^{\prime(\mathrm{l})} = \mathrm{sgn}(i-j)s_{ij}^{(\mathrm{l})}, \quad (27)$$

$$P(\mathrm{sgn}(i-j)|y_{ij}) = \sigma\big(s_{ij}^{\prime(\mathrm{l})}\big). \quad (28)$$

This can be effortlessly incorporated into the edge score by adding in the log of this probability $-\log(1+\exp(-s_{ij}^{\prime(\mathrm{l})}))$. Error is not backpropagated to this submodule through the final attachment loss; instead, it is trained with its own cross entropy, with error only computed on gold edges. This ensures that the model learns the conditional probability *given a true edge*, rather than just learning to predict the linear order of two words.

For $P(\mathrm{abs}(i-j)|y_{ij})$, we use another deep biaffine scorer to generate a distance score. Distances are always no less than 1, so we apply $1+\mathrm{softplus}$ to predict the distance between i and j when there's an edge between them:

$$s_{ij}^{(\mathrm{d})} = \text{Deep-Biaff}^{(\mathrm{d})}(\mathbf{h}_i, \mathbf{h}_j), \quad (29)$$

$$s_{ij}^{\prime(\mathrm{d})} = 1 + \mathrm{softplus}\big(s_{ij}^{(\mathrm{d})}\big). \quad (30)$$

where $\mathrm{softplus}(x) = \log(1+\exp(x))$. The distribution of edge lengths in the treebanks roughly follows a Zipfian distribution, to which the Cauchy distribution is closely related, only the latter is more stable for values at or near zero. Thus, rather than modeling the probability of an arc's length, we can use the Cauchy distribution to model the probability of an arc's *error* in predicted length, namely how likely it is for the predicted distance and the true distance to have a difference of $\delta_{ij}^{(\mathrm{d})}$:

$$\mathrm{Zipf}(k;\alpha,\beta) \propto (k^{\alpha}/\beta)^{-1}, \quad (31)$$

$$\mathrm{Cauchy}(x;\gamma) \propto (1+x^2/\gamma)^{-1} \quad (32)$$

$$\delta_{ij}^{(\mathrm{d})} = \mathrm{abs}(i-j) - s_{ij}^{\prime(\mathrm{d})}, \quad (33)$$

$$P(\mathrm{abs}(i-j)|y_{ij}) \propto (1+\delta_{ij}^{2(\mathrm{d})}/2)^{-1}. \quad (34)$$

When the difference $\delta_{ij}^{(\mathrm{d})}$ is small or zero, there will be effectively no penalty; but when the model expects a significantly longer or shorter arc than the observed distance between i and j, it is discouraged from assigning an edge between them. As with the linear order probability, the log of the distance probability is added to the edge score, and trained with its own cross-entropy on gold edges.[6]

At inference time, the Chu-Liu/Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) is used to ensure a maximum spanning tree. Dependency relations are assigned to gold (at training time) or predicted (at inference time) edges $y_{i*}^{(\mathrm{e})}$ using another deep biaffine classifier, following (Dozat et al., 2017) with no augmentations:

$$\mathbf{s}_i^{(\mathrm{r})} = \text{Deep-Biaff}^{(\mathrm{r})}\big(\mathbf{h}_i, \mathbf{h}_{y_{i*}^{(\mathrm{e})}}\big), \quad (35)$$

$$P\big(y_{ik}^{(\mathrm{r})}|y_{i*}^{(\mathrm{e})}\big) = \mathrm{softmax}_k\big(\mathbf{s}_i^{(\mathrm{r})}\big). \quad (36)$$

3 Training Details

Except where otherwise stated, our system is a pipeline: given a document of raw text, the tokenizer/sentence segmenter/MWT expander first splits it into sentences of syntactic words; the tagger then assigns UPOS, XPOS and UFeat tags to each word; the lemmatizer takes the predicted word and UPOS tag and outputs a lemma; finally, the parser takes all annotations as input and predicts the head and dependency label for each word.

All components are trained with early stopping on the dev set when applicable. When a dev set is unavailable, we split the training set into an approximately 7-to-1 split for training and development. All components (except the dependency parser) are trained and evaluated on the development set assuming all related components had oracle implementations. This means the tokenizer/sentence segmenter assumes all correctly predicted MWTs will be correctly expanded, the MWT expander assumes gold word segmentation, and all downstream tasks assume gold word segmentation, along with gold annotations of all prerequisite tasks. The dependency parser is trained with predicted tags and morphological features from the POS/UFeats tagger.

Treebanks without training data. For treebanks without training data, we adopt a heuristic approach for finding replacements. Where a larger treebank in the same language is available (*i.e.*, all PUD treebanks and Japanese-Modern), we used the models from the largest treebank available in that language. Where treebanks in related languages are available (as determined by language families from Wikipedia), we use models from the largest treebank in that related language. We

[6] Note that the penalty assigned to the edge score in this way is proportional to $\ln \delta_{ij}^{(\mathrm{d})}$ for high $\delta_{ij}^{(\mathrm{d})}$; using a Gamma or Poisson distribution to model the distance directly, or using a normal distribution instead of Cauchy, respectively, assigns penalties roughly proportional to $\delta_{ij}^{(\mathrm{d})}$, $\ln\Gamma(\delta_{ij}^{(\mathrm{d})})$, and $\delta_{ij}^{2(\mathrm{d})}$. Thus, the Cauchy is more numerically stable during training.

ended up choosing the models from English-EWT for Naija (an English-based pidgin), Irish-IDT for Breton (both are Celtic), and Norwegian-Nynorsk for Faroese (both are West Scandinavian). For Thai, since it uses a different script from all other languages, we use UDPipe 1.2 for all components.

Hyperparameters. The tokenizer/sentence segmenter uses BiLSTMs with 64d hidden states in each direction and takes 32d character embeddings as input. During training, we employ dropout to the input embeddings and hidden states at each layer with $p = .33$. We also randomly replace the input unit with a special `<UNK>` unit with $p = .33$, which would be used in place of any unseen input at test time. We add noise to the gating mechanism in Eq. (6) by randomly setting the gates to 1 with $p = .02$ and setting its temperature to 2 to make the model more robust to tokenization errors at test time. Optimization is performed with Adam (Kingma and Ba, 2015) with an initial learning rate of .002 for up to 20,000 steps, and whenever dev performance deteriorates, as is evaluated every 200 steps after the 2,000$^{\text{th}}$ step, the learning rate is multiplied by .999. For the convolutional component we use filter sizes of 1 and 9, and for each filter size we use 64 channels (same as one direction in the BiLSTM). The convolutional outputs are concatenated in the hidden layer, before an affine transform is applied to serve as a residual connection for the BiLSTM. For the MWT expander, we use BiLSTMs with 256d hidden states in each direction as the encoder, a 512d LSTM decoder, 64d character embeddings as input, and dropout rate $p = .5$ for the inputs and hidden states. Models are trained up to 100 epochs with the standard Adam hyperparameters, and the learning rate is annealed similarly every epoch after the 15$^{\text{th}}$ epoch by a factor of 0.9. Beam search of beam size 8 is employed in evaluation.

The lemmatizer uses BiLSTMs with 100d hidden states in each direction of the encoder, 50d character embeddings as input, and dropout rate $p = .5$ for the inputs and hidden states. The decoder is an LSTM with 200d hidden states. During training we jointly minimize (with equal weights) the cross-entropy loss of the edit classifier and the negative log-likelihood loss of the seq2seq lemmatizer. Models are trained up to 60 epochs with standard Adam hyperparameters.

The tagger and parser share most of their hyperparameters. We use 75d uncased frequent word and lemma embeddings, and 50d POS tag and UFeat embeddings. Pretrained embeddings and character-based word representations are both transformed to be 125d. During training, all embeddings are randomly replaced with a `<drop>` symbol with $p = .33$. We use 2-layer 200d BiLSTMs for the tagger and 3-layer 400d BiLSTMs for the parser. We employ dropout in all feed-forward connections with $p = .5$ and all recurrent connections (Gal and Ghahramani, 2016) with $p = .25$ (except $p = .5$ in the tagger BiLSTM). All classifiers use 400d FC layers (except 100d for UFeats) with the ReLU nonlinearity. We train the systems with Adam ($\alpha = .003$, $\beta_1 = .9$, $\beta_2 = .95$) until dev accuracy decreases, at which point we switch to AMSGrad (Reddi et al., 2018) until 3,000 steps pass with no dev accuracy increases.

4 Results

The main results are shown in Table 1. As can be seen from the table, our system achieves competitive performance on nearly all of the metrics when macro-averaged over all treebanks. Moreover, it achieves the top performance on several metrics when evaluated only on big treebanks, showing that our systems can effectively leverage statistical patterns in the data. Where it is not the top performing system, our system also achieved competitive results on each of the metrics on these treebanks. This is encouraging considering that our system is comprised of single-system components, whereas some of the best performing teams used ensembles (*e.g.*, HIT-SCIR (Che et al., 2018)).

When taking a closer look, we find that our UFeats classifier is very accurate on these treebanks as well. Not only did it achieve the top performance on UFeats F_1, but also it helped the parser achieve top MLAS as well on big treebanks, even when the parser is not the best-performing as evaluated by other metrics. We also note the contribution from our consistency modeling in the POS tagger/UFeats classifier: in both settings the individual metrics (UPOS, XPOS, and UFeats) achieve a lower advantage margin over the reference systems when compared to the AllTags metric, showing that these reference systems, though sometimes more accurate on each individual task, are not as consistent as our system overall.

The biggest disparity between the all-treebanks and big-treebanks results comes from sentence

(a) Results on all treebanks

System	Tokens	Sent	Words	Lemmas	UPOS	XPOS	UFeats	AllTags	UAS	CLAS	**LAS**	**MLAS**	**BLEX**
Stanford	96.19	76.55	95.99	88.32	89.01	85.51	85.47	79.71	76.78	68.73	72.29	60.92	64.04
Reference	98.42†	83.87†	98.18‡	91.24*	90.91‡	86.67*	87.59‡	80.30*	80.51†	72.36†	75.84†	61.25*	66.09*
Δ	–2.23	–7.32	–2.19	–2.92	–1.90	–1.16	–2.12	–0.59	–3.73	–3.63	–3.55	–0.33	–2.05
Stanford+	97.42	85.46	97.23	89.17	89.95	86.50	86.20	80.36	79.04	70.39	74.16	62.08	65.28
Δ	–1.00	**+1.59**	–0.95	–2.07	–0.96	–0.17	–1.39	**+0.06**	–1.47	–1.97	–1.68	**+0.83**	–0.81

(b) Results on big treebanks only

System	Tokens	Sent	Words	Lemmas	UPOS	XPOS	UFeats	AllTags	UAS	CLAS	**LAS**	**MLAS**	**BLEX**
Stanford	99.43	89.52	99.21	95.25	95.93	94.95	94.14	91.50	86.56	79.60	83.03	72.67	75.46
Reference	99.51†	87.73†	99.16†	96.08*	96.23†	95.16†	94.11*	91.45*	87.61†	81.29†	84.37†	71.71*	75.83*
Δ	–0.08	**+1.79**	**+0.05**	–0.83	–0.30	–0.21	**+0.03**	**+0.05**	–1.05	–1.69	–1.34	**+0.96**	–0.37

Table 1: Evaluation results (F_1) on the test set, on all treebanks and big treebanks only. For each set of results on all metrics, we compare it against results from reference systems. A reference system is the top performing system on that metric if we are not top, or the second-best performing system on that metric. Reference systems are identified by superscripts (†: HIT-SCIR, ‡: Uppsala, ⋆: TurkuNLP, ∗: UDPipe Future). Shaded columns in the table indicate the three official evaluation metrics. "Stanford+" is our system after a bugfix evaluated unofficially; for more details please see the main text.

Treebanks	System	**LAS**	**MLAS**	**BLEX**
Small	Stanford+	**83.90**	**72.75**	**77.30**
	Reference	69.53†	49.24‡	54.89‡
Low-Res	Stanford+	**63.20**	**51.64**	**53.58**
	Reference	27.89*	6.13*	13.98*
PUD	Stanford+	**82.25**	**74.20**	**74.37**
	Reference	74.20†	58.75*	63.25$^{\bullet}$

Table 2: Evaluation results (F_1) on low-resource treebank test sets. Reference systems are identified by symbol superscripts (†: HIT-SCIR, ‡: ICS PAS, ⋆: CUNI x-ling, ∗: Stanford, •: TurkuNLP).

segmentation. After inspecting the results on smaller treebanks and double-checking our implementation, we noticed issues with how we processed data in the tokenizer that negatively impacted generalization on these treebanks.[7] This is devastating for these treebanks, as all downstream components process words at the sentence level.

We fixed this issue, and trained new tokenizers with all hyperparameters identical to our system at submission. We further built an unofficial evaluation pipeline, which we verified achieves the same evaluation results as the official system, and evaluated our entire pipeline by *only* replacing the tokenizer. As is shown in Table 1, the resulting system (Stanford+) is much more accurate overall, and we would have ranked 2$^{\text{nd}}$, 1$^{\text{st}}$, and 3$^{\text{rd}}$ on the official evaluation metrics LAS, MLAS, and BLEX, respectively.[8] On big treebanks, all metrics changed within only 0.02% F_1 and are thus not included. On small treebanks, however, this effect is more pronounced: as is shown in Table 2, our corrected system outperforms all submission systems on all official evaluation metrics on all low-resource treebanks by a large margin.

5 Analysis

In this section, we perform ablation studies on the new approaches we proposed for each component, and the contribution of each component to the final pipeline. For each component, we assume access to an oracle for all other components in the analysis, and show their efficacy on the dev sets.[9] For the ablations on the pipeline, we report macro-averaged F_1 on the test set.

[7] Specifically, our tokenizer was originally designed to be aware of newlines (`\n`) in double newline-separated paragraphs, but we accidentally prepared training and dev sets for low resource treebanks by putting each sentence on its own line in the text file. This resulted in the sentence segmenter overfitting to relying on newlines. In later experiments, we replaced all in-paragraph whitespaces with space characters.

[8] We note that the only system that is more accurate than ours on LAS is HIT's ensemble system, and we achieve very close performance to their system on MLAS (only 0.05% F_1 lower, which is likely within the statistical variation reported in the official evaluation).

[9] We perform treebank-level paired bootstrap tests for each ablated system against the top performing system in ablation with 10^5 bootstrap samples, and indicate statistical significance in tables with symbol superscripts (*:$p < 0.05$, **:$p < 0.01$, ***:$p < 0.001$).

System	Tokens	Sentences	Words
Stanford+	99.46	91.33	**99.27**
– *gating*	**99.47**	**91.34**	**99.27**
– *conv*	99.45	91.03	98.67
– *seq2seq*	–	–	98.97
– *dropout*	99.22*	88.78***	98.98*

Table 3: Ablation results for the tokenizer. All metrics in the table are macro-averaged dev F_1.

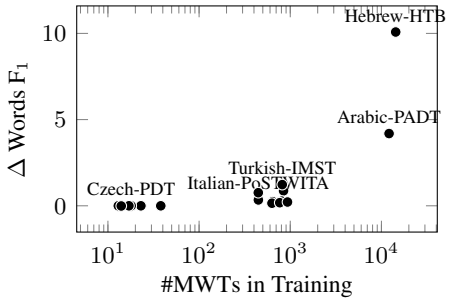

Figure 2: Effect of the seq2seq component for MWT expansion in the tokenizer.

Tokenizer. We perform ablation studies on the less standard components in the tokenizer, namely the gating mechanism in Eq. (6) (*gating*), the convolutional residual connections (*conv*), and the seq2seq model in the MWT expander (*seq2seq*), on all 61 big treebanks. As can be seen in Table 3, all but the gating mechanism make noticeable differences in macro F_1. When taking a closer look, we find that both *gating* and *conv* show a mixed contribution to each treebank, and we could have improved overall performance further through treebank-level component selection. One surprising discovery is that *conv* greatly helps identify MWTs in Hebrew (+34.89 Words F_1) and sentence breaks in Ancient Greek-PROIEL (+18.77 Sents F_1). In the case of *seq2seq*, although the overall macro difference is small, it helps with the word segmentation performance on all treebanks where it makes any meaningful difference, most notably +10.08 on Hebrew and +4.19 on Arabic in Words F_1 (see also Figure 2). Finally, we note that *dropout* plays an important role in safeguarding the tokenizer from overfitting.

POS/UFeats Tagger. The main novelty in our tagger is the explicit conditioning of XPOS and UFeats predictions on the UPOS prediction. We compare this against a tagger that simply shares the hidden features between the UPOS, XPOS, and UFeats classifiers. Since we used full-rank tensors in the biaffine classifier, treebanks with large, composite XPOS tagsets would incur prohibitive memory requirements. We therefore exclude treebanks that either have more than 250 XPOS tags or don't use them, leaving 36 treebanks for this analysis. We also measure consistency between tags by their pointwise mutual information

$$\text{PMI} = \log\left(\frac{p_c(\text{AllTags})}{p_c(\text{UPOS})p_c(\text{XPOS})p_c(\text{UFeats})}\right),$$

where $p_c(X)$ is the accuracy of X. This quantifies (in nats) how much more likely it is to get all tags right than we would expect given their individual accuracies, if they were independent. As can be seen in Table 4, the added parameters do not affect UPOS performance significantly, but do help improve XPOS and UFeats prediction. Moreover, the biaffine classifier is markedly more consistent than the affine one with shared representations.

System	UPOS	XPOS	UFeats	AllTags	PMI
Stanford	**96.50**	**95.87**	**95.01**	**92.52**	**.0514**
– *biaff*	96.47	95.71*	94.13***	91.32***	.0497*

Table 4: Ablation results for the tagger. All metrics are macro-averaged dev F_1, except PMI, which is explained in detail in the main text.

System	Big	Small	LowRes	All
Stanford	**96.56**	91.72*	**69.21**	**94.22**
– *edit & seq2seq*	89.97***	82.68***	63.50**	87.45***
– *edit*	96.48*	**91.80**	68.30	94.10
– *dictionaries*	95.37***	90.43***	66.02*	92.89***

Table 5: Ablation results for the lemmatizer, split by different groups of treebanks. All metrics in the table are macro-averaged dev F_1.

Lemmatizer. We perform ablation studies on three individual components in our lemmatizer: the edit classifier (*edit*), the sequence-to-sequence module (*seq2seq*) and the dictionaries (*dictionaries*). As shown in Table 5, we find that our lemmatizer with all components achieves the best overall performance. Specifically, adding the neural components (i.e., *edit & seq2seq*) drastically improves overall lemmatization performance over a simple dictionary-based approach (+6.77 F_1), and the gains are consistent over different treebank groups. While adding the edit classifier slightly decreases the F_1 score on small treebanks, it improves the performance on low-resource languages substantially (+0.91 F_1), and therefore leads to an overall gain of 0.11 F_1. Tree-

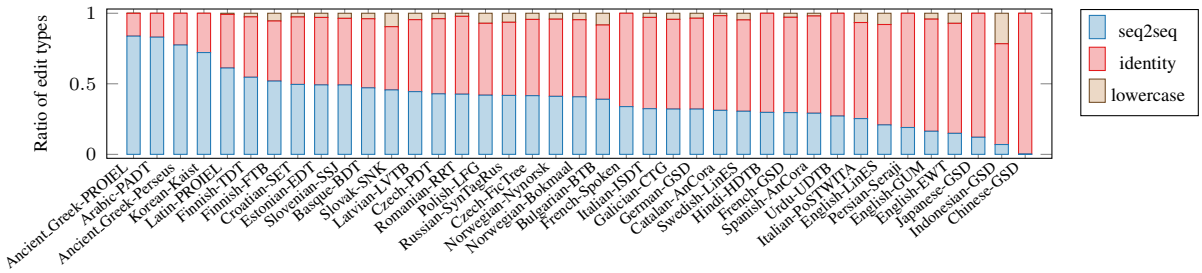

Figure 3: Edit operation types as output by the *edit* classifier on the official dev set. Due to space limit only treebanks containing over 120k dev words are shown and sorted by the ratio of *seq2seq* operation.

System	LAS	CLAS
Stanford	**87.60**	**84.68**
– *linearization*	87.55*	84.62*
– *distance*	87.43***	84.48***

Table 6: Ablation results for the parser. All metrics in the table are macro-averaged dev F_1.

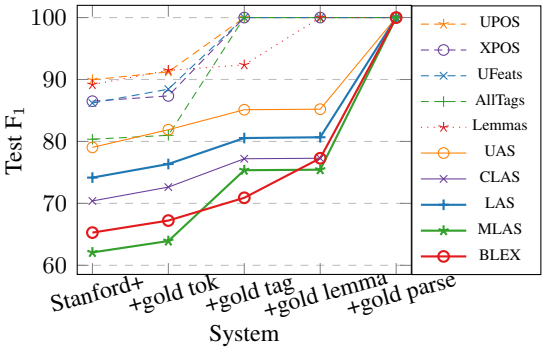

Figure 4: Pipeline ablation results. Dashed, dotted, and solid lines represent tagger, lemmatizer, and parser metrics, respectively. Official evaluation metrics are highlighted with thickened lines.

banks where the largest gains are observed include Upper_Sorbian-UFAL (+4.55 F_1), Kurmanji-MG (+2.27 F_1) and English-LinES (+2.16 F_1). Finally, combining the neural lemmatizer with dictionaries helps capture common lemmatization patterns seen during training, leading to substantial improvements on all treebank groups.

To further understand the behavior of the edit classifier, for each treebank we present the ratio of all predicted edit types on dev set words in Figure 3. We find that the behavior of the edit classifier aligns well with linguistic knowledge. For example, while Ancient Greek, Arabic and Korean require a lot of complex edits in lemmatization, the vast majority of operations in Chinese and Japanese are simple identity mappings.

Dependency Parser. The main innovation for the parsing module is terms that model locations of a dependent word relative to possible head words in the sentence. Here we examine the impact of these terms, namely linearization (Eq. (28)) and distance (Eq. (34)). For this analysis, we exclude six treebanks with very small dev sets. As can be seen in Table 6, both terms contribute significantly to the final parser performance, with the distance term contributing slightly more.

Pipeline Ablation. We analyze the contribution of each pipeline component by incrementally replacing them with gold annotations and observing performance change. As shown in Figure 4, most downstream systems benefit moderately from gold sentence and word segmentation, while the parser largely only benefits from improved POS/UFeats tagger performance (aside from BLEX, which is directly related to lemmatization performance and benefits notably). Finally, we note that the parser still is far from perfect even given gold annotations from all upstream tasks, but our components in the pipeline are very effective at closing the gap between predicted and gold annotations.

6 Conclusion & Future Directions

In this paper, we presented Stanford's submission to the *CoNLL 2018 UD Shared Task*. Our submission consists of neural components for each stage of a pipeline from raw text to dependency parses. The final system was very competitive on big treebanks; after fixing our preprocessing bug, it would have outperformed all official systems on all metrics for low-resource treebank categories.

One of the greatest opportunities for further gains is through the use of context-sensitive word embeddings, such as ELMo (Peters et al., 2018) and ULMfit (Howard and Ruder, 2018). Although this requires a large resource investment, HIT-SCIR (Che et al., 2018) has shown solid improvements from incorporating these embeddings.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR* .

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5:135–146. http://aclweb.org/anthology/Q17-1010.

Wanxiang Che, Yijia Liu, Yuxuan Zheng Bo Wang, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017. Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica* 14:1396–1400.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 Shared Task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. pages 20–30. http://www.aclweb.org/anthology/K/K17/K17-3002.pdf.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards* 71:233–240.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*. pages 1050–1059.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR* .

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing*.

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.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the convergence of Adam and beyond. *ICLR* .

Iulian Vlad Serban, Tim Klinger, Gerald Tesauro, Kartik Talamadupula, Bowen Zhou, Yoshua Bengio, and Aaron C Courville. 2017. Multiresolution recurrent neural networks: An application to dialogue response generation. In *AAAI*. pages 3288–3294.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. In *Proceedings of the Deep Learning Workshop at the International Conference on Machine Learning*.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics, Brussels, Belgium, pages 1–20.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinkova, Jan Hajic jr., Jaroslava Hlavacova, Václava Kettnerová, Zdenka Uresova, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria dePaiva, Kira Droganova, Héctor Martínez Alonso, Çağr Çöltekin, Umut Sulubacak, 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 Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonca, 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, pages 1–19.

Yuhao Zhang, Peng Qi, and Christopher D. Manning. 2018. Graph convolution over pruned dependency

trees improves relation extraction. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing*.