Dependency Parsing with Bottom-up Hierarchical Pointer Networks

Daniel Fernández-González and Carlos Gómez-Rodríguez
Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Group
Dept. de Ciencias de la Computación y Tecnologías de la Información
Campus de Elviña, s/n, 15071 A Coruña, Spain
d.fgonzalez@udc.es, carlos.gomez@udc.es

Abstract

Dependency parsing is a crucial step towards deep language understanding and, therefore, widely demanded by numerous Natural Language Processing applications. In particular, left-to-right and top-down transition-based algorithms that rely on Pointer Networks are among the most accurate approaches for performing dependency parsing. Additionally, it has been observed for the top-down algorithm that Pointer Networks’ sequential decoding can be improved by implementing a hierarchical variant, more adequate to model dependency structures. Considering all this, we develop a bottom-up-oriented Hierarchical Pointer Network for the left-to-right parser and propose two novel transition-based alternatives: an approach that parses a sentence in right-to-left order and a variant that does it from the outside in. We empirically test the proposed neural architecture with the different algorithms on a wide variety of languages, outperforming the original approach in practically all of them and setting new state-of-the-art results on the English and Chinese Penn Treebanks for non-contextualized and BERT-based embeddings.

1 Introduction

Dependency parsing consists in representing the grammatical structure of a given sentence by attaching each word to another (which will be considered its head or parent), to finally gather all these directed links into a dependency tree as the one in Figure 1. Additionally, these directed links or dependencies are enhanced with labels that describe syntactic functions.

This syntactic information accurately provided by parsers as dependency trees has been demonstrated highly useful for a huge variety of Natural Language Processing (NLP) tasks. In particular, dependency parsing has been recently used for machine translation (Zhang et al., 2019; Yang et al., 2020; Zhang et al., 2021), opinion mining (Zhang et al., 2020a), relation and event extraction (Nguyen and Verspoor, 2019), question answering (Cao et al., 2021), sentiment classification (Bai et al., 2021), sentence classification (Zhang et al., 2021), summarization (Balachandran et al., 2021) or semantic role labeling and named entity recognition (Sachan et al., 2021), among others.

Pointer Networks (Vinyals et al., 2015) have notably succeeded in implementing highly accurate versions of one of the most widely used dependency parsing paradigms: transition-based dependency parsers. In particular, two different algorithms have been proposed: a top-down approach (Ma et al., 2018) that, at each step and starting from the root node, connects each word to one of its children; and a left-to-right variant (Fernández-González and Gómez-Rodríguez, 2019) that, starting from the left, attaches each word of the sentence to its parent. Apart from doubling the speed of the former, the latter outperforms the top-down variant in terms of accuracy in practically all datasets tested so far.

As transition-based algorithms (Nivre, 2003), both perform the parsing process as a sequential decoding where, at each step or parsing state, all permissible actions (which do not violate the single-head or acyclicity constraints) are evaluated and the highest-scoring one is greedily applied, generating a new state. This sequential decoding differs from the approach followed by their main competitors: graph-based algorithms (McDonald et al., 2005). These previously score all possible arcs (or
sets of arcs) and then, during decoding, search for the highest-scoring valid dependency tree. While recent graph-based models use a head-selection strategy (Zhang et al., 2017; Dozat and Manning, 2017) similar to that followed by the left-to-right transition-based parser (Fernández-González and Gómez-Rodríguez, 2019), the training and decoding processes are completely different: at each decoding step, only permissible arc/transitions are evaluated in the transition-based approach, and scores are computed taking into account a sequence of previous decisions encoded through the decoder. On the contrary, these simplified graph-based models independently score all possible parents for each word and then, during decoding, only the highest-scoring ones are kept regardless of the parents chosen for the other words. Then, at the end of the process, a maximum spanning tree algorithm is applied (if necessary) to output a well-formed tree. This difference can be also seen empirically, as the left-to-right transition-based parser achieves higher accuracies than the approach by Dozat and Manning (2017) in several widely-known benchmarks.

While this sequential decoding (typically implemented by a recurrent neural network) seems to be beneficial for dependency parsing, it might lead to accuracy losses due to error propagation: mistaken past decisions will affect future actions, especially harming performance on long-range dependencies and attachments created in final steps. This limitation present in classic transition-based parsers also affects recent algorithms based on Pointer Networks, as decoder states located at the end of the sequence tend to forget relevant information from the past and are more prone to suffer from error propagation. In fact, Liu et al. (2019) propose a hierarchical decoding for the top-down algorithm (Ma et al., 2018) by having access, at each step, to information about the focus word’s parent and siblings created in the past, introducing not only knowledge about distant decoder states relevant for future decisions, but also an underlying tree structure to the decoding process (more appropriate for modelling dependency graphs).

In this paper, we initially develop a general Hierarchical Pointer Network architecture with a bottom-up structured decoding for the left-to-right transition-based algorithm (Fernández-González and Gómez-Rodríguez, 2019). This will allow the decoder to have access to information about partial structures created in the past and will help to make better future decisions. Alternatively to this algorithm, we design two novel bottom-up-oriented transition systems that can be easily implemented on the proposed neural model.

Finally, we empirically show that the presented architecture with any transition-based algorithm provides improvements in accuracy on ten different languages from Universal Dependencies (Nivre et al., 2016) and achieves state-of-the-art scores on the widely-known English and Chinese Penn Treebanks (Marcus et al., 1993; Xue et al., 2005).

The remainder of this article is organized as follows: Section 2 introduces the baseline left-to-right parser by Fernández-González and Gómez-Rodríguez (2019) and briefly presents the top-down Hierarchical Pointer Network by Liu et al. (2019). In Section 3, we describe in detail the novel bottom-up Hierarchical Pointer Network and how it is adapted to the left-to-right algorithm. Section 4 presents new transition systems and how they are implemented on the novel neural architecture. In Section 5, we extensively evaluate the proposed neural model with each parsing strategy on numerous datasets, as well as include a thorough analysis of their performance. Lastly, Section 6 contains a final discussion.

2 Preliminaries

2.1 Left-to-Right Transition System

Fernández-González and Gómez-Rodríguez (2019) propose an efficient transition system that is defined by a focus word pointer, which is used to point to the word currently being processed, and a single SHIFT-ATTACH transition, which attaches the current focus word \(w_i\) to its parent word \(w_p\) in position \(p\) (producing the dependency arc \(w_p \rightarrow w_i\)) and then moves the focus to the next word \(w_{i+1}\). This algorithm starts at the first word \((w_1)\) of a sentence of length \(n\) and sequentially (from left to right) connects each word \(w_i\) to its parent in just \(n\) steps. In Figure 2(a), we depict the sequential prediction of SHIFT-ATTACH transitions that pro-

---

1Additionally, while in transition-based parsers like (Ma et al., 2018) and (Fernández-González and Gómez-Rodríguez, 2019), the sequential decoding makes it possible to easily add high-order features and projection constraints without increasing runtime complexity, these enhancements in graph-based models lead to a performance penalty.

2Source code available at https://github.com/danig/BottomUp-Hierarchical-PtrNet.

3The top-down transition system by Ma et al. (2018) needs \(2n-1\) steps to parse a sentence of length \(n\).
duces all arcs in the dependency tree in Figure 1 following a left-to-right transition system.

In order to incrementally build a well-formed dependency tree during decoding, only \textsc{Shift-Attach-}\(p\) transitions that do not generate cycles in the already-built dependency graph are allowed. Additionally, the left-to-right transition system can efficiently produce unrestricted dependency graphs, including non-projective structures; however, the projectivity constraint can be enforced, discarding transitions that produce crossing dependencies in those treebanks with a negligible presence of non-projectivity (Fernández-González and Gómez-Rodríguez, 2020).

2.2 Pointer Networks for Left-to-Right Dependency Parsing

The left-to-right algorithm can be easily implemented by a Pointer Network (Vinyals et al., 2015) to perform non-projective dependency parsing. These sequence-to-sequence neural networks are able to learn the conditional probability of an output sequence of discrete numbers that correspond to positions from an input sequence. In our case, the output is the sequence \(p = p_1, p_2, \ldots, p_n\) of values necessary to parameterize the \textsc{Shift-Attach-}\(p\) transition and build a dependency graph for the input sentence \(w = w_1, \ldots, w_n\). These positions (or values of \(p\) in the left-to-right transition system) are selected by a mechanism of attention (Bahdanau et al., 2014) over the input sentence. This avoids having to keep a fixed size for the output dictionary, which will vary for each input sentence with respect to its length.

More specifically, the encoder-decoder architecture of the original left-to-right parser (Fernández-González and Gómez-Rodríguez, 2019) is designed as follows:

**Encoder** Given an input sentence \(w = w_1, \ldots, w_n\), each word \(w_i\) is initially represented as \(x_i\): i.e., a concatenation of character-level representations \((e_{ci}^i)\) (extracted by max-pooling-based Convolutional Neural Networks (Ma and Hovy, 2016)), POS tag embeddings \((e_{pi}^i)\) and word embeddings \((e_{wi}^i)\):

\[ x_i = e_{ci}^i \oplus e_{wi}^i \oplus e_{pi}^i \]

Then, each \(x_i\) is fed one-by-one into a BiLSTM to generate the vector representation \(h_i\):

\[ h_i = \text{BiLSTM}(x_i) \]

This results in a sequence of encoder hidden states \(h_0, \ldots, h_n\), where \(h_0\) is a special vector representation for the ROOT node.

We also report accuracies with our encoder enhanced with deep contextualized word embeddings extracted from language model BERT (Devlin et al., 2019). In those cases, BERT-based word embeddings are directly concatenated to the resulting basic word representation before feeding the BiLSTM-based encoder:

\[ x_i' = x_i \oplus e_{wi}^{BERT} ; h_i = \text{BiLSTM}(x_i') \]

**Decoder** The sequential decoding that models the transition-based behaviour is implemented by a unidirectional LSTM. At each time step \(t\), a new decoder hidden state \(s_t\) is generated, receiving as input the encoder hidden state \(h_i\) of the current focus word \(w_i\) and, as a recurrent neural network, being also conditioned by the previous decoder state \(s_{t-1}\):

\[ s_t = \text{LSTM}(h_i) \]

In the original work, authors added to \(h_i\) the previous and next encoder hidden states as extra features;
however, in a follow-up, they removed those additions since the bidirectional LSTM is already encoding such information and, therefore, they do not lead to significant improvements (Fernández-González and Gómez-Rodríguez, 2020).

The resulting decoder state $s_t$ (which represents the current focus word plus the past decisions made so far) is used for computing scores of all possible words $w_j$ from the input (represented by encoder hidden representations $h_j$ with $j \in [0, n]$ and $j \neq i$) as parent of $w_i$. These scores are obtained by a biaffine scoring function (Dozat and Manning, 2017) to finally compute the attention distribution in vector $\alpha_t$:

$$
\mathbf{v}_{tj} = \text{score}(s_t, h_j) = f_1(s_t)^T \mathbf{W} f_2(h_j) + U^T f_1(s_t) + V^T f_2(h_j) + b;
\alpha_t = \text{softmax}(v_t)
$$

where $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are the weights and $f_1(\cdot)$ and $f_2(\cdot)$ are two single-layer multilayer perceptrons (MLP) with ELU activation.

The attention vector $\alpha_t$ is then used as a pointer over the input, selecting, at each step, the highest-scoring position (where the parent word $w_p$ is located) and, therefore, providing a value $p$ necessary for applying a SHIFT-ATTACH-$p$ transition and building the arc $w_p \rightarrow w_i$. Finally, as proposed by (Ma et al., 2018), a biaffine classifier (Dozat and Manning, 2017) is separately trained to predict the arc label of each dependency created by the pointer.

This transition-based algorithm can process unrestricted non-projective sentences in $O(n^2)$ time complexity, since the decoding requires $n$ attachments to successfully parse a sentence with length $n$, and at each step the attention vector $\alpha_t$ is computed over the whole input.

### 2.3 Top-down Hierarchical Pointer Networks

Liu et al. (2019) introduced Hierarchical Pointer Networks for the top-down transition-based algorithm developed by Ma et al. (2018). The top-down transition system consists of two actions and a stack: one transition for connecting the word on top of the stack to one of its children and push it into the stack, and another action for popping the current focus word on top. Liu et al. (2019) design a structured decoding for the original Pointer Network, where not only the immediately-previous decoder state $s_{t-1}$ of the LSTM is considered for choosing the next action to be applied on the current word on top, but also decoder hidden states generated when the current focus word’s parent and last sibling were assigned following the top-down transition system. More graphically, in Figure 1, when, for instance, the current focus word to be processed is John, decoder hidden states of its parent play and last-assigned sibling together are used for choosing John’s children. With this strategy, they manage to introduce an explicit structural inductive bias into the original linear decoder, allowing a more adequate modelling for the generation of dependency graphs. Liu et al. (2019) also empirically show that their top-down Hierarchical Pointer Networks are beneficial for increasing the accuracy on arcs created in final steps, which is especially crucial on long sentences.

### 3 Bottom-up Hierarchical Pointer Networks

Based on the research work by Liu et al. (2019), we propose a Hierarchical Pointer Network for the left-to-right transition-based algorithm (Fernández-González and Gómez-Rodríguez, 2019) and other related parsing strategies where dependents of the focus word may already have been attached (as we will design more such algorithms later). In particular, instead of parent or sibling information, we design a bottom-up structured tree decoding by considering decoder hidden states of the focus word’s relevant dependents. More concretely, to alleviate the gradual loss of relevant information during the sequential decoding, we propose to consider decoder hidden states of the leftmost and rightmost dependents already attached to the current focus word, as well as the most recently attached left and right dependents. Note that not all this information is available in all strategies, e.g., the left-to-right algorithm has no access to right dependencies of the focus word.

All this will not only provide valuable knowledge about words processed in the past (especially those attached by long-range dependencies), but will also keep an underlying tree structure that will certainly help the decoder to make good decisions throughout the parsing process. For instance, when the sentence from Figure 1 is parsed following a purely bottom-up strategy and current focus word play must be attached to its parent, it would be

---

4 Assuming that in the top-down transition system, right children are attached first in an inside-out order.

5 Please note that the left-to-right transition system is not purely bottom-up since, when a word is attached to its parent, all its children might have not been assigned yet.
helpful to have access to information about its leftmost and rightmost dependents (John and together, respectively), as well as the more recently-attached right dependent (tennis).

### 3.1 Structured Tree Decoder

To implement this novel bottom-up Hierarchical Pointer Network variant, we keep the encoder as the original approach (Fernández-González and Gómez-Rodríguez, 2019), but use a bottom-up hierarchical decoder instead.

More in detail, at each step $t$ with current focus word $w_t$, the generated decoder hidden state $s_t$ is directly conditioned, apart from by the previous decoder state $s_{t-1}$ and the encoder hidden state $h_t$, by the decoder states of the already-attached leftmost ($s_{l(m)(t)}$) and rightmost ($s_{r(m)(t)}$) dependents of $w_t$, as well as the decoder states of the left ($s_{l(a)(t)}$) and right ($s_{r(a)(t)}$) last-attached dependents.\(^6\)

$$s_t = f(s_{l(m)(t)}, s_{r(m)(t)}, s_{l(a)(t)}, s_{r(a)(t)}, s_{t-1}, h_t)$$

where $f(\cdot)$ is a fusion function that combines all components into a single decoder state.

Following Liu et al. (2019), we do not directly feed these six components to the decoder, but utilize a gating mechanism to allow our model to adequately extract the most useful information at each decoding step. In particular, we implement two different gating functions:

\[
g_t = \text{sigmoid}(W_{g\rho}s_{t-1} + W_{glm}s_{l(m)(t)} + W_{grm}s_{r(m)(t)} + W_{gla}s_{l(a)(t)} + W_{gra}s_{r(a)(t)} + b_g)
\]

\[
g_t = \text{sigmoid}(W_{glm}(s_{t-1} \odot s_{l(m)(t)}) + W_{grm}(s_{t-1} \odot s_{r(m)(t)}) + W_{gla}(s_{t-1} \odot s_{l(a)(t)}) + W_{gra}(s_{t-1} \odot s_{r(a)(t)}) + b_g)
\]

where $W_{g\rho}$, $W_{glm}$, $W_{grm}$, $W_{gla}$, $W_{gra}$ and $b_g$ are gating weights. While all decoder states are equally weighted in GATE1, the element-wise product used in GATE2 has the effect of similarity comparison among the previous decoder state $s_{t-1}$ and the other components.

These gating functions are then used to define the fusion function $f$ as follows:

\[
h_t' = \tanh(W_{p}s_{t-1} + W_{lm}s_{l(m)(t)} + W_{rm}s_{r(m)(t)} + W_{la}s_{l(a)(t)} + W_{ra}s_{r(a)(t)});
\]

\[
h_t'' = g_t \odot h_t';
\]

\[
s_t = \text{LSTM}(h_t', h_t)
\]

where $W_{lm}$, $W_{rm}$, $W_{la}$ and $W_{ra}$ are the weights for combining the five different decoder states into one intermediate hidden state $h_t'$. After that, the gating mechanism $g_t$ is applied to control the information flow and generate the hidden state $h_t''$, which will be fed into the decoder together with the encoder hidden state $h_t$.

As shown in Section 2.2, the resulting decoder state $s_t$ is then used for computing the attention vector $a_t$ that will work as a pointer over the input sentence.

### 3.2 Model Specifics for the Left-to-Right Transition System

While the left-to-right transition system can be directly implemented on the described Hierarchical Pointer Network, it cannot use its full potential. This transition-based approach does not follow a fully bottom-up strategy during the left-to-right decoding\(^7\) and, therefore, right dependents are not available when they are needed. In the example, when the word play is under processing, only the leftmost dependent John is available since no right dependents were created yet and, when the words tennis and together will be attached, they are no longer needed since the word play will have already been processed. This led us to simplify the definition of the gating-based fusion function $f$ for the left-to-right transition system by removing decoder states of right dependents:

\[
s_t = f(s_{l(m)(t)}, s_{l(a)(t)}, s_{t-1}, h_t) \quad (\text{l-adapted})
\]

Since, in some languages, it might be the case that words have either only one left dependent ($s_{l(m)(t)}$ and $s_{l(a)(t)}$ being the same) or the left last-attached dependent was created in the previous time step ($s_{l-1}$ and $s_{l(a)(t)}$ being the same), we also experiment with a simpler variant of $f$ that just considers decoder states $s_{t-1}$ and $s_{l(m)(t)}$:

\[
s_t = f(s_{l(m)(t)}, s_{t-1}, h_t) \quad (\text{l-simple})
\]

\(^6\)Note that $s_{l(m)(t)}$ and $s_{r(m)(t)}$ might have the same values as $s_{l(a)(t)}$ and $s_{r(a)(t)}$, respectively, when just one left or right dependent were assigned for the current focus word.

\(^7\)Left dependents are added bottom–up and right dependents top–down.
Finally, it is worth mentioning that the hierarchical decoder does not penalize the quadratic runtime complexity of the left-to-right parser and, as the original work (Fernández-González and Gómez-Rodríguez, 2019), it is trained by minimizing the total log loss (cross entropy) for choosing the correct sequence of \textsc{Shift-Attach-}\(p\) transitions to build a gold dependency tree for the input sentence \(w\) (i.e. predicting the correct sequence of indices \(p\), with each decision (\(p_t\)) being conditioned by previous ones (\(p_{<t}\))):

\[
\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log P_{\theta}(p_t | p_{<t}, w)
\]

The labeler is simultaneously trained by optimizing the sum of their objectives.

4 Alternative Transition-based Algorithms

Apart from the existing left-to-right algorithm, other transition systems can be implemented on the proposed bottom-up Hierarchical Pointer Network. In particular, we develop two alternative approaches to sequentially parse a sentence by attaching each word to its parent.

Right-to-Left While a left-to-right hierarchical decoding might be more adequate for modelling left-branching languages where long-range dependencies tend to be leftward arcs (such as Turkish and Korean), for right-branching languages with a high predominance of long rightward arcs (such as Arabic and Hebrew), a right-to-left transition system could be a perfect fit.\(^8\) This parses a sentence starting from the last word \(w_n\) and it provides a \textsc{Shift-Attach-}\(p\) transition that, at each step, assigns a parent to the current focus word \(w_i\) and moves \(i\) to point to word \(w_{i-1}\). In Figure 2(b), we describe the order in which arcs are created with the right-to-left transition system to parse the sentence in Figure 1 and the available dependents at each decoding step. Symmetrically to the left-to-right algorithm, we adjust the fusion function \(f\) to process only decoder states of right dependents:

\[
s_t = f(s_{rm(t)}, s_{ra(t)}, s_{t-1}, h_t) \quad (\text{r-adapted})
\]

Additionally, following the same reasoning as for the left-to-right algorithm, we also implement a simplified alternative that removes \(s_{ra(t)}\) from the equation:

\[
s_t = f(s_{rm(t)}, s_{t-1}, h_t) \quad (\text{r-simple})
\]

Outside-in In order to fully use the proposed neural architecture and, therefore, have access to left and right dependents during the decoding process, we design a novel outside-in transition system: it parses a sentence starting from the leftmost word \((w_1)\) and continuing with the rightmost word \((w_n)\) towards the middle of the sentence, alternating words from the left and the right. To that end, a canonical order \((w_1, w_n, w_2, w_{n-1}, \ldots, w_{\lfloor n/2 \rfloor})\) is determined at the beginning and at each decoding step the \textsc{Shift-Attach-}\(p\) transition, apart from attaching \(w_1\) to its parent, moves \(i\) according to that order. In Figure 2(c), we can see an example of how this outside-in algorithm works and how left and right dependents are available during the decoding process. Since this strategy allows us to use decoder states of left and right dependents, the fusion function \(f\) can be fully used. For the same reasons as stated for the other transition systems, we also propose a simplification that discards last-attached dependents as follows:

\[
s_t = f(s_{lm(t)}, s_{rm(t)}, s_{t-1}, h_t) \quad (\text{simple})
\]

Note that both alternative transition systems are guaranteed to parse a sentence of length \(n\) in just \(n\) steps, keeping the same runtime complexity as the left-to-right algorithm, and are likewise trained, except for the order in which transitions (values of \(p\)) are predicted.

Figure 3 depicts the proposed Hierarchical Pointer Network architecture and the decoding procedure for parsing the sentence in Figure 1 with the novel outside-in transition system. In this sketch, it can be graphically seen how decoder hidden states of dependents influence the decoding process.

Finally, it is worth mentioning that it is not possible to design a purely bottom-up transition system on the Pointer Network framework using \(n\) transitions (one per word) as, regardless of the order

---

\(^8\)Please note that, following common usage in the parsing literature, naming conventions in this paper use the terms “left” and “right” to refer to temporal order (under the convention that a word is to the “left” of another if it comes first), not to spatial order when the words are written in a given script. This means that in languages with a right-to-left writing system, such as Arabic and Hebrew, the original left-to-right parser will begin reading each line of text from the right (i.e., the beginning), and the right-to-left algorithm will begin from the left (i.e., the end). Similarly, the graphical representation of what we call a “leftward arc” would be an arrow pointing to the right in those languages.
in which words are considered, it is not possible
to guarantee that dependents are always processed
before heads (e.g., the first node considered will
not necessarily be a leaf).

5 Experiments

5.1 Data

We conduct experiments on a wide variety of lan-
guages from Universal Dependencies v2.6 (Nivre
et al., 2016). Following (Kulmizev et al., 2019),
we choose ten treebanks from different language
families, with different morphological complexity
and with different predominances of long-range
dependencies. These are detailed in Table 1.

Additionally, we test our approach on two widely-used benchmarks: the Stanford Dependenc-
ies (de Marneffe and Manning, 2008) conver-
sion (using Stanford parser v3.3.0)\(^9\) of the
English Penn Treebank (PTB) (Marcus et al., 1993),
with standard splits and, following (Fernández-
González and Gómez-Rodríguez, 2020), without
any PoS tags;\(^10\) and the converted dependency vari-
ant (Zhang and Clark, 2008) of the Chinese Penn
Treebank 5.1 (CTB) (Xue et al., 2005) with gold
POS tags.

\(^9\)https://nlp.stanford.edu/software/lex-parser.shtml
\(^10\)It has been shown that using predicted PoS tags does not lead to accuracy gains in parsers built on Pointer Networks (Ma et al., 2018).

| Language | Treebank | Family | Order | Size | %long | %left |
|----------|----------|--------|-------|------|-------|-------|
| Arabic   | PADT     | AA     | VSO   | 6.1k | 16.62 | 10.30 |
| Basque   | BDT      | LI     | SOV   | 5.4k | 20.17 | 44.46 |
| Chinese  | GSD      | ST     | SVO   | 4.0k | 25.31 | 59.44 |
| English  | EWT      | IE     | SVO   | 12.5k| 16.43 | 25.49 |
| Finnish  | TDT      | UR     | SVO   | 12.2k| 17.45 | 26.83 |
| Hebrew   | HTB      | AA     | SOV   | 5.2k | 17.82 | 20.60 |
| Korean   | GSD      | KO     | SOV   | 4.4k | 14.76 | 78.40 |
| Swedish  | Talbanken| IE     | SVO   | 4.3k | 19.22 | 29.51 |
| Turkish  | IMST     | TU     | SOV   | 3.7k | 14.84 | 70.69 |

Table 1: Detailed treebanks used in our experiments.

Following standard practice, we just exclude
punctuation for evaluating on PTB and CTB, and,
due to random initializations, we report the average
Labelled and Unlabelled Attachment Scores (LAS
and UAS) over 3 repetitions for each experiment.

5.2 Settings

We use the code of the original parser by Fernández-
González and Gómez-Rodríguez (2019) (with the
modifications proposed for the single-task parser
in (Fernández-González and Gómez-Rodríguez,
2020)) and implement the bottom-up Hierarchical
Pointer Network plus the two novel transition sys-
tems, allowing a homogeneous comparison against the baseline. Similarly to (Liu et al., 2019), we experiment with the two available gate mechanisms in combination with the proposed transition systems and fusion function implementations.

Character and PoS tag embeddings are randomly initialized and word vectors are initialized with pre-trained structured-skipgram embeddings (Ling et al., 2015) for English and Chinese, and Polyglot embeddings (Al-Rfou’ et al., 2013) for other languages, being all of them fine-tuned during training.

For PTB and CTB, we additionally concatenate deep contextualized word embeddings from the pre-trained language model BERT (Devlin et al., 2019). We follow the greener and less resource-consuming approach undertaken by (Fernández-González and Gómez-Rodríguez, 2020) and directly feed fixed weights extracted from BERT as described in Section 2.2, without any fine-tuning to our specific task. More specifically, we extract a combination of weights from 17-20 layers of BERT\textsubscript{LARGE} for PTB and weights from the second-to-last layer of BERT\textsubscript{BASE} for CTB, averaging BERT-based embeddings when a word is tokenized into more than one subword.

Finally, we use beam size 10 for PTB and CTB, and 1 for UD, the Adam optimizer (Kingma and Ba, 2014) and follow (Ma et al., 2018; Dozat and Manning, 2017) for parameter optimization and hyper-parameter selection. These are detailed in Table 2.

| Architecture hyper-parameters |
|--------------------------------|
| CNN window size                | 3 |
| CNN number of filters          | 50 |
| BiLSTM encoder layers          | 3 |
| BiLSTM encoder size            | 512 |
| LSTM decoder layers            | 1 |
| LSTM decoder size              | 512 |
| LSTM layers dropout            | 0.33 |
| Word/POS/Char. embedding dimension | 100 |
| English BERT embedding dimension | 1024 |
| Chinese BERT embedding dimension | 768 |
| Embeddings dropout             | 0.33 |
| MLP layers                     | 1 |
| MLP activation function        | ELU |
| Arc MLP size                   | 512 |
| Label MLP size                 | 128 |
| UNK replacement probability    | 0.5 |
| Beam size                      | 10 |

| Adam optimizer hyper-parameters |
|--------------------------------|
| Initial learning rate          | 0.001 |
| $\beta_1$, $\beta_2$          | 0.9 |
| Batch size                     | 32 |
| Decay rate                     | 0.75 |
| Gradient clipping              | 5.0 |

Table 2: Model hyper-parameters.

5.3 Results

In Table 3, we report LAS on dev splits of UD treebanks obtained by the proposed bottom-upHierarchical Pointer Networks with different transition systems and parser configurations (gate and fusion function) and choose the best configuration on average to be evaluated on test splits. As shown in the reported results, the proposed architecture with the three algorithms improves over the baseline parser in practically all datasets, obtaining higher gains on those languages with a larger amount of long arcs (such as Basque and Chinese). We can also observe that the simplified fusion functions improve over the adapted and full versions on average, meaning that long-distance dependents are more valuable than last-assigned ones and, in some cases, the usage of the latter harms parsing accuracy. Regarding the novel transition systems, the right-to-left parser outperforms other algorithms on languages with a significant predominance of long rightward arcs (such as Arabic), and the outside-in algorithm clearly underperforms the other transition systems on average in spite of having access to left and right relevant dependents, which probably means that a sequential human-like strategy is more suitable for parsing natural languages, or that the outside-in order is too complex to learn effectively. Finally, GATE\textsuperscript{1} obtains higher accuracy in general.

We also compare our novel approach with the three implemented transition systems against other state-of-the-art dependency parsers on PTB and CTB. In order to choose the best parser configuration for the Penn Treebanks, we use scores on PTB development splits reported in Table 4. Since GATE\textsuperscript{1} outperforms GATE\textsuperscript{2} in general (as shown for UD treebanks in Table 3), we simply use GATE\textsuperscript{1} in all experiments and vary the fusion function implementation. Lastly, for BERT augmentations, we directly use simplified fusion functions, significantly reducing training time. As we can see in Table 5, the left-to-right approach achieves the highest accuracy obtained so far on PTB and CTB test splits with neither contextualized word embeddings nor extra constituent information. Regarding BERT augmentations, all transition systems not only outperform the baseline (Fernández-González and Gómez-Rodríguez, 2020), but also the right-to-left model achieves the best LAS to date among approaches that use (or fine-tune) BERT (even improving over those enhanced with constituent information on PTB). These results provide some
While, at a first glance, we might think that the To further understand the availability of dependent
branching languages, where rightward dependency
information for each transition system and its im-
measure the impact of error propagation in transition systems
errors relative to dependency length and word position. This might explain the good
results of the right-to-left approach on UD datasets on average (since the majority of tested languages have a higher percentage of long rightward arcs and long-range dependents are considered more valuable for reducing error propagation) and the fact that the l-adapted fusion function has a better performance on the left-to-right parser (since this function gathers information about closer dependents and this algorithm has access to a notable amount of dependent information). Finally, we can also observe that, as expected, the right-to-left algorithm is more adequate to model languages with a high predominance of rightward arcs such as Hebrew and especially Arabic, where it has access to a remarkable amount of right dependents not seen by the other transition systems throughout the parsing process.

| tran. | f   | gate | ar   | en   | eu   | fi   | he   | it   | ko   | sv   | tr   | zh   | Avg. |
|-------|-----|------|------|------|------|------|------|------|------|------|------|------|------|
| L2R   | -   | -    | 83.82| 90.29| 83.80| 88.46| 89.11| 92.52| 83.29| 86.78| 64.75| 82.54| 84.54|
| L2R   | l-adapted | GATE1 | 84.15| 90.51| 84.67| **88.97**| 89.27| **92.63**| 83.81| 87.41| 65.35| 83.62| **85.04**|
|       | l-adapted | GATE2 | 84.05| 90.44| 84.53| 88.95| 89.22| 92.60| 83.79| 87.32| 65.24| 83.48| 84.96|
|       | l-simple  | GATE1 | 84.06| 90.49| 84.84| 89.90| 89.30| 92.59| 83.84| 87.44| **65.40**| 83.65| **85.05**|
|       | l-simple  | GATE2 | 84.05| 90.29| 84.77| 88.89| 89.20| 92.62| **83.86**| 87.43| 65.12| 83.57| 84.98|
| R2L   | r-adapted | GATE1 | 84.21| 90.41| 84.83| 89.95| 89.21| 92.47| 83.66| 87.28| 65.25| 83.50| 84.98|
|       | r-adapted | GATE2 | 84.20| 90.26| **84.93**| 88.81| 89.21| 92.52| 83.68| 87.14| 65.09| 83.53| 84.94|
|       | r-simple  | GATE1 | 84.20| 90.52| 84.75| 88.86| 89.28| 92.56| 83.71| 87.41| 65.33| **83.69**| **85.04**|
|       | r-simple  | GATE2 | 84.26| 90.46| 84.88| 88.85| 89.20| 92.40| 83.69| 87.26| 65.36| 83.52| 84.99|
| O-I   | full    | GATE1 | 84.07| 90.43| 84.59| 88.74| 89.28| 92.51| 83.46| 86.91| 65.14| 83.54| 84.87|
|       | full    | GATE2 | 84.00| 90.33| 84.42| 88.80| 89.09| 92.38| 83.45| 86.96| 65.07| 83.47| 84.80|
|       | simple  | GATE1 | 84.13| 90.48| 84.54| 88.77| **89.36**| 92.55| 83.56| 87.06| 65.12| 83.49| **84.91**|
|       | simple  | GATE2 | 84.04| 90.35| 84.53| 88.77| 89.10| 92.33| 83.53| 87.07| 65.14| 83.38| 84.81|

Table 3: LAS comparison of the original left-to-right parser with sequential decoding against the available three transition systems on Hierarchical Pointer Networks combined with different fusion functions implementations and gating mechanisms on ten treebanks from UD. Only best models on average on dev splits are evaluated on test sets. In Appendix A.1, we report the standard deviations over 3 runs on test splits. We use ISO 639-1 codes to represent languages.

| tran. | f   | gate | UAS  | LAS  |
|-------|-----|------|------|------|
| L2R   | -   | -    | 96.03| 94.15|
|       | l-adapted | GATE1 | 96.01| **94.15**|
|       | l-simple  | GATE1 | **96.04**| **94.16**|
| R2L   | r-adapted | GATE1 | 96.00| 94.14|
|       | r-simple  | GATE1 | **96.00**| **94.14**|
| O-I   | full    | GATE1 | 95.96| 94.10|
|       | simple  | GATE1 | **96.00**| **94.14**|

Table 4: Accuracy performance on PTB dev splits. We mark in bold the chosen configurations.

5.4 Dependent Information Availability

To further understand the availability of dependent information for each transition system and its impact on parsing performance, we report in Table 6 the number of dependents per sentence that are accessible for each transition system on UD datasets. While, at a first glance, we might think that the outside-in algorithm is the strategy that leverages dependent information the most since it receives both left and right dependents, it seems that, on overall and according to gold trees, the left-to-right parser is the strategy that has access to a larger amount of (in this case, left) dependents and, the right-to-left approach, the one that uses more long-range (right) dependents (typically present in right-branching languages, where rightward dependency arcs tend to be longer). This might explain the good results of the right-to-left approach on UD datasets on average (since the majority of tested languages have a higher percentage of long rightward arcs and long-range dependents are considered more valuable for reducing error propagation) and the fact that the l-adapted fusion function has a better performance on the left-to-right parser (since this function gathers information about closer dependents and this algorithm has access to a notable amount of dependent information). Finally, we can also observe that, as expected, the right-to-left algorithm is more adequate to model languages with a high predominance of rightward arcs such as Hebrew and especially Arabic, where it has access to a remarkable amount of right dependents not seen by the other transition systems throughout the parsing process.

5.5 Error Analysis

We study the mitigation of error propagation by the proposed neural architecture and characterize errors relative to sentence length and word position.

---

11 Errors relative to dependency length (as described in McDonald and Nivre, 2007) are no longer an effective method to measure the impact of error propagation in transition systems designed for Pointer Networks (except for the top-down algorithm (Ma et al., 2018)), since these require a single SHIFT-ATTACH-β transition to directly build any arc (regardless of its length), not involving several decoder states and, therefore, not directly reflecting the presence of error propagation.
Figure 4: Parsing performance of the original left-to-right parser and the proposed variants implemented on Hierarchical Pointer Networks relative to sentence length and word position.

In particular, we show in Figures 4(a) and (b) the accuracy relative to sentence lengths of the three novel transition systems plus the baseline parser on UD and PTB, respectively. On UD, we see accuracy gains regardless of the sentence length; however, while the improvements tend to be higher as sentences are longer, these narrow when the length is larger than 40 (especially affecting the left-to-right and outside-in variants). On PTB, the novel approaches not only outperform the baseline on long sentences (as expected), but also surprisingly on the shortest ones. We also note that, while the right-to-left parser obtains the highest performance on the longest sentences (better dealing with error-propagation), the outside-in algorithm is suffering a drop on sentences with a length greater than 40 on UD and, with length between 31 and 40, on PTB (possibly because the information about left and right dependents is exclusively used by this transition system and it tends to be available at final steps – as shown in Figure 2(c) –, where it might be difficult to manage when sentences are substantially long and the amount of information is probably significantly large).

The reduction of error propagation can be seen more clearly in Figures 4(c) and (d), where we report the LAS relative to word positions within the sentence. In comparison to the sequential variant, the left-to-right transition system with structured decoding obtains the highest gains in accuracy on attachments made on words at final positions of the sentence (the most affected by error propagation). We can also observe how the right-to-left and outside-in approaches significantly outperform the left-to-right algorithm on words at the end of the sentence, since these transition systems attach those words in initial steps of the parsing process.

---

In order to prevent data sparsity, we randomly select the number of sentences that approximately gather 10,000 tokens per treebank.

---

12In order to prevent data sparsity, we randomly select the number of sentences that approximately gather 10,000 tokens per treebank.
Lastly, there is a significant drop in accuracy by the outside-in model in words at middle positions on PTB (from 31 to 40). This is probably due to the fact that arcs in those positions are created in final steps following the outside-in strategy, being the most affected by error-propagation.

6 Conclusions

We manage to introduce structural knowledge to the sequential decoding of the left-to-right dependency parser with Pointer Networks. We addition-}

| Parser                        | PTB UAS  | LAS  | UAS  | LAS  |
|-------------------------------|----------|------|------|------|
| Zhang et al. (2017)           | 94.10    | 91.90| 87.84| 86.15|
| Ma and Hovy (2017)            | 94.88    | 92.96| 89.05| 87.74|
| Dozat and Manning (2017)      | 95.74    | 94.08| 89.30| 88.23|
| Li et al. (2018)              | 94.11    | 92.08| 88.78| 86.23|
| Ma et al. (2018)              | 95.87    | 94.19| 90.59| 89.29|
| Ji et al. (2019)              | 95.97    | 94.31| -    | -    |
| Wang and Tu (2020)            | 95.98    | 94.34| 90.81| 89.57|
| Hier. Ptr. Net. L2R           | 96.18    | 94.59| 90.76| 89.67|
| Hier. Ptr. Net. R2L           | 96.14    | 94.53| 90.72| 89.62|
| Hier. Ptr. Net. O-I          | 96.07    | 94.48| 90.64| 89.50|
| +BERT                         |          |      |      |      |
| Li et al. (2020)              | 96.44    | 94.63| 90.89| 89.73|
| Moham. & Hend. (2020)*        | 96.66    | 95.01| 92.86| 91.11|
| Wang and Tu (2020)*           | 96.91    | 95.34| 92.55| 91.38|
| Fdez-G & Gómez-R (2020)       | 96.91    | 95.35| 92.58| 91.42|
| Hier. Ptr. Net. L2R           | 97.05    | 95.47| 92.70| 91.50|
| Hier. Ptr. Net. R2L           | 97.01    | 95.48| 92.75| 91.62|
| Hier. Ptr. Net. O-I          | 96.95    | 95.36| 92.65| 91.47|
| Zhou and Zhao (2019)          | 96.09    | 94.68| -    | -    |
| Fdez-G & Gómez-R (2020)       | 96.25    | 94.64| 90.79| 89.69|
| Zhou and Zhao (2019)*         | 97.00    | 95.43| 91.21| 89.15|
| Fdez-G & Gómez-R (2020)       | 96.97    | 95.46| 92.78| 90.65|
| Liu et al. (2019)             | 96.09    | 95.03| -    | -    |

Table 5: Accuracy comparison of state-of-the-art dependency parsers on PTB and CTB. Second block gathers approaches that are enhanced with constituent information and, the last block, includes the performance of the top-down transition-based model with Hierarchical Pointer Networks, since only scores with gold PoS tags are reported. Models that fine-tune BERT are marked with *. Those parsers marked with † report scores based on the best single run on the development set, instead of reporting the average score on the test set over several runs, i.e., instead of averaging to mitigate the effect of random seeds in reported accuracy, they use model selection to choose the most promising seed using the dev set (following this method, our best model Hier.Ptr.Net L2R w/o BERT obtains UAS 96.19 LAS 94.61 on PTB).

Table 6: Number of all and long-range dependents per sentence that are available when processing gold trees in UD dev splits with each proposed transition system. Note that the number of long-range dependents (with arc lengths > 4) per sentence is notably low since short sentences are also considered for the computation.

 Acknowledgments

We acknowledge the European Research Council (ERC), which has funded this research under the European Union’s Horizon 2020 research and innovation programme (FASPARSE, grant agreement No 714150), MINECO (ANSWER-ASAP, TIN2017-85160-C2-1-R), Xunta de Galicia (ED431C 2020/11), and Centro de Investigación de Galicia “CITIC”, funded by Xunta de Galicia and the European Union (ERDF - Galicia 2014-2020 Program), by grant ED431G 2019/01.

References

Rami Al-Rfou’, Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual NLP. In Proceedings of the Seventeenth Conference on Computational Natural Language Learning, pages 183–192, Sofia, Bulgaria. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473.

Jiangang Bai, Yujing Wang, Yiren Chen, Yaming Yang, Jing Bai, Jing Yu, and Yunhai Tong. 2021. Syntax-BERT: Improving pre-trained transformers with syntax trees. In Proceedings of the 16th Conference of
the European Chapter of the Association for Computational Linguistics: Main Volume, pages 3011–3020, Online. Association for Computational Linguistics.

Vidhisha Balachandran, Artidoro Pagnoni, Jay Yoon Lee, Dheeraj Rajagopal, Jaime Carbonell, and Yulia Tsvetkov. 2021. StructSum: Summarization via structured representations. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 2575–2585, Online. Association for Computational Linguistics.

Qingxing Cao, Xiaodan Liang, Bailin Li, and Liang Lin. 2021. Interpretable visual question answering by reasoning on dependency trees. IEEE transactions on pattern analysis and machine intelligence, 43(3):887—901.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In ICRL.OpenReview.net.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Left-to-right dependency parsing with pointer networks. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 710–716, Minneapolis, Minnesota. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Multitask pointer network for multi-representational parsing.

M. Haspelmath, M.S. Dryer, Oxford University, H.J. Bibiko, Oxford University Press, Max-Planck-Institut für Evolutionäre Anthropologie, J. Hagen, D. Gil, B. Comrie, and C. Schmidt. 2005. The World Atlas of Language Structures. Oxford linguistics. OUP Oxford.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 2475–2485, Florence, Italy. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. 2019. Deep contextualized word embeddings in transition-based and graph-based dependency parsing - a tale of two parsers revisited. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 2755–2768. Hong Kong, China. Association for Computational Linguistics.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In Proceedings of the 27th International Conference on Computational Linguistics, pages 3203–3214, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Zuchao Li, Hai Zhao, and Kevin Parnow. 2020. Global greedy dependency parsing. In the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-2020).

Wang Ling, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015. Two/too simple adaptations of Word2Vec for syntax problems. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics, pages 1299–1304, Denver, Colorado. Association for Computational Linguistics.

Linlin Liu, Xiang Lin, Shafiq Joty, Simeng Han, and Lidong Bing. 2019. Hierarchical pointer net parsing. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 1007–1017, Hong Kong, China. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1064–1074. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective mst parsing. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 59–69. Asian Federation of Natural Language Processing.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics, 19:313–330.
Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation, pages 1–8, Manchester, UK. Coling 2008 Organizing Committee.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05), pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 122–131, Prague, Czech Republic. Association for Computational Linguistics.

Ali Reza Mohammadshahi and James Henderson. 2020. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement.

Dat Quoc Nguyen and Karin Verspoor. 2019. From pos tagging to dependency parsing for biomedical event extraction. BMC Bioinformatics, 20(1):72.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In Proceedings of the Eighth International Conference on Parsing Technologies, pages 149–160, Nancy, France.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. 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 Tenth International Conference on Language Resources and Evaluation (LREC 2016), pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

Devendra Sachan, Yuhao Zhang, Peng Qi, and William L. Hamilton. 2021. Do syntax trees help pre-trained transformers extract information? In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 2647–2661, Online. Association for Computational Linguistics.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 2692–2700. Curran Associates, Inc.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training.

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. Nat. Lang. Eng., 11(2):207–238.

Baosong Yang, Derek F. Wong, Lidia S. Chao, and Min Zhang. 2020. Improving tree-based neural machine translation with dynamic lexicalized dependency encoding. Knowledge-Based Systems, 188:105042.

Bo Zhang, Yue Zhang, Rui Wang, Zhenghua Li, and Min Zhang. 2020a. Syntax-aware opinion role labeling with dependency graph convolutional networks. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 3249–3258, Online. Association for Computational Linguistics.

Meishan Zhang, Zhenghua Li, Guohong Fu, and Min Zhang. 2019. Syntax-enhanced neural machine translation with syntax-aware word representations. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 1151–1161, Minneapolis, Minnesota. Association for Computational Linguistics.

Meishan Zhang, Zhenghua Li, Guohong Fu, and Min Zhang. 2021. Dependency-based syntax-aware word representations. Artificial Intelligence, 292:103427.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers, pages 665–676.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020b. Efficient second-order TreeCRF for neural dependency parsing. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 3295–3305, Online. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, pages 562–571, Honolulu, Hawaii. Association for Computational Linguistics.

Junru Zhou and Hai Zhao. 2019. Head-driven phrase structure grammar parsing on Penn treebank. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.
## Appendices

### A.1 Standard deviations

| Parser                  | PTB | CTB |
|-------------------------|-----|-----|
|                         | UAS | LAS | UAS | LAS |
| Hier. Ptr. Net. L2R     | ±0.02| ±0.03| ±0.05| ±0.06|
| Hier. Ptr. Net. R2L     | ±0.04| ±0.03| ±0.06| ±0.08|
| Hier. Ptr. Net. O-I     | ±0.03| ±0.05| ±0.04| ±0.06|
| +BERT                    |     |     |     |     |
| Hier. Ptr. Net. L2R     | ±0.01| ±0.02| ±0.04| ±0.05|
| Hier. Ptr. Net. R2L     | ±0.03| ±0.02| ±0.06| ±0.05|
| Hier.Ptr. Net. O-I      | ±0.02| ±0.02| ±0.03| ±0.04|

Table 7: Standard deviations over 3 runs on test splits for scores reported in Table 5.
| tran. | f   | gate   | ar ±   | en ±   | eu ±   | fi ±   | he ±   | it ±   | ko ±   | sv ±   | tr ±   | zh ±   |
|-------|-----|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| L2R   | -   | -      | ±0.06  | ±0.01  | ±0.05  | ±0.06  | ±0.14  | ±0.04  | ±0.06  | ±0.03  | ±0.13  | ±0.04  |
| L2R   | l-simp. GATE1 | ±0.02  | ±0.04  | ±0.06  | ±0.02  | ±0.08  | ±0.02  | ±0.04  | ±0.04  | ±0.08  | ±0.04  |
| R2L   | r-simp. GATE1 | ±0.06  | ±0.04  | ±0.02  | ±0.06  | ±0.12  | ±0.02  | ±0.06  | ±0.05  | ±0.11  | ±0.08  |
| O-I   | simple GATE1 | ±0.02  | ±0.08  | ±0.05  | ±0.04  | ±0.06  | ±0.07  | ±0.09  | ±0.02  | ±0.12  | ±0.08  |

Table 8: Standard deviations over 3 runs on test splits for scores reported in Table 3.