Graph-Based Universal Dependency Parsing in the Age of the Transformer: What Works, and What Doesn't

Stefan Grünewald[★♦] **Annemarie Friedrich**[♦] **Jonas Kuhn**[★]
[★]Institut für Maschinelle Sprachverarbeitung, University of Stuttgart
[♦]Bosch Center for Artificial Intelligence, Renningen, Germany
stefan.gruenewald|annemarie.friedrich@de.bosch.com
jonas.kuhn@ims.uni-stuttgart.de

Abstract

Current state-of-the-art graph-based dependency parsers differ on various dimensions. Among others, these include (a) the choice of pre-trained word embeddings or language models used for representing tokens, (b) training setups performing only parsing or additional tasks such as part-of-speech-tagging, and (c) their mechanism of constructing trees or graphs from edge scores. Because of this, it is difficult to estimate the impact of these architectural decisions when comparing parsers.

In this paper, we perform a series of experiments on STEPS, a new modular graph-based parser for basic and enhanced Universal Dependencies, analyzing the effects of architectural configurations. We find that pre-trained embeddings have by far the greatest and most clear-cut impact on parser performance. The choice of factorized vs. unfactorized architectures and a multi-task training setup affect parsing accuracy in more subtle ways, depending on target language and output representation (trees vs. graphs). Our parser achieves new state-of-the-art results for a wide range of languages on both basic as well as enhanced Universal Dependencies, using a unified and comparatively simple architecture for both parsing tasks.[1]

1 Introduction

Recent years have seen considerable improvements in the performance of syntactic dependency parsers. For graph-based parsers, these improvements can in large part be attributed to two developments: the introduction of deep biaffine classifiers (Dozat and Manning, 2017), which now constitute the de-facto standard approach for graph-based dependency parsing, and the rise of pre-trained distributed word representations, particularly transformer-based contextualized embeddings such as BERT (Devlin et al., 2019) or RoBERTa (Liu et al., 2019). As shown in Table 1, both features are present in the best-performing systems (Che et al., 2018; Kanerva et al., 2018, 2020) of the CoNLL-18 and IWPT-20 Shared Tasks on Universal Dependency parsing (Zeman et al., 2018; Bouma et al., 2020).

Universal Dependencies (UD, de Marneffe et al., 2014) is a community-driven, cross-linguistic framework for syntactic dependency annotation. In addition to classical dependency trees ("basic UD"), which are annotated in all available treebanks, more and more treebanks are adding non-tree dependency graph annotations ("enhanced UD," Schuster and Manning, 2016). Enhanced UD graphs include a number of additional dependencies in order to make relations between content words more explicit and have been designed with shallow natural language understanding tasks in mind. The parsing of these graph structures can be seen as similar to other bi-lexical semantic parsing tasks (Oepen et al., 2016) such as Prague Semantic Dependencies (Hajič et al., 2012) or Enju Predicate–Argument Structures (Miyao and Tsujii, 2004).

Despite the ubiquity of biaffine classifiers and pre-trained contextualized word embeddings in recent neural dependency parsers (Che et al., 2018; Straka et al., 2019; Kondratyuk and Straka, 2019), there remain a considerable number of implementation and configuration choices whose impact on parser performance is less well understood; a fact which is evidenced by the many different model configurations present in parsers that have achieved top results in the CoNLL-17 (Zeman et al., 2017), CoNLL-18 (Zeman et al., 2018), and IWPT-20 (Bouma et al., 2020) Shared Tasks on Universal Dependency Parsing (see Table 1). In particular, these choices include (a) the concrete pre-trained word

[1]We will release our code and pre-trained models at github.com/boschresearch/steps-parser.

embeddings or language model to use, particularly when working with many different languages; (b) whether to predict edge existence and edge labels using two separate modules ("factorized architecture") or a single module ("unfactorized architecture"); and (c) whether to use a multi-task training setup to simultaneously predict other UD properties (such as morphological features or POS tags) during parsing.

In this paper, we disentangle the effects of the above factors in order to determine their impact on parser performance. We do so by performing a series of experiments on the Universal Dependencies treebanks of a diverse set of languages using STEPS (the **S**tuttgart **T**ransformer-based **E**xtensible **P**arsing **S**ystem), a modular graph-based dependency parser implementation based on PyTorch (Paszke et al., 2019). Our setup facilitates estimating the impact of the various architecture and configuration decision in a comparable way.

We make the following contributions:

1. We introduce STEPS, a new graph-based parser for Universal Dependencies inspired by UDify (Kondratyuk and Straka, 2019) and designed to be modular and easily extensible. To the best of our knowledge, STEPS constitutes the first dependency parser that has been designed from the ground up to produce both basic as well as enhanced UD using a unified architecture. It achieves new state-of-the-art results on a large majority of evaluated treebanks, sometimes by very wide margins.

2. We compare the performance of multilingual BERT (Devlin et al., 2019), language-specific BERTs, and (multilingual) XLM-R (Conneau et al., 2019), finding that XLM-R yields the best results on the vast majority of treebanks.

3. We compare the performance of a factorized and an unfactorized architecture (Dozat and Manning, 2018) for both tree parsing ("basic UD") and graph parsing ("enhanced UD"). We find that the factorized approach consistently outperforms the unfactorized approach for tree parsing, whereas both approaches overall perform roughly equally well for graph parsing. However, training factorized models is faster in almost all cases.

4. We evaluate parsing performance in a multi-task setup in which part-of-speech tags and morphological features are predicted simultaneously with syntactic dependencies. We find that this setting can result in small improvements in parsing performance for some languages, but that only at the cost of small decreases in accuracy of the tagging tasks.

5. We will make our code and pre-trained models for 12 languages available at `github.com/boschresearch/steps-parser`.

Parser	Pre-trained embeddings	Multi-task
Basic UD		
StanfordNLP	word2vec, fastText	no
UDPipe 2.0	word2vec, fastText	yes
HIT-SCIR	fastText, ELMo	no
UDify	mBERT	yes
Enhanced UD		
TurkuNLP	langBERTs	yes
ShanghaiTech	XLM-R[a]	no

Table 1: Settings for a number of previously state-of-the-art graph-based dependency parsers. [a]Additionally for Tamil: fastText + Flair.

The rest of this paper is structured as follows. Sec. 2 gives the necessary background on relevant state-of-the-art neural graph-based dependency parsers as well as related work on analysing and comparing parsers. Sec. 3 describes the architecture and configuration options for our new STEPS parser, at the same time introducing the various factors that are the parameters studied in our experiments (Sec. 4). Finally, we discuss implications for parser choice and future parser design (Sec. 5).

2 Related Work

This section provides an outline of the architectural details of recently developed graph-based dependency parsers, in each case referring to the versions used to generate the main submissions for the respective Shared Tasks. In addition, we briefly review recent related work on dependency parser analysis.

Table 1 shows the configurations of six previously state-of-the-art parsers, most of which were among the best-performing systems in the CoNLL 2017 and CoNLL 2018 Shared Tasks on basic UD parsing, or in the IWPT 2020 Shared Task on enhanced UD parsing. The column labeled "pre-trained embeddings" lists the pre-trained word embeddings employed by the respective parser in

order to represent input tokens,[2] and "Multi-task" states whether the parser is also trained to predict other UD properties such as POS tags or morphological features.

2.1 Tree Parsers for Basic UD

This subsection describes tree parsers for basic UD, i.e., parsers that produce "classical" syntactic dependency trees.

StanfordNLP (Dozat et al., 2017) was one of the first systems to apply the biaffine graph-based parser architecture to Universal Dependency parsing. It scored 1st in the CoNLL 2017 Shared Task by a wide margin.

UDPipe 2.0 (Straka, 2018) uses a multi-task setup in which POS and feature tagging, lemmatization, and dependency parsing share layers. It provides a "loosely joint" model in which only input embeddings are shared, and a "tightly joint" model in which BiLSTM layers are shared. This approach allowed the system to score 1st in terms of MLAS (Morphology-Aware Labeled Attachment Score) in the CoNLL 2018 Shared Task.

HIT-SCIR (Che et al., 2018) was one of the first UD parsers to make use of contextualized pre-trained word embeddings (in the form of ELMo; Peters et al., 2018), allowing it to score 1st in the CoNLL 2018 Shared Task in terms of LAS.

UDify (Kondratyuk and Straka, 2019) differs from previous UD parsers in a number of ways. First, it does not use an LSTM to create vector representations of tokens, instead using a learned scalar mixture of mBERT layers and fine-tuning mBERT during training. Different scalar mixture coefficients are learned for POS tagging, feature prediction, and lemmatization, which are trained jointly with dependency parsing in a multi-task setting. Second, UDify utilizes multilingual training: Only a single model is learned for all languages using a concatenation of all UD 2.5 training sets. This combination of factors enabled UDify to achieve new state-of-the-art results for dependency parsing on a large number of UD 2.5 treebanks.

2.2 Graph Parsers for Enhanced UD

This subsection describes parsers that produce enhanced UD graphs.

TurkuNLP (Kanerva et al., 2020) is based on a version of UDify modified to produce enhanced UD graphs instead of basic trees and scored 1st in the official evaluation of the IWPT 2020 Shared Task. Instead of fine-tuning a single multilingual BERT model on all UD treebanks, it uses carefully selected language-specific BERT models for the different treebanks in the Shared Task.

ShanghaiTech (Wang et al., 2020) was the first UD parser to make use of XLM-R embeddings, which are used as inputs to a BiLSTM whose outputs are then fed to a second-order inference algorithm for dependency arc predictions. It scored 1st in the unofficial post-evaluation of the IWPT 2020 Shared Task.

2.3 Parser Analyses and Comparisons

Prior to the advent of end-to-end neural parsing systems, dependency parsers commonly used structural features (e.g., morpho-syntactic constraints; see Seeker and Kuhn (2013)) in order to increase parsing accuracy. However, recent studies show that such explicit features are rendered obsolete by modern neural sequence processing methods. For example, Falenska and Kuhn (2019) demonstrate that when using BiLSTMs, structural features extracted from partial subtrees do not help parsing accuracy. Conversely, using structural probes, a number of recent studies have demonstrated that token representations derived from contextualized word embeddings such as ELMo and BERT already contain a large amount of implicit syntactic information (Tenney et al., 2019a,b; Hewitt and Manning, 2019).

As a result, recent work has studied in detail the effect of different transformer-based token embeddings on parsing accuracy. Straka et al. (2019) compare a baseline of UDPipe 2.0 against a modified version of the same parser (henceforth referred to as **UDPipe+**) that incorporates BERT and Flair embeddings (Akbik et al., 2018) where available. They find that using these contextualized embeddings improves parsing accuracy considerably, with BERT yielding the greatest improvements.

Comparing multilingual BERT and language-specific BERT models trained on Wikipedia data ("WikiBERT"), Pyysalo et al. (2020) find that the latter outperform the former on average, but with large inter-language variation. Similar results are reported by Kanerva et al. (2020), who generally find language-specific BERT models to outperform mBERT on the development data provided by the IWPT 2020 Shared Task.

[2] Most of these parsers do not only use pre-trained embeddings, but additionally also train their own embeddings and sometimes use CharLSTMs for token representation.

3 STEPS: A Modular Graph-Based Dependency Parser

In this section, we describe our modular dependency parser STEPS (**S**tuttgart **T**ransformer-based **E**xtensible **P**arsing **S**ystem). Each subsection focuses on a particular aspect of the parser setup, providing background on its usage, its potential impact on parser performance, and our implementation.

3.1 Input Token Representation

STEPS provides a number of different options for input token representation. As Table 1 shows, parsers have made use of a variety of pre-trained embeddings, with transformer-based language models having become the predominant current approach. We hence focus on the latter and compare multilingual BERT (**mBERT**; Devlin et al., 2019), language-specific BERTs (**langBERT**), and the multilingual **XLM-R** model (Conneau et al., 2019). Table 2 provides a detailed overview over all transformer-based models used in our experiments.

As proposed by Kondratyuk and Straka (2019), we compute token embeddings as weighted sums of the representations of the respective tokens given by the internal transformer encoder layers. Coefficients for this weighted sum are learned during training, and layer dropout is applied in order to prevent the model from focusing on particular layers. In case our model has more than one output (see Sections 3.3, 3.5), different coefficients are learned for each output task.

3.2 Biaffine Classifier

STEPS makes use of biaffine classifiers as proposed by Dozat and Manning (2017), which have become the de-facto standard method for graph-based dependency parsing. In a first step, a head representation $\mathbf{h}_i^{head}$ and a dependent representation $\mathbf{h}_i^{dep}$ are created for each input token i (represented as embedding vector $\mathbf{r}_i$, the result of the weighted sum explained in Sec. 3.1) via two single-layer feedforward neural networks:

$$\mathbf{h}_i^{head} = \text{FNN}^{head}(\mathbf{r}_i) \quad (1)$$

$$\mathbf{h}_i^{dep} = \text{FNN}^{dep}(\mathbf{r}_i) \quad (2)$$

These representations are then fed into the biaffine function, which maps head–dependent pairs (i, j) onto vectors $\mathbf{s}_{i,j}$ of arbitrary size:

$$\mathbf{s}_{i,j} = \text{Biaff}\big(\mathbf{h}_i^{head}, \mathbf{h}_j^{dep}\big) \quad (3)$$

$$\text{Biaff}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^\top \mathbf{U}\mathbf{x}_2 + W(\mathbf{x}_1 \oplus \mathbf{x}_2) + \mathbf{b} \quad (4)$$

$\mathbf{U}$, W and $\mathbf{b}$ are learned parameters; $\oplus$ denotes the concatenation operation. The scores $s_{i,j}$ can now be leveraged in different ways to construct an output tree or graph; this will be described next.

3.3 Factorized and Unfactorized Models for Constructing Trees and Graphs

STEPS is able to output both tree and graph structures using a unified architecture. It is hence applicable to both basic and enhanced UD and may in theory even be trained to output both layers at once.

Both trees and graphs can be constructed either using a **factorized model** or an **unfactorized model**. Factorized models make use of two instantiations of a biaffine classifier as described in Sec. 3.2 in order to produce labeled dependency structures. The first classifier (the "arc scorer") is responsible for predicting which (unlabeled) edges exist in the output structure, and the second classifier (the "label scorer") then assigns dependency labels to edges predicted in the first step. By contrast, unfactorized models only employ a label scorer. They predict the existence of edges in parallel with their label by encoding non-existence of a dependency as simply another label ($\varnothing$).

Tree parsing. To extract dependency trees using a factorized model, we follow the approach proposed by Dozat and Manning (2017) and use the arc scorer to predict, for each token, a probability distribution over potential syntactic heads (i.e., all other tokens in the sentence). We then feed the log-probabilities to the Chu-Liu/Edmonds maximum spanning tree algorithm (Chu and Liu, 1965; Edmonds, 1967) and label the resulting tree using the label scorer.

In the unfactorized case, we proceed similarly, but compute edge weights for the spanning tree algorithm as $\log(1 - P(\varnothing))$ for each pair of tokens. To the best of our knowledge, our study is the first to apply an unfactorized architecture to dependency tree parsing.

Graph parsing. To extract dependency graphs using a factorized model, we follow the approach proposed by Dozat and Manning (2018). In this case, the arc scorer performs, for each ordered pair

Language	BERT model	Citation	Model size
Arabic	ArabicBERT-large	Safaya et al. (2020)	335M
Chinese	Chinese BERT	Devlin et al. (2019)	110M
Czech	Slavic-BERT	Arkhipov et al. (2019)	110M
English	RoBERTa-large	Liu et al. (2019)	355M
Finnish	FinBERT	Virtanen et al. (2019)	110M
German	German BERT (dbmdz)	`github.com/dbmdz/berts`	110M
Hindi	WikiBERT-Hindi	Pyysalo et al. (2020)	110M
Italian	Italian BERT (dbmdz)	`github.com/dbmdz/berts`	110M
Japanese	WikiBERT-Japanese	Pyysalo et al. (2020)	110M
Korean	KR-BERT	Lee et al. (2020)	110M
Latvian	WikiBERT-Latvian	Pyysalo et al. (2020)	110M
Russian	RuBERT	Kuratov and Arkhipov (2019)	110M
Multilingual	mBERT	Devlin et al. (2019)	110M
Multilingual	XLM-R	Conneau et al. (2019)	550M

Table 2: Transformer-based language models used in our experiments.

of tokens, a binary classification of whether an edge exists between them or not. Predicted edges are then labeled using the label scorer.

Similar to Dozat and Manning (2018), we adjust the weighting of the loss for each output module. We do so by adding two hyperparameters λ_{edge}, λ_{label} so that the overall loss ℓ of the system is computed as

$$\ell = \lambda_{edge}\ell_{edge} + \lambda_{label}\ell_{label}. \quad (5)$$

In the unfactorized case, for each pair of tokens, we simply add the edge with the highest-scoring dependency label predicted by the label scorer, or no edge if $\varnothing$ received the highest score.

Enhanced UD imposes the constraint that graphs must be connected and every node must be reachable from the root. Hence, we extend our graph parsers (both factorized and unfactorized) with a post-processing step that greedily adds the highest-scoring edge from a node that is reachable from the root to a node that is unreachable from the root until the condition is fulfilled.

3.4 Label Lexicalization

For certain relations such as *nmod* or *obl*, enhanced UD allows for the inclusion of lexical material (such as prepositions) in dependency labels. To avoid data sparsity issues resulting from the increase in the number of dependency labels, we follow the label (de)lexicalization strategy proposed by Grünewald and Friedrich (2020):

1. At training time, we replace lexical material in labels with placeholders such as *obl:[case]*.

2. At prediction time, we use a set of rules to re-lexicalize placeholder labels.

   (a) The basic rule is to check whether the token has a dependent that is attached via the placeholder of the de-lexicalized relation in question. If so, we lexicalize the relation with the word form of this dependent. For example, in a phrase like *"at school"*, we lexicalise *obl:[case]* to *obl:at* because *at* is attached to *school* via the *case* relation.

   (b) If the basic rule fails to find lexical material, we utilize to a set of more complex rules which are designed to find lexical material in the presence of conjunctions. For example, in a phrase like *"at work or school"*, we also lexicalize the incoming *obl:[case]* relation to *work* as *obl:at*, even though *at* is not a dependent of *work*. For more details on these rules, see Grünewald and Friedrich (2020).

   (c) In case the complex rules also fail, we simply delete the placeholder without substituting any lexical material.

3.5 Multi-Task Training

We study the effects of a multi-task training setup by implementing two different approaches to training our parser: (a) **dep-only**, in which the model is trained only on syntactic dependencies; and (b) multi-task learning (**MTL**), in which the model is additionally trained to predict universal part-of-speech tags (UPOS) and morphological features (UFeats). We follow Kondratyuk and Straka (2019) by learning specific coefficients for the transformer layers for these tagging tasks (see Sec. 3.1) and then using a single-layer feed-forward neural network to extract logit vectors over the respective

label vocabularies. The loss for the entire system is computed as the sum of losses for the individual output modules (UPOS tagger, UFeats tagger, and dependency parser). As for factorized parsing, we add the option of scaling the loss for these additional output modules.

4 Experiments

This section describes our experimental setup and reports the results of our experiments on (a) pre-trained embeddings; (b) parser architecture (factorized vs. unfactorized); and (c) multi-task training.

4.1 Experimental Setup

We evaluate our different parser variants on basic as well as enhanced UD parsing.

Languages and treebanks. For basic UD, we select 12 languages, covering a diverse range of language families and writing systems, by applying linguistic criteria similar to those outlined by de Lhoneux et al. (2017). For each language, we then select the largest available treebank from UD 2.6 for which token data is freely available. For enhanced UD, we pick 7 treebanks from the IWPT 2020 Shared Task, again trying to cover as diverse a set of languages as possible. However, the selection is necessarily narrower since the set of languages in the Shared Task is much smaller than the entire set of languages for which UD treebanks exist.

In all of our experiments, we always train a single model per treebank and test on the respective test set of the same treebank.

Evaluation metrics. The primary evaluation metrics for basic UD are UAS (Unlabeled Attachment Score, i.e., the fraction of tokens assigned the correct syntactic head) and LAS (Labeled Attachment Score, i.e., the fraction of tokens assigned the correct syntactic head and the correct dependency label). We compute UAS and LAS using the official evaluation script for the CoNLL 2018 Shared Task.[3]

The primary evaluation metrics for enhanced UD are ELAS (Enhanced Labeled Attachment Score, i.e., F1 score over the set of enhanced dependencies in the system output and the gold standard) and EULAS (Enhanced Universal Labeled Attachment Score, i.e., ELAS but ignoring label subtypes including lexical material). We compute ELAS and EULAS using the official evaluation script for the IWPT 2020 Shared Task.[4]

Gold vs. raw text input. For basic UD, we use gold tokens as input for our systems, as this is the setting used in most recent relevant related work (Straka et al., 2019; Kondratyuk and Straka, 2019). For enhanced UD, we use raw text as input, as was the setting in the IWPT 2020 Shared Task (Bouma et al., 2020). In this case, we tokenize and segment the text prior to parsing using the Stanza toolkit (Qi et al., 2020).

4.2 Parser Implementation

Our parser is implemented in Python, using Pytorch (Paszke et al., 2019) and the Huggingface Transformers library (Wolf et al., 2019). We will publish our code and pointers to pre-trained models at https://github.com/boschresearch/steps-parser.

Hyperparameters. We aim to obtain a simple yet high-performing hyperparameter configuration. We do so by using the configuration of UDify (the existing parser that is architecturally most similar to STEPS) as a starting point and tuning parameters on a small development set (consisting of English, Arabic, and Korean), aiming at a simplified setup that achieves good results across these diverse languages. Table 3 shows our final hyperparameter settings. In large part, our settings correspond to UDify's values with the following exceptions: We use AdamW instead of Adam; we perform neither label smoothing nor gradient clipping; we do not use differential learning rates. In addition, we do not train for a fixed number of epochs, but instead stop once performance on the validation set does not increase for 15 epochs, or after at most 24 hours. Training is performed on a single nVidia Tesla V100 GPU.

4.3 Impact of Pre-Trained Word Embeddings

We evaluate how parsing performance differs between parsers using mBERT, language-specific BERT models (langBERT), or XLM-R as underlying pre-trained language models. For these experiments, we use a factorized parser architecture and only perform dependency parsing.

Basic dependencies. Table 4 shows results for basic UD parsing on the 12 languages evaluated.

[3] https://universaldependencies.org/conll18/conll18_ud_eval.py

[4] https://universaldependencies.org/iwpt20/iwpt20_xud_eval.py

Transformer LM	
Token mask probability	0.15
Layer dropout	0.1
Hidden dropout	0.2
Attention dropout	0.2
Output dropout	0.5
Biaffine classifier	
Arc scorer dimension	768 or 1024[a]
Label scorer dimension	256 or 768/1024[b]
Dropout	0.33
Optimization	
Optimizer	AdamW
β_1, β_2	0.9, 0.999
Weight decay	0
Batch size	32
Base learning rate	$4e^{-5}$
LR schedule	Noam
LR warmup	1 epoch
λ_{edge}	1.00
λ_{label}	1.00 or 0.05[c]

Table 3: Hyperparameter values. [a]Identical to hidden size of the transformer encoder. [b]256 in factorized models, hidden size of transformer encoder in unfactorized models. [c]1.00 for tree parsing, 0.05 for graph parsing.

"UDPipe+" refers to the version of UDPipe enhanced with BERT and Flair embeddings proposed by Straka et al. (2019) (see Sec. 2.3), and results for UDify are given for the model trained on all languages, without treebank-specific fine-tuning.

Parsing accuracy of STEPS is very high across the board, with new state-of-the-art results being achieved on all languages except Japanese, often by considerable margins (>1.0 LAS F1). For most languages, the best results are achieved using $\text{STEPS}_{\text{XLM-R}}$, with $\text{STEPS}_{\text{langBERT}}$ coming in second for most languages. In contrast, using mBERT is not the best option on any treebank. In fact, the only language for which mBERT achieves better results than langBERT in our experiments is Latvian.[5]

While using langBERT usually yields worse parsing accuracy than XLM-R, results are roughly on par for Arabic and English. We note that the language-specific models we chose for these treebanks (ArabicBERT-large and RoBERTa-large, respectively) are the only ones with a number of trainable parameters similar to XLM-R, while all others have a considerably smaller number of parameters (see Table 2). This highlights the importance of model size in pre-trained word embeddings.

We notice a slight overall trend for $\text{STEPS}_{\text{mBERT}}$ to achieve higher parsing accuracy than UDify, which is somewhat surprising. Not only do both models use mBERT and are architecturally very similar, but Kondratyuk and Straka (2019) also found UDify to generally perform worse when only fine-tuning on a single treebank versus training on all concatenated treebanks.[6] As we train only on single languages, this should work to UDify's favor in this comparison. Possible reasons for the discrepancy are that we perform early stopping while UDify is trained for a fixed number of epochs, or that we found a better hyperparameter configuration. Nonetheless, future work includes reproducing UDify's multilingual training approach in STEPS in order to be able to perform a more rigorous comparison between the two systems.

Finally, we note that on Finnish and Latvian, $\text{STEPS}_{\text{XLM-R}}$ outperforms the previously best results by extremely high margins (nearly 5.0 and 7.0 LAS F1, respectively). Future work will include a detailed study of the results for these morphologically rich languages.

Enhanced dependencies. Table 5 shows results for enhanced UD parsing. Per-treebank results for TurkuNLP and ShanghaiTech are as reported in the IWPT 2020 Shared Task.[7]

We observe similar trends compared to the basic case, with $\text{STEPS}_{\text{XLM-R}}$ achieving the best results, langBERT-based parsers coming in second, and $\text{STEPS}_{\text{mBERT}}$ usually performing worst. Although ShanghaiTech also uses XLM-R to represent input tokens, we find $\text{STEPS}_{\text{XLM-R}}$ to achieve better results on all treebanks except Arabic and Czech, suggesting that its overall architecture (not using an LSTM and fine-tuning XLM-R during training) may be better suited to dependency parsing. However, further experiments are needed to confirm this.

Finally, we note that even though originally developed for English, the simple dependency label lexicalization strategy proposed by Grünewald and Friedrich (2020) also works well for Czech, Italian, and Latvian. (Finnish and Russian do not have lexicalized labels.) In contrast, it performs poorly on Arabic (as evidenced by the large gap between EULAS and ELAS scores). This is because lexical material in dependency labels is supposed to be

[5]In a similar study comparing mBERT- and langBERT-based parsers, Kanerva et al. (2020) also found Latvian to be one of the few languages for which mBERT outperformed the language-specific (WikiBERT) version.

[6]Though note that this effect was much more pronounced on low-resource languages.

[7]https://universaldependencies.org/iwpt20/Results.html

	ar-PADT		cs-PDT		de-GSD		en-EWT		fi-TDT		hi-HDTB	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
UDPipe	87.54	82.94	93.33	91.31	85.53	81.07	89.63	86.97	89.88	87.46	94.85	91.83
UDPipe+	89.01	84.62	94.43	92.56	88.11	84.06	92.50	90.40	91.66	89.49	95.56	92.50
UDify[a]	87.72	82.88	94.73	92.88	87.81	83.59	90.96	88.50	86.42	82.03	95.13	91.46
$\text{STEPS}_{\text{mBERT}}$	88.72	83.98	94.42	92.72	88.48	84.19	91.69	89.21	91.15	88.70	94.92	91.45
$\text{STEPS}_{\text{langBERT}}$	**90.90**	86.42	94.68	93.03	89.90	86.03	93.94	**92.01**	95.06	93.59	94.71	91.20
$\text{STEPS}_{\text{XLM-R}}$	90.88	**86.47**	**95.89**	**94.58**	**90.11**	**86.10**	**93.95**	91.94	**95.72**	**94.42**	**96.06**	**93.29**

	it-ISDT		ja-GSD		ko-Kaist		lv-LVTB		ru-SynTagRus		zh-GSD	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
UDPipe	93.49	91.54	95.06	93.73	88.42	86.48	87.20	83.35	93.80	92.32	84.64	80.50
UDPipe+	94.97	93.38	**95.55**	**94.27**	89.35	87.54	88.05	84.50	94.92	93.68	90.13	86.74
UDify[a]	95.54	93.69	94.37	92.08	87.57	84.52	89.33	85.09	94.83	93.13	87.93	83.75
$\text{STEPS}_{\text{mBERT}}$	94.66	92.94	94.11	92.26	55.47	24.51	89.14	85.29	95.09	93.76	88.08	84.99
$\text{STEPS}_{\text{langBERT}}$	95.82	94.38	94.94	93.09	86.92	84.66	87.10	82.94	95.68	94.50	89.26	86.32
$\text{STEPS}_{\text{XLM-R}}$	**96.26**	**94.83**	95.47	93.98	**91.79**	**89.99**	**94.45**	**92.07**	**96.36**	**95.31**	**90.75**	**87.76**

Table 4: Parsing results for different pre-trained embeddings (basic dependency parsing, gold tokenzation). [a]Results for multilingual training without language-specific fine-tuning.

	ar-PADT		cs-PDT		en-EWT		fi-TDT		it-ISDT		lv-LVTB		ru-SynTagRus	
	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS
TurkuNLP	79.68	**77.83**	89.51	88.17	86.71	86.14	89.86	89.24	92.29	91.54	85.51	84.94	91.57	90.69
ShanghaiTech	79.91	77.73	**91.90**	**90.63**	86.95	86.30	90.93	89.97	92.17	91.49	88.19	87.64	92.73	92.31
$\text{STEPS}_{\text{mBERT}}$	76.15	65.79	89.02	86.01	84.74	84.33	85.31	84.63	91.01	90.03	82.10	81.01	92.18	91.75
$\text{STEPS}_{\text{langBERT}}$	**80.01**	68.63	89.47	86.57	**87.54**	**87.17**	90.62	90.28	92.18	91.33	79.72	78.66	93.14	92.73
$\text{STEPS}_{\text{XLM-R}}$	**80.01**	68.66	91.42	88.65	87.31	86.91	**91.89**	**91.52**	**93.16**	**92.37**	**89.63**	**88.58**	**94.01**	**93.80**

Table 5: Parsing results for different pre-trained embeddings (enhanced dependency parsing, raw text).

lemmatized, but we do not lemmatize tokens during parsing. While not a significant problem for the other languages (where lexical material is rarely inflected), this decreases the accuracy of lexicalized labels drastically for Arabic.

4.4 Impact of Factorized vs. Unfactorized Biaffine Architecture

In a second set of experiments, we analyze how performance changes when using a factorized vs. an unfactorized biaffine architecture for dependency parsing. For these experiments, we use XLM-R as input embeddings and only perform dependency parsing.

For basic dependencies (see Table 6), we make the following two main observations: (a) In terms of LAS, factorized models outperform unfactorized models on all treebanks except German. (b) In terms of UAS, the difference is smaller and unfactorized models even perform better for a small number of languages (Arabic, German, Hindi, and Korean).

For enhanced dependencies (see Table 7), we observe that neither unfactorized nor factorized is clearly superior to the other, but performance is highly dependent on language. Furthermore, these differences in performance are relatively small.

For both UD representations, we also observe that although running a single epoch is faster for the unfactorized models (ca. 4.5 vs 5.5 minutes), it takes considerably longer to train them overall; i.e., unfactorized models take longer to reach the stopping criterion (no improvement on the validation set for 15 epochs). This effect is present on all treebanks except Japanese, where training the unfactorized model is 20 % faster than training the factorized model. On average, it takes ca. 75 % more epochs to train an unfactorized model, and ca. 40 % more runtime.

Overall, our results suggest that the tasks of edge prediction and edge labeling are indeed different enough to result in (generally) lower parsing accuracy and higher training time when trying to perform them with a single output module. This particularly applies for basic UD, where the factorized model seems to be able to much better exploit the single-head property by predicting actual

	ar-PADT		cs-PDT		de-GSD		en-EWT		fi-TDT		hi-HDTB	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Unfact	**90.96**	86.29	95.79	94.37	**90.19**	**86.24**	93.78	91.69	95.55	94.13	**96.20**	93.18
Fact	90.88	**86.47**	**95.89**	**94.58**	90.11	86.10	**93.95**	**91.94**	**95.72**	**94.42**	96.06	**93.29**

	it-ISDT		ja-GSD		ko-Kaist		lv-LVTB		ru-SynTagRus		zh-GSD	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Unfact	96.18	94.66	95.42	93.47	**91.81**	89.89	94.21	91.67	96.24	95.09	90.51	87.54
Fact	**96.26**	**94.83**	**95.47**	**93.98**	91.79	**89.99**	**94.45**	**92.07**	**96.36**	**95.31**	**90.75**	**87.76**

Table 6: Results for factorized vs. unfactorized parsing (basic, gold tokenization).

	ar-PADT		cs-PDT		en-EWT		fi-TDT		it-ISDT		lv-LVTB		ru-SynTagRus	
	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS	EULAS	ELAS
Unfact	80.00	68.52	91.24	88.43	**87.65**	**87.26**	**91.97**	**91.61**	**93.26**	**92.47**	89.35	88.27	**94.06**	93.68
Fact	**80.01**	**68.66**	**91.42**	**88.65**	87.31	86.91	91.89	91.52	93.16	92.37	**89.63**	**88.58**	94.01	**93.80**

Table 7: Results for factorized vs. unfactorized parsing (enhanced, raw text).

probability distributions over syntactic heads. Conversely, in enhanced UD (where this single-head constraint is not present), unfactorized models do indeed perform better on some languages, indicating that they may learn *semantic* relations between tokens in a more natural way. In this regard, our results resemble those of Dozat and Manning (2018), who found factorized and unfactorized models to perform equally well for a number of semantic dependency formalisms. Nonetheless, training takes considerably longer for the unfactorized models. Under practical considerations, this means that factorized models may be considered the preferable approach (or at least a better starting point) in most circumstances, but unfactorized models constitute a valid alternative that should be considered at least for enhanced UD parsing.

4.5 Impact of Multi-Task Approach

Finally, we analyze how performance changes when predicting POS tags and morphological features in addition to syntactic dependencies. For these experiments, we use XLM-R as input embeddings and a factorized architecture.

Table 8 shows the results of our experiments for basic dependencies on 6 languages. $\text{STEPS}_{\text{MTL}}$ achieves very high accuracies for UPOS and UFeats, outperforming previous systems on all evaluated languages except Chinese. However, we find that compared to the dependency-only system, parsing accuracy drops considerably in the multi-task setting (up to almost 1 LAS F1 for Finnish).

During training, we noticed that accuracy on the validation set increased very rapidly for the tagging tasks and reached levels close to the final values after only a few epochs, while accuracy for the parsing task increased much slower. This suggests that the loss for the tagging tasks might overwhelm the system as a whole, causing the parser modules to underfit in a manner similar to the one described by Dozat and Manning (2018) for factorized graph parsing, where they found the label scorer to converge much more quickly than the arc scorer. We therefore implement another version of our multi-task system, $\text{STEPS}_{\text{MTLscale}}$, in which the loss for UPOS and UFeats is scaled down to 5% during training (as for the label scorer in factorized graph parsing). In this setting, we achieve much better parsing results than with the "naive" MTL approach, to the point that we get close to the performance of the original (dependency-only) system or even outperform it. In turn, however, accuracy for UPOS and especially UFeats drops considerably.

We conclude that the effect of multi-task training – at least in our naive implementation – is more accurately described as a tradeoff rather than a synergy. When weighing the losses of the subtasks equally, it achieves good tagging results at the cost of lower parsing accuracy; when re-scaling the losses, it achieves good parsing results at the cost of lower tagging accuracy. However, we also note that for some languages, including UPOS and UFeats prediction in the training while performing loss scaling can push parsing accuracy past a dependency-only setup.

TREEBANK	MODEL	UPOS	UFEATS	UAS	LAS
Arabic PADT (ar_padt)	UDPipe	96.83	94.11	87.54	82.94
	UDPipe+	96.98	94.72	89.01	84.62
	UDify	96.58	91.77	87.72	82.88
	$\text{STEPS}_{\text{dep-only}}$	–	–	90.88	86.47
	$\text{STEPS}_{\text{MTL}}$	**97.26**	**94.94**	90.39	86.04
	$\text{STEPS}_{\text{MTLscale}}$	96.46	87.85	**91.01**	**86.57**
Czech PDT (cs_pdt)	UDPipe	99.18	97.23	93.33	91.31
	UDPipe+	99.34	97.67	94.43	92.56
	UDify	99.18	96.69	94.73	92.88
	$\text{STEPS}_{\text{dep-only}}$	–	–	95.89	94.58
	$\text{STEPS}_{\text{MTL}}$	**99.40**	**98.23**	95.72	94.36
	$\text{STEPS}_{\text{MTLscale}}$	99.09	95.78	**95.94**	**94.62**
English EWT (en_ewt)	UDPipe	96.29	97.10	89.63	86.97
	UDPipe+	97.59	97.82	92.50	90.40
	UDify	96.21	96.02	90.96	88.50
	$\text{STEPS}_{\text{dep-only}}$	–	–	**93.95**	**91.94**
	$\text{STEPS}_{\text{MTL}}$	**97.93**	**98.03**	93.50	91.61
	$\text{STEPS}_{\text{MTLscale}}$	96.91	96.91	93.86	91.88
Finnish TDT (fi_tdt)	UDPipe	97.45	95.43	89.88	87.46
	UDPipe+	97.57	95.80	91.66	89.49
	UDify	94.43	90.48	86.42	82.03
	$\text{STEPS}_{\text{dep-only}}$	–	–	**95.72**	**94.42**
	$\text{STEPS}_{\text{MTL}}$	**98.54**	**96.94**	94.88	93.43
	$\text{STEPS}_{\text{MTLscale}}$	98.19	87.33	95.67	94.35
Hindi HDTB (hi_hdtb)	UDPipe	97.52	94.15	94.85	91.83
	UDPipe+	97.58	94.24	95.56	92.50
	UDify	97.12	92.59	95.13	91.46
	$\text{STEPS}_{\text{dep-only}}$	–	–	96.06	93.29
	$\text{STEPS}_{\text{MTL}}$	**98.09**	**94.55**	96.01	93.14
	$\text{STEPS}_{\text{MTLscale}}$	97.62	89.58	**96.27**	**93.59**
Chinese GSD (ja_gsd)	UDPipe	94.88	99.22	84.64	80.50
	UDPipe+	97.07	**99.58**	90.13	86.74
	UDify	95.35	99.35	87.93	83.75
	$\text{STEPS}_{\text{dep-only}}$	–	–	**90.75**	**87.76**
	$\text{STEPS}_{\text{MTL}}$	**97.19**	99.53	90.06	87.15
	$\text{STEPS}_{\text{MTLscale}}$	95.56	98.82	90.54	87.69

Table 8: Results for dependency-only vs. multi-task parsing (basic dependencies, gold tokenization).

Furthermore, our experiments indicate that when comparing parser performance, it is crucial to take potential multi-task setups into account. If the respective setups differ, ignoring them may result in misleading interpretations of performance of model architectures. Metrics such as MLAS, which combine parsing accuracy with UPOS and UFeats accuracy, may help in conducting fair comparisons between models with different setups.

4.6 Summary

Our experimental findings can be summarized as follows: (a) Choice of pre-trained embeddings has the greatest impact on parser performance, with XLM-R yielding the best results in most cases; (b) unfactorized models achieve worse results than factorized models except on some enhanced UD treebanks, and also take longer to train; (c) when using a multi-task approach incorporating UPOS and UFeats prediction, there is a tradeoff between tagging and parsing accuracy, but the latter can be boosted slightly for some treebanks when utilizing loss scaling.

5 Discussion and Conclusion

In this paper, we have introduced STEPS, a new modular dependency parser which allows to easily perform experiments using a variety of settings and principled comparisons. Using STEPS, we have conducted a series of experiments in order analyze how different implementation choices affect the performance of graph-based dependency parsing.

Our experimental results indicate that high-quality pre-trained token embeddings are by far the most crucial factor towards achieving high parsing accuracy. In our case, the best results were achieved by using XLM-R, or, on some treebanks, language-specific BERT models.

Furthermore, we found that an unfactorized parser architecture is competitive with a factorized architecture on enhanced UD, but achieves worse results on basic UD and takes longer to train for both representations. This suggests that in most cases, factorized models may be preferable to unfactorized models when parsing UD.

Finally, we have evaluated the impact of a multi-task setup in which the parsing system is also trained to predict part-of-speech tags and morphological features. We have found that doing so can increase parsing accuracy modestly for some languages, but that this requires scaling down the loss of the tagging tasks, reducing their accuracy in turn. To achieve ideal results across the parsing and tagging tasks, it may thus be preferable to use a pipeline model that handles them separately, or to use more sophisticated modes of interactions between the tasks, such as stack propagation (Zhang and Weiss, 2016).

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. 2019. Tuning multilingual transformers for language-specific named entity recognition. In *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 89–93, Florence, Italy. Association for Computational Linguistics.

Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2020. Overview of the IWPT 2020 shared task on parsing into enhanced Universal Dependencies. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 151–161, Online. Association for Computational Linguistics.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, 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*, pages 55–64, Brussels, Belgium. 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.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*.

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 *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

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, Vancouver, Canada. Association for Computational Linguistics.

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

Agnieszka Falenska and Jonas Kuhn. 2019. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics.

Stefan Grünewald and Annemarie Friedrich. 2020. RobertNLP at the IWPT 2020 shared task: Surprisingly simple enhanced UD parsing for English. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 245–252, Online. Association for Computational Linguistics.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing Prague Czech-English Dependency Treebank 2.0. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3153–3160, Istanbul, Turkey. European Language Resources Association (ELRA).

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in 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 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Jenna Kanerva, Filip Ginter, Niko Miekka, Akseli Leino, and Tapio Salakoski. 2018. Turku neural parser pipeline: An end-to-end system for the CoNLL 2018 shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 133–142, Brussels, Belgium. Association for Computational Linguistics.

Jenna Kanerva, Filip Ginter, and Sampo Pyysalo. 2020. Turku enhanced parser pipeline: From raw text to enhanced graphs in the IWPT 2020 shared task. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 162–173, Online. Association for Computational Linguistics.

Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing universal dependencies universally. 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 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Yuri Kuratov and Mikhail Arkhipov. 2019. Adaptation of deep bidirectional multilingual transformers for russian language. *arXiv preprint arXiv:1905.07213*.

Sangah Lee, Hansol Jang, Yunmee Baik, Suzi Park, and Hyopil Shin. 2020. Kr-bert: A small-scale korean-specific language model. *arXiv preprint arXiv:2008.03979*.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *The 15th Treebanks and Linguistic Theories Workshop (TLT)*, pages 99–110.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland. European Language Resources Association (ELRA).

Yusuke Miyao and Jun'ichi Tsujii. 2004. Deep linguistic analysis for the accurate identification of predicate-argument relations. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 1392–1398, Geneva, Switzerland. COLING.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Zdeňka Urešová. 2016.

Towards comparability of linguistic graph Banks for semantic parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3991–3995, Portorož, Slovenia. European Language Resources Association (ELRA).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Sampo Pyysalo, Jenna Kanerva, Antti Virtanen, and Filip Ginter. 2020. Wikibert models: deep transfer learning for many languages. *arXiv preprint arXiv:2006.01538*.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Ali Safaya, Moutasem Abdullatif, and Deniz Yuret. 2020. Kuisail at semeval-2020 task 12: Bert-cnn for offensive speech identification in social media.

Sebastian Schuster and Christopher D. Manning. 2016. Enhanced English Universal Dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 2371–2378, Portorož, Slovenia. European Language Resources Association (ELRA).

Wolfgang Seeker and Jonas Kuhn. 2013. Morphological and syntactic case in statistical dependency parsing. *Computational Linguistics*, 39(1):23–55.

Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium. Association for Computational Linguistics.

Milan Straka, Jana Straková, and Jan Hajič. 2019. Evaluating contextualized embeddings on 54 languages in pos tagging, lemmatization and dependency parsing. *arXiv preprint arXiv:1908.07448*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019a. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. 2019b. What do you learn from context? probing for sentence structure in contextualized word representations. *arXiv preprint arXiv:1905.06316*.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: Bert for finnish. *arXiv preprint arXiv:1912.07076*.

Xinyu Wang, Yong Jiang, and Kewei Tu. 2020. Enhanced Universal Dependency parsing with second-order inference and mixture of training data. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 215–220, Online. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

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*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, 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 de Paiva, Kira Droganova, Héctor Martínez Alonso,

Çağrı 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 Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566, Berlin, Germany. Association for Computational Linguistics.