

Parsing as Pretraining

David Vilares,¹ Michalina Strzyz,¹ Anders Søgaard,^{2,3} Carlos Gómez-Rodríguez¹

¹ Universidade da Coruña, CITIC, Ciencias de la Computación y Tecnologías de la Información (CC&TI), A Coruña, Spain

² University of Copenhagen, Department of Computer Science, Copenhagen, Denmark

³ Google Research, Berlin, Germany

david.vilares@udc.es, michalina.strzyz@udc.es, soegaard@di.ku.dk, carlos.gomez@udc.es

Abstract

Recent analyses suggest that encoders pretrained for language modeling capture certain morpho-syntactic structure. However, probing frameworks for word vectors still do not report results on standard setups such as constituent and dependency parsing. This paper addresses this problem and does full parsing (on English) relying *only* on pretraining architectures – and *no* decoding. We first cast constituent and dependency parsing as sequence tagging. We then use a single feed-forward layer to directly map word vectors to labels that encode a linearized tree. This is used to: (i) see how far we can reach on syntax modelling with just pretrained encoders, and (ii) shed some light about the syntax-sensitivity of different word vectors (by freezing the weights of the pretraining network during training). For evaluation, we use bracketing F1-score and LAS, and analyze in-depth differences across representations for span lengths and dependency displacements. The overall results surpass existing sequence tagging parsers on the PTB (93.5%) and end-to-end EN-EWT UD (78.8%).

Introduction

Traditionally, natural language processing (NLP) models represented input sentences using one-hot vectors, together with weighting schemes such as TF-IDF. Such vectors can encode shallow linguistic information, like term frequency or local context if n-grams are allowed. However, they cannot capture complex linguistic structure due to the inability to consider non-local context, their orderless nature, and the curse of dimensionality.

This paradigm has however become obsolete for many NLP tasks in favour of continuous vector representations, also known as word embeddings. The idea is to encode words as low-dimensional vectors ($\vec{v} \in \mathbb{R}^n$), under the premise that words with similar context should have similar vectors – which seemingly better capture semantic and morpho-syntactic information. In practice, these architectures have been widely adopted because they have not only made it possible to obtain more accurate models but also conceptually simpler ones, reducing the number of features required to obtain state-of-the-art results.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

In parsing, two research paths have arisen with respect to word vectors: (i) whether and to what extent pretraining architectures can offer help creating parsers which avoid the need for dozens of hand-crafted structural features, *ad-hoc* parsing algorithms, or task-specific decoders; and (ii) how to explain what sort of syntactic phenomena are encoded in such pretraining encoders. In related work, to test (i) it is common to rely on ablation studies to estimate the impact of removing features, or to further contextualize word embeddings with powerful, task-specific neural networks to show that richer linguistic contextualization translates into better performance. But to the best of our knowledge, there is no work that has tried to do (full) parsing relying *uniquely* on word embeddings, i.e. no features beyond words, no parsing algorithms, and no task-specific decoders. To test (ii), the most common probing framework consists in using models with limited expression (e.g. feed-forward networks on top of the word vectors) to solve tasks that can give us insights about the linguistic information that word vectors can encode (Tenney et al. 2018). However, these recent studies do not provide results on full parsing setups, but instead on simplified versions, which sometimes are even limited to analyzing capabilities on specific syntactic phenomena.

Contribution In this paper, we try to give an answer to these questions using a unified framework. Our approach consists in casting both (full) constituent and dependency parsing as pretraining from language modelling.¹ To do so, we first reduce constituent and dependency parsing to sequence labeling. Then, under this paradigm we can directly map, through a single feed-forward layer, a sequence of word embeddings of length n into an output sequence (also of length n) that encodes a linearized tree. The novelty of the paper is twofold: (i) we explore to what extent it is possible to do parsing relying *only* on pretrained encoders, (ii) we shed light on the syntactic abilities of existing encoders.

Related work

We now review previous work on the two research directions that we will be exploring in our paper.

¹In this paper, we limit our analysis to English.

Word vectors for simpler parsing

Syntactic parsers traditionally represented input sentences as one-hot vectors of discrete features. Beyond words, these included part-of-speech (PoS) tags, morphological features and dozens of hand-crafted features. For instance, in transition-based dependency parsers it was common to refer to daughter or grand-daughter features of a given term in the stack or the buffer, in order to provide these models with more contextual information (Zhang and Nivre 2011). We will be referring to these features as *structural features*.

Chen and Manning (2014) were one of the first to use word vectors to train a transition-based dependency parser using a feed-forward network. They showed that their model performed comparably to previous parsers, while requiring fewer structural features. Later, Kiperwasser and Goldberg (2016) demonstrated that replacing Chen and Manning’s feed-forward network with bidirectional long short-term memory networks (BiLSTMs) (Hochreiter and Schmidhuber 1997) led to more accurate parsers that at the same time required even fewer features. Following this trend, Shi, Huang, and Lee (2017) proposed a minimal feature set, by taking advantage of BiLSTMs and dynamic programming algorithms. Furthermore, this redundancy of structural features in neural parsers was empirically demonstrated by Falenska and Kuhn (2019). However, in addition to small sets of hand-crafted features, these approaches relied on *ad-hoc* parsing algorithms. In this vein, recent research has showed that task-specific sequence-to-sequence and sequence labeling decoders suffice to perform competitively even without small sets of structural features nor parsing algorithms (Li et al. 2018; Strzyz, Vilares, and Gómez-Rodríguez 2019).

For constituent parsing, the tendency has run parallel. Transition-based systems (Zhu et al. 2013) used templates of hand-crafted features suiting the task at hand. More recent research has shown that when using word embeddings and neural networks such templates can be simplified (Dyer et al. 2016) or even ignored (Kitaev and Klein 2018a; 2018b). Finally, it has been proved that transition-based or chart-based parsers are not required to do constituent parsing, and that task-specific neural decoders for sequence-to-sequence and sequence labeling suffice (Vinyals et al. 2015; Gómez-Rodríguez and Vilares 2018).

Probing syntax-sensitivity of word vectors

These improvements in parsing have raised the question of whether and to what extent pretraining architectures capture syntax-sensitive phenomena. For example, Mikolov et al. (2013) and Pennington, Socher, and Manning (2014) already discussed the syntax-sensitivity of `word2vec` and `GloVe` vectors, evaluating them on syntactic word analogy (e.g. ‘seat is to seating as breath is to x ’). However this only provides indirect and shallow information about what syntactic information such vectors accommodate.

State-of-the-art pretrained encoders such as `ELMo` (Peters et al. 2018) or `BERT` (Devlin et al. 2018), trained with a language modeling objective and self-supervision, seemingly encode some syntactic properties too. Goldberg (2019) discusses this precisely using `BERT`, and reports results

on subject-verb agreement tasks (e.g. ‘The *dog* of my uncle *eats*’ vs ‘The *dog* of my uncles *eat*’). He used similar methods to the ones employed to assess how recurrent neural networks (RNN) capture structure. For example, Linzen, Dupoux, and Goldberg (2016) studied how LSTMs perform on subject-verb agreement, and trained the network using different objectives to know whether it predicted the grammatical number of the next word. Gulordava et al. (2018) additionally incorporated the concept of ‘colorless green ideas’ (Chomsky 1957), i.e. they replaced content words with random terms with the same morphological information, to force the model to attend to syntactic patterns and not words.

Another common strategy consists in analyzing the capabilities of word vectors using models with limited expression, i.e. simple models such as n -layer feed-forward networks that take word vectors as input and are used to solve structured prediction tasks in NLP. This is the angle taken by Tenney et al. (2018), who evaluated contextualized embeddings on problems such as part-of-speech tagging or named-entity labeling. They also included *simplified and partial* versions of constituent and dependency parsing. They ask the model to predict the type of phrase of a span or the dependency type between two specific words, i.e., not to predict the full syntactic structure for a sentence. In a similar fashion, Liu et al. (2019) analyze syntactic properties of deep contextualized word vectors using shallow syntactic tasks, such as CCG supertagging (Clark 2002).

In a different vein, Hewitt and Manning (2019) proposed a structural probe to evaluate whether syntax trees are embedded in a linear transformation of an `ELMo` and `BERT` word representation space. Although their study does not support full parsing analysis either, it reports partial metrics.²

Parsing only with word vectors

Nevertheless, it remains an open question how much of the workload these encoders can take off a parser’s shoulders. To try to answer this, we build on top of previous work reducing parsing to sequence tagging as well as on using models with limited capacity. The goal is to bring to bear what pretrained encoders have learned from distributional evidence alone.

Notation We will be denoting a raw input sequence by $w = [w_0, w_1, \dots, w_{|w|}]$, with $w_i \in V$ and mark vectors and matrices with arrows (e.g. \vec{v} and \vec{W}).

Parsing as sequence labeling

Sequence labeling is a structured prediction task where given an input sequence, $w_{|w|}$, the goal is to generate one output label for every w_i . Part-of-speech tagging, chunking or named-entity recognition are archetype tasks of this type of problem. In this vein, recent work has demonstrated that is possible to design reductions to address full constituent and dependency parsing as sequence labeling too.³

²These partial metrics refer to analysis such as *undirected* Unlabeled Attachment Score (UUAS) as well as the average Spearman correlation of true to predicted distances.

³Whether sequence labeling is the most adequate paradigm, in terms of performance, for obtaining syntactic representations (in

Constituent parsing as sequence labeling Gómez-Rodríguez and Vilares (2018) reduced constituent parsing to a pure sequence labeling task, defining a linearization function $\Phi_{|w|} : T_{|w|} \rightarrow L^{|w|}$ to map the constituent tree of an input sentence $w=[w_0, w_1, \dots, w_{|w|}]$ into a sequence of $|w|$ labels, i.e. they proposed a method to establish a one-to-one correspondence between words and labels that encode a syntactic representation for each word with enough information to encode the full constituent tree of the sentence. In particular, each label $l_i \in L$ is a 3-tuple (n_i, c_i, u_i) where:⁴

- n_i encodes the number of tree levels in common between w_i and w_{i+1} (computed as the relative variation with respect to n_{i-1}).
- c_i encodes the lowest non-terminal symbol shared between those two words.
- u_i encodes the leaf unary branch located at w_i (if any).

Figure 1 illustrates the encoding with an example.

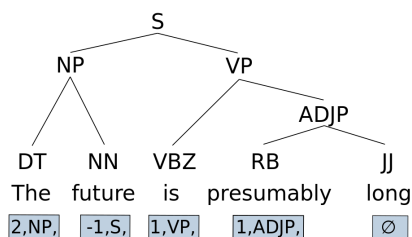


Figure 1: A linearized constituent tree according to Gómez-Rodríguez and Vilares (2018). For example, $n_{\text{'The'}} = 2$ since ‘The’ and ‘future’ have in common the top two levels in the tree (and there is no previous n), $c_{\text{'The'}} = \text{NP}$ because that is the non-terminal symbol shared at that lowest common ancestor, and $u_{\text{'The'}} = \emptyset$ as there is no unary branch for ‘The’. If we move one step forward, for ‘future’ we have $n_{\text{'future'}} = -1$ (in terms of the variation with respect to the previous timestep), since ‘future’ and ‘is’ just share the top level of the tree, i.e. one level less than for $n_{\text{'the'}}$.

Postprocessing The linearization function is complete and injective, but not surjective, i.e. postprocessing needs to be applied in order to ensure the validity of a predicted output tree. We follow the original authors’ strategy to solve the two potential sources of incongruities:

1. conflicting non-terminals, i.e. a nonterminal c can be the lowest common ancestor of more than two pairs of contiguous words (w_i, w_j) with $c_i \neq c_j$. In such case, we ignore all but the first prediction.
2. empty intermediate unary branches, i.e. decoding the sequence of n_i ’s might generate a predicted tree where some

contrast to algorithms) is a fair question. That said, the aim of this work is not to outperform other paradigms, but to find a way to estimate the ‘amount of syntax’ that is encoded in embeddings. To achieve this, we consider that sequence labeling with limited expressivity is a natural probing framework for syntax-sensitivity.

⁴The index is omitted when not needed.

intermediate unary branches are not assigned any non-terminal. If so, we simply delete that empty level.

Dependency parsing as sequence labeling In a similar fashion to Gómez-Rodríguez and Vilares (2018), Strzyz, Vilares, and Gómez-Rodríguez (2019) define a function to encode a dependency tree as a sequence of labels, i.e. $\Upsilon_{|w|} : T_{d,|w|} \rightarrow L_d^{|w|}$. Let $r_i \in L_d$ be a particular label that encodes the head term of w_i , they also represent it as a 3-tuple (o_i, p_i, d_i) where:⁵

- The pair (o_i, p_i) encodes the index of the head term. Instead of using the absolute index of the head term as a part of the label, the head of w_i is represented as the o_i th closest word to the right with PoS tag p_i if $o_i > 0$, and the $-o_i$ th closest word to the left with PoS tag p_i , if $o_i < 0$.
- d_i denotes the dependency relation between the head and the dependent.

Figure 2 illustrates the encoding with an example.

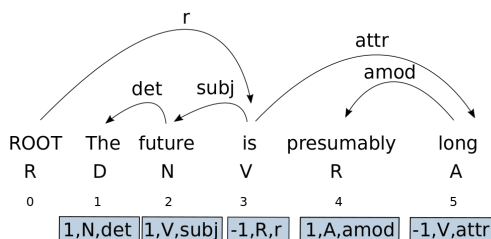


Figure 2: A linearized dependency tree according to the PoS tag-based encoding used in Strzyz, Vilares, and Gómez-Rodríguez (2019). For instance, $(o_{\text{'The'}}, p_{\text{'The'}}) = (1, N)$ which means that ‘The’ is the head of ‘future’ to the left whose part-of-speech tag is N, i.e. ‘future’; and $d_{\text{'The'}} = \text{det}$, which denotes the syntactic relationship existing between the head ‘future’ and the dependent ‘The’.

Postprocessing We ensure that the predicted dependency tree is acyclic and single-headed:

1. If no token is selected as syntactic root (by setting its head to be index 0, which we use as a dummy root, as seen in Figure 2), take as root the first one with $d=\text{root}$, or (if none exists) the first token. If multiple roots have been assigned, the first one is considered as the only root, and the rest become children of that one.
2. If a token has an invalid head index, attach it to the real syntactic root.
3. If there are cycles, the first token involved is assigned to the real root. The process is repeated until the tree is acyclic.

⁵Note that a dependency tree can be trivially encoded as a sequence of labels using a naïve positional encoding that uses the word absolute index and the dependency type, but in previous work this did not lead to robust results.

Models with limited expression

Parsing as sequence labeling can be used to set up a direct, one-to-one mapping from words to some sort of syntactic labels that help establish conclusions about the syntactic properties that such word vectors accommodate. However, previous work on parsing as sequence labeling did not exploit this paradigm to study this. Instead, they simply trained different task-specific decoders such as BILSTMs, which can potentially mask poor syntactic abilities of word representations.

We propose to remove any task-specific decoder and instead directly map n word vectors (extracted from a given pretrained architecture) to n syntactic labels that encode the tree (using a single feed-forward layer, the simplest mapping strategy). Such architecture can be used to study the two research questions we address. First, it can be used to explore how far we can reach relying only on pretraining architectures. At the same time, these models can be used to probe syntax-sensitivity of continuous vector representations. More particularly, they minimize the risk of neural architectures, training objectives or specific parsing algorithms implicitly hiding, modifying and biasing the syntactic abilities captured by word representations during pretraining. This is in line with Tenney et al. (2018), who explore properties of contextualized vectors by fixing their representations and training a 2-layer perceptron for certain tasks. The aim was to have a model with limited expression, to focus on the information that is directly represented in the vectors. However, when evaluating syntax-sensitivity, Tenney et al. relied on a simplified and partial version of constituent and dependency parsing, as mentioned in the related work section. It is also worth remarking that we take a more extreme approach, and use just a single feed-forward layer.

Model architecture

We denote the output of a pretrained encoder by $\vec{x} = [\vec{x}_0, \vec{x}_1, \dots, \vec{x}_{|x|}]$, where \vec{x}_i is the vector for the word indexed at i . PoS tags and character embeddings are not used to train the model.⁶ Our architecture is depicted in Figure 3: we use a feed-forward layer on top of the pretrained encoder to predict each label, y_i , followed by a softmax activation function:

$$P(y = j | \vec{x}_i) = \text{softmax}(\vec{W} \cdot \vec{x}_i + \vec{b}) = \frac{e^{\vec{W}_j \cdot \vec{x}_i}}{\sum_k e^{\vec{W}_k \cdot \vec{x}_i}} \quad (1)$$

The model is optimized using categorical cross-entropy:

$$\mathcal{L} = - \sum \log(P(y | \vec{x}_i)) \quad (2)$$

We allow to freeze or fine-tune the word vectors during training, and refer to the models as `ff` (feed-forward) and `ff-ft`, respectively. Freezing word representations aims to not adapt the syntax-sensitivity inherently encoded. The goal of fine-tuning is to test the best performance that we can achieve using just pretraining networks.

⁶Note that for dependency parsing we have predicted PoS tags separately to rebuild the tree in CoNLL format, as the output labels contain the PoS tag of the head term. We understand this could lead to some latent learning of PoS tagging information.

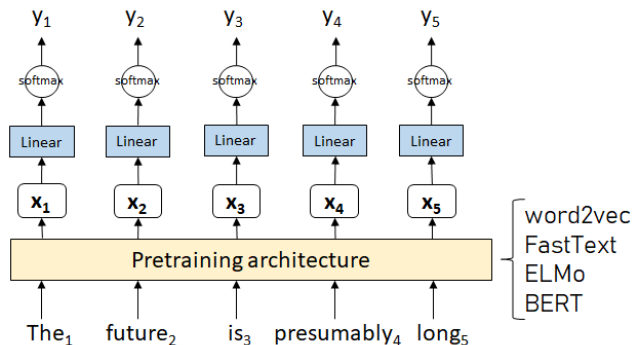


Figure 3: High level architecture of parsing as pretraining

Models with task-specific decoders We also build models using task-specific decoders. More particularly, we use for this setup 2-stacked BILSTMs. This is similar to Gómez-Rodríguez and Vilares; Strzyz, Vilares, and Gómez-Rodríguez (2018; 2019). This has a secondary goal: to further illustrate if the tendencies observed with the models with limited expression remain, and thus to provide a more complete evaluation on how choices of embeddings interact with the presence or absence of task-specific decoding. We again will freeze and fine-tune the pretraining architectures, and will refer these models as `lstm` and `lstm-ft`.

Pretrained encoders

We test both *precomputed* (lookup tables that map words to vectors), and *contextualized* representations (where each word vector is dependent on the context and generated by a pretrained neural network). In this work, we test:

- **Random embeddings:** Uniformly distributed, in the interval $[-\sqrt{(3.0)/d}, \sqrt{(3.0)/d}]$, where $d = 300$ is the dimension of the embedding. We tried other dimensions corresponding to the size of other tested methods, but we did not observe significant differences. We consider this can roughly be seen as a suitable baseline to know whether the tested embedding methods learn syntax above expectation by chance.
- **Skip-gram word2vec** (Mikolov et al. 2013). The approach learns to predict the window context based on the word, learning to generate the representation for that word in the process.⁷
- **Structured skip-gram word2vec** (Ling et al. 2015). A variant of the standard word2vec that is sensitive to word order, keeping separate matrices to learn to predict the word at each position in the window context.⁸
- **Cbow-based FastText** (Bojanowski et al. 2017). A cbow word2vec extension. It tries to predict a word

⁷For word2vec, we use GoogleNews-vectors-negative300 (<https://code.google.com/archive/p/word2vec/>)

⁸For structured skip-gram word2vec, we use the vectors at <https://github.com/clab/lstm-parser/blob/master/README.md>

based on its context, learning to generate its representation in the process.⁹

- GloVe (Pennington, Socher, and Manning 2014). It creates precomputed word vectors combining matrix factorization with local window methods.¹⁰
- ELMo (Peters et al. 2018). Each word vector is a weighted average: (i) a context-independent vector computed through a character convolutional network, (ii) an output vector from a 2-layer left-to-right LSTM, and (iii) and output vector from a 2-layer right-to-left LSTM. Following the canonical paper, we let the fine-tuned models learn a linear combination of these representations, but will freeze the language modeling layers.¹¹
- BERT (Devlin et al. 2018). Uses a Transformer to generate the output word vectors. As the Transformer purely relies on attention mechanisms, the positional information is encoded through positional embeddings, which poses an interesting difference with respect to ELMo, where such information is inherently encoded through the recurrent network.¹²

When available, we chose 300-dimensional vectors, but ELMo vectors have 1024 dimensions, BERT 768, and the Ling et al. (2015) ones just 100. Despite this, we stuck to available pretrained models since we feel the experiments are more useful if performed with the standard models used in NLP. Also, it lies out of the standard computational capacity to train BERT and ELMo models from scratch to force them to have the same number of dimensions.

We build on top of the framework by Yang and Zhang (2018) to run all vectors under our setup, except BERT, for which we use a pytorch wrapper and its hyperparameters.¹³

Experiments

The source code is accessible at <https://github.com/aghie/parsing-as-pretraining>.

Corpora

We use the English Penn Treebank (PTB) (Marcus, Santorini, and Marcinkiewicz 1993) for evaluation on constituent parsing, and the EN-EWT UD treebank (v2.2) for dependency parsing (Nivre and others 2017). To train our sequence labeling models, we add dummy beginning- and end-of-sentence tokens, similarly to previous work on parsing as sequence labeling. The labels are predicted atomically.

⁹FastText can consider subword level information. In early experiments we tried both wiki-news-300d-1M-subword and wiki-news-300d-1M pretrained vectors, choosing the latter because they behaved better (<https://fasttext.cc/docs/en/english-vectors.html>).

¹⁰For GloVe, we use the glove.840B.300 vectors (<https://nlp.stanford.edu/projects/glove/>)

¹¹ELMo can be downloaded from <https://allennlp.org/elmo>

¹²For BERT, we used bert-base-cased (<https://github.com/google-research/bert>)

¹³<https://github.com/huggingface/pytorch-pretrained-BERT>. For ff/lstm, the learning rate was set to 5e-4.

Metrics

For constituents, we use labeled bracketing F1-score and the COLLINS.PRM parameter file. We also break down the results according to different span lengths. For dependencies, we use UAS¹⁴ and LAS¹⁵. We use the EN-EWT UD treebank with predicted segmentation by UDPipe (Straka 2018), for comparison against related work. We also compute *dependency displacements*, i.e., signed distances between the head and the dependent terms (where the dependency type is predicted correctly too).

Results and discussion

Table 1 shows the F1-scores for constituent parsing on the PTB, and both UAS and LAS scores for dependency parsing on the EN-EWT UD; evaluating both models where the pretraining network is frozen (ff) and fine-tuned (ff-ft) during training. For a better understanding of how the tendencies of these results can be influenced by a task-specific decoder, Table 2 replicates the same set of experiments using instead a 2-layered BILSTM decoder, allowing to freeze (lstm) and fine-tune (lstm-ft) the pretraining weights.

| Vectors | PTB | | EN-EWT | | | |
|-----------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | ff | ff-ft | ff | | ff-ft | |
| | F1 | F1 | UAS | LAS | UAS | LAS |
| Random | 32.9 | 42.8 | 37.3 | 30.6 | 46.4 | 39.7 |
| GloVe | 37.9 | 42.9 | 44.7 | 38.0 | 47.1 | 40.3 |
| Struct.word2vec | 40.0 | 43.2 | 45.4 | 37.8 | 47.0 | 40.2 |
| word2vec | 39.1 | 43.5 | 45.6 | 37.9 | 46.8 | 40.2 |
| FastText | 41.4 | 43.0 | 46.6 | 39.0 | 47.5 | 40.3 |
| ELMo | 69.7 | 75.8 | 65.2 | 60.3 | 67.6 | 62.4 |
| BERT | 78.2 | 93.5 | 68.1 | 63.0 | 81.0 | 78.8 |

Table 1: Labeled F1-score on the PTB test set, and UAS/LAS on the EN-EWT UD test set (with predicted segmentation)

| Vectors | PTB | | EN-EWT | | | |
|-----------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | lstm | lstm-ft | lstm | | lstm-ft | |
| | F1 | F1 | UAS | LAS | UAS | LAS |
| Random | 80.7 | 88.0 | 62.3 | 56.5 | 73.2 | 69.7 |
| GloVe | 89.2 | 89.5 | 74.3 | 71.2 | 75.6 | 72.5 |
| Struct.word2vec | 89.0 | 89.5 | 73.2 | 69.5 | 74.7 | 71.5 |
| word2vec | 89.1 | 89.6 | 72.1 | 68.5 | 74.7 | 71.5 |
| FastText | 89.0 | 89.5 | 73.2 | 69.9 | 74.6 | 71.5 |
| ELMo | 92.5 | 92.7 | 78.8 | 76.5 | 79.5 | 77.4 |
| BERT | 92.2 | 93.7 | 78.4 | 75.7 | 81.1 | 79.1 |

Table 2: Same scores reported in Table 1 on the PTB and EN-EWT UD, but using instead a (task-specific) 2-layer BILSTM decoder (freezing and fine-tuning the pretraining network).

Parsing with pretraining architectures

We first discuss the performance of the ff and ff-ft models. Among the precomputed word vectors, window-

¹⁴Unlabeled Attachment Score: Percentage of relations for which the head has been assigned correctly.

¹⁵Labeled Attachment Score: Percentage of relations for which the head and the dependency type have been assigned correctly.

based approaches perform slightly better than alternatives such as GLoVe, but they all do clearly surpass the control method (random embeddings). With respect to contextualized ones, for ff , both ELMo and BERT get a decent sense of phrase structures, with BERT slightly superior. For the $ff-ft$ models differences are larger: while ELMo is well behind a robust model (we believe this could be due to keeping the BiLM weights fixed as done by default in (Peters et al. 2018)), BERT $ff-ft$ performs competitively. From the results in Table 2 using task-specific decoders, we observe that even if the tendency remains, the differences across different vectors are smaller. This is due to the capacity of these decoders to obtain richer contextualized representations for the target task and to mask poor syntax capabilities of the input, even when the pretrained network is kept frozen.

Tables 3 and 4 compare the BERT-based models without task-specific decoders against the state of the art for constituent and dependency parsing, respectively. In this context, it is worth remarking that all our models only use words as features. For constituents, BERT obtains a 93.5% F1-score vs the 95.8% reported by Kitaev and Klein (2018b) and Zhou and Zhao (2019), which are the current best performing models and use BERT representations as input (together with PoS tag embeddings, and in the case of Zhou and Zhao also char embeddings). Surprisingly, the performance of BERT $ff-ft$ is superior to strong models such as (Dyer et al. 2016), and it outperforms traditional parsers that used dozens of hand-crafted features to explicitly give the model structural information, such as Zhu et al. (2013)’s parser. Against other sequence tagging constituent parsers, we improve by more than two points the 91.2% by Vilares, Abdou, and Søgaard (2019), which was the current state of the art for this particular paradigm, and used task-specific decoders and multi-task learning, together with PoS tags and character embeddings. For dependencies, we observe similar tendencies. With respect to the state of the art, the winning system (Che et al. 2018) at the CoNLL-UD shared task reported a LAS of 84.57% with an ensemble approach that incorporated ELMo vectors (and also additional features such as PoS tag embeddings). BERT $ff-ft$ also performs close to widely used approaches such as (Kiperwasser and Goldberg 2016). When comparing against sequence labeling dependency parsers, we obtain a LAS of 78.8% versus the 78.6% reported by (Strzyz, Vilares, and Gómez-Rodríguez 2019), which included task-specific decoders and linguistic features, such as PoS tags and character embeddings.

Analysis of syntactic abilities

Here, we consider the ff models for two reasons: they (i) do not fine-tune word vectors, and (ii) have limited expression.

Figure 4 shows the F1-score on span identification for different lengths. All precomputed vectors surpass the random model. For contextualized vectors, although BERT does consistently better than ELMo, both fairly succeed at identifying labeled spans. Figure 5 shows the performance by span label. Labels coming from shorter and frequent spans (e.g. NP) are easier to handle by precomputed vectors. However, these struggle much more than contextualized ones when these

| Models | PTB F1 |
|---|-------------|
| ff_{BERT} | 78.2 |
| $ff-ft_{BERT}$ | 93.5 |
| Vinyals et al. (2015) | 88.3 |
| Gómez-Rodríguez and Vilares (2018) [◊] | 90.7 |
| Zhu et al. (2013) | 90.4 |
| Vilares, Abdou, and Søgaard (2019) [◊] | 91.2 |
| Dyer et al. (2016) (generative) | 92.1 |
| Kitaev and Klein (2018a) | 95.1 |
| Kitaev and Klein (2018b) | 95.8 |
| Zhou and Zhao (2019) | 95.8 |

Table 3: Comparison against related work on the PTB. [◊] are other sequence tagging parsers.

| Models | EN-EWT | |
|--|-------------|-------------|
| | LAS | UAS |
| ff_{BERT} | 63.0 | 68.1 |
| $ff-ft_{BERT}$ | 78.8 | 81.0 |
| Strzyz, Vilares, and Gómez-Rodríguez (2019) [◊] | 78.6 | 81.5 |
| Kiperwasser and Goldberg (2016) | 79.0 | 82.2 |
| Straka (2018) (UDpipe 2.0) | 82.5 | 85.0 |
| Che et al. (2018) | 84.6 | 86.8 |

Table 4: Comparison against related work on the EN-EWT UD test set. [◊] are other sequence tagging parsers.

come from larger or less frequent spans (e.g. VP).¹⁶

Figure 6 shows the analysis for dependencies, using dependency displacements. The results for precomputed embeddings are similar across positive and negative displacements. For contextualized vectors, the differences between both ELMo and BERT are small, and smaller than for constituents. They also perform closer to precomputed vectors, suggesting that these models could be less suited for dependency structures. The results also show similarities to the analysis on the constituent and dependency subtasks proposed by Tenney et al. (2018), who expose that ELMo and BERT perform closer to each other when they are asked to find out dependency relations. Figure 7 shows the performance on common dependency relations. Simple relations such as `det` are handled accurately by both precomputed and contextualized vectors,¹⁷ while harder ones such as `root`, `nsubj` or `obj` need deep contextualized ones.

In general, we feel Figures 4-7 distill that ELMo and BERT representations respond better to phrase-structure representations. Our intuition is that this might be due to language modelling objectives, where learning the concept of ‘phrase’ seems more natural than the one of ‘dependency’, although the answer to this is left as an open question.

¹⁶For unary chains, we only pick up the uppermost element. Also, the span containing the whole sentence is ignored.

¹⁷Other simple relations such as `punct` or `case` also showed a high performance, but only when evaluating relation labels only (i.e. when failing the head index was not penalized, as opposed to Figure 7). These results are not included due to space limitations.

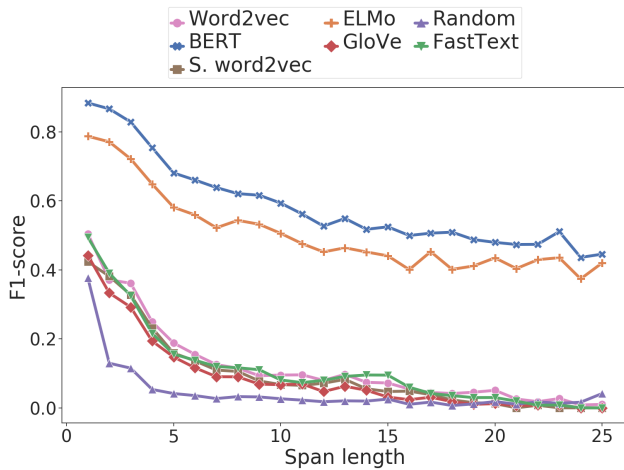


Figure 4: Span length F1-score on the PTB test set for the $\mathcal{E}\mathcal{F}$ models

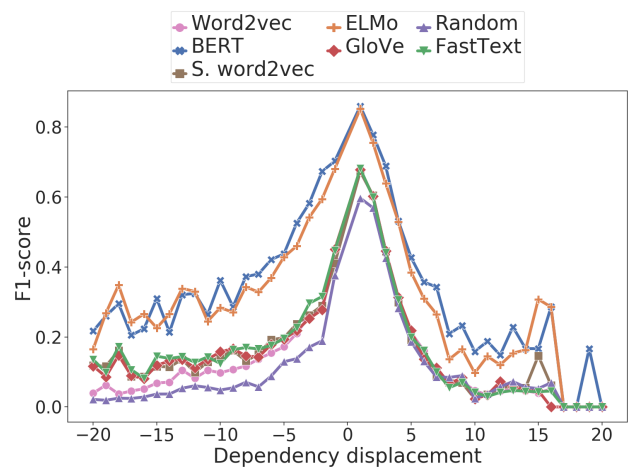


Figure 6: Dependency displacement F1-score on the EN-EWT UD test set for the $\mathcal{E}\mathcal{F}$ models (with gold segmentation)

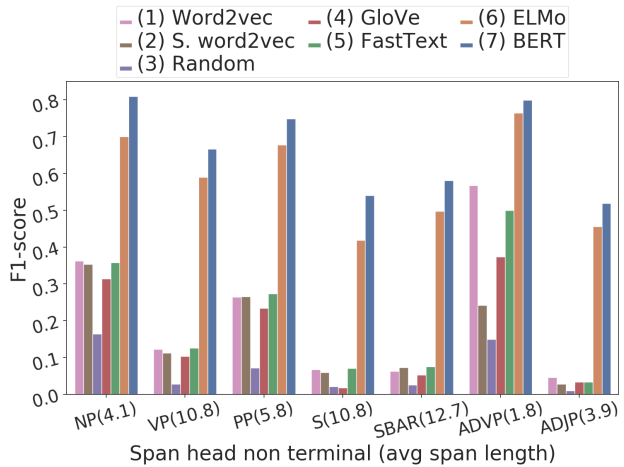


Figure 5: Span label F1-score on the PTB test set for the $\mathcal{E}\mathcal{F}$ models.

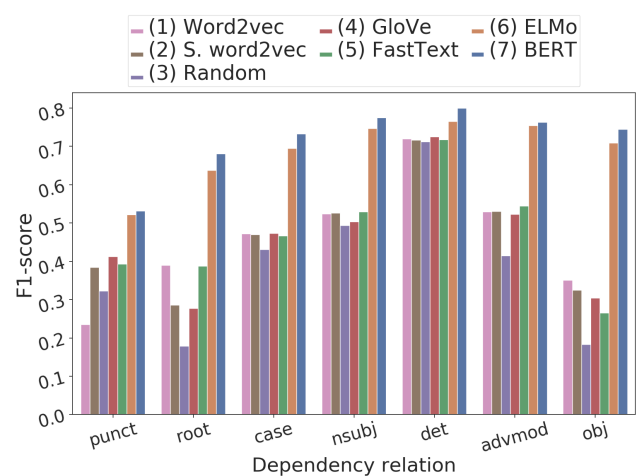


Figure 7: F1-score for the most common relations on the EN-EWT UD test set for $\mathcal{E}\mathcal{F}$ models (with gold segmentation)

Conclusion

We proposed a method to do constituent and dependency parsing relying solely on pretraining architectures – that is, without defining any parsing algorithm or task-specific decoders. Our goal was twofold: (i) to show to what extent it is possible to do parsing relying only in word vectors, and (ii) to study if certain linguistic structures are learned in pre-training networks. To do so, we first cast parsing as sequence labeling, to then map, through a linear layer, words into a sequence of labels that represent a tree. During training, we considered to both freeze and fine-tune the pretraining networks. The results showed that (frozen) pretraining architectures such as ELMo and BERT get a sense of the syntactic structures, and that a (tuned) BERT model suffices to parse. Also, by freezing the weights we have provided different analyses regarding the syntax-sensitivity of word vectors.

Contemporaneously to this work, Hewitt and Liang (2019) proposed to complement probing frameworks that

test linguistic abilities of pretrained encoders with *control tasks*, i.e. tasks that can be only learned by the probing framework (e.g. classifying words into random categories). If the pretrained network is encoding the target property, the probing framework should perform well on the target task and poorly on the control one. As future work, we plan to add this strategy to our analyses, and expand our experiments to languages other than English.

Acknowledgements

We thank Mark Anderson and Daniel Herscovich for their comments. DV, MS and CGR are funded by the ERC under the European Union’s Horizon 2020 research and innovation programme (FASTPARSE, grant No 714150), by the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and by Xunta de Galicia (ED431B 2017/01). AS is funded by a Google Focused Research Award.

References

- Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2017. Enriching word vectors with subword information. *TACL* 5:135–146.
- Che, W.; Liu, Y.; Wang, Y.; Zheng, B.; and Liu, T. 2018. Towards better ud parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. *CoNLL 2018* 55.
- Chen, D., and Manning, C. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP 2014*, 740–750.
- Chomsky, N. 1957. *Syntactic structures*. Mouton.
- Clark, S. 2002. Supertagging for combinatory categorial grammar. In (*TAG*), 19–24.
- Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Dyer, C.; Kuncoro, A.; Ballesteros, M.; and Smith, N. A. 2016. Recurrent neural network grammars. In *NAACL 2016*, 199–209.
- Falenska, A., and Kuhn, J. 2019. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *ACL 2019*, 117–128.
- Goldberg, Y. 2019. Assessing bert’s syntactic abilities. *arXiv preprint arXiv:1901.05287*.
- Gómez-Rodríguez, C., and Vilares, D. 2018. Constituent parsing as sequence labeling. In *EMNLP 2018*, 1314–1324.
- Gulordava, K.; Bojanowski, P.; Grave, E.; Linzen, T.; and Baroni, M. 2018. Colorless green recurrent networks dream hierarchically. In *NAACL 2018*, volume 1, 1195–1205.
- Hewitt, J., and Liang, P. 2019. Designing and interpreting probes with control tasks. In *EMNLP 2019*, 2733–2743.
- Hewitt, J., and Manning, C. D. 2019. A structural probe for finding syntax in word representations. In *NAACL 2019*, volume 2.
- Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.
- Kiperwasser, E., and Goldberg, Y. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *TACL* 4:313–327.
- Kitaev, N., and Klein, D. 2018a. Constituency parsing with a self-attentive encoder. In *ACL 2018*, volume 1, 2676–2686.
- Kitaev, N., and Klein, D. 2018b. Multilingual constituency parsing with self-attention and pre-training. *CoRR* abs/1812.11760.
- Li, Z.; Cai, J.; He, S.; and Zhao, H. 2018. Seq2seq dependency parsing. In *COLING 2018*, 3203–3214.
- Ling, W.; Dyer, C.; Black, A. W.; and Trancoso, I. 2015. Two/too simple adaptations of word2vec for syntax problems. In *NAACL 2015*, 1299–1304.
- Linzen, T.; Dupoux, E.; and Goldberg, Y. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *TACL* 4:521–535.
- Liu, N. F.; Gardner, M.; Belinkov, Y.; Peters, M. E.; and Smith, N. A. 2019. Linguistic knowledge and transferability of contextual representations. In *NAACL 2019*, 1073–1094.
- Marcus, M.; Santorini, B.; and Marcinkiewicz, M. A. 1993. Building a large annotated corpus of english: The penn treebank.
- Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS 2013*, 3111–3119.
- Nivre, J., et al. 2017. Universal dependencies 2.1. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.
- Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *EMNLP 2014*, 1532–1543.
- Peters, M.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *NAACL 2018*, volume 1, 2227–2237.
- Shi, T.; Huang, L.; and Lee, L. 2017. Fast (er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *EMNLP 2017*, 12–23.
- Straka, M. 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*, 197–207.
- Strzyz, M.; Vilares, D.; and Gómez-Rodríguez, C. 2019. Viable dependency parsing as sequence labeling. In *NAACL 2019*, 717–723.
- Tenney, I.; Xia, P.; Chen, B.; Wang, A.; Poliak, A.; McCoy, R. T.; Kim, N.; Van Durme, B.; Bowman, S.; Das, D.; et al. 2018. What do you learn from context? probing for sentence structure in contextualized word representations.
- Vilares, D.; Abdou, M.; and Søgaard, A. 2019. Better, faster, stronger sequence tagging constituent parsers. In *NAACL 2019*, 3372–3383.
- Vinyals, O.; Kaiser, Ł.; Koo, T.; Petrov, S.; Sutskever, I.; and Hinton, G. 2015. Grammar as a foreign language. In *NeurIPS 2015*, 2773–2781.
- Yang, J., and Zhang, Y. 2018. Ncrf++: An open-source neural sequence labeling toolkit. *ACL 2018, System Demonstrations* 74–79.
- Zhang, Y., and Nivre, J. 2011. Transition-based dependency parsing with rich non-local features. In *ACL 2011*, 188–193.
- Zhou, J., and Zhao, H. 2019. Head-driven phrase structure grammar parsing on Penn treebank. In *ACL 2019*, 2396–2408.
- Zhu, M.; Zhang, Y.; Chen, W.; Zhang, M.; and Zhu, J. 2013. Fast and accurate shift-reduce constituent parsing. In *ACL 2013*, 434–443.