# An Empirical Exploration of Local Ordering Pre-training for Structured Prediction

**Zhisong Zhang, Xiang Kong, Lori Levin, Eduard Hovy**
Language Technologies Institute, Carnegie Mellon University
{zhisongz,xiangk,lsl,hovy}@cs.cmu.edu

## Abstract

Recently, pre-training contextualized encoders with language model (LM) objectives has been shown an effective semi-supervised method for structured prediction. In this work, we empirically explore an alternative pre-training method for contextualized encoders. Instead of predicting words in LMs, we "mask out" and predict word order information, with a local ordering strategy and word-selecting objectives. With evaluations on three typical structured prediction tasks (dependency parsing, POS tagging, and NER) over four languages (English, Finnish, Czech, and Italian), we show that our method is consistently beneficial. We further conduct detailed error analysis, including one that examines a specific type of parsing error where the head is misidentified. The results show that pre-trained contextual encoders can bring improvements in a structured way, suggesting that they may be able to capture higher-order patterns and feature combinations from unlabeled data.

## 1 Introduction

Recently, pre-trained contextualized encoders (Peters et al., 2018; Radford et al., 2019; Devlin et al., 2019) have been shown to be beneficial for NLP tasks, including structured prediction (Kulmizev et al., 2019; Kondratyuk and Straka, 2019). Most of the pre-training objectives are based on variants of language models (LM), that is, the model is trained to predict lexical items with partial inputs. Masked Language Model (MaskLM) is a typical example, popularized by BERT (Devlin et al., 2019), which masks out lexical tokens in the input sequences and predicts their identities. Since natural sentences contain not only lexical tokens but also their linearized word orders, it is a natural question if we can perform pre-training by "masking out" and recovering word order information.

Word order is an important method of grammatical encoding (Dryer, 2007), and can play an important role in predicting basic sentence structures (Naseem et al., 2012; Täckström et al., 2013; Ammar et al., 2016; Ahmad et al., 2019). Recently, Wang et al. (2018) pre-train an explicit word reordering model and show that its contextualized representations improve dependency parsing.

In this work, we explore a local ordering pre-training strategy with word-selection objectives. Instead of completely discarding original word order information, we segment the input sentence into local bags of words and keep the ordering of these bags. Inside each bag, we discard all the local word orders and train the model to recover them. Furthermore, we simplify the training objectives: instead of training explicit word linearizers which require extra unidirectional decoders, we only ask the model to select original neighboring words. This scheme simplifies the pre-training procedure and enhances the encoder since it can take information from the whole sentence.

A similar idea is explored in StructBERT (Wang et al., 2020), which adopts a word structural objective by shuffling and re-predicting randomly selected subsets of trigrams. Our method is different in that we make local bags of words instead of shuffling and we adopt simpler and cheaper word-selection objectives. Moreover, we focus on empirical experiments and error analysis on structured prediction tasks.

We evaluate on three structured prediction tasks (dependency parsing, part-of-speech (POS) tagging, and Named Entity Recognition (NER)) over four languages (English, Finnish, Czech, Italian). The highlights of our findings are:

- For local ordering pre-training, the best performance is obtained when partially masking out information in a suitable degree. (§3.2.1)



*Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1770–1783
November 16 - 20, 2020. 

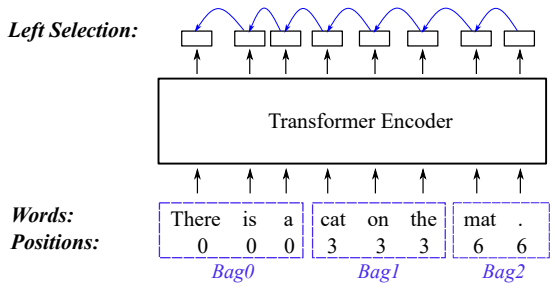


Figure 1: Illustration of the local ordering pre-training strategy. We segment the input sentence into local bags (bag size is fixed to three here) and discard word order information inside each bag by assigning same position indexes. Training objectives are to select original neighboring words. Here, we only show the scenario for direct left-neighbor selection, while selections for other positions will be similar.

- Even when pre-trained with a small amount of data (1M Wikipedia sentences), our method can improve the performances of structured predictors in a consistent way. Our method performs comparably to MaskLM and there can be further improvements when combining the two objectives, especially for parsing, which is the most structured task we explore. (§3.2.2, §3.3)
- The pre-trained models make fewer structured errors, suggesting that they may be able to capture higher-order patterns and feature combinations from unlabeled data. (§3.4)

## 2 Local Ordering Pre-training

Word reordering or linearization itself is an interesting task, aiming to arrange a bag of words into a natural sentence (Liu et al., 2015; Zhang and Clark, 2015; Schmaltz et al., 2016). Wang et al. (2018) show that representations from an explicit reordering model can benefit dependency parsing. However, there may be two issues with an explicit reordering model for pre-training. Firstly, the input is a bag of words without any positional information. This could discard too much information, leading to relatively large discrepancies between pre-training and fine-tuning. Moreover, training explicit reordering models requires unidirectional decoders, which are only aware of contexts from one direction and cannot make full use of the bidirectional information at one time.

To mitigate these issues, we explore a local ordering pre-training strategy with word-selection objectives. Inspired by MaskLM, where only some of the tokens are masked out, we "mask out" partial ordering information by segmenting the input sentence into multiple local bags of words, and only discarding word orders inside each bag (§2.1). Moreover, we adopt simpler training objectives of selecting original neighboring words, which avoids the need of unidirectional decoders and focuses the pre-training on the encoder (§2.2).

### 2.1 Local Bags of Words

Instead of discarding all positional information, we keep the overall ordering and only discard local word orders. This is achieved by segmenting the input sentence into a sequence of local bags of words. In this way, the model is not aware of the local word orders inside each bag, but the overall ordering of the bags is kept. Figure 1 provides a simplified example to illustrate this scheme. We specify special positional encodings to "mask out" local word orders: inside each local bag, all the tokens get the same positional indexes. For example, the position indexes in the first bag {There, is, a} are all set to 0, while in the second bag {cat, on, the}, the position indexes are all casted to 3.

The above example illustrates a simplified scheme, whereas in actual pre-training, we adopt several variations to make it more flexible. 1) First, for the position indexes inside each bag, we do not fix them to the index of the first token, but randomly pick a representative token and adopt its index. For example, in the second bag, we randomly choose a representative index from $\{3, 4, 5\}$, and then set all position indexes to this value. 2) Moreover, for each local bag, we randomly sample its bag size from a pre-defined range, instead of using a fixed size. 3) In addition, we randomly pick half of the bags and keep the original position indexes in them, which is another way of retaining partial ordering information.

### 2.2 Word-selection Objectives

Since the aim of pre-training is not the pre-training task itself but the encoder, we do not need an explicit word reordering model, which may require unidirectional decoders. In some way, an explicit reordering model can be regarded as a LM which constrains candidate words to come from the input sentence. Therefore, it may suffer from the same problem as unidirectional LMs: at one time, contexts from only one direction can be utilized instead of from both directions. This is the bias of unidirectional decoders and we replace them with simpler word selectors.

Specifically, we only ask the model to select original neighbors for each word that loses its local word order information. Figure 1 illustrates the case for left-neighbor selection. This task is non-trivial since the model is unaware of word orders inside each bag. In many scenarios, it needs to capture certain global sentence structures. For example, in the second bag {cat, on, the}, if looking only locally, we may pick "the" as the left neighbor of "cat". However, if we notice that there is another determiner "a" in the first bag, then "the" will not be the only choice.

In actual running, we adopt four classification tasks corresponding to different original offsets: two for the selection of the original left neighbor (-1) and the left of the left neighbor (-2) and two for the right ones. Each word selector gets its own parameters. Since the word selection task is similar to dependency parsing (Zhang et al., 2017), we adopt the biaffine scorer (Dozat and Manning, 2017). The training objectives are negative log likelihoods on selecting the correct words.

Formally, assume that we have an input sequence of $w_0, w_1, \ldots, w_{n-1}$, and we generate their corrupted positions $p_0, p_1, \ldots, p_{n-1}$ with our local bag strategy. For a specific word $w_i$ (where $p_i \neq i$) and a specific selection offset $\delta$ ($\delta \in \{-2, -1, 1, 2\}$), its loss objective will be (for brevity, we omit the conditions on the inputs):

$$\ell_{w_i,\delta} = -\log \frac{\exp \text{Score}_\delta(w_i, w_{i+\delta})}{\sum_j \exp \text{Score}_\delta(w_i, w_j)}$$

Here, $\text{Score}_\delta$ denotes the scores of two tokens having positional differences $\delta$.

Notice that the simplified tasks are not necessarily easier than the explicit reordering task, since we can recover the original word order if we know all the local neighboring information. The word-selection objectives get rid of the explicit decoder as well as its unidirectional bias. At the same time, the model is still as efficient as word reordering models, since we only need to select among the words that appear in the input sentence, and there is no need to do the computationally expensive normalizations over the whole vocabulary as in LMs.

### 2.3 Hybrid Training

We further perform multi-task hybrid training, including both ordering and MaskLM objectives. Actually, our local ordering strategy can be integrated with MaskLM in a natural way. Since half of the local bags preserve the original position indexes, we randomly select words inside those bags to mask and predict. This scheme is nearly as effective as the original one because we can segment local bags and mask words at the same time and thus there is no need to run through the encoder twice. The encoder produces one set of contextualized representations, which we can feed to the corresponding modules of the two tasks. We adopt equal weights (both set to 0.5) for the two objectives.

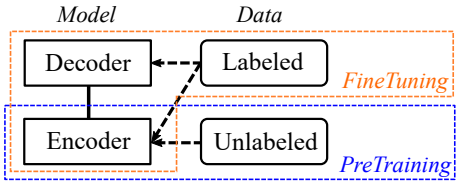


Figure 2: Illustration of the overall training scheme. The encoder is pre-trained in the pre-training stage with the unlabeled data. Later, the task-specific decoder is stacked and both modules are further fine-tuned with task-specific labeled data.

## 3 Experiments

### 3.1 Settings

In this sub-section, we briefly describe our main experiment settings[1]. Please refer to the Appendix for more details.

**Scheme** Figure 2 shows our overall training scheme. We take a two-step approach: pre-training plus fine-tuning. First, the encoder is pre-trained using a relatively large unlabeled corpus, then the task-specific decoders are stacked upon the pre-trained encoder and all the modules are fine-tuned with task-specific labeled data, which is much smaller than the pre-training data.

**Data** We explore four languages to evaluate our pre-training strategy: English (en), Finnish (fi), Czech (cs), and Italian (it). For the unlabeled data in pre-training, we collect Wikipedia corpora from the 2018-Fall Wiki-dump. Due to limitation of computational resources, we sample 1M sentences for each language. For POS tagging and dependency parsing, we utilize Universal Dependencies (UD) v2.4 (Nivre et al., 2019). For NER, we utilize CoNLL03 (Tjong Kim Sang and De Meulder, 2003) for English, Digitoday (Ruokolainen et al., 2019) for Finnish, Czech Named Entity Corpus (Ševčíková et al., 2007) for Czech and

[1]Our implementation is publicly available at https://github.com/zzsfornlp/zmsp

EVALITA 2009 (Speranza, 2009) for Italian. We mainly follow the default dataset splittings, except for the training sets. To investigate middle- and low-resource scenarios, we explore three settings of different training sizes, sampling 1k, 5k and 10k sentences from the original training set. We adopt standard evaluation criteria: accuracies for POS tagging, first-level (language-independent) Labeled Attachment Score (LAS) for dependency parsing, and F1 score for NER.

**Encoders** We adopt encoders with the same architecture: a 6-layer Transformer, whose head number, model dimension and feed-forward hidden dimension are set to 8, 512 and 1024, respectively. In addition, we adopt relative positional encodings (Shaw et al., 2018; Dai et al., 2019) within the Transformer, since in preliminary experiments we find this helpful for target tasks. In contrast to BERT, we adopt words[2] as basic input and modeling units. We further include a character-level Convolutional Neural Network (CNN) to capture internal structures of words.

**Decoders** For the decoders of specific tasks, we adopt typical solutions. For dependency parsing, we adopt the biaffine graph-based decoder (Dozat and Manning, 2017). For POS tagging, we simply add a single-layer classifier over all tags (Yang et al., 2018). For NER, we adopt a standard CRF layer (Lafferty et al., 2001).

**Training** For model training, we adopt the Adam optimizer (Kingma and Ba, 2014) with a warming-up styled learning rate schedule. In pre-training, each mini-batch includes 480 sentences and we train the model for 200k steps, in which the first 5k steps are specified for linearly increasing the learning rate towards 4e-4. The pre-training stage takes around 3 days with one RTX 2080 Ti GPU. In task-specific training, we adopt a mini-batch size of 80 sentences and train the model for maximally 250 epochs over the training set, which generally takes several hours using a single GPU.

## 3.2 Effects of Pre-training Strategies

In this sub-section, we explore the effects of pre-training strategies. Here, we take the English dependency parsing dataset for development.

[2]Except for those which directly utilize BERT, all models adopt the same word-based input scheme. We adopt this mainly to follow the conventions of the target tasks and to compare with baselines without pre-trained encoders.

| R | 3 | 5 | 7 | 9 | 11 | $\infty$ |
|---|---|---|---|---|---|---|
| 10k | 86.83 | 87.72 | 87.75 | **87.91** | 87.64 | 86.98 |
| 5k | 85.61 | 86.54 | **86.70** | **86.70** | 86.38 | 85.64 |
| 1k | 80.87 | 82.07 | **82.25** | 81.91 | 82.17 | 79.06 |

Table 1: Comparisons of bag size ranges ([$\frac{R+1}{2}$, R]) for the local ordering strategy. "R=$\infty$" indicates that all words from one input sentence fall into one bag. Evaluations are performed with the English dependency parsing task (LAS on development set). Each row represents different (target task) training sizes.

### 3.2.1 Bag Size Range

As described in §2.1, we adopt variable bag sizes for the ordering pre-training. The aim is to make the model more flexible and prevent it from always seeing the same patterns associated with fixed bag sizes. The neighbor selection process is not affected by this since it does not care about the bag boundaries, and selects among all the input tokens. The bag size range is a major setting in this strategy. To reduce the number of hyper-parameters, we specify a maximum bag size $R$, and set the bag size range to [$\frac{R+1}{2}$, R]. For example, if $R$ is set to 7, then for each bag, its size is randomly selected from 4 to 7. We also include a setting where $R$ is $\infty$, which corresponds to the case where all words fall into one global bag, as in the full word reordering model.

The results are shown in Table 1. Firstly, in the case of $R = \infty$, the model generally performs worse than those with local bags. This shows the effectiveness of keeping partial ordering information for pre-training, which may possibly reduce the discrepancies between pre-training and fine-tuning, matching our intuition of the local ordering strategy. Furthermore, when the bag size is too small as in the case of $R = 3$, the performances are also worse, possibly because the task becomes so simple that the model learns little in pre-training. Among the middle-ranged settings of $R$, which partially mask out information in suitable degrees, the results do not differ too much. In the following experiments, we fix $R$ to 7, which performs well overall.

### 3.2.2 Comparisons

We compare various pre-training strategies and show the results in Table 2. As split in this table, we arrange the models into three groups:

(1) The first group includes models without pre-trained encoders. "Random" gets random initialization, and "fastText" gets its word lookup table

| | Random | fastText | BiLM | MaskLM | LBag | Hybrid | BERT |
|---|---|---|---|---|---|---|---|
| 10k | $83.70_{\pm 0.36}$ | $86.00_{\pm 0.10}$ | $87.28_{\pm 0.16}$ | $87.96_{\pm 0.09}$ | $87.75_{\pm 0.13}$ | $88.27_{\pm 0.11}$ | $89.60_{\pm 0.10}$ |
| 5k | $80.75_{\pm 0.35}$ | $83.17_{\pm 0.24}$ | $86.16_{\pm 0.03}$ | $87.09_{\pm 0.10}$ | $86.70_{\pm 0.13}$ | $87.35_{\pm 0.10}$ | $88.47_{\pm 0.11}$ |
| 1k | $69.93_{\pm 0.32}$ | $72.84_{\pm 0.25}$ | $80.75_{\pm 0.03}$ | $82.65_{\pm 0.04}$ | $82.25_{\pm 0.07}$ | $83.28_{\pm 0.26}$ | $84.62_{\pm 0.28}$ |

Table 2: Comparisons of different pre-training strategies with the English dependency parsing task (LAS on development set, averaged over three runs). Each row represents different (target task) training sizes.

initialized from static fastText embeddings[3].
(2) The second group includes models whose encoders are pre-trained with the same settings on the 1M Wiki corpus. "BiLM" denotes Elmo-styled (Peters et al., 2018) Bidirectional LM (BiLM), where we train left-to-right and right-to-left language models with causality attention masks. "MaskLM" means the BERT-styled MaskLM, where 15% of the words are masked out and predicted. "LBag" denotes our Local-Bag based ordering strategy and "Hybrid" is the multi-task hybrid model trained with both ordering and MaskLM objectives.
(3) The third group only contains "BERT", which directly utilizes pre-trained BERT[4].

In the first group, where there are no pre-trained encoders, the performances drop drastically in low-resource cases. The pre-trained static word embeddings help in some way, but its degree of performance drop is very similar to the baseline: there are performance gaps of nearly 14 points between 10k and 1k training sizes. If we adopt pre-trained encoders, as in the second and third group, the performance clearly improves for all training sizes. Particularly, in the low-resource (1k) settings, the performance drops from the 10k settings are much smaller than those in the first group.

The more interesting comparisons are among those in the second group, where the settings are kept the same except for pre-training strategies. Firstly, BiLM performs worst in this group. The reason may be that BiLM contains unidirectional decoders, which cannot make full use of the inputs. The performance of our local ordering strategy (LBag) is very close to those of the MaskLM, with performance gaps of only 0.2 to 0.4 in LAS. Furthermore, if we combine the ordering and MaskLM objectives as in the Hybrid model, there can be further improvements. This suggests that local ordering pre-training may capture orthogonal information from MaskLM. Overall, the model performances in the second group do not differ too much, suggesting that the effectiveness of contextualized pre-training can be realized as long as the model is capable enough.

Unsurprisingly, BERT performs the best, possibly due to its larger model and training corpus. Nevertheless, if calculating the gaps between the second group and BERT, we can find that they are relatively consistent as training sizes get smaller. In contrast, the gaps between the first group and BERT obviously get larger in lower-resource settings. This again suggests the effectiveness of contextualized pre-training.

For the pre-trained models in the following experiments, we focus on three strategies: MaskLM, LBag and Hybrid, since they are the ones that we are most interested to compare.

### 3.3 Main Results

Figure 3 shows the main results on the test sets. The patterns are very similar to the development results. Pre-trained BERT obtains the best results, while our smaller pre-trained models lag behind by small gaps, which are relatively consistent across different training sizes. Those without pre-trained encoders mostly get worse results, especially in low-resource cases. For the parsing task, our local ordering strategy can get comparable results to those of MaskLM and overall there can be further improvements by combining the two objectives. For the other two sequence labeling tasks, the results are mixed, possibly because in these cases the lexical information may be more important, and the LM-styled pre-training may be better at capturing them. Nevertheless, our strategy still generally obtains comparable results to MaskLM.

### 3.4 Analysis

It is not surprising that contextualized pre-training can help structured prediction, since pre-trained encoders may have already captured structured patterns from unlabeled data. We perform detailed

[3]https://fasttext.cc/docs/en/pretrained-vectors.html

[4]We use `bert-base-multilingual-cased` in this work. Since there are various aspects (model size, pre-training data size, etc.) making our models not directly comparable to BERT, we include BERT results mainly as a reference of how much better we may possibly get with larger models and more pre-training data.

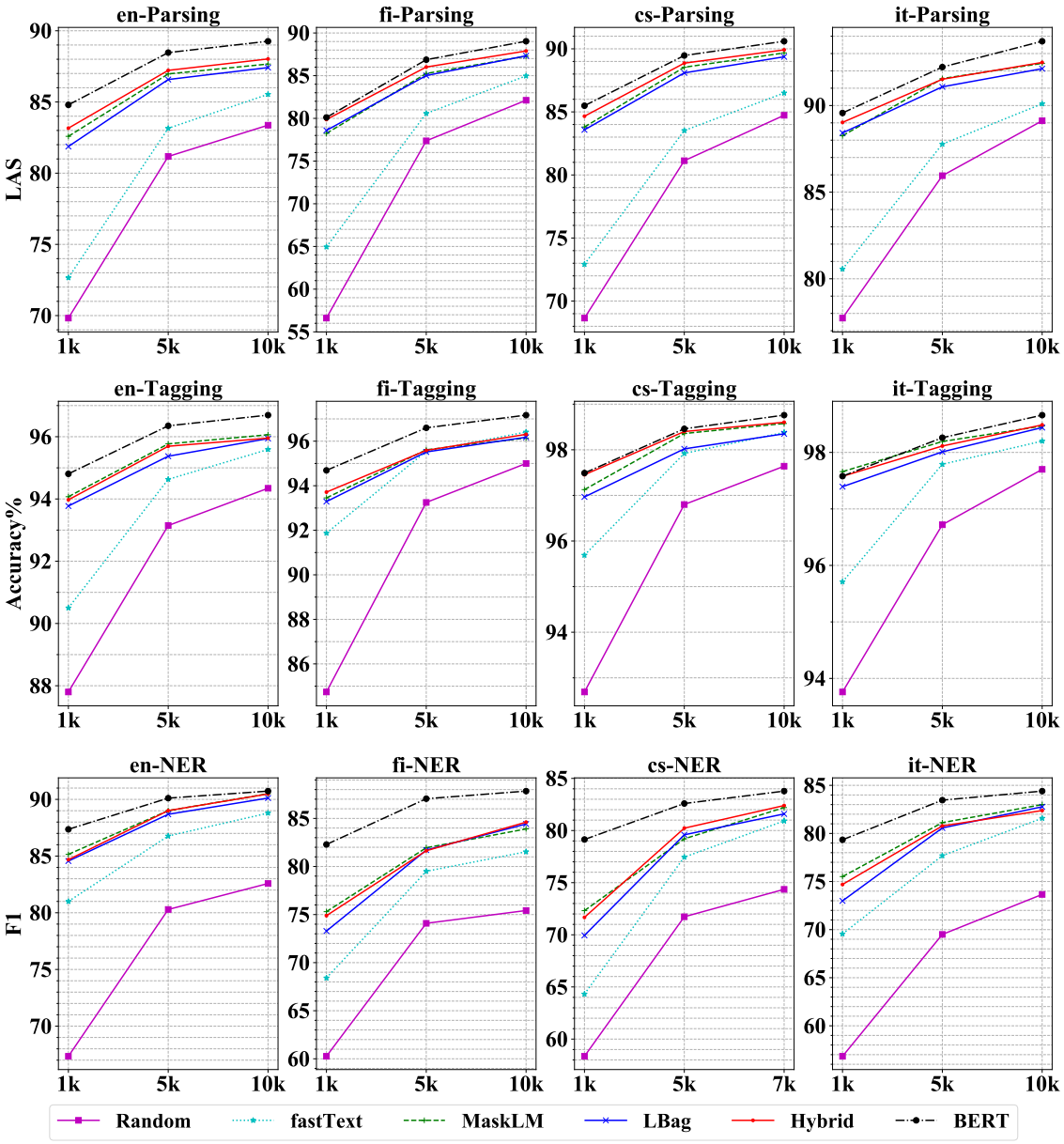


Figure 3: Test results for dependency parsing (LAS), POS tagging (Accuracy%) and NER (F1 score).

analysis to investigate in what aspects pre-training are helpful. We select low-resource dependency parsing (with 1k training size) as the analyzing task, since parsing is the most structurally complex task we explore and there may be more obvious patterns in low-resource scenarios. For error analysis of parsing, Kulmizev et al. (2019) provide detailed error breakdowns on various factors, along the lines of (McDonald and Nivre, 2007, 2011). In this work, we explore different aspects, especially focusing on the structured nature of the task.

### 3.4.1 On Word Frequencies

Since pre-training is performed on a much larger corpus than the task-specific training set, we would expect that pre-trained models perform better on out-of-vocabulary (OOV) and rare words, since they would be seen more often in pre-training.

To investigate this, we split the words of the development set into four bins according to their frequency ranking in the (target task) training vocabulary. Except for the OOV bin where words do not appear in training, the other three bins get the same number of running word counts.

Figure 4 shows a breakdown of the results. First, if comparing fastText against the Random baseline, we can find that overall, the most improvements come from low frequency and OOV words. For words with high and middle frequency, static em-

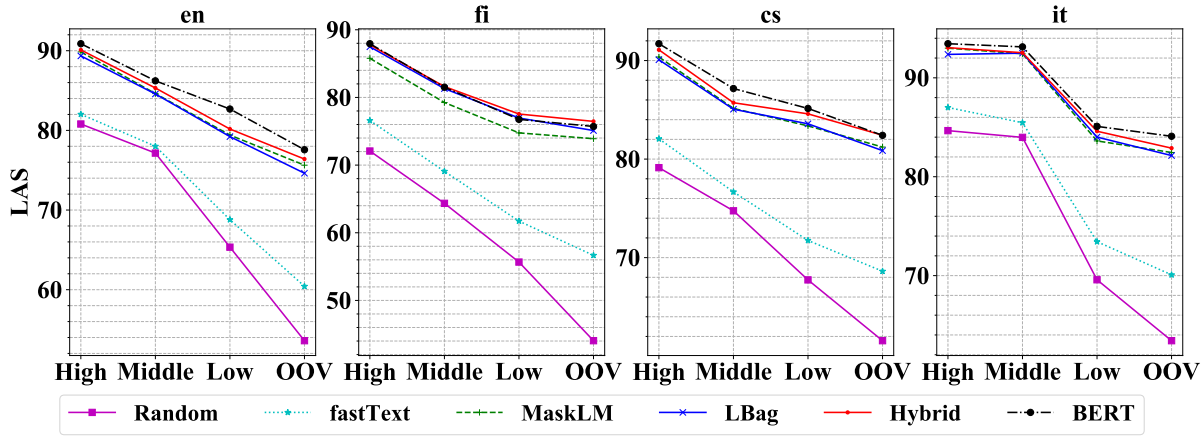


Figure 4: Performance breakdown of dependency parsing (LAS on development sets, trained with 1k sentences) on word frequencies. Non-OOV words are evenly divided into the first three bins according to frequency ranking in (target task) training vocabularies.

beddings provide less or sometimes even no obvious improvements. With pre-trained encoders, not only do the results on rare and OOV words get much better, but even high frequent words improve by a large margin. This suggests that the benefits of pre-training include not just that each individual word is known better, which may also be captured by static embeddings, but also that contextualized pre-training may be able to identify higher-order structured patterns.

When comparing the models with pre-trained encoders, the trends are very similar to the overall LAS scores. A slightly surprising phenomenon is that, although our models are trained on much less data than BERT, the performance gaps are still relatively consistent across different frequency bins. This may suggest that even for rare or OOV words, their contexts can be signals that are strong enough for syntax prediction.

### 3.4.2 On Higher-order Matches

A dependency tree is a collection of dependency edges, which are not individual but interact with each other, forming higher-order structures. To investigate how pre-trained encoders help predicting higher-order structures, we specify some frame patterns and calculate the higher-order matching accuracies. Here, we use "frame" to denote a collection of dependency edges which form a pre-defined pattern. Accuracy is calculated by counting how many times all the dependency edges in the specific frame are correctly predicted.

We investigate five frame patterns: 1) *pred*: all edges connecting a predicate and its core argument children, 2) *mwe*: all multi-word expression (MWE) edges connected to the head word of an MWE phrase, 3) *conj*: all edges related to a conjunction, 4) *expl*: an expletive edge and its core argument siblings, 5) *acl*: an adjectival clause modifier and all its core argument children. Please refer to the Appendix for examples and more detail about the extraction of these higher-order patterns.

Figure 5 shows the results. We can again observe that static word embeddings improve higher-order accuracies very limitedly, while pre-trained encoders give totally different stories. For the "pred" patterns, the trends are very similar to the overall LAS results, where LBag is slightly worse than MaskLM and Hybrid is better. The interesting cases are "mwe" and "conj", where LBag mostly performs better than MaskLM. The reason might be that these patterns are more fixed in aspects of word order, which may be captured better by ordering pre-training. For the last two types, the results are mixed for different languages. Nevertheless, the ordering pre-trained models can still achieve comparable or sometimes better results than MaskLM.

### 3.4.3 On Head Errors

Finally, we investigate a special error pattern in dependency parsing, for which Figure 6 shows an example. Here, all the predicted edges are wrong, but there seems to be only one head selection error: "Epic" is an apposition modifier of "movie", but the model picks "Epic" as the head, leading to all other errors. In constituency trees, an attachment error may lead to multiple wrong brackets (Kummerfeld et al., 2012). In contrast, in dependency trees, a

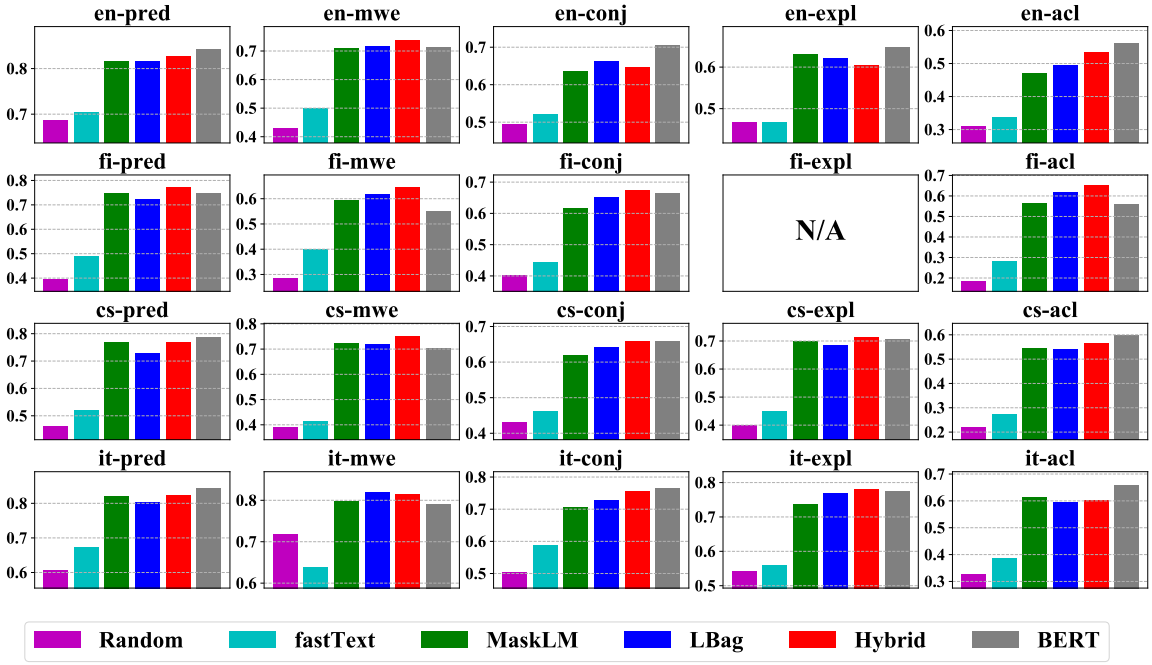


Figure 5: Comparisons of higher-order matching accuracies on dependency parsing (on development sets, with 1k training). There are no results for "fi-expl" since in the Finnish (TDT) Treebank we adopt, "expl" is not used.

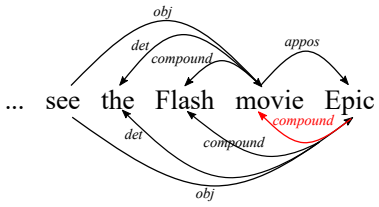


Figure 6: An example of head error. Here, the edges above the tokens are gold ones and the edges below are predictions. The red edge indicates the back edge, which is directly reversed in this case.

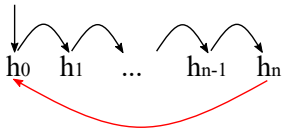


Figure 7: Illustration of multi-step back edge. Here, the edges above the tokens are gold ones (Notice that in actual sequence, the tokens do not necessarily appear in left-to-right order). The red edge below indicates a $n$-step back edge for the gold tree.

pure attachment error may influence no other edges, but head errors may lead to multiple related errors.

In the pattern of head errors, the predicted edge that forms a back edge in the original gold tree can usually be the signature. The prediction of a back edge indicates that a word is wrongly attached to one of its descendants in the gold tree. In addition to the wrongly predicted back edge itself, there must be at least another error, since loops are not allowed in trees. The example in Figure 6 shows a special case where the back edge is a directly reversed one, where the head and the modifier are reversely predicted. This type of 1-step back edges usually indicates local head errors, while there can be back edges involving multiple steps, which usually suggest more complex structured errors.

Figure 8 shows the results on back edges. Firstly, as the trends in previous analyses, the pre-trained models obviously predict fewer back edges and thus make fewer head errors, again suggesting structural improvements. Moreover, comparing the 1-step back-edge percentages, the pre-trained models also have higher rates, indicating that their head errors are more local. Further comparing different pre-training strategies, we can see that, except for Finnish, the MaskLM predicts fewer back edges and makes more local head errors (indicated by higher 1-step back edge percentages) than LBag. This suggests that, LM pre-training, which directly predicts lexical items, may be more sensitive to the information of head words.

We further investigate errors[5] that might be related with head errors. We adopt a relatively simple

[5]For simplicity, in this analysis, we ignore dependency labels and focus on unlabeled errors.

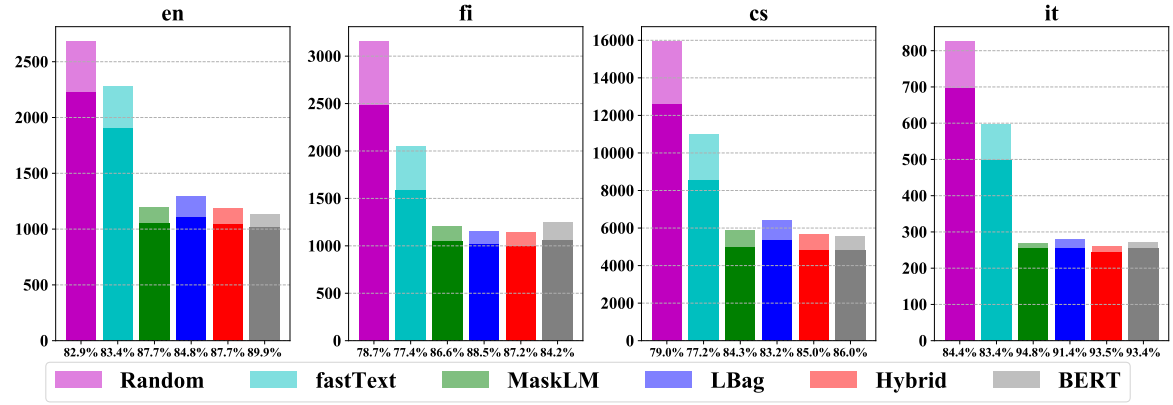


Figure 8: Results on back edges (on development sets, with 1k training). The light bars indicate the number of all back edges, while the darker and shaded parts represent the number of 1-step back edges. The numbers on the $x$-axis indicate the percentage of 1-step back edges (which indicate more local errors) among all back edges.

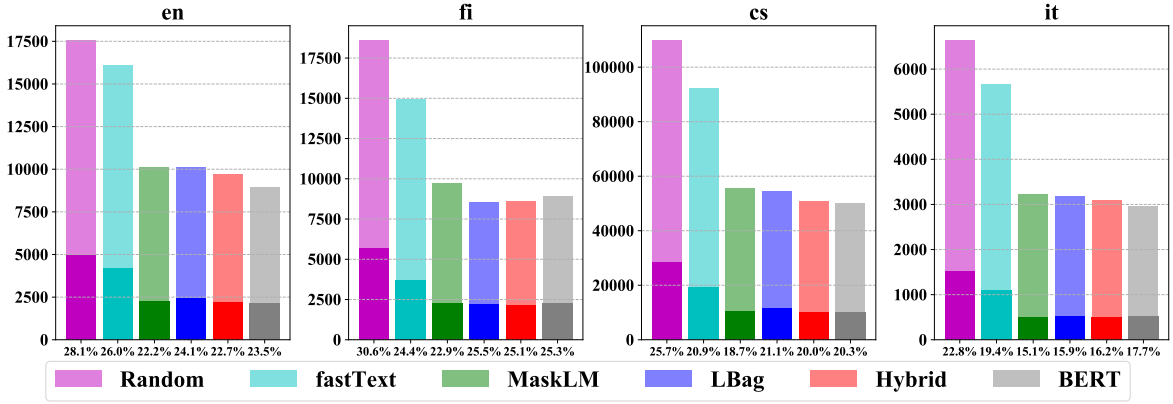


Figure 9: Results on head-error related errors (on development sets, with 1k training). The light bars indicate the number of total erroneous edges, while the darker and shaded parts represent the number of the ones that are related with head errors. The numbers on the $x$-axis indicate the relatedness rates: the percentage of head-error related erroneous edges among all erroneous edges.

strategy: first identify all back edges, and then include other erroneous edges that might be *related* with any head error. We use the diagram in Figure 7 to illustrate our criterion for *relatedness*. We mark three types of erroneous edges as head-error related: 1) the back edge itself ($h_n \rightarrow h_0$), 2) any wrongly predicted children of $h_n$ whose gold head should be one of $[h_0, h_1, ..., h_{n-1}]$, 3) any errors for the head prediction of the tokens $[h_0, h_1, ..., h_{n-1}]$. This criterion may miss or over-predict related errors, nevertheless we find it a reasonable approximation.

Figure 9 shows the results. First, as in Figure 8, the pre-trained models are less influenced by head errors, again suggesting structural improvements. Further comparing different pre-training strategies, generally MaskLM is less influenced by head errors, as shown by either lower head-error related error counts or relatedness rates.

## 4 Conclusion

In this work, we empirically explore an alternative pre-training strategy for contextualized encoders. Instead of training variants of language models, we adopt a local word ordering strategy, which segments the inputs into local bags of words, together with order-based word-selection objectives. Evaluated on typical structured prediction tasks, we show the effectiveness of this method. With further analysis on one typical structured task, we show that pre-trained encoders can bring improvements in a structured way. We hope this empirical work can shed some light and inspire future work on exploring how pre-trained contextualized encoders capture language structures.

## References


Wasi Ahmad, Zhisong Zhang, Xuezhe Ma, Eduard Hovy, Kai-Wei Chang, and Nanyun Peng. 2019. On difficulties of cross-lingual transfer with order differences: A case study on dependency parsing. 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 2440–2452, Minneapolis, Minnesota. Association for Computational Linguistics.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

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 *ICLR*.

Matthew S Dryer. 2007. Word order. *Language typology and syntactic description*, 1:61–131.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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.

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

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059, Jeju Island, Korea. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289.

Yijia Liu, Yue Zhang, Wanxiang Che, and Bing Qin. 2015. Transition-based syntactic linearization. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 113–122, Denver, Colorado. Association for Computational Linguistics.

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

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

Tahira Naseem, Regina Barzilay, and Amir Globerson. 2012. Selective sharing for multilingual dependency parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 629–637, Jeju Island, Korea. Association for Computational Linguistics.

Joakim Nivre, Mitchell Abrams, Željko Agić, and et al. 2019. Universal dependencies 2.4. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Teemu Ruokolainen, Pekka Kauppinen, Miikka Silfverberg, and Krister Lindén. 2019. A finnish news corpus for named entity recognition. *arXiv preprint arXiv:1908.04212*.

Allen Schmaltz, Alexander M. Rush, and Stuart Shieber. 2016. Word ordering without syntax. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2319–2324, Austin, Texas. Association for Computational Linguistics.

Magda Ševčíková, Zdeněk Žabokrtský, and Oldřich Krůza. 2007. Named entities in czech: Annotating data and developing NE tagger. In *Lecture Notes in Artificial Intelligence, Proceedings of the 10th International Conference on Text, Speech and Dialogue*, Lecture Notes in Computer Science, pages 188–195, Berlin / Heidelberg. Springer.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Manuela Speranza. 2009. The named entity recognition task at evalita 2009. In *EVALITA 2009*.

Oscar Täckström, Ryan McDonald, and Joakim Nivre. 2013. Target language adaptation of discriminative transfer parsers. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1061–1071, Atlanta, Georgia. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Wei Wang, Bin Bi, Ming Yan, Chen Wu, Jiangnan Xia, Zuyi Bao, Liwei Peng, and Luo Si. 2020. Structbert: Incorporating language structures into pre-training for deep language understanding. In *International Conference on Learning Representations*.

Wenhui Wang, Baobao Chang, and Mairgup Mansur. 2018. Improved dependency parsing using implicit word connections learned from unlabeled data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2857–2863, Brussels, Belgium. Association for Computational Linguistics.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design challenges and misconceptions in neural sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3879–3889, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

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

Yue Zhang and Stephen Clark. 2015. Discriminative syntax-based word ordering for text generation. *Computational Linguistics*, 41(3):503–538.

# Appendices

## A Detailed Experiment Settings

In this subsection, we describe the details of our experiment settings, mainly including datasets and hyper-parameter settings.

### A.1 Datasets

**Languages** In this work, we explore four languages from different language family subdivisions: English (Germanic), Finnish (Uralic), Czech (Slavic) and Italian (Romance). It may be interesting to see how the effects of pre-training are influenced by specific language characteristics, for example, the agglutination in Finnish and relatively free word order in Czech. We would like to include more languages in future work, especially those in different language families.

**Unlabeled data** For pre-training, we use the unlabeled data collected from the 2018-Fall Wiki-dump[6]. We extract raw texts using `WikiExtractor`[7] and then do sentence-splitting and tokenization using `UDPipe`[8]. Due to the limitation of computational resources, for each language, we sample 1M sentences whose length is between 5 and 80 for the purpose of pre-training. Our empirical results show that for the basic structured prediction tasks explored in this work, such relative small amount of unlabeled data is already enough to bring obvious improvements.

**Vocabularies** Except for models that directly use pre-trained BERT, all models regard words as the basic inputting and modeling units. Therefore, for pre-trained encoders, we collect vocabularies from the unlabeled corpus, filtering out rare words that appear less than five times. Table 4 summaries

[6]https://dumps.wikimedia.org
[7]https://github.com/attardi/wikiextractor
[8]http://ufal.mff.cuni.cz/udpipe

| Lang. | NER | | | Parsing/POS | | |
|---|---|---|---|---|---|---|
| | Train | Dev | Test | Train | Dev | Test |
| en | 15.0k/203.6k | 3.5k/51.4k | 3.7k/46.4k | 12.5k/204.6k | 2.0k/25.1k | 2.1k/25.1k |
| fi | 13.5k/180.2k | 1.0k/13.6k | 3.5k/46.4k | 12.2k/162.8k | 1.4k/18.3k | 1.6k/21.1k |
| cs | 7.2k/160.0k | 0.9k/20.0k | 0.9k/20.1k | 68.5k/1.2m | 9.3k/159.3k | 10.1k/173.9k |
| it | 10.0k/189.1k | 1.2k/23.4k | 4.1k/86.4k | 13.1k/276.0k | 0.6k/11.9k | 0.5k/10.4k |

Table 3: Statistics (#Sent./#Token) of the original Parsing/POS and NER datasets. In our experiments, we adopt the original development and test sets, but sample training sets with different sizes from the original training sets.

| Lang. | #Sent. | #Token | #Vocab | OOV% |
|---|---|---|---|---|
| en | 1M | 23.6M | 103k | 2.7% |
| fi | 1M | 14.1M | 177k | 10.9% |
| cs | 1M | 19.2M | 175k | 5.1% |
| it | 1M | 25.3M | 128k | 2.6% |

Table 4: Statistics of the unlabeled Wiki corpus for pre-training. For each language (Lang.), we sample 1M sentences ("#Sent."). "#Token" indicates the number of tokens (words), "#Vocab" denotes the vocabulary size after rare words filtering. The final column represents the out-of-vocabulary (OOV) rate over the 1M corpus.

the related statistics. We adopt word-level inputs mainly to follow the conventions of the target tasks explored in this work and to compare with baseline models without pre-trained encoders. It will be interesting to explore other input schemes (such as sub-words as in BERT) in future work, which is orthogonal to the main focus of this work.

**Target tasks** We explore three typical structured prediction tasks: dependency parsing, part-of-speech (POS) tagging and Named Entity Recognition (NER). For the tagging and parsing tasks, we utilize annotations from UDv2.4[9]. Specifically, we use the following treebanks: "English-EWT", "Finnish-TDT", "Czech-PDT" and "Italian-ISDT". For NER, we utilize various datasets, including CoNLL03[10] (Tjong Kim Sang and De Meulder, 2003) for English, Digitoday[11] (Ruokolainen et al., 2019) for Finnish, Czech Named Entity Corpus[12] (Ševčíková et al., 2007) for Czech and EVALITA 2009[13] (Speranza, 2009) for Italian. We only adopt simple settings for the NER tasks, specifically, ignoring nested annotations for Finnish NER and considering Supertypes for Czech NER. For it-NER, we take the first 10k sentences as training set and the rest 1.2k as development set. Table 3 lists the statistics of the original datasets.

We mainly follow the default dataset splittings, but for the training set, we explore three different training sizes by sampling 1k, 5k and 10k sentences[14]. These settings aim at exploring how pre-trained encoders can improve the structured learners in middle- and low-resource settings. For evaluations, POS tagging is evaluated by tagging accuracies and NER is evaluated by the standard F1 scores. For dependency parsing, we report first-level Labeled Attachment Scores (LAS) over all tokens including punctuations.

[9]http://hdl.handle.net/11234/1-2988
[10]https://www.clips.uantwerpen.be/conll2003/ner/
[11]https://github.com/mpsilfve/finer-data
[12]http://ufal.mff.cuni.cz/cnec
[13]http://www.evalita.it/2009/tasks/entity

| | | |
|---|---|---|
| Embeddings | $d_{emb}$ | 300 |
| | $d_{char}$ | 50 |
| | $d_{proj.}$ | 512 |
| Encoder | $N_{layer}$ | 6 |
| | $d_{model}$ | 512 |
| | $d_{ff}$ | 1024 |
| | position-encoding | Relative |
| PreTrain | optimizer | Adam |
| | learning-rate | 4e-4 |
| | warmup-steps | 5k |
| | total-steps | 200k |
| | batch-size | 480 |
| Decoding | POS | Enumeration |
| | Parsing | Graph-based(o1) |
| | NER | CRF |
| FineTune | optimizer | Adam |
| | learning-rate | 2e-4 |
| | total-epochs | 250 |
| | batch-size | 80 |

Table 5: Hyper-parameter settings of the model and training.

## A.2 Hyper-parameter Settings

Table 5 lists our main hyper-parameter settings.

**Encoder** Throughout our experiments, we adopt Transformer encoders with almost the same architecture. For the inputting parts of the encoder, we include representations of words and characters. Word representations are from a randomly initial-

[14]Only Czech-NER has less than 10k training sentences, therefore we take the whole 7k training set for the 10k setting.

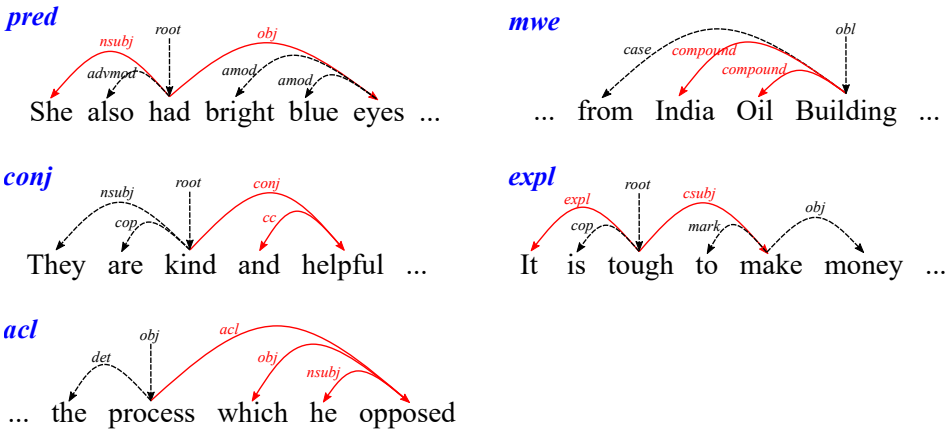


Figure 10: Examples of the higher-order frame patterns. The red solid edges are included, while others (black dotted ones) are not included.

ized word lookup table, while character representations are from a character-level CNN. Further, a linear layer is added to project these input features to the model dimension. Notice that there are no other input factors, since these are the ones that are directly available from the unlabeled corpus.

**Pre-training** We adopt almost identical pre-training schemes for all pre-training strategies, including optimizer, learning rate scheme and batch sizes. We employ one RTX 2080 Ti GPU for the pre-training. To fit the GPU memory, we split one mini-batch into multiple pieces and do gradient accumulation. The pre-training stage takes around 3 days for the MaskLM, LBag and Hybrid strategies, while the BiLM requires around 5 days.

**Decoders** For specific target tasks, we specify corresponding decoders. Since our main focus is not on decoders, we adopt the standard choices for these tasks. For dependency parsing, we adopt non-projective first-order (o1) graph-based decoder. For POS tagging, we do simple enumeration and select the maximally scored POS tag for each word. Since dependency parsing and POS tagging share the same datasets, we apply simple multi-task learning and train one joint model for these two tasks. For NER, we adopt a standard CRF layer and perform decoding with the Viterbi algorithm.

**Fine-tuning** For the training or fine-tuning of the target tasks, we also adopt similar schemes. In addition, the learning rate is decreased by a decay rate of 0.75 every 8 epochs when there are no improvements on the development set, which is also utilized for model selection. The training on target tasks usually takes several hours, depending on training sizes.

## B Details of Analysis

### B.1 Details on Higher-order Matches

We provide extraction details and examples for the five patterns we explore. We first define several groupings of dependency relations according to the UD documentation[15]:

- **PRED**={csubj, ccomp, xcomp, advcl, acl, root}. This set denotes dependency relations where the modifier is usually a clausal predicate.
- **CORE**={nsubj, obj, iobj, csubj, ccomp, xcomp}. This set includes the core arguments of predicates.
- **MWE**={fixed, flat, compound}. This set includes the Multi-Word Expression (MWE) dependency relations.

To extract the specified patterns, we go through each word $w$ and apply a filter to decide whether there is a frame which we are looking for. If there is, then we apply the extractor to obtain all the related dependency edges, forming the frame that we want to extract. Table 6 describes the extraction rules (the filters and extractors) and Figure 10 further provides some examples.

## C Extra Results

### C.1 Results on Development Sets

Figure 11 shows the results on development sets, whose patterns are similar to those of the test sets as shown in the main contents.

[15] https://universaldependencies.org/u/dep/index.html

| Pattern | Filter | Extractor |
|---|---|---|
| *pred* | lambda w: w.label in **PRED** | [c for c in w.children if c.label in **CORE**] |
| *mwe* | lambda w: any(c.label in **MWE** for c in w.children) | [c for c in w.children if c.label in **MWE**] |
| *conj* | lambda w: any(c.label==‘conj’ for c in w.children) | [c for c in w.children if c.label==‘conj’]+[g for g in w.grandchildren if g.label==‘cc’] |
| *expl* | lambda w: any(c.label==‘expl’ for c in w.children) | [c for c in w.children if c.label==‘expl’]+[c for c in w.children if c.label in **CORE**] |
| *acl* | lambda w: w.label==‘acl’ | [w]+[c for c in w.children if c.label in **CORE**] |

Table 6: Filter and extractor functions for the frame pattern extraction (in Python-styled pseudocode). We go through each word $w$ and apply the filter. If the filter returns True, then the extractor is applied to extract all related dependency edges, forming the desired frame.

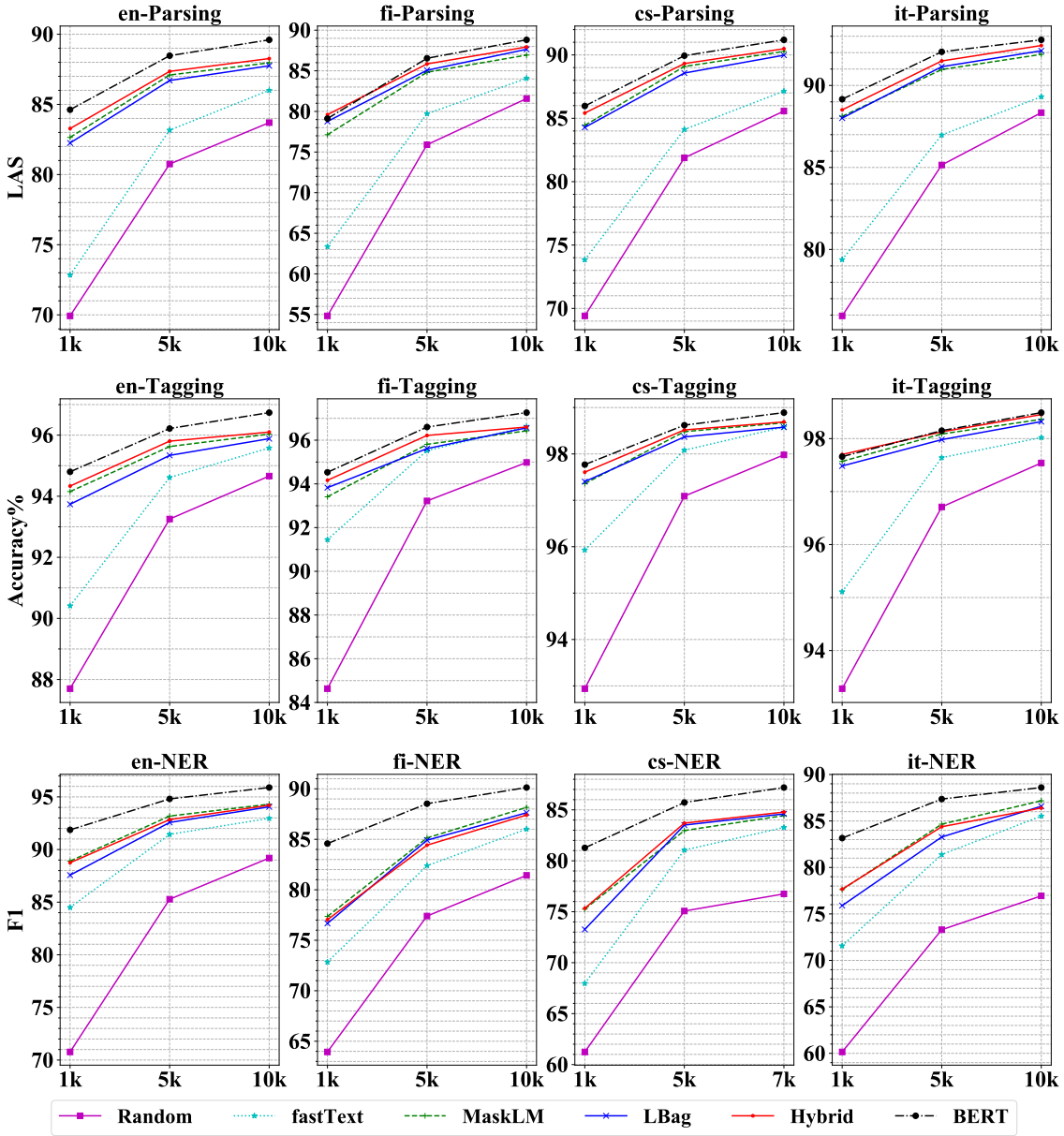


Figure 11: Development results for dependency parsing (LAS), POS tagging (Accuracy%) and NER (F1 score).