# Transition-based Parsing with Lighter Feed-Forward Networks

**David Vilares**
Universidade da Coruña
FASTPARSE Lab, LyS Group
Departamento de Computación
Campus de Elviña s/n, 15071
A Coruña, Spain
david.vilares@udc.es

**Carlos Gómez-Rodríguez**
Universidade da Coruña
FASTPARSE Lab, LyS Group
Departamento de Computación
Campus de Elviña s/n, 15071
A Coruña, Spain
carlos.gomez@udc.es

## Abstract

We explore whether it is possible to build lighter parsers, that are statistically equivalent to their corresponding standard version, for a wide set of languages showing different structures and morphologies. As testbed, we use the Universal Dependencies and transition-based dependency parsers trained on feed-forward networks. For these, most existing research assumes *de facto standard* embedded features and relies on pre-computation tricks to obtain speed-ups. We explore how these features and their size can be reduced and whether this translates into speed-ups with a negligible impact on accuracy. The experiments show that *grand-daughter* features can be removed for the majority of treebanks without a significant (negative or positive) LAS difference. They also show how the size of the embeddings can be notably reduced.

## 1 Introduction

Transition-based models have achieved significant improvements in the last decade (Nivre et al., 2007; Chen and Manning, 2014; Rasooli and Tetreault, 2015; Shi et al., 2017). Some of them already achieve a level of agreement similar to that of experts on English newswire texts (Berzak et al., 2016), although this does not generalize to other configurations (e.g. lower-resource languages). These higher levels of accuracy often come at higher computational costs (Andor et al., 2016) and lower bandwidths, which can be a disadvantage for scenarios where speed is more relevant than accuracy (Gómez-Rodríguez et al., 2017). Furthermore, running neural models on small devices for tasks such as part-of-speech tagging or word segmentation has become a matter of study (Botha et al., 2017), showing that small feed-forward networks are suitable for these challenges. However, for parsers that are trained using neural networks, little exploration has been done beyond the application of pre-computation tricks, initially intended for fast neural machine translation (Devlin et al., 2014), at a cost of affordable but larger memory.

**Contribution** We explore efficient and light dependency parsers for languages with a variety of structures and morphologies. We rely on neural feed-forward dependency parsers, since their architecture offers a competitive *accuracy vs bandwidth* ratio and they are also the inspiration for more complex parsers, which also rely on embedded features but previously processed by bidirectional LSTMs (Kiperwasser and Goldberg, 2016). In particular, we study if the *de facto standard* embedded features and their sizes can be reduced without having a significant impact on their accuracy. Building these models is of help in downstream applications of natural language processing, such as those running on small devices and also of interest for syntactic parsing itself, as it makes it possible to explore how the same configuration affects different languages. This study is made on the Universal Dependencies v2.1, a testbed that allows us to compare a variety of languages annotated following common guidelines. This also makes it possible to extract a robust and fair comparative analysis.

## 2 Related Work

### 2.1 Computational efficiency

The usefulness of dependency parsing is partially thanks to the efficiency of existing transition-based algorithms, although to the date it is an open question which algorithms suit certain languages better. To predict projective structures, a number of algorithms that run in $\mathcal{O}(n)$ with respect to the length of the input string are available. Broadly speaking, these parsers usually keep two structures: a stack (containing the words that are waiting for some arcs to be created) and a buffer (containing words awaiting to be processed). The ARC-STANDARD parser (Nivre, 2004) follows a strictly bottom-up strategy, where a word can only be assigned a head (and removed from the stack) once every daughter node has already been processed. The ARC-EAGER parser avoids this restriction by including a specific transition for the reduce action. The ARC-HYBRID algorithm (Kuhlmann et al., 2011) mixes characteristics of both algorithms. More recent algorithms, such as ARC-SWIFT, have focused on the ability to manage *non-local* transitions (Qi and Manning, 2017) to reduce the limitations of transition-based parsers with respect to graph-based ones (McDonald et al., 2005; Dozat and Manning, 2017), that consider a more global context. To manage non-projective structures, there are also different options available. The Covington (2001) algorithm runs in $\mathcal{O}(n^2)$ in the worst case, by comparing the word in the top of the buffer with a subset of the words that have been already processed, deciding whether or not to create a link with each of them. More efficient algorithms such as SWAP (Nivre, 2009) manage non-projectivity by learning when to swap pairs of words that are involved in a crossing arc, transforming it into a projective problem, with expected execution in linear time. The 2-PLANAR algorithm (Gómez-Rodríguez and Nivre, 2010) decomposes trees into at most two planar graphs, which can be used to implement a parser that runs in linear time. The NON-LOCAL COVINGTON algorithm (Fernández-González and Gómez-Rodríguez, 2018) combines the advantages of the wide coverage of the Covington (2001) algorithm with the non-local capabilities of the Qi and Manning (2017) transition system, running in quadratic time in the worst case.

### 2.2 Fast dependency parsing strategies

Despite the advances in transition-based algorithms, dependency parsing still is the bottleneck for many applications. This is due to collateral issues such as the time it takes to extract features and the multiple calls to the classifier that need to be made. In traditional dependency parsing systems, such as MaltParser (Nivre et al., 2007), the oracles are trained relying on machine learning algorithms, such as support vector machines, and hand-crafted (Huang et al., 2009; Zhang and Nivre, 2011) or automatically optimized sets of features (Ballesteros and Nivre, 2012). The goal usually is to maximize accuracy, which often comes at a cost of bandwidth. In this sense, efforts were made in order to obtain speed-ups. Using linear classifiers might lead to faster parsers, at a cost of accuracy and larger memory usage (Nivre and Hall, 2010). Bohnet (2010) illustrates that mapping the features into weights for a support vector machine is the major issue for the execution time and introduces a hash kernel approach to mitigate it. Volokh (2013) made efforts on optimizing the feature extraction time for the Covington (2001) algorithm, defining the concept of *static features*, which can be reused through different configuration steps. The concept itself does not imply a reduction in terms of efficiency, but it is often employed in conjunction with the reduction of *non-static features*, which causes a drop in accuracy.

In more modern parsers, the oracles are trained using feed-forward networks (Titov and Henderson, 2007; Chen and Manning, 2014; Straka et al., 2015) and sequential models (Kiperwasser and Goldberg, 2016). In this sense, to obtain significant speed improvements it is common to use the pre-computation trick from Devlin et al. (2014), initially intended for machine translation. Broadly speaking, they precompute the output of the hidden layer for each individual feature and each position in the input vector where they might occur, saving computation time during the test phase, with an affordable memory cost. Vacariu (2017) proposes an optimized parser and also includes a brief evaluation about reducing features that have a high cost of extraction, but the analysis is limited to English and three treebanks. However, little analysis has been made on determining if these features are relevant across a wide variety of languages that show different particularities. Our work is also in line with this line of research. In particular, we focus on feed-forward transition-based parsers, which already offer a very competitive accuracy vs bandwidth ratio. The models used in this work do not use any pre-computation trick, but it is worth pointing out that the insights of this paper could be used in conjunction with it, to obtain further bandwidth improvements.

## 3 Motivation

Transition-based dependency parsers whose oracles are trained using feed-forward neural networks have adopted as the *de facto standard* set of features the one proposed by Chen and Manning (2014) to parse the English and Chinese Penn Treebanks (Marcus et al., 1993; Xue et al., 2005).

We hypothesize this *de facto standard* set of features and the size of the embeddings used to represent them can be reduced for a wide variety of languages, obtaining significant speed-ups at a cost of a marginal impact on their performance. To test this hypothesis, we are performing an evaluation over the Universal Dependencies v2.1 (Nivre et al., 2017) a wide multilingual testbed to approximate relevant features over a wide variety of languages from different families.

## 4 Methods and Materials

This section describes the parsing algorithms (§4.1), the architecture of the feed-forward network (§4.2) and the treebanks (§4.3).

### 4.1 Transition-based algorithms

Let $w = [w_1, w_2, ..., w_{|w|}]$ be an input sentence, a *dependency tree* for $w$ is an edge-labeled directed tree $T = (V, A)$ where $V = \{0, 1, 2, \dots, |w|\}$ is the set of nodes and $A = V \times D \times V$ is the set of labeled arcs. Each arc $a \in A$, of the form $(i, d, j)$, corresponds to a syntactic *dependency* between the words $w_i$ and $w_j$; where $i$ is the index of the *head* word, $j$ is the index of the *child* word and $d$ is the *dependency type* representing the kind of syntactic relation between them. Each transition configuration is represented as a 3-tuple $c = (\sigma, \beta, A)$ where:

- $\sigma$ is a stack that contains the words that are awaiting for remaining arcs to be created. In $\sigma|i$, $i$ represents the first word of the stack.
- $\beta$ is a buffer structure containing the words that still have not been processed (awaiting to be moved to $\sigma$. In $i|\beta$, $i$ denotes the first word of the buffer.
- $A$ is the set of arcs that have been created.

We rely on two transition-based algorithms: the stack-based ARC-STANDARD (Nivre, 2008) algorithm for projective parsing and its corresponding version with the SWAP operation (Nivre, 2009) to manage non-projective structures. The election of the algorithms is based on their computational complexity as both run in $\mathcal{O}(n)$ empirically. The set of transitions is shown in Table 1. Let $c_i = ([0], \beta, \{\})$ be an initial configuration, the parser will apply transitions until a final configuration $c_f = ([0], [], A)$ is reached.

| | **Transition** | **Step t** | **Step *t+1*** |
|---|---|---|---|
| STANDARD | LEFT-ARC$_l$ | $(\sigma\|i\|j, \beta, A)$ | $(\sigma\|j, \beta, A \cup (j, l, i))$ |
| (projective) | RIGHT-ARC$_l$ | $(\sigma\|i\|j, \beta, A)$ | $(\sigma\|i, \beta, A \cup (i, l, j))$ |
| | SHIFT | $(\sigma, i\|\beta, A)$ | $(\sigma\|i, \beta, A)$ |
| SWAP | SWAP | $(\sigma\|i\|j, \beta, A)$ | $(\sigma\|j, i\|\beta, A)$ |
| (above +) | | | |

Table 1: Transitions for the projective version of the stack-based ARC-STANDARD algorithm and its non-projective version including the SWAP operation

### 4.2 Feed-forward neural network

We reproduce the Chen and Manning (2014) architecture and more in particular the Straka et al. (2015) version. These two parsers report the fastest architectures for transition-based dependency parsing (using the pre-computation trick from Devlin et al. (2014)), and obtain results close to the state of the art. Let $\text{MLP}_\theta(\mathbf{v})$ be an abstraction of our multilayered perceptron parametrized by $\theta$, the output for an input $\mathbf{v}$ (in this paper, a concatenation of embeddings, as described in §5) is computed as:

$$\text{MLP}_\theta(\mathbf{v}) = \mathit{softmax}(\mathbf{W_2} \cdot \mathit{relu}(\mathbf{W_1} \cdot \mathbf{v} + \mathbf{b_1}) + \mathbf{b_2}) \tag{1}$$

where $\mathbf{W_i}$ and $\mathbf{b_i}$ are the weights and bias tensors to be learned at the $i$th layer and $\mathit{softmax}$ and $\mathit{relu}$ correspond to the activation functions in their standard form.

### 4.3 Universal Dependencies v2.1

Universal dependencies (UD) v2.1 (Nivre et al., 2017) is a set of 101 dependency treebanks for up to 60 different languages. They are labeled in the CoNLLU format, heavily inspired in the CoNLL format (Buchholz and Marsi, 2006). For each word in a sentence there is available the following information: ID, WORD, LEMMA, UPOSTAG (universal postag, available for all languages), XPOSTAG (language-specific postag, available for some languages), FEATS (additional morphosyntactic information, available for some languages), HEAD, DEPREL and other optional columns with additional information.

In this paper, we are only considering experiments on the *unsuffixed* treebanks (where UD_English is an unsuffixed treebank and UD_English-PUD is a suffixed treebank). The motivation owes to practical issues and legibility of tables and discussions.

## 5 Experiments

We followed the training configuration proposed by Straka et al. (2015). All models where trained using mini-batches (size=10) and stochastic gradient descent (SGD) with exponential decay ($lr = 0.02$, decay computed as $lr \times e^{-0.2 \times epoch}$). Dropout was set to 50%. With our implementation dropout was observed to work better than regularization with less effort in terms of tuning. We used internal embeddings, initialized according to a Glorot uniform (Glorot and Bengio, 2010), which are learned together with the oracle during the training phase. In the experiments we use no external embeddings, following the same criteria as Straka et al. (2015). The aim was to evaluate all parsers under a homogeneous configuration, and high-quality external embeddings may be difficult to obtain for some languages.

The experiments explore two paths: (1) is it possible to reduce the number of features without a significant loss in terms of accuracy? and (2) is it possible to reduce the size of the embeddings representing those features, also without causing significant loss in terms of accuracy? To evaluate this, we used as baseline the following configuration.

### 5.1 Baseline configuration

This configuration reproduces that of Straka et al. (2015) which is basically a version of the Chen and Manning (2014) parser whose features were specifically adapted to the UD treebanks:

**De facto standard features** The initial set of features, which we call the *de facto standard* features, is composed of: FORM, UPOSTAG and FEATS for the first 3 words in $\beta$ and the first 3 words of $\sigma$. The FORM, UPOSTAG, FEATS and DEPREL[1] for the 2 leftmost and rightmost children of the first 2 words in $\sigma$. And the FORM, UPOSTAG, FEATS and DEPREL of the leftmost of the leftmost and rightmost of the rightmost children of the first 2 words in $\sigma$. This makes a total of 18 elements and 66 different features. In the case of UD treebanks, it is worth noting that for some languages the FEATS features are not available. We thought of two strategies in this situation: (1) not to consider any FEATS vector as input or (2) assume that a dummy input vector is given to represent the FEATS of an element of the tree. The former would be more realistic in a real environment, but we believe the latter offers a fairer comparison of speeds and memory costs, as the input vector is homogeneous across all languages. Thus, this is the option we have implemented. The dummy vector is expected to be given no relevance by the neural network during the training phase. We also rely on gold UPOSTAGs and FEATS to measure the impact of the reduced features and their reduced size in an isolated environment.[2]

**Size of the embeddings** The embedding size for the FORM features is set to 50 and for the UPOSTAG, FEATS and DEPREL features it is set to 20. Given an input configuration, the final dimension of the input vector is 1860: 540 dimensions from directly accessible nodes in $\sigma$ and $\beta$, 880 dimensions corresponding to daughter nodes and 440 dimensions corresponding to grand-daughter nodes.

**Metrics** We use LAS (Labeled Attachment Score) to measure the performance. To determine whether the gain or loss with respect to the *de facto standard* features is significant or not, we used Bikel's randomized parsing evaluation comparator ($p < 0.05$), a stratified shuffling significance test. The null hypothesis is that the two outputs are produced by equivalent models and so the scores are equally likely. To refute it, it first measures the difference obtained for a metric by the two models. Then, it shuffles scores of individual sentences between the two models and recomputes the evaluation metrics, measuring if the new difference is smaller than the original one, which is an indicator that the outputs are significantly different. *Thousands of tokens parsed per second* is the metric used to compare the speed between different feature sets. To diminish the impact of running time outliers, this is averaged across five runs.

**Hardware** All models were run on the test set on a single thread on a Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz.

[1]Once it has been assigned

[2]The use of predicted PoS-tags and/or tokenization would make harder to measure which is the actual impact of using different features and size of embeddings.

**No precomputation trick** All the parsers proposed in this work do not use the precomputation trick from Devlin et al. (2014). There is no major reason for this, beyond measuring the impact of the strategies in a simple scenario. We would like to remark that the speed-ups obtained here by reducing the number of features could also be applied to the parsers implementing this precomputation trick, in the sense that the feature extraction time is lower. No time will be further gained in terms of computation of the hidden activation values. However, in this context, at least in the case of the Chen and Manning (2014) parser, the pre-computation trick is only applied to the 10 000 most common words. The experiments here proposed are also useful to save memory resources, even if the trick is used.

## 5.2 Reducing the number of features

Table 2 shows the impact of ignoring features that have a larger cost of extraction, i.e., daughter and grand-daughter nodes, for both the ARC-STANDARD and SWAP algorithms. It compares three sets of features in terms of performance and speed: (1) *de facto standard* features, (2) *no grand-daughter* (NO-GD) features (excluding every leftmost of leftmost and rightmost of rightmost feature) and (3) *no daughter* (NO-GD/D) features (excluding every daughter and grand-daughter feature from nodes of $\sigma$).

**Impact of using the NO-GD feature set** The results show that these features can be removed without causing a significant difference in most of the cases. In the case of the ARC-STANDARD algorithm, for 47 out of 52 treebanks there is no significant accuracy loss with respect to the *de facto standard* features. In fact, for 22 treebanks there was a gain with respect to the original set of features, from which 5 of them were statistically significant. With respect to SWAP, we observe similar tendencies. For 38 out of 52 treebanks there is no loss (or the loss is again not statistically significant). There is however a larger number of differences that are statistically significant, both gains (11) and losses (13). On average, the ARC-STANDARD models trained with these features lost 0.1 LAS points with respect to the original models, while the average speed-up was $\sim$23%. The models trained with SWAP gained instead 0.15 points and the bandwidth increased by $\sim$28%.

**Impact of the NO-GD/D features** As expected, the results show that removing *daughter* features in conjunction with *grand-daughter* causes a big drop in performance for the vast majority of cases (most of them statistically significant). Due to this issue and despite the (also expected) larger speed-ups, we are not considering this set of features for the next section.

## 5.3 Reducing the embedding size of the selected features

We now explore whether by reducing the size of the embeddings for the FORM, POSTAG, FEATS and DEPREL features the models can produce better bandwidths without suffering a lack of accuracy. We run separate experiments for the ARC-STANDARD and SWAP algorithms, using as the starting point the NO-GD feature set, which had a negligible impact on accuracy, as tested in Table 2. Table 3 summarizes the experiments when reducing the size of each embedding from 10% to 50%, at a step size of 10 percentage points, for the ARC-STANDARD. The results include information indicating whether the difference in performance is statistically significant from that obtained by the *de facto* standard set. In general terms, reducing the size of the embeddings causes a small but constant drop in the performance. However, for the vast majority of languages this drop is not statistically significant. Reducing the size of the embeddings by a factor of 0.2 was the configuration with the minimum number of significant losses (6), and reducing them by a factor of 0.5 the one with the largest (14). On average, the lightest models lost 0.45 LAS points to obtain an speed-up of $\sim$40%. Similar tendencies were observed in the case of the non-projective algorithm, whose results reducing the size of the embeddings by a factor of 0.1 and 0.5 can be found in Table 4.

## 5.4 Discussion

Different deep learning frameworks to build neural networks might present differences and implementation details that might cause the speed obtained empirically to differ from theoretical expectations.

From a theoretical point of view, both tested approaches (§5.2, 5.3) should have a similar impact, as their use directly affects the size of the input to the neural network. The smaller the input size, the lighter and faster parsers are obtained. As a side note, with respect to the case of reducing the

| Treebank | ARC-STANDARD |  |  |  |  |  | SWAP |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | STANDARD |  | NO-GD/D |  | NO-GD |  | STANDARD |  | NO-GD/D |  | NO-GD |  |
|  | LAS | kt/s | LAS | kt/s | LAS | kt/s | LAS | kt/s | LAS | kt/s | LAS | kt/s |
| Afrikaans | 82.72 | 3.3 | 71.67$^{--}$ | 8.4 | 82.42$^{-}$ | 4.0 | 82.55 | 3.0 | 70.59$^{--}$ | 7.7 | 82.96$^{+}$ | 3.8 |
| Anc Greek | 56.85 | 3.5 | 50.27$^{--}$ | 8.9 | 56.63$^{-}$ | 4.3 | 58.97 | 2.9 | 51.36$^{--}$ | 7.8 | 58.48$^{-}$ | 3.8 |
| Arabic | 77.46 | 3.1 | 70.69$^{--}$ | 7.7 | 77.87$^{+}$ | 3.7 | 76.77 | 3.0 | 70.4$^{--}$ | 7.4 | 77.5$^{++}$ | 3.7 |
| Basque | 74.26 | 3.6 | 68.13$^{--}$ | 9.0 | 74.05$^{-}$ | 4.4 | 73.98 | 3.2 | 67.31$^{--}$ | 8.4 | 72.44$^{--}$ | 3.8 |
| Belarusian | 70.12 | 2.4 | 61.43$^{--}$ | 5.8 | 67.73$^{-}$ | 2.9 | 69.75 | 2.4 | 62.81$^{--}$ | 5.6 | 70.33$^{+}$ | 2.9 |
| Bulgarian | 88.42 | 3.4 | 77.88$^{--}$ | 8.4 | 88.24$^{-}$ | 4.1 | 87.95 | 3.2 | 77.41$^{--}$ | 8.2 | 87.98$^{+}$ | 4.2 |
| Catalan | 87.57 | 3.4 | 76.79$^{--}$ | 8.9 | 87.5$^{-}$ | 4.2 | 87.01 | 3.1 | 76.48$^{--}$ | 8.4 | 87.06$^{+}$ | 3.9 |
| Chinese | 79.23 | 3.2 | 64.66$^{--}$ | 8.3 | 79.2$^{-}$ | 4.0 | 78.26 | 3.2 | 62.74$^{--}$ | 8.0 | 78.8$^{+}$ | 4.0 |
| Coptic | 78.68 | 1.9 | 71.32$^{--}$ | 4.9 | 76.0$^{--}$ | 2.3 | 77.25 | 1.3 | 70.08$^{--}$ | 3.1 | 77.44$^{+}$ | 1.5 |
| Croatian | 81.23 | 3.2 | 72.11$^{--}$ | 7.8 | 81.4$^{+}$ | 3.8 | 80.63 | 3.0 | 70.54$^{--}$ | 7.6 | 81.39$^{++}$ | 3.6 |
| Czech | 85.74 | 3.5 | 78.1$^{--}$ | 8.3 | 86.09$^{++}$ | 4.2 | 85.55 | 3.4 | 77.9$^{--}$ | 7.9 | 85.42$^{-}$ | 4.2 |
| Danish | 80.93 | 3.1 | 70.54$^{--}$ | 7.3 | 81.28$^{+}$ | 3.7 | 79.79 | 2.9 | 65.4$^{--}$ | 7.1 | 79.55$^{-}$ | 3.6 |
| Dutch | 78.67 | 3.3 | 66.82$^{--}$ | 8.4 | 79.41$^{+}$ | 4.1 | 77.02 | 3.1 | 64.83$^{--}$ | 7.8 | 78.15$^{++}$ | 3.8 |
| English | 84.16 | 3.6 | 72.68$^{--}$ | 8.8 | 84.42$^{+}$ | 4.4 | 83.19 | 3.6 | 72.76$^{--}$ | 8.7 | 84.09$^{++}$ | 4.4 |
| Estonian | 81.57 | 3.1 | 72.63$^{--}$ | 7.6 | 81.74$^{+}$ | 3.8 | 80.65 | 2.9 | 72.68$^{--}$ | 6.6 | 81.06$^{+}$ | 3.7 |
| Finnish | 81.25 | 3.3 | 69.08$^{--}$ | 8.0 | 82.08$^{++}$ | 4.1 | 81.47 | 3.3 | 69.36$^{--}$ | 7.8 | 80.4$^{--}$ | 3.9 |
| French | 84.65 | 3.0 | 73.15$^{--}$ | 7.2 | 84.83$^{+}$ | 3.5 | 83.54 | 2.7 | 72.31$^{--}$ | 6.8 | 83.27$^{-}$ | 3.5 |
| Galician | 80.51 | 3.5 | 68.82$^{--}$ | 8.5 | 80.29$^{-}$ | 4.2 | 79.85 | 3.3 | 69.25$^{--}$ | 8.3 | 80.01$^{+}$ | 4.2 |
| German | 79.86 | 3.3 | 72.3$^{--}$ | 8.1 | 79.67$^{-}$ | 4.1 | 78.52 | 3.1 | 70.69$^{--}$ | 8.2 | 77.65$^{--}$ | 3.4 |
| Gothic | 74.57 | 3.2 | 66.19$^{--}$ | 8.0 | 74.18$^{-}$ | 3.8 | 72.92 | 2.7 | 65.92$^{--}$ | 7.6 | 73.19$^{+}$ | 3.4 |
| Greek | 84.71 | 3.1 | 77.53$^{--}$ | 7.8 | 85.07$^{+}$ | 3.7 | 84.13 | 3.0 | 76.54$^{--}$ | 7.5 | 84.21$^{+}$ | 3.8 |
| Hebrew | 82.16 | 3.2 | 67.9$^{--}$ | 7.9 | 82.63$^{+}$ | 3.8 | 81.87 | 3.1 | 68.77$^{--}$ | 7.6 | 82.06$^{+}$ | 4.0 |
| Hindi | 90.8 | 3.5 | 81.8$^{--}$ | 8.8 | 90.69$^{-}$ | 4.3 | 90.46 | 3.2 | 80.5$^{--}$ | 8.3 | 90.01$^{--}$ | 4.0 |
| Hungarian | 73.34 | 3.1 | 66.39$^{--}$ | 7.0 | 73.14$^{-}$ | 3.5 | 72.33 | 2.9 | 66.88$^{--}$ | 6.8 | 73.36$^{++}$ | 3.1 |
| Indonesian | 79.47 | 2.9 | 62.58$^{--}$ | 7.3 | 79.3$^{-}$ | 3.5 | 78.86 | 2.8 | 63.62$^{--}$ | 7.1 | 79.07$^{+}$ | 3.5 |
| Irish | 60.07 | 2.8 | 52.66$^{--}$ | 7.4 | 59.94$^{-}$ | 3.5 | 61.82 | 2.8 | 54.28$^{--}$ | 7.0 | 60.3$^{--}$ | 3.4 |
| Italian | 89.21 | 2.9 | 78.16$^{--}$ | 7.2 | 89.33$^{+}$ | 3.3 | 88.34 | 2.9 | 78.13$^{--}$ | 7.0 | 88.81$^{++}$ | 3.6 |
| Japanese | 92.16 | 3.3 | 74.2$^{--}$ | 8.6 | 92.19$^{+}$ | 4.1 | 91.95 | 3.3 | 74.09$^{--}$ | 9.0 | 91.91$^{-}$ | 4.2 |
| Kazakh | 22.78 | 3.4 | 16.1$^{--}$ | 8.9 | 27.41$^{++}$ | 4.3 | 29.32 | 3.4 | 20.47$^{--}$ | 8.7 | 33.0$^{++}$ | 4.1 |
| Korean | 60.84 | 3.5 | 46.27$^{--}$ | 8.9 | 60.13$^{-}$ | 4.4 | 60.46 | 3.5 | 47.63$^{--}$ | 8.8 | 57.98$^{--}$ | 4.3 |
| Latin | 43.31 | 3.3 | 41.33$^{--}$ | 7.8 | 44.16$^{++}$ | 3.9 | 47.11 | 2.6 | 45.33$^{--}$ | 7.1 | 46.54$^{-}$ | 3.4 |
| Latvian | 75.14 | 3.4 | 64.88$^{--}$ | 8.3 | 75.36$^{+}$ | 4.1 | 74.73 | 3.3 | 65.11$^{--}$ | 8.1 | 75.55$^{++}$ | 4.1 |
| Lithuanian | 42.74 | 1.5 | 40.75$^{-}$ | 3.5 | 42.64$^{-}$ | 1.8 | 46.79 | 1.4 | 38.21$^{--}$ | 3.4 | 43.4$^{--}$ | 1.8 |
| Marathi | 66.02 | 1.7 | 61.89$^{--}$ | 4.4 | 62.14$^{--}$ | 2.1 | 65.05 | 1.7 | 59.95$^{--}$ | 4.2 | 66.26$^{+}$ | 2.2 |
| Old Church Slavonic | 78.33 | 3.3 | 71.07$^{--}$ | 8.4 | 78.97$^{++}$ | 4.1 | 79.48 | 3.0 | 69.65$^{--}$ | 8.3 | 79.76$^{+}$ | 3.8 |
| Persian | 82.1 | 3.1 | 66.16$^{--}$ | 7.6 | 81.1$^{--}$ | 3.8 | 80.79 | 3.0 | 65.35$^{--}$ | 7.3 | 81.08$^{+}$ | 3.8 |
| Polish | 90.92 | 3.5 | 83.03$^{--}$ | 8.8 | 90.9$^{-}$ | 4.3 | 90.49 | 3.5 | 83.46$^{--}$ | 8.7 | 90.29$^{-}$ | 4.4 |
| Portuguese | 86.27 | 3.1 | 74.09$^{--}$ | 8.3 | 86.58$^{+}$ | 4.0 | 83.87 | 2.7 | 71.84$^{--}$ | 7.0 | 85.23$^{++}$ | 3.6 |
| Romanian | 82.12 | 3.3 | 68.89$^{--}$ | 8.2 | 81.73$^{-}$ | 4.1 | 80.92 | 3.3 | 68.82$^{--}$ | 7.9 | 81.05$^{+}$ | 4.1 |
| Russian | 79.47 | 3.0 | 70.0$^{--}$ | 7.5 | 79.14$^{-}$ | 3.7 | 78.55 | 2.9 | 68.63$^{--}$ | 7.4 | 77.62$^{--}$ | 3.7 |
| Serbian | 84.9 | 3.3 | 76.57$^{--}$ | 8.4 | 85.17$^{+}$ | 4.0 | 85.8 | 3.2 | 76.04$^{--}$ | 7.7 | 85.64$^{-}$ | 4.0 |
| Slovak | 85.54 | 3.2 | 76.44$^{--}$ | 7.8 | 85.45$^{-}$ | 4.0 | 84.96 | 3.2 | 77.72$^{--}$ | 8.2 | 84.25$^{--}$ | 3.9 |
| Slovenian | 88.73 | 3.2 | 78.06$^{--}$ | 7.3 | 88.74$^{+}$ | 3.8 | 89.35 | 3.0 | 77.67$^{--}$ | 7.1 | 88.31$^{--}$ | 3.7 |
| Spanish | 85.16 | 2.8 | 71.78$^{--}$ | 6.7 | 83.75$^{--}$ | 3.5 | 84.32 | 2.7 | 71.78$^{--}$ | 7.2 | 82.96$^{--}$ | 3.5 |
| Swedish | 83.73 | 3.4 | 71.21$^{--}$ | 8.6 | 83.63$^{-}$ | 4.2 | 84.37 | 3.4 | 69.96$^{--}$ | 8.5 | 83.78$^{--}$ | 4.3 |
| (Sw) Sign Language | 23.4 | 1.1 | 25.53$^{+}$ | 2.5 | 22.7$^{-}$ | 1.3 | 10.64 | 0.9 | 23.05$^{++}$ | 2.4 | 21.99$^{++}$ | 1.2 |
| Tamil | 69.18 | 2.0 | 66.47$^{--}$ | 4.8 | 69.58$^{+}$ | 2.4 | 71.04 | 1.7 | 67.77$^{--}$ | 4.7 | 71.09$^{+}$ | 2.4 |
| Telugu | 75.17 | 1.5 | 74.76$^{-}$ | 3.1 | 74.48$^{-}$ | 1.6 | 74.2 | 1.4 | 75.45$^{+}$ | 3.0 | 75.03$^{++}$ | 1.6 |
| Turkish | 59.51 | 3.2 | 53.47$^{--}$ | 7.4 | 59.29$^{-}$ | 3.9 | 60.32 | 3.0 | 53.14$^{--}$ | 7.7 | 59.35$^{-}$ | 3.8 |
| Ukrainian | 81.29 | 3.2 | 71.95$^{--}$ | 7.9 | 81.71$^{+}$ | 3.8 | 81.8 | 3.0 | 69.82$^{--}$ | 7.6 | 81.19$^{--}$ | 3.9 |
| Urdu | 83.12 | 3.3 | 71.84$^{--}$ | 8.5 | 83.03$^{-}$ | 4.1 | 81.29 | 2.9 | 69.42$^{--}$ | 7.9 | 80.93$^{-}$ | 3.4 |
| Vietnamese | 64.73 | 3.4 | 54.71$^{--}$ | 9.0 | 64.39$^{-}$ | 4.3 | 64.35 | 3.5 | 55.51$^{--}$ | 8.8 | 63.71$^{-}$ | 4.3 |
| AVERAGE | 75.67 | 3.0 | 66.42 | 7.5 | 75.57 | 3.7 | 75.29 | 2.8 | 66.07 | 7.2 | 75.44 | 3.6 |

Table 2: Performance for the (1) de facto standard, (2) NO-GD/D and (3) NO-GD set of features, when used to train oracles with the ARC-STANDARD and SWAP algorithms. Red cells indicate a significant loss (- -) with respect to the baseline, the yellow ones a non-significant gain(+)/loss (-) and the green ones a significant gain (++).

| | ARC-STANDARD | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | STANDARD | | NO GD | | | | | | | | | |
| | | | size -10% | | size -20% | | size -30% | | size -40% | | size -50% | |
| **Treebank** | **LAS** | **kt/s** | **LAS** | **kt/s** | **LAS** | **kt/s** | **LAS** | **kt/s** | **LAS** | **kt/s** | **LAS** | **kt/s** |
| Afrikaans | 82.72 | 3.3 | $82.66^-$ | 4.1 | $82.64^-$ | 4.4 | 82.72 | 4.3 | $82.5^-$ | 4.6 | $83.11^+$ | 4.5 |
| Anc Greek | 56.85 | 3.5 | $57.03^+$ | 4.4 | $56.52^-$ | 4.6 | $56.56^-$ | 4.6 | $56.24^{--}$ | 4.8 | $56.87^+$ | 4.8 |
| Arabic | 77.46 | 3.1 | $77.24^{--}$ | 3.8 | $76.55^{--}$ | 4.0 | $77.97^{++}$ | 4.0 | $77.18^-$ | 4.2 | $77.41^-$ | 4.7 |
| Basque | 74.26 | 3.6 | $74.78^{++}$ | 4.5 | $74.05^-$ | 4.7 | $74.12^-$ | 4.7 | $73.8^-$ | 4.9 | $74.21^-$ | 4.9 |
| Belarusian | 70.12 | 2.4 | $68.67^-$ | 2.9 | $68.74^-$ | 3.0 | $68.02^{--}$ | 3.0 | $69.18^-$ | 3.1 | $66.86^{--}$ | 3.3 |
| Bulgarian | 88.42 | 3.4 | $87.62^{--}$ | 4.2 | $87.95^-$ | 4.5 | $87.58^{--}$ | 4.4 | $87.9^{--}$ | 4.6 | $87.53^{--}$ | 4.6 |
| Catalan | 87.57 | 3.4 | $86.77^{--}$ | 4.3 | $87.63^+$ | 4.7 | $87.22^{--}$ | 4.6 | $87.28^{--}$ | 4.8 | $87.35^{--}$ | 4.7 |
| Chinese | 79.23 | 3.2 | $79.0^-$ | 4.1 | $79.31^+$ | 4.3 | $79.15^-$ | 4.3 | $79.13^-$ | 4.5 | $78.8^-$ | 4.4 |
| Coptic | 78.68 | 1.9 | $76.58^-$ | 2.4 | $78.68^+$ | 2.5 | $79.73^+$ | 2.5 | $77.25^-$ | 2.6 | $75.62^{--}$ | 2.5 |
| Croatian | 81.23 | 3.2 | $80.76^-$ | 3.9 | $81.44^+$ | 4.1 | $81.2^-$ | 4.0 | $81.58^+$ | 4.2 | $80.5^{--}$ | 4.3 |
| Czech | 85.74 | 3.5 | $85.98^{++}$ | 4.3 | $85.88^{++}$ | 4.5 | $86.01^{++}$ | 4.4 | $86.02^{++}$ | 4.6 | $85.39^{--}$ | 4.9 |
| Danish | 80.93 | 3.1 | $81.02^+$ | 3.8 | $80.68^-$ | 4.0 | $80.61^{--}$ | 4.0 | $80.83^-$ | 4.2 | $80.81^-$ | 4.4 |
| Dutch | 78.67 | 3.3 | $78.63^-$ | 4.2 | $78.63^-$ | 4.4 | $78.87^+$ | 4.4 | $78.13^-$ | 4.6 | $79.36^{++}$ | 4.6 |
| English | 84.16 | 3.6 | $84.09^-$ | 4.5 | $83.91^-$ | 4.7 | $84.49^+$ | 4.7 | $84.35^+$ | 4.9 | $83.78^-$ | 4.5 |
| Estonian | 81.57 | 3.1 | $82.2^+$ | 3.9 | $81.55^-$ | 4.0 | $81.9^+$ | 4.0 | $81.05^-$ | 4.2 | $81.48^-$ | 4.5 |
| Finnish | 81.25 | 3.3 | $81.37^+$ | 4.2 | $81.8^+$ | 4.4 | $81.52^+$ | 4.3 | $81.71^+$ | 4.5 | $81.03^-$ | 4.6 |
| French | 84.65 | 3.0 | $84.88^+$ | 3.6 | $85.18^+$ | 3.8 | $84.74^+$ | 3.8 | $84.51^-$ | 3.9 | $85.19^+$ | 4.3 |
| Galician | 80.51 | 3.5 | $79.67^{--}$ | 4.3 | $80.24^-$ | 4.5 | $79.88^{--}$ | 4.4 | $80.36^-$ | 4.7 | $80.59^+$ | 4.8 |
| German | 79.86 | 3.3 | $79.0^{--}$ | 4.2 | $79.54^-$ | 4.5 | $79.65^-$ | 4.4 | $79.54^-$ | 4.6 | $79.38^-$ | 4.4 |
| Gothic | 74.57 | 3.2 | $74.77^+$ | 3.9 | $74.63^+$ | 4.2 | $73.75^{--}$ | 4.1 | $73.96^{--}$ | 4.3 | $73.93^-$ | 4.5 |
| Greek | 84.71 | 3.1 | $84.87^+$ | 3.7 | $84.45^-$ | 4.0 | $84.61^-$ | 3.9 | $84.18^{--}$ | 4.1 | $85.21^+$ | 4.5 |
| Hebrew | 82.16 | 3.2 | $81.94^-$ | 3.8 | $82.13^-$ | 4.1 | $81.83^-$ | 4.0 | $81.42^-$ | 4.2 | $81.67^-$ | 4.4 |
| Hindi | 90.8 | 3.5 | $90.92^+$ | 4.4 | $90.66^-$ | 4.7 | $90.46^{--}$ | 4.6 | $90.73^-$ | 4.8 | $90.42^{--}$ | 4.8 |
| Hungarian | 73.34 | 3.1 | $72.78^-$ | 3.5 | $73.02^-$ | 3.7 | $72.73^-$ | 3.7 | $72.6^-$ | 3.9 | $72.85^-$ | 4.2 |
| Indonesian | 79.47 | 2.9 | $78.81^{--}$ | 3.5 | $79.07^-$ | 3.7 | $79.23^-$ | 3.7 | $79.31^-$ | 3.9 | $79.1^{--}$ | 4.0 |
| Irish | 60.07 | 2.8 | $59.17^{--}$ | 3.5 | $59.72^-$ | 3.7 | $57.94^{--}$ | 3.7 | $58.6^{--}$ | 3.9 | $57.55^{--}$ | 3.8 |
| Italian | 89.21 | 2.9 | $89.34^+$ | 3.3 | $89.16^-$ | 3.5 | $88.33^{--}$ | 3.5 | $89.16^-$ | 3.6 | $89.57^+$ | 3.8 |
| Japanese | 92.16 | 3.3 | $92.14^-$ | 4.3 | $91.95^-$ | 4.4 | $91.97^-$ | 4.4 | $92.27^+$ | 4.6 | $91.86^-$ | 4.7 |
| Kazakh | 22.78 | 3.4 | $26.79^{++}$ | 4.5 | $24.82^{++}$ | 4.8 | $24.22^{++}$ | 4.7 | $20.17^{--}$ | 4.9 | $23.64^{++}$ | 4.7 |
| Korean | 60.84 | 3.5 | $58.97^{--}$ | 4.5 | $58.8^{--}$ | 4.6 | $59.89^{--}$ | 4.7 | $59.85^{--}$ | 4.8 | $59.37^{--}$ | 4.9 |
| Latin | 43.31 | 3.3 | $43.59^{++}$ | 4.0 | $43.45^{++}$ | 4.2 | $42.19^{--}$ | 4.2 | $43.84^+$ | 4.4 | $40.22^{--}$ | 4.5 |
| Latvian | 75.14 | 3.4 | $75.83^{++}$ | 4.2 | $75.23^+$ | 4.5 | $75.26^+$ | 4.4 | $74.85^-$ | 4.7 | $75.1^-$ | 4.6 |
| Lithuanian | 42.74 | 1.5 | $44.06^+$ | 1.8 | $44.43^+$ | 1.9 | $40.75^-$ | 1.8 | $41.98^-$ | 1.9 | $40.94^-$ | 2.0 |
| Marathi | 66.02 | 1.7 | $64.32^-$ | 2.2 | $65.53^-$ | 2.3 | $64.81^-$ | 2.3 | $62.86^-$ | 2.3 | $63.11^-$ | 2.4 |
| Old Church Slavonic | 78.33 | 3.3 | $78.86^+$ | 4.2 | $79.01^{++}$ | 4.5 | $78.81^{++}$ | 4.4 | $78.55^+$ | 4.6 | $78.86^{++}$ | 4.5 |
| Persian | 82.1 | 3.1 | $81.95^-$ | 3.9 | $82.23^+$ | 4.1 | $82.3^+$ | 4.1 | $82.12^+$ | 4.2 | $82.63^+$ | 4.3 |
| Polish | 90.92 | 3.5 | $90.87^-$ | 4.4 | $90.34^{--}$ | 4.7 | $90.65^-$ | 4.7 | $90.44^-$ | 4.9 | $90.02^{--}$ | 4.7 |
| Portuguese | 86.27 | 3.1 | $86.47^+$ | 4.0 | $86.5^+$ | 4.3 | $86.72^+$ | 4.2 | $86.02^-$ | 4.5 | $86.53^+$ | 4.2 |
| Romanian | 82.12 | 3.3 | $81.71^-$ | 4.2 | $80.47^{--}$ | 4.3 | $81.28^{--}$ | 4.4 | $81.13^{--}$ | 4.5 | $81.55^-$ | 4.6 |
| Russian | 79.47 | 3.0 | $79.49^+$ | 3.7 | $79.28^-$ | 3.9 | $79.1^-$ | 3.9 | $79.4^-$ | 4.0 | $79.21^-$ | 4.0 |
| Serbian | 84.9 | 3.3 | $85.15^+$ | 4.1 | $85.81^{++}$ | 4.4 | $85.16^+$ | 4.3 | $85.02^+$ | 4.5 | $85.71^{++}$ | 4.4 |
| Slovak | 85.54 | 3.2 | $85.07^{--}$ | 4.1 | $85.52^-$ | 4.4 | $85.02^-$ | 4.3 | $84.49^{--}$ | 4.5 | $85.06^{--}$ | 4.5 |
| Slovenian | 88.73 | 3.2 | $88.6^-$ | 3.9 | $88.63^-$ | 4.1 | $88.59^-$ | 4.0 | $88.46^-$ | 4.2 | $88.43^-$ | 4.4 |
| Spanish | 85.16 | 2.8 | $85.1^-$ | 3.6 | $84.58^{--}$ | 3.8 | $84.63^-$ | 3.7 | $84.47^{--}$ | 3.9 | $84.93^-$ | 4.1 |
| Swedish | 83.73 | 3.4 | $84.22^{++}$ | 4.3 | $83.91^+$ | 4.6 | $84.0^+$ | 4.5 | $82.73^{--}$ | 4.8 | $83.51^-$ | 4.6 |
| (Sw) Sign Language | 23.4 | 1.1 | $24.47^+$ | 1.4 | $27.3^+$ | 1.4 | $24.82^+$ | 1.4 | $24.11^+$ | 1.5 | $22.7^-$ | 1.4 |
| Tamil | 69.18 | 2.0 | $69.83^+$ | 2.4 | $69.28^+$ | 2.5 | $68.58^-$ | 2.5 | $70.04^+$ | 2.6 | $70.14^{++}$ | 2.7 |
| Telugu | 75.17 | 1.5 | $75.73^+$ | 1.7 | $75.73^+$ | 1.7 | $74.48^-$ | 1.7 | $73.65^-$ | 1.7 | $74.06^-$ | 1.9 |
| Turkish | 59.51 | 3.2 | $60.49^{++}$ | 4.0 | $59.74^+$ | 4.1 | $59.53^+$ | 4.0 | $59.6^+$ | 4.3 | $59.64^+$ | 4.5 |
| Ukrainian | 81.29 | 3.2 | $82.05^{++}$ | 3.9 | $81.61^+$ | 4.1 | $81.79^+$ | 4.0 | $82.08^{++}$ | 4.2 | $81.46^+$ | 4.2 |
| Urdu | 83.12 | 3.3 | $83.79^{++}$ | 4.1 | $83.41^+$ | 4.4 | $83.65^{++}$ | 4.3 | $83.68^{++}$ | 4.6 | $83.48^+$ | 4.5 |
| Vietnamese | 64.73 | 3.4 | $63.73^{--}$ | 4.4 | $63.5^{--}$ | 4.7 | $64.32^-$ | 4.6 | $64.7^-$ | 4.9 | $63.96^{--}$ | 4.8 |
| AVERAGE | 75.67 | 3.0 | 75.65 | 3.8 | 75.67 | 3.9 | 75.45 | 3.9 | 75.29 | 4.1 | 75.22 | 4.2 |

Table 3: ARC-STANDARD baseline configuration versus different runs with the NO-GD feature set and embedding size reduction from 10% to 50%. See Table 2 for color scheme definition.

| | STANDARD | | NO-GD | | | | | | STANDARD | | NO-GD | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | size -10% | | size -50% | | | | | | size -10% | | size -50% | |
| **Treebank** | **LAS** | **kt/s** | **LAS** | **kt/s** | **LAS** | **kt/s** | | **Treebank** | **LAS** | **kt/s** | **LAS** | **kt/s** | **LAS** | **kt/s** |
| Afrikaans | 82.55 | 3.0 | $83.19^{+}$ | 3.9 | $80.55^{--}$ | 4.2 | | Anc Greek | 58.97 | 2.9 | $59.04^{+}$ | 3.6 | $60.3^{++}$ | 3.8 |
| Arabic | 76.77 | 3.0 | $76.8^{+}$ | 3.8 | $76.71^{-}$ | 4.1 | | Basque | 73.98 | 3.2 | $73.74^{-}$ | 4.2 | $72.48^{--}$ | 4.3 |
| Belarusian | 69.75 | 2.4 | $69.54^{-}$ | 2.9 | $68.52^{-}$ | 3.2 | | Bulgarian | 87.95 | 3.2 | $87.06^{--}$ | 4.2 | $87.73^{-}$ | 4.9 |
| Catalan | 87.01 | 3.1 | $87.16^{+}$ | 3.9 | $86.8^{-}$ | 4.6 | | Chinese | 78.26 | 3.2 | $79.76^{++}$ | 4.1 | $78.19^{-}$ | 4.6 |
| Coptic | 77.25 | 1.3 | $78.01^{+}$ | 1.7 | $76.29^{-}$ | 1.5 | | Croatian | 80.63 | 3.0 | $81.28^{+}$ | 4.0 | $81.37^{++}$ | 4.1 |
| Czech | 85.55 | 3.4 | $83.49^{--}$ | 4.3 | $84.97^{--}$ | 4.7 | | Danish | 79.79 | 2.9 | $78.79^{--}$ | 3.7 | $78.55^{--}$ | 4.0 |
| Dutch | 77.02 | 3.1 | $77.93^{+}$ | 3.9 | $77.53^{+}$ | 4.3 | | English | 83.19 | 3.6 | $83.61^{+}$ | 4.5 | $83.92^{++}$ | 4.9 |
| Estonian | 80.65 | 2.9 | $80.01^{-}$ | 3.7 | $80.72^{+}$ | 4.1 | | Finnish | 81.47 | 3.3 | $80.98^{--}$ | 4.1 | $81.25^{-}$ | 4.4 |
| French | 83.54 | 2.7 | $82.53^{--}$ | 3.5 | $83.78^{+}$ | 3.9 | | Galician | 79.85 | 3.3 | $80.44^{++}$ | 4.4 | $80.01^{+}$ | 4.8 |
| German | 78.52 | 3.1 | $77.53^{--}$ | 3.5 | $77.6^{--}$ | 4.2 | | Gothic | 72.92 | 2.7 | $72.96^{+}$ | 3.5 | $71.43^{--}$ | 3.8 |
| Greek | 84.13 | 3.0 | $83.91^{-}$ | 3.9 | $84.11^{-}$ | 4.4 | | Hebrew | 81.87 | 3.1 | $82.07^{+}$ | 4.0 | $82.41^{+}$ | 4.5 |
| Hindi | 90.46 | 3.2 | $89.86^{--}$ | 4.0 | $89.58^{--}$ | 4.4 | | Hungarian | 72.33 | 2.9 | $73.77^{++}$ | 3.7 | $72.8^{+}$ | 3.3 |
| Indonesian | 78.86 | 2.8 | $79.0^{+}$ | 3.6 | $79.19^{+}$ | 3.9 | | Irish | 61.82 | 2.8 | $60.67^{--}$ | 3.5 | $60.88^{--}$ | 3.9 |
| Italian | 88.34 | 2.9 | $88.39^{+}$ | 3.4 | $88.51^{+}$ | 4.0 | | Japanese | 91.95 | 3.3 | $92.02^{+}$ | 4.3 | $91.91^{-}$ | 4.7 |
| Kazakh | 29.32 | 3.4 | $29.64^{++}$ | 4.3 | $29.77^{+}$ | 4.8 | | Korean | 60.46 | 3.5 | $59.66^{--}$ | 4.4 | $58.75^{--}$ | 5.1 |
| Latin | 47.11 | 2.6 | $45.05^{--}$ | 3.3 | $44.3^{--}$ | 3.6 | | Latvian | 74.73 | 3.3 | $75.05^{+}$ | 4.2 | $75.05^{+}$ | 4.8 |
| Lithuanian | 46.79 | 1.4 | $44.72^{-}$ | 1.8 | $44.91^{-}$ | 1.9 | | Marathi | 65.05 | 1.7 | $64.81^{-}$ | 2.2 | $65.53^{+}$ | 2.4 |
| Old Church Slavonic | 79.48 | 3.0 | $80.07^{+}$ | 3.9 | $77.62^{--}$ | 4.4 | | Persian | 80.79 | 3.0 | $81.54^{+}$ | 3.7 | $80.84^{+}$ | 4.3 |
| Polish | 90.49 | 3.5 | 90.49 | 4.4 | $90.17^{-}$ | 4.9 | | Portuguese | 83.87 | 2.7 | $83.09^{-}$ | 2.9 | $84.41^{+}$ | 3.6 |
| Romanian | 80.92 | 3.3 | $80.08^{--}$ | 4.1 | $80.1^{--}$ | 4.5 | | Russian | 78.55 | 2.9 | $77.78^{--}$ | 3.7 | $77.75^{--}$ | 4.1 |
| Serbian | 85.8 | 3.2 | $85.02^{--}$ | 4.1 | $85.24^{-}$ | 4.5 | | Slovak | 84.96 | 3.2 | $85.25^{+}$ | 3.9 | $84.27^{--}$ | 4.4 |
| Slovenian | 89.35 | 3.0 | $88.85^{--}$ | 3.6 | $88.88^{--}$ | 4.2 | | Spanish | 84.32 | 2.7 | $83.84^{-}$ | 3.6 | $83.28^{--}$ | 3.7 |
| Swedish | 84.37 | 3.4 | $82.8^{--}$ | 4.3 | $83.72^{--}$ | 4.8 | | (Sw) Sign Language | 10.64 | 0.9 | $21.28^{++}$ | 1.2 | $17.73^{++}$ | 1.2 |
| Tamil | 71.04 | 1.7 | $70.54^{-}$ | 3.2 | $70.79^{-}$ | 2.6 | | Telugu | 74.2 | 1.4 | $75.17^{++}$ | 1.7 | $73.79^{-}$ | 1.7 |
| Turkish | 60.32 | 3.0 | $59.71^{-}$ | 3.9 | $58.45^{--}$ | 4.3 | | Ukrainian | 81.8 | 3.0 | $80.66^{--}$ | 4.0 | $80.86^{--}$ | 4.4 |
| Urdu | 81.29 | 2.9 | $81.42^{+}$ | 3.7 | $82.3^{++}$ | 4.0 | | Vietnamese | 64.35 | 3.5 | $64.32^{-}$ | 4.4 | $64.67^{+}$ | 5.0 |

Table 4: SWAP baseline configuration versus different runs with the NO-GD feature set and embedding size reduction by a factor of 0.1 and 0.5. The average LAS/speed for the baseline is 75.29/2.8, for the NO-GD feature set with embedding reduction by a factor of 0.1 is 75.27/3.6, and with embedding reduction by a factor of 0.5 75.02/4.0. See Table 2 for color scheme definition.

number of features (§5.2), an additional speed improvement is expected, as less features need to be collected. But broadly speaking, the speed obtained by skipping half of the features should be in line with that obtained by reducing the size of the embeddings of the original features by a factor of 0.5.

For a practical point of view, in this work we relied on `keras` (Chollet et al., 2015). With respect to the part reported in §5.2, the experiments went as expected. Taking as examples the results for the ARC-STANDARD algorithm, using no *grand-daughter* features implies to diminish the dimension of the input vector from 1860 dimensions to 1420, a reduction of ∼23%. The average thousands of tokens parsed per second of the *de facto standard* features was 3.0 and the average obtained without *grand-daughter* features was 3.7, a gain of ∼20%. If we also skip *daughter* features and reduce the size of the input vector by ∼71%, the speed increased by a factor of 2.5. Similar tendencies were observed with respect to the SWAP algorithm. When reducing the size of the embeddings (§5.3), the obtained speed-ups were however lower than those expected in theory. In this sense, an alternative implementation or a use of a different framework could lead to reduce these times to values closer to the theoretical expectation.

Trying other neural architectures is also of high interest, but this is left as an open question for future research. In particular, in the popular BIST-based parsers (Kiperwasser and Goldberg, 2016; de Lhoneux et al., 2017; Vilares and Gómez-Rodríguez, 2017), the input is first processed by a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) that computes an embedding for each token, taking into account its left and right context. These embeddings are then used to extract the features for transition-based algorithms, including the head of different elements and their leftmost/rightmost children. Those features are then fed to a feed-forward network, similar to the one evaluated in this work. Thus, the results of this work might be of future interest for this type of parsers too, as the output of the LSTM can be seen as improved and better contextualized word embeddings.

## 6 Conclusion

We explored whether it is possible to reduce the number and size of embedded features assumed as *de facto standard* by feed-forward network transition-based dependency parsers. The aim was to train efficient and light parsers for a vast amount of languages showing a rich variety of structures and morphologies.

To test the hypothesis we used a multilingual testbed: the Universal Dependencies v2.1. The study considered two transition-based algorithms to train the oracles: a stack-based ARC-STANDARD and its non-projective version, by adding the SWAP operation. We first evaluated three sets of features, clustered according to their extraction costs: (1) the *de facto standard* features that usually are fed as input to feed-forward parsers and consider *daughter* and *grand-daughter* features, (2) a *no grand-daughter* feature set and (3) a *no grand-daughter/daughter* feature set. For the majority of the treebanks we found that the feature set (2) did not cause a significant loss, both for the stack-based ARC-STANDARD and the SWAP algorithms. We then took that set of features and reduced the size of the embeddings used to represent each feature, up to a factor of 0.5. The experiments also show that for both the ARC-STANDARD and the SWAP algorithms these reductions did not cause, in general terms, a significant loss. As a result, we obtained a set of lighter and faster transition-based parsers that achieve a better *accuracy vs bandwidth* ratio than the original ones. It was observed that these improvements were not restricted to a particular language family or specific morphology.

As future work, it would be interesting to try alternative experiments to see whether reducing the size of embeddings works the same for words as for other features. Also, the results are compatible with existent optimizations and can be used together to obtain further speed-ups. Related to this, quantized word vectors (Lam, 2018) can save memory and be used to outperform traditional embeddings.

## Acknowledgments

We would like to thank the anonymous reviewers for their useful suggestions and detailed comments. This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the TELEPARES-UDC project (FFI2014-51978-C2-2-R) and the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia (ED431B 2017/01). We gratefully acknowledge NVIDIA Corporation for the donation of a GTX Titan X GPU.

## References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Miguel Ballesteros and Joakim Nivre. 2012. Maltoptimizer: A system for maltparser optimization. In *LREC*, pages 2757–2763.

Yevgeni Berzak, Yan Huang, Andrei Barbu, Anna Korhonen, and Boris Katz. 2016. Anchoring and agreement in syntactic annotations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2215–2224, Austin, Texas. Association for Computational Linguistics.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 89–97, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jan A. Botha, Emily Pitler, Ji Ma, Anton Bakalov, Alex Salcianu, David Weiss, Ryan McDonald, and Slav Petrov. 2017. Natural language processing with small feed-forward networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2879–2885. Association for Computational Linguistics.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

François Chollet et al. 2015. Keras. https://github.com/keras-team/keras.

Michael A Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014.

Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1370–1380.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the 5th International Conference on Learning Representations*.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018. Non-projective dependency parsing with non-local transitions. 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 693–700. Association for Computational Linguistics.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Carlos Gómez-Rodríguez, Iago Alonso-Alonso, and David Vilares. 2017. How important is syntactic parsing accuracy? an empirical evaluation on rule-based sentiment analysis. *Artificial Intelligence Review*.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Liang Huang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1222–1231. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 673–682. Association for Computational Linguistics.

M. Lam. 2018. Word2Bits - Quantized Word Vectors. *ArXiv e-prints*.

Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017. From raw text to universal dependencies-look, no tags! *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 207–217.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational linguistics*, 19(2):313–330.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, IncrementParsing '04, pages 50–57, Stroudsburg, PA, USA. Association for Computational Linguistics.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 351–359. Association for Computational Linguistics.

Joakim Nivre and Johan Hall. 2010. A quick guide to MaltParser optimization. *http://maltparser.org/guides/opt/quick-opt.pdf*.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Joakim Nivre 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.

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–117, Vancouver, Canada. Association for Computational Linguistics.

Mohammad Sadegh Rasooli and Joel Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *arXiv preprint arXiv:1503.06733*.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23. Association for Computational Linguistics.

Milan Straka, Jan Hajic, Jana Straková, and Jan Hajic jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *International Workshop on Treebanks and Linguistic Theories (TLT14)*, pages 208–220.

Ivan Titov and James Henderson. 2007. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proc. of the CoNLL shared task. Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, Czech Republic.

Andrei Vlad Vacariu. 2017. *A high-throughput dependency parser*. Ph.D. thesis, Applied Sciences: School of Computing Science, Simon Fraser University.

David Vilares and Carlos Gómez-Rodríguez. 2017. A non-projective greedy dependency parser with bidirectional LSTMs. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 152–162.

Alexander Volokh. 2013. *Performance-Oriented Dependency Parsing*. Doctoral dissertation, Saarland University, Saarbrücken, Germany.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. 2005. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2):207–238.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 188–193. Association for Computational Linguistics.