Encoding morphological annotations for universal dependency parsing

August 2017

Author:
Kuan Yu
kuan.yu@student.uni-tuebingen.de

Advisor:
Dr. Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

Abstract

We investigated different methods of encoding morphological annotations for the neural network classifier in a transition-based dependency parser. The most effective and efficient method we found was embedding morphological entries as real vectors which compose through a simple componentwise operation such as max.

1 Introduction

Morphological annotations provide valuable linguistic information. Figure 1 gives an example for the word "am", annotated by an attribute-value structure (AVS) with five entries, where each entry is an attribute-value pair.

```
Mood=Ind|Number=Sing|Person=1|Tense=Pres|VerbForm=Fin
```

Figure 1: Morphological AVS for "am".

Encoding such structured information in machine learning for language processing is however not entirely straightforward. The number of entries in a morphological AVS is uncertain. In English, it ranges from zero (such as for prepositions) to five and more. It is also difficult to determine when a morphological attribute is applicable, simply based on the word form or the word class. Consider for the verb "like", the `Person` attribute is applicable in "I like" and "He likes" but not in "He does like".

We investigated different methods of encoding morphological annotations for machine learning, and evaluated these methods in the task of universal dependency parsing. Our program is open source under the MIT license.[1]

We will first describe the parsing task (Section 2–4), then our methods (Section 5–6), and finally our experiments and the results (Section 7–9).

2 Universal dependencies

Universal Dependencies (UD) (Nivre, Marneffe, et al. 2016) is a cross-linguistically consistent annotation scheme for dependency-based treebanks.[2] UD treebanks conform to the CoNLL-U format.[3] In this format, each node in a dependency-graph has ten fields named `ID`, `FORM`, `LEMMA`, `UPOSTAG`, `XPOSTAG`, `FEATS`, `HEAD`, `DEPREL`, `DEPS`, and `MISC`. A labeled directed arc $label(parent, child)$ is given by $DEPREL(HEAD, ID)$. Figure 2 gives an example of a dependency tree. The arc $conj(1, 6)$ crosses $root(0, 2)$ as well as $punct(2, 9)$, making this tree non-projective.

From UD version 2.0 (UD2) (Nivre et al. 2017a; Nivre, Agić, Ahrenberg, et al. 2017b), we selected 15 treebanks representing a variety of language families and morphological typologies, although fusional Indo-European languages are the most available choices. Our selections also vary in sizes and

[1] https://github.com/ysmiraak/darc

[2] http://universaldependencies.org

[3] http://universaldependencies.org/format.html

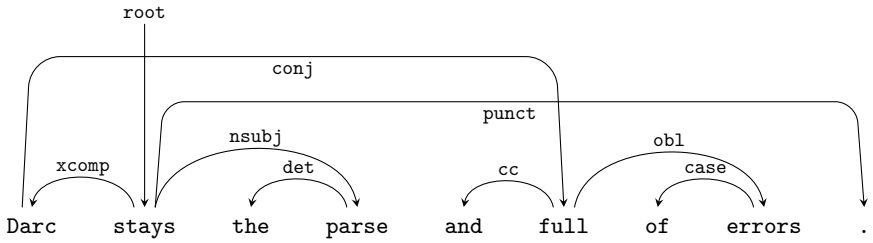

Figure 2: A non-projective dependency tree.

ID	FORM	LEMMA	UPOSTAG	FEATS	HEAD	DEPREL
1	Darc	darc	ADJ	Degree=Pos	2	xcomp
2	stays	stay	VERB	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	0	root
3	the	the	DET	Definite=Def|PronType=Art	4	det
4	parse	parse	NOUN	Number=Sing	2	nsubj
5	and	and	CCONJ	_	6	cc
6	full	full	ADJ	Degree=Pos	1	conj
7	of	of	ADP	_	8	case
8	errors	error	NOUN	Number=Plur	6	obl
9	.	.	PUNCT	_	2	punct

degrees of non-projectivity. Each treebank consists of a train-, a dev-, and a test-set. Table 1 lists the statistics from the train-sets.

treebank	#tree	#node	%nonp	family	type
Ancient_Greek	11 476	159 895	62.77	IE.Hellenic	fus
Arabic	6 075	223 881	10.77	AA.Semitic	fus
Basque	5 396	72 974	33.52	Vasconic	agg
Bulgarian	8 907	124 336	3.17	IE.Balto-Slavic	fus
Chinese	3 997	98 608	0.63	Sino-Tibetan	iso
Croatian	7 689	169 283	8.56	IE.Balto-Slavic	fus
Dutch	12 330	186 467	20.28	IE.Germanic	fus
Finnish-FTB	14 981	127 602	7.47	Uralic	agg
Hebrew	5 241	137 680	1.26	AA.Semitic	fus
Italian	12 838	270 703	4.24	IE.Italic	fus
Latin-PROIEL	14 192	147 044	26.52	IE.Italic	fus
Persian	4 798	121 064	6.86	IE.Iranian	fus
Polish	6 100	62 501	0.85	IE.Balto-Slavic	fus
Swedish	4 303	66 645	3.18	IE.Germanic	fus
Turkish	3 685	38 082	11.23	Turkic	agg

Table 1:

#tree	number of trees
#node	number of syntactic nodes
%nonp	percentage of non-projective trees
family	IE: Indo-European, AA: Afro-Asiatic
type	iso: isolating, agg: agglutinative, fus: fusional

3 Transition-based parsing

Dependency parsing produces a directed acyclic graph from a sequence of nodes $[w_0, w_1, \ldots, w_n]$, which are the syntactic tokens of a sentence, where w_0 is a pseudo root node. A transition-based parsing algorithm defines transition actions over configurations. Each configuration is a triple consisting of a stack σ, a buffer β, and a set of arcs A. From the initial configuration $(\sigma\colon [w_0], \beta\colon [w_1, w_2, \ldots, w_n], A\colon \{\})$, a series of transition actions are taken to produce a terminal configuration $(\sigma\colon [w_0], \beta\colon [\,], A)$.

We adopted a non-projective variant of the arc-standard algorithm (Nivre 2008; Nivre 2009). The transition actions are listed below. For labeled parsing, some actions are parameterized over *arc*, the set of possible dependency labels, in which case a different action is defined for each label.

$$
\begin{array}{rll}
\texttt{shift}: & (\sigma, [w_i|\beta], A) \mapsto ([w_i|\sigma], \beta, A) & \\
\texttt{left(arc)}: & ([w_i, w_j|\sigma], \beta, A) \mapsto ([w_i|\sigma], \beta, A \cup \{arc(w_i, w_j)\}) & \\
\texttt{right(arc)}: & ([w_i, w_j|\sigma], \beta, A) \mapsto ([w_j|\sigma], \beta, A \cup \{arc(w_j, w_i)\}) & 0 \neq j \\
\texttt{swap}: & ([w_i, w_j|\sigma], \beta, A) \mapsto ([w_i|\sigma], [w_j|\beta], A) & 0 < j < i
\end{array}
$$

A classifier is trained for estimating the probabilities of each transition action given the current state of the configuration. For training the classifier, we implemented the static oracle designed by Nivre, Kuhlmann, and Hall (2009). The static oracle produces a transition sequence which leads to the gold-standard parse of a sentence. Details about the classifier is discussed in Section 4. Our parser is greedy, which means that only the most probable legal transition action is taken.

The parsing results are evaluated the same way as in the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies (Zeman et al. 2017). The evaluation metric is the labeled attachment score (LAS), namely the percentage of syntactic nodes with the correct HEAD and DEPREL. Non-syntactic nodes in the CoNLL-U format, i.e. empty nodes and multi-word tokens, are simply ignored. Only the UD labels count for scoring, and not the language-specific subtypes. For example, `acl:relcl` is considered to be the same as `acl`. However, our parser always keeps the complete DEPREL given by the treebank.

In UD, a sentence has a single root, which means that a unique syntactic node is attached to the pseudo root node. However, the transition algorithm

may attach multiple ones. Our parser keeps the first attachment, and re-attach the other nodes to the first one with a `parataxis` arc.

4 Neural network classifier

Following Chen and Manning (2014), we implemented a simple feedforward neural network classifier in Keras (Chollet et al. 2015) with the TensorFlow backend (Martín Abadi et al. 2015).

The input features are extracted from 18 graph nodes.

- 3 nodes from the stack top, with the top two being the nodes of interest
- 3 lookaheads from the buffer
- 2 leftmost & rightmost children of the two
- 1 leftmost-of-leftmost & rightmost-of-rightmost children of the two

Some nodes may be missing in a configuration, in which case dummy values are supplied as input features.

The inputs are successively transformed by two hidden layers each with 256 rectified linear units (ReLU), and then outputted through a softmax layer with as many units as transition actions. The softmax output assigns a probability for each action. Successive layers are fully connected. The weights for the connections are initialized in a random uniform distribution following He et al. (2015). The parameters are updated through backpropagation using the AdaMax algorithm (Kingma and Ba 2014) in batches of 32 training instances, with 25% dropout (Srivastava et al. 2014) applied to the hidden layers.[4]

5 Nominal features

All the input features for our classifier are nominal values, i.e. symbols.

- Lexical features: FORM & LEMMA
- Part-of-speech information: UPOSTAG
- Morphological information: FEATS
- Partial graph information: DEPREL, from the children nodes

[4]We experimented with a higher dropout rate, and found that while it could improve the accuracy of the model slightly, it also required extending the sizes of the layers, which would quadratically increase the number of parameters.

Table 2 lists some statistics about these nominal categories. Details about FEATS are discussed in Section 6. Analogously to word classes, UPOSTAG and DEPREL are closed in the sense that new members are not expected, while FORM and LEMMA are open classes for assuming Zipfian distributions where the majority of the members appear only once or twice. Since an open nominal class anticipates new symbols which are unseen in the training data, we add a catch-all symbol for unknowns, and all symbols with unique appearances are replaced with it for parameter training.

treebank	#upos	#drel	#form	#form*	#lemma	#lemma*
Ancient_Greek	14	25	33 237	11 856	10 932	6 076
Arabic	16	32	23 149	13 242	14 842	8 793
Basque	16	30	19 222	6 483	8 605	4 246
Bulgarian	16	35	25 047	9 682	13 292	6 877
Chinese	15	42	17 610	7 096	17 604	7 091
Croatian	17	41	34 968	13 340	17 672	8 325
Dutch	16	32	26 694	10 053	19 687	8 200
Finnish-FTB	17	38	39 717	10 579	18 767	6 991
Hebrew	16	43	16 156	8 127	9 497	5 609
Italian	17	44	28 126	13 365	18 407	9 483
Latin-PROIEL	14	34	22 258	9 762	6 803	4 345
Persian	15	37	13 248	6 907	6 743	4 376
Polish	15	30	20 784	5 572	11 567	4 805
Swedish	16	39	12 911	4 885	8 284	3 831
Turkish	14	29	13 780	3 901	4 909	2 657

Table 2:

#upos	number of unique UPOSTAG
#drel	number of unique DEPREL
#form	number of unique FORM
#form*	number of unique FORM with multiple appearances
#lemma	number of unique LEMMA
#lemma*	number of unique LEMMA with multiple appearances

In machine learning, nominal values are usually represented by embedding the nominal category in a real vector space. An embedding is an injective structure-preserving map. It is often difficult, however, to discern the exact structure of the nominal category. Ideally, the embedding function should preserve all the relevant information for solving the task at hand. Such an embedding function is normally reified as a real matrix. The embedding matrix has a column for each nominal value (injective up to isomorphism) and as many rows as the dimension of the target space.[5]

[5]The actual embedding matrix is transposed for efficient access in the row-major order.

When the identity matrix is used, the embedding is know as one-hot encoding. The nominal values are represented as orthonormal vectors, and thus the nominal category is treated as completely unstructured, i.e. a set. A major drawback of this treatment is that the dimension of the target space must be as high as the number of nominal values. With vocabularies, it is usually in the magnitude of 10^4. Nevertheless, one-hot encoding can be efficiently and effectively used in many machine learning algorithms, such as generalized linear models and support vector machines.

In neural network models, the embedding matrix can be trained through backpropagation. This allows the embedding function to adapt to the given task. However, the size of the embedding matrix adds to the number of parameters of the model, and while the number of rows is flexible, the number of columns is not. For big nominal categories such as FORM and LEMMA, the embedding matrix has a substantial amount of parameters to be trained. Quite often they make up the majority of the parameters in the model. Chen and Manning (2014) showed that it is helpful in dependency parsing to initialize these parameters with pre-trained embeddings. For training general-purpose word embeddings, a popular method is word2vec (Mikolov, Sutskever, et al. 2013; Mikolov, Chen, et al. 2013). Ling et al. (2015) extended the method to be better at learning syntactic information. We used this tool for pre-training FORM and LEMMA embeddings in our experiments.[6] Embeddings trained by this method have scalar values in normal distributions with means around zero and standard deviations around one half. In our experiments, embeddings which were not pre-trained were randomly initialized from a uniform distribution with the interval $[-0.5, 0.5]$, resulting in a standard deviation of $\frac{1}{\sqrt{12}}$.

6 Encoding morphological AVS

Embedding FEATS is the topic of investigation in our experiments. Table 3 lists some relevant statistics. Despite UD2 having a finite morphological feature inventory, the set of FEATS, unlike UPOSTAG, is not a closed class in practice. New members emerge regularly in all treebanks except for Chinese, as illustrated by the `#feats!` column. The set of morphological entries is virtually closed, but only a small subset of them are actually active in any specific treebank, and their numbers differ greatly between treebanks, as

[6] https://github.com/wlin12/wang2vec/ with options `{-type 3 -hs 0 -min-count 2 -window 7 -sample 0.1 -negative 7 -iter 20}`; Though in fact `[-min-count 2]` had no effect, as we had all hapaxes replaced with the symbol for unknowns.

treebank	%feats	|feats|	⌈feats⌉	#feats	#feats!	#entry
Ancient_Greek	60.84	3.88	7	638	117	32
Arabic	75.26	3.09	7	293	39	36
Basque	63.10	2.92	11	572	185	69
Bulgarian	63.93	3.93	8	343	37	45
Chinese	11.09	1.01	2	14	0	12
Croatian	80.17	3.26	9	1020	175	52
Dutch	91.83	1.99	8	255	7	65
Finnish-FTB	71.38	3.30	8	1309	356	79
Hebrew	59.64	2.59	7	443	100	48
Italian	61.20	2.80	5	175	23	40
Latin-PROIEL	73.18	4.26	7	760	99	44
Persian	64.96	1.50	6	104	6	30
Polish	74.20	4.11	9	813	189	56
Swedish	65.27	3.50	6	154	23	39
Turkish	63.35	4.39	9	857	269	62

Table 3:

%feats	percentage of nodes with FEATS
|feats|	average number of entries per FEATS
⌈feats⌉	maximum number of entries
#feats	number of unique FEATS
#feats!	number of unique FEATS with unique appearances
#entry	number of unique entries in FEATS

illustrated by the `#entry` column.

The morphological annotation may have language-specific variations. An attribute can be layered, for example with `Gender=Masc|Gender[psor]=Fem`, both the gender of the noun and the gender of the possessor are marked, in which case we treat them as different attributes. The value part of an entry may be composite such as `Gender=Fem,Masc`. Entries like this rarely occur, thought when they do, they appear as regular members within the treebank.[7] We could treat an AVS as a multimap, where multiple values may coexist under the same attribute. However, this is only sensible when composite entries arise due to ambiguities, which is not always the case. They may exist because of the language and the annotation scheme, for example, `Int` and `Rel` always occur together under `PronType` in the Polish treebank, and `Aux` and `Cop` always occur together under `VerbType` in the Dutch treebank. For simplicity, we treat such composite values as new

[7] 1–4 such entries are present with an average frequency of 1267 in Croatian, Dutch, Hebrew, Latin-PROIEL, and Polish. An exception is Finnish-FTB, where 8 such entries exist, all under the `Clitic` attribute, among which 5 are irregular, with frequencies of 1–3.

atomic values, and an AVS as a simple associative structure.

With these considerations in mind, we devised five treatments.

Atomic

The entire AVS is treated as an atomic symbol.

The CoNLL-U format requires the entries and values of FEATS be sorted alphabetically, which guarantees a bijection between the AVS and the string representation. That string representation is treated as an atomic symbol. These atomic symbols form a nominal category which needs to be treated as an open class in practice, and so members with unique appearances are replaced with the symbol for unknowns. The symbol for dummy values and the symbol for the pseudo root are also added into this nominal category. Then an embedding matrix is trained as part of the model.

The number of parameters in the embedding matrix can be considerable. The number of rows (`#feats` − `#feats!` + 3) is in the magnitude of 10^2 for most treebanks. However, the choice of the column dimension, i.e. the dimension of the AVS vectors, has heavier overall impact on the number of parameters in the model, because 18 such vectors are fully connected with 256 ReLU units (Section 4), which means that each dimension adds 18×256 parameters in the first hidden layer.

This is the most common treatment adopted by dependency parsers with a similar architecture, such as the MaltParser (Nivre, Hall, and Nilsson 2006) and Parsito (Straka, Hajič, Straková, and Hajič jr. 2015).[8]

Binary

The AVS is deconstructed and encoded as a vector of binary values.

Here we consider the nominal category of morphological entries present in the train-set, plus the dummy and the root symbols. Each nominal value is assigned a disparate dimension in the target vector space, and each AVS is embedded in the target space as a vector of binary values indicating the presence or absence of the corresponding nominal values. In case some unseen entry appears in the dev- or test-set, it is simply ignored. An AVS containing only unknown entries is mapped to the zero vector.

This treatment can be more expensive than the atomic one, despite not adding a trainable embedding matrix. The AVS vectors, while not of a very large dimension (`#entry` + 2), are sparse (mostly zeros), which means that the dense vectors in the atomic treatment could encode the same amount of

[8] `http://www.maltparser.org/` `https://ufal.mff.cuni.cz/parsito`

information with a lower dimensionality. Take the Latin-PROIEL treebank for example. It has a large number of unique FEATS and a medium number of unique entries. The binary treatment adds 211 968 parameters to the model, all in the first hidden layer, while the atomic treatment with $32d$-embedding adds 168 704 in total.

One-hot

Similar to the binary treatment, except that the embedded vectors are concatenated one-hot vectors, guaranteed to have the same norm.

Here each attribute forms its own nominal category, consisting of all possible values for that attribute, plus a sentinel value for when that attribute is inapplicable, which guarantees that the image of its embedding are one-hot vectors. The dummy and the root symbols form a special category.

This treatment is notably more expensive than the binary one, because of all the sentinel values. It is also slower in feature construction, because of the complicated procedure. Its advantage is having all AVS vectors equal in norm, while the binary vectors vary according to the number of entries.

attribute	value	binary
-	Dummy	0
	Root	0
Number	Plur	1
	Sing	0
Person	1	1
	2	0
	3	0

attribute	value	one-hot
-	-	1
	Dummy	0
	Root	0
Number	-	0
	Plur	1
	Sing	0
Person	-	0
	1	1
	2	0
	3	0

Table 4: `Number=Plur|Person=1` in binary and one-hot treatments, assuming only the listed entries are observed in the train-set of the treebank.

Summed

The AVS is the sum of its entries.

Still we consider the nominal category of entries plus the dummy and the root symbols. An embedding matrix is trained as part of the model, mapping the entries to real vectors. The entry vectors compose the AVS vector

as their componentwise sum. An AVS containing only unknown entries is mapped to the zero vector, as with the binary treatment.

The following calculation demonstrates the summed treatment for the example in Table 4.

$$\begin{aligned}
&\text{input feature: } F = \begin{bmatrix} 0 & & & \cdots & & \\ & 0 & & & & \\ & & 1 & & \ddots & \vdots \\ & & & 0 & & \\ \vdots & & \ddots & & 1 & \\ & & & & & 0 \\ & & \cdots & & & & 0 \end{bmatrix} && \in \mathbb{R}^{7\times 7} \\
&\text{entry embedding: } E && \in \mathbb{R}^{d\times 7} \\
&\sum_{1\le j\le 7} (EF)_{\,i,j} = E_{i,3} + E_{i,5} && \in \mathbb{R}^{d}
\end{aligned}$$

This treatment is less expensive than the previous ones. The size of the embedding matrix is negligible, and the dimension of the dense AVS vectors can be low. The computation can also be implemented efficiently: Apply the sum operation to the embedding matrix along the input axis with absent entries masked.

Maxout

Same as the summed treatment, only replaces the sum operation with max.

This is equivalent to a maxout layer (Goodfellow et al. 2013). Each row of the entry embedding matrix corresponds to a maxout unit. A row transforms the input feature matrix into a vector, and the maxout activation reduces the vector down to a scalar. However, same with the summed treatment, the actual implementation does not require constructing the input feature as a matrix, but simply as a masking vector.

7 Experiments on encodings

We investigated how different methods of encoding FEATS impact the parsing accuracy. At this stage, we did not include any lexical feature such as

FORM or LEMMA. Lexical information could partially replace morphological information. The inclusion of lexical features decreases the impact of morphological representations, while increasing the impact of random initialization by adding a substantial amount of parameters into the model, which complicates the analysis of the results. Therefore the baseline model was only given UPOSTAG and DEPREL as inputs, via $12d$- and $16d$-embeddings respectively.

The five treatments discussed in Section 6 were implemented with variations. The binary treatment produces vectors with varying norms, for which we tried different normalization procedures to project them on unit n-spheres. L^1-normalization for a binary vector can also be interpreted as a transformation into a probability distribution over entries, which was proposed by Alberti, Weiss, and Petrov (2015) and dubbed set-valued features. The varying norm problem does not exist in the one-hot treatment, and is not disruptive in the atomic or the maxout treatment, but it is disruptive in the summed treatment, for which we tried averaging and normalizations. In total, we tried ten methods plus the baseline.

0. `Baseline`
1. `Atomic:` $32d$`-embedding`
2. `Binary`
3. `Binary:` L^1`-normalized`
4. `Binary:` L^2`-normalized`
5. `One-hot`
6. `Summed:` $32d$`-embedding`
7. `Summed:` $32d$`-embedding, averaged`
8. `Summed:` $32d$`-embedding,` L^1`-normalized`
9. `Summed:` $32d$`-embedding,` L^2`-normalized`
10. `Maxout`

For each treebank and each method, we trained a model for 25 epochs on the train-set. After each epoch, we took the model to parse the dev-set, and calculated the LAS.[9]

Method 6 was numerically unstable and failed to train for all but the Chinese treebank. The reason why the Chinese treebanks was spared is clear from the [`feats`] column in Table 3.

[9] We also checked the unlabeled attachment scores (UAS) and came to the same judgments. The data are available in csv format: https://github.com/ysmiraak/darc/tree/master/thesis/data

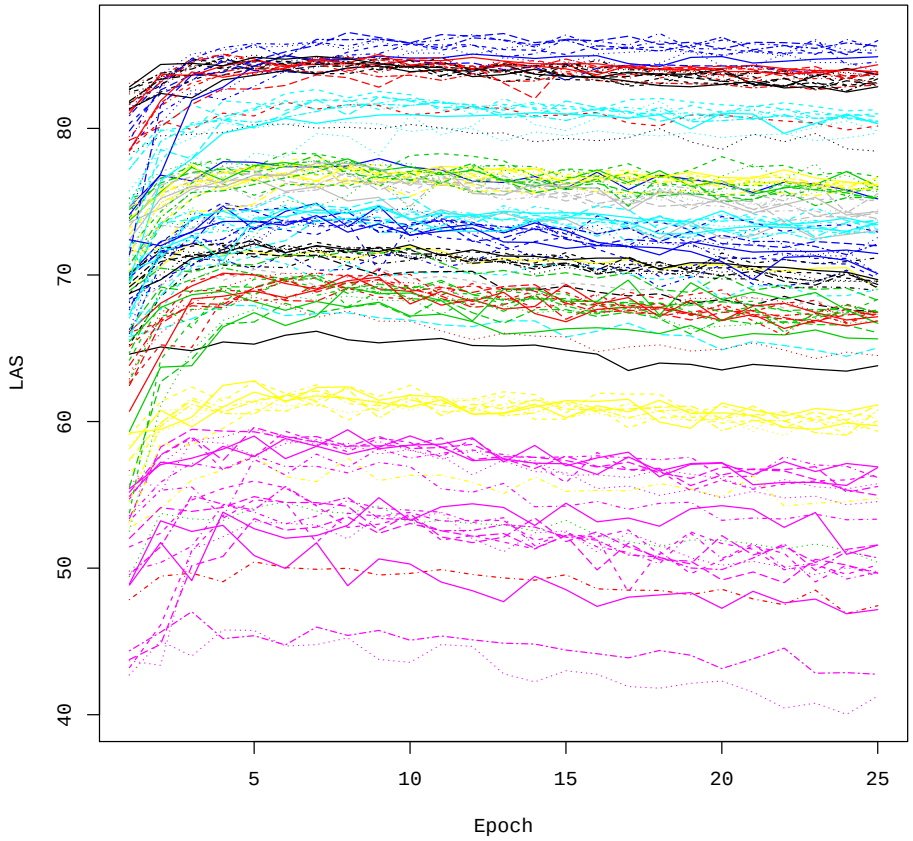

Figure 3: Training processes (excluding method 6). A color for each treebank.

Figure 3 shows that for most processes, no apparent over-fitting occurred, but mere slight fluctuations, and that for most treebanks, the scores produced by different methods are extremely close. For these reasons, we took the top three scores from each process, instead of only the best ones. This gives us more data points for analysis, and also counters random luck whereby some processes happened to peak at the end of a training epoch while others passed their peaks in the middle of an epoch.

The scores are not comparable across treebanks, and our goal was to investigate if some methods are consistently better than others. Therefore instead of testing for each treebank whether one method is significantly

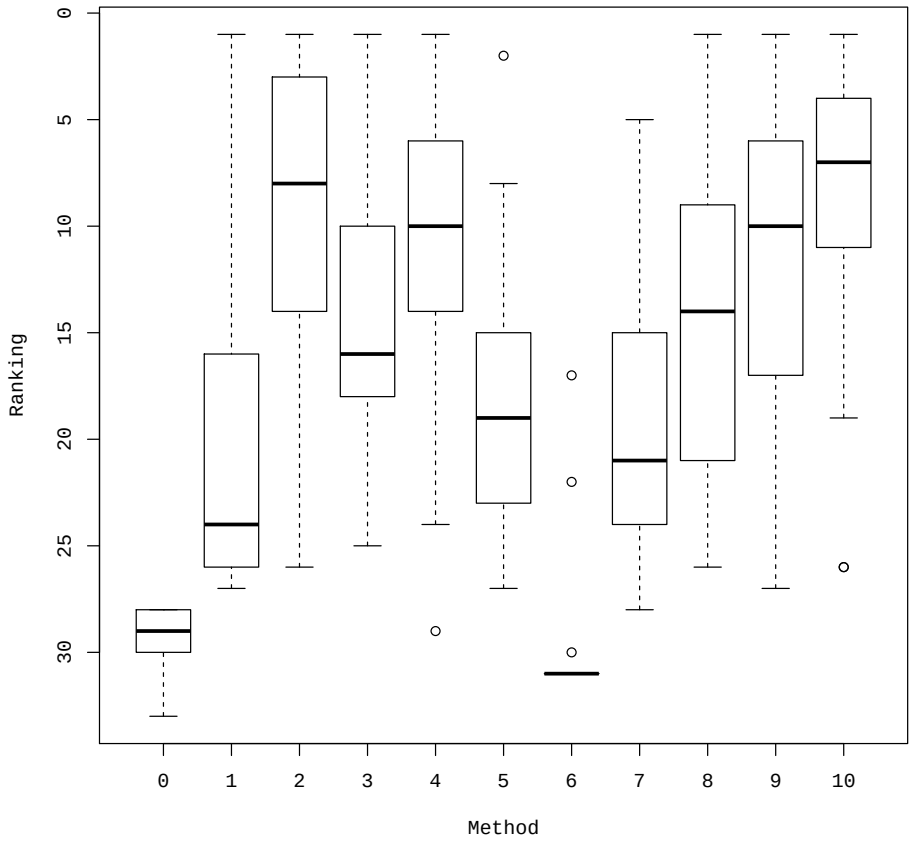

Figure 4: Ranking of methods.

better than another by some ad hoc definition of significance, we simply compared the ranks. The top three scores from each processes were ranked within each treebank,[10] and the ranks were grouped by the methods.[11]

In Figure 4, the low position of the baseline method (0) shows that morphological features are helpful for all treebanks, including Chinese. It is also evident that the binary methods (2–4), the normalized summed methods (8–9), and the maxout method (10) are better than the others. For the

[10]Taking the top one score leads to the same judgments, only with the differences appearing to be more extreme.

[11]The scores were rounded to two decimals. Equal scores were given equal ranks.

binary methods, normalization seems unnecessary, while for the summed methods, L^2-normalization appears to have the best potential.

We picked the unnormalized binary method, the L^2-normalized summed method, and the maxout method for further investigations. From this point forward, the first two are simply referred to as the binary and the summed methods.

8 Further investigations

We continued the contest between the binary, the summed, and the maxout methods while augmenting the model with lexical features and applying regularizations to the trainable embeddings.

For adding lexical features, we tried two arrangements.

- `FORM, 64`d`-embedding`
- `FORM & LEMMA, 32`d`-embedding each`

This means that the dimension of the real vector space where we embed the lexical representations is fixed to 64. The lexical space is either entirely dedicated to FORM, or split between FORM and LEMMA. This way the number of parameters in the first hidden layer stays constant. Adding LEMMA on top of FORM actually decreases the total number of parameters, because the LEMMA embedding matrix has fewer rows (`#form*` and `#lemma*` in Table 2).

For applying regularizations to the trainable embeddings, we tried three arrangements.

- `Nothing`
- `Unit-norm constraint`
- `Unit-norm constraint, 25% dropout`

The unit-norm constraint normalizes the rows of the embedding matrix to unit vectors after each update.[12]

In total, we conducted 19 trials. The ranking of the results are plotted in Figure 5. The unit-norm constraint makes embedding training more effective (7–12 vs 1–6). Dropout further helps (13–18 vs 7–12). Without regularizations, adding LEMMA seems to have made no difference (4–6 vs

[12]The max-norm constraint may be a better option, which only enforces an upper bound on the norm. However, the optimal upper bound is a dataset-specific hyperparameter, which we wish to avoid.

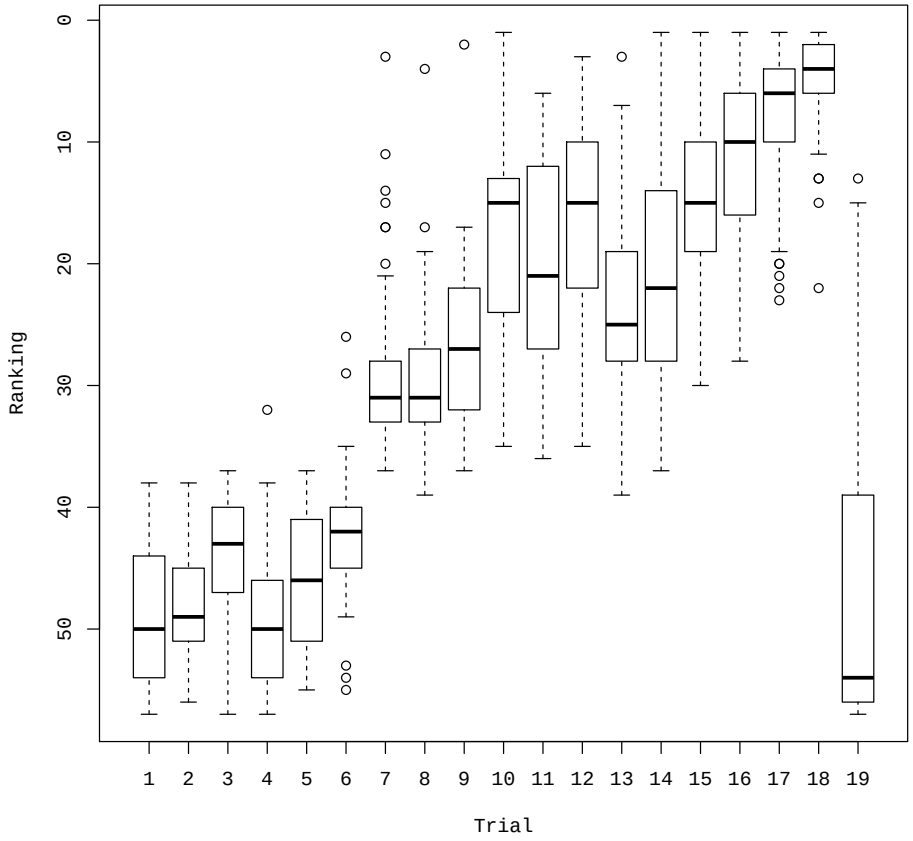

Figure 5:

trial	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
binary	•			•			•			•			•			•			
summed		•			•			•			•			•			•		
maxout			•			•			•			•			•			•	
+LEMMA				•	•	•				•	•	•				•	•	•	•
unit-norm							•	•	•	•	•	•	•	•	•	•	•	•	•
dropout													•	•	•	•	•	•	•

1–3); It is however clearly superior, when regularizations are added (10–12 vs 7–9, 16–18 vs 13–15). Trial 19 is the control group for 16–18, without using inputs from FEATS. It is evident that morphological features are helpful even with the presence of lexical features.

We took the best three models (16–18) for each treebank, picked the parameters from the epoch which yielded the best LAS on the dev-set, and

	epoch			dev LAS			test LAS		
treebank	b	s	m	b	s	m	b	s	m
Ancient_Greek	18	17	18	66.54	**67.76**	*67.45*	66.25	**67.23**	*67.10*
Arabic	10	24	17	79.36	*79.52*	**79.70**	78.05	**79.23**	*79.17*
Basque	20	19	24	76.61	*77.35*	**77.44**	76.47	*77.96*	**78.43**
Bulgarian	24	16	24	88.20	*88.91*	**89.17**	88.53	*88.64*	**89.00**
Chinese	23	24	23	*77.74*	77.43	**77.79**	78.95	*79.70*	**79.94**
Croatian	14	24	13	81.94	*82.25*	**82.41**	82.36	*82.56*	**82.61**
Dutch	22	20	19	82.78	**83.34**	*83.14*	78.85	*79.20*	**79.81**
Finnish-FTB	19	24	18	**87.89**	87.53	*87.63*	*86.95*	86.79	**87.08**
Hebrew	23	19	14	83.50	*83.70*	**83.72**	*82.18*	**82.53**	81.81
Italian	20	23	19	**88.94**	88.72	*88.86*	*88.89*	**89.19**	88.76
Latin-PROIEL	16	17	23	76.32	**76.64**	*76.60*	74.90	*74.97*	**76.68**
Persian	19	19	15	*83.30*	83.21	**83.54**	82.51	**83.29**	*82.97*
Polish	13	23	23	89.50	*89.94*	**90.11**	88.98	*90.05*	**90.41**
Swedish	14	21	15	**81.62**	81.51	**81.62**	**84.10**	83.96	*84.08*
Turkish	8	20	18	62.23	**62.99**	*62.95*	**62.96**	*62.92*	61.90

Table 5: Best dev models.
b: binary, s: summed, m: maxout.

parsed the test-set. The results listed in Table 5 shows that in general the maxout method outperforms the summed method which in turn outperforms the binary method. In fact, the dimension of the entry embedding in the maxout and the summed methods can be tuned for a better performance.

9 Conclusion

We have shown that morphological information is valuable in transition-based dependency parsing, and that a most effective and efficient way to encode morphological annotations is training an embedding from morphological entries to real vector space as part of the model, and construct the vector representation for an AVS from its entries, through scalar composition such as the max operation.

We participated in the CoNLL 2017 Shared Task with our parser, darc (Yu et al. 2017), and achieved results comparable to UDPipe 1.0 and 2.0 systems (Straka, Hajič, and Straková 2016),[13] despite the many limitations of our system in contrast. We did lose to UDPipe 2.0, however at the time, we applied the binary method without the knowledge that the maxout

[13] https://ufal.mff.cuni.cz/udpipe

	UAS		LAS	
treebank	UDPipe 2.0	darc maxout	UDPipe 2.0	darc maxout
Ancient_Greek	69.2	**71.6**	64.4	**67.1**
Arabic	82.9	**83.8**	77.9	**79.2**
Basque	**82.3**	82.1	78.4	78.4
Bulgarian	**92.6**	92.1	**89.1**	89.0
Chinese	**84.1**	82.8	**81.4**	79.9
Croatian	**87.1**	86.6	**83.2**	82.6
Dutch	82.9	**83.7**	79.4	**79.8**
Finnish-FTB	88.8	**89.2**	86.5	**87.1**
Hebrew	**87.8**	85.6	**84.3**	81.8
Italian	**91.3**	90.4	**89.7**	88.8
Latin-PROIEL	79.0	**80.2**	75.0	**76.7**
Persian	**87.7**	86.0	**84.9**	83.0
Polish	92.9	**93.4**	89.5	**90.4**
Swedish	**88.0**	87.1	**85.0**	84.1
Turkish	66.8	**67.3**	61.1	**61.9**
average	**84.2**	84.1	80.7	80.7

Table 6: Darc vs UDPipe.

method would be better. Table 6 compares the parsing scores of the two systems given the gold-standard segmentation and tagging information.

Better parsing algorithms and architectures have been proposed, and our method for encoding morphological annotations is applicable to any neural network models. However, good morphological annotations may not always be available. Bojanowski et al. (2016) proposed a method for constructing morphological word representations by learning representations for character n-grams and composing words as the sum. Their method extracts morphological information directly from the orthography, a natural source of linguistic annotations, and our method utilizes an external source, namely the morphological tags. Both methods compose the vector representation for a complex structure from the vector representations of its components by an componentwise operation. This approach is generally worth considering in machine learning where some input comes with polymorphic information, such as instances with soft clusters or types with type classes.

References

Nivre, Joakim, Johan Hall, and Jens Nilsson (2006). "Maltparser: A data-driven parser-generator for dependency parsing". In: *Proceedings of LREC*. Vol. 6, pp. 2216–2219.

Nivre, Joakim (2008). "Algorithms for deterministic incremental dependency parsing". In: *Computational Linguistics* 34.4, pp. 513–553.

— (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*. Association for Computational Linguistics, pp. 351–359.

Nivre, Joakim, Marco Kuhlmann, and Johan Hall (2009). "An improved oracle for dependency parsing with online reordering". In: *Proceedings of the 11th international conference on parsing technologies*. Association for Computational Linguistics, pp. 73–76.

Goodfellow, Ian J et al. (2013). "Maxout networks". In: *arXiv preprint arXiv:1302.4389*.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Chen, Danqi and Christopher D Manning (2014). "A Fast and Accurate Dependency Parser using Neural Networks." In: *EMNLP*, pp. 740–750.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Alberti, Chris, David Weiss, and Slav Petrov (2015). "Improved transition-based parsing and tagging with neural networks". In:

Chollet, François et al. (2015). *Keras*. https://github.com/fchollet/keras.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

Ling, Wang et al. (2015). "Two/Too Simple Adaptations of word2vec for Syntax Problems". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Hu-*

man Language Technologies. Denver, Colorado: Association for Computational Linguistics.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: http://tensorflow.org/.

Straka, Milan, Jan Hajič, Jana Straková, and Jan Hajič jr. (2015). "Parsing Universal Dependency Treebanks using Neural Networks and Search-Based Oracle". In: *Proceedings of Fourteenth International Workshop on Treebanks and Linguistic Theories (TLT 14)*.

Bojanowski, Piotr et al. (2016). "Enriching word vectors with subword information". In: *arXiv preprint arXiv:1607.04606*.

Nivre, Joakim, Marie-Catherine de Marneffe, et al. (2016). "Universal Dependencies v1: A Multilingual Treebank Collection". In: *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. Portorož, Slovenia: European Language Resources Association, pp. 1659–1666. ISBN: 978-2-9517408-9-1.

Straka, Milan, Jan Hajič, and Jana Straková (2016). "UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing". In: *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. Portorož, Slovenia: European Language Resources Association. ISBN: 978-2-9517408-9-1.

Nivre, Joakim et al. (2017a). *Universal Dependencies 2.0*. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, http://hdl.handle.net/11234/1-1983. URL: http://hdl.handle.net/11234/1-1983.

Nivre, Joakim, Željko Agić, Lars Ahrenberg, et al. (2017b). *Universal Dependencies 2.0 – CoNLL 2017 Shared Task Development and Test Data*. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. URL: http://hdl.handle.net/11234/1-2184.

Yu, Kuan et al. (2017). "The parse is darc and full of errors: Universal dependency parsing with transition-based and graph-based algorithms". In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, pp. 126–133. URL: http://www.aclweb.org/anthology/K17-3013.

Zeman, Daniel et al. (2017). "CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies". In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Uni-*

versal Dependencies. Vancouver, Canada: Association for Computational Linguistics, pp. 1–19. URL: http://www.aclweb.org/anthology/K17-3001.