Eberhard Karls Universität Tübingen
Seminar für Sprachwissenschaft

Enriching Topological Field Tagging with Clause Information

Bachelor Thesis

presented to the examination office
of the philosophical faculty

Author: Sebastian Pütz

Supervisor: Dr. Daniël de Kok

Contents

1 Introduction **1**

2 Clause and Field Structure of German **3**
2.1 Overview 4
2.2 Challenging Phenomena 5

3 Related Work **6**
3.1 Constituency Parsing 6
3.2 Topological Field Chunking 7
3.3 Topological Field Tagging with Neural Networks 7

4 Classifying Clause Relations **10**

5 Models **11**

6 Evaluation **12**
6.1 Data 13
6.2 Preprocessing 14
6.3 Results and Discussion 16

7 Conclusion **24**

A Hyperparameters **29**
A.1 Word embeddings 29
A.2 Topo and Clause models 29

Abstract

We explore a novel scheme of modeling clause information by defining *is-in-same-clause* relations between token pairs in order to enrich the predictions of a topological field tagger. In this work, we develop a fast and accurate model to classify these relations and evaluate it against a state-of-the-art constituency parser. Moreover, we show that a statistical transition-based dependency parser benefits from the information provided by our model.

1 Introduction

While German is known to be a language with a relatively free word order and therefore complex syntactical structure, there are certain regularities on a clause level that are traditionally explained by the topological field model (Herling, 1821; Erdmann, 1886; Drach, 1937; Höhle and Schöne, 1986). Depending on the position of verbal components, a relatively fixed sequence of topological fields is found, where some fields impose limitations on what can be placed inside.

Predicting the topological field structure of a sentence is a fruitful first step for parsing its full syntactical structure. In constituency parsing it has been used to reduce the search space of an HPSG algorithm (Frank et al., 2003) and in dependency parsing it has been demonstrated by de Kok and Hinrichs (2016) that the local decisions of a transition-based statistical dependency parser can be improved through information about the topological field structure.

Dipper (2014) showed on the TIGER corpus (Brants et al., 2002) that, although full syntactic annotations are present, it is not a trivial task to infer the correct topological structure. Currently the only large scale, hand-annotated treebank including topological fields is the TüBa-D/Z (Telljohann et al., 2004). Thus, linguists, interested in e.g. extracting statistics over the topological structure of sentences, are limited to this treebank.

The work on predicting the topological field structure of sentences can be divided into three strains: Parsing, Chunking and closely related to chunking, Sequence Tagging. Kübler (2005); Maier (2006); Cheung and Penn (2009) trained probabilistic constituency parsers to predict topological field structure, Veenstra et al. (2002); Liepert (2003) applied different methods such as Finite State Transducers and Memory Based Learners to topological field chunking and de Kok and Hinrichs (2016) employed a neural sequence tagging architecture to topological field modeling. In parsing, the field structure is either predicted implicitly as part of the full constituency structure or explicitly by only predicting the clause and field nodes.

Constituency parsers have to make a trade-off between expressive power and speed; they are either restricted to projective structures or otherwise have to deal with high runtime. Typically, the constituency structure of German sentences is not projective, 43.69% of the sentences in TüBa-D/Z include crossing edges. While the non-projectivity is mostly limited to punctuation, there remains a

fair amount of other content, like parenthetical constructions, that introduces crossing edges.

Sequence taggers, in contrast, only assign a topological field tag to each token, these field tags yield no information with regard to the underlying structure that these fields were derived from. All hierarchy is lost because taggers only predict flat sequences, but those sequences can be based on non-projective trees.

Considering information available at the token level, a difference between the two approaches becomes apparent: the sequence tagging approach directly associates structural information with each token, while the parsers provide tree structures that such information can be derived from. Surface level information, as given by sequence taggers, plays a role in token-driven analyses of sentences like transition-based dependency parsing. Transition-based dependency parsers build a tree representing the syntactic structure of a sentence by introducing directed and labeled relations between tokens. Thus, they have to decide, solely based on features available for tokens or token pairs, whether to introduce a dependency relation or not.

Token-driven dependency parsers have recently been used to produce dependency treebanks based on large-scale web corpora (Schäfer, 2015; de Kok, 2014). Although sentences with hundreds of tokens are not particularly frequent in newspaper texts that are usually used to develop NLP systems, they are somewhat prevalent in texts from the web. For this reason, the runtime complexity for all involved systems is to be considered. If a dependency parser should, for instance, be supplied with information about clause structure in addition to topological field information, both could be derived from constituency trees. Constituency parsers often have the undesirable property of cubic runtime growth, which makes it hardly feasible to employ such a parser to provide the dependency parser with structural information. The runtime of sequence taggers, on the other hand, only grows linearly in relation to sequence length and they provide a flat sequence of topological field tags that are associated with tokens and can directly be used by the dependency parser. Unlike the constituency parser, a sequence tagger cannot easily provide clause information and is thus also not fit to provide information about clause membership.

As a solution, we propose binary *is-in-same-clause* relations between tokens as a novel scheme to directly model the clause structure of sentences without any overhead. The definition of this scheme is quite simple and almost trivial: If two tokens are located within the same clause, the relation holds. In that sense only the annotation of the clause structure is required to train a classifier, which requires far less manual work than annotating a corpus with constituency trees and in turn opens our scheme to low resource languages, where large treebanks with constituency annotations might not be prevalent.

In this thesis, we extend the recent work in topological field tagging by de Kok and Hinrichs (2016) who used a neural sequence tagger to assign a topological field label to each token in a sentence. We use this sequence tagger's output to determine the *is-in-same-clause* relation of any token pair with an accuracy higher than 97%.

Our model is best fit when downstream applications, like the previously

described transition-based dependency parsers, only require local information about the clause structure. Moreover, our model can be effiently applied to searching a dependency treebank, since such queries typically rely on token-level features or relations between tokens.

The contributions of this work can be enumerated as:

- Novel scheme for modeling binary clause relations between Tokens
- Developing a fast and accurate model to predict these relations.
- Evaluation of our model and comparison to a constituency parser.
- Application of our model to dependency parsing.

Following this introduction, we will first describe the clause and field structure of German sentences before going into existing approaches and topics related to topological field and clause prediction. After diving into the related work, we will discuss our novel scheme to model clause relations and finally introduce our own models. Then we will shortly describe the data used for our experiments and report our preprocessing procedures. The main part of this thesis will be finished by evaluating the models considered in this work. This evaluation will mostly be intrinsically and quantitative, but will also include an extrinsic part where we test the different models' predictions on a dependency parser. The thesis will close with the conclusion and an outlook to future work.

2 Clause and Field Structure of German

German Clause Types					
V1		LK Ist	(MF) er das	(VC) gewesen?	(NF)
V2	VF Das	LK ist	(MF) er	(VC) gewesen.	(NF)
VL		(C) dass	(MF) er das	VC gewesen sei.	(NF)

Table 1: Overview of German clause types with some simplifications. Usually punctuation is not annotated within the topological field structure but is included here for demonstration purposes. Non-obligatory fields are parenthesized. *VF - pre field, LK - Left sentence bracket, C - Complementizer, MF - middle field, VC - verbal complex, NF - post field*

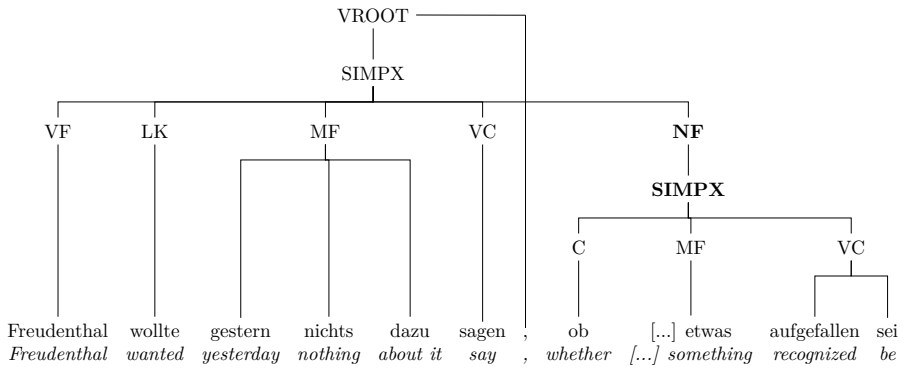

Figure 1: Clause and field structure with an embedded clause. Some words and sentence final punctuation are elided for space reasons.
Sentence 20 from TüBa-D/Z: *"Freudenthal did not want to say anything yesterday about, whether she found anything [...]"*

2.1 Overview

German clauses can be divided into three types, each one corresponding to a specific position of the finite verb: verb-initial (V1), verb-second (V2) and verb-last (VL). Within the clause types, the sequence of topological fields is predictable (Höhle and Schöne, 1986) (see Table 1 for a simplified overview). In V1 and V2 clauses, the finite verb component occupies the left sentence bracket (*Linke Satzklammer* – LK) while the non-finite verb parts are found in the right sentence bracket (*Rechte Satzklammer* or *Verb Komplex* – VC). In VL clauses, the LK is replaced by the complementizer (C). Together, the sentence brackets delineate the other topological fields.

The prefield (*Vorfeld* – VF) precedes the LK and allows just a single constituent. This constituent is often the subject of the clause but can also be replaced by virtually any other constituent if the subject is realized in the middle field (*Mittelfeld* – MF). The MF is located between the sentence brackets, the LK to the left and if present the VC on the right. If no VC is given, the right boundary for the MF is set by the postfield (*Nachfeld* – NF). The MF can contain multiple constituents which often include verb arguments, adverbs and the like. The NF is located at the end of clauses and allows arbitrary numbers of constituents. It is the place where clausal constituents, e.g. relative clauses, with their own topological field structure are often found. Additionally, a coordination field (KOORD) preceding all other fields is assumed for conjunction on the clause level. Finally, a field for the left-dislocated phrase in resumptive construction (*Linksversetzung* – LV) is placed between KOORD and the initial field in V1 and V2 sentences. Telljohann et al. (2006, chap. 3.1.1) offers a summary of the above described structures.

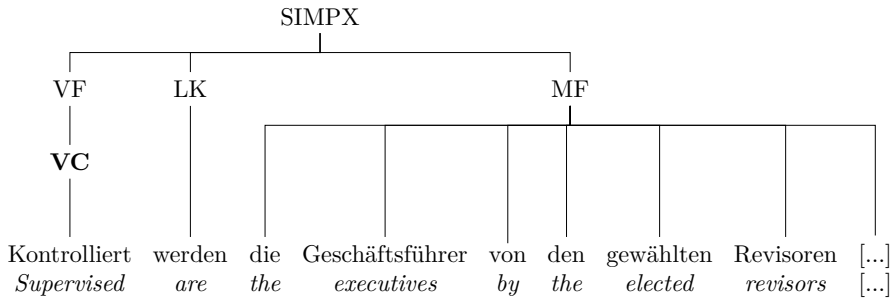

Figure 2: Topological field structure with topicalized VC. End of the sentence is elided for space reasons.
TüBa-D/Z sentence 19: "*The executives are supervised by the elected revisors [...]*"

2.2 Challenging Phenomena

In Table 1 and the previous section, we described a rather simple, sequential picture of clause structure in German. In the following, we will briefly introduce some phenomena that complicate the topological field model, such as recursion, coordination and topicalization.

Recursion. Since subordinate clauses participate in the topological structure of the matrix clause but also include their own topological structure, the topological structure of German is recursive and hierarchical. The topological field model only makes assumptions about the sequence of fields within a clause. Through the recursive nature of the structures, the surface sequence of topological fields can thus differ on a sentence level. See Figure 1 for an example.

Fronting. The surface sequence of topological fields within a clause also does not always adhere to the previously described sequences. For example, topicalization is a challenge since it moves fields or parts of fields into the VF at the beginning of the clause. Figure 2 shows part of a sentence including a fronted VC. In a non-topicalized variant the subject "die Geschäftsführer" would be placed in the VF and the VC would be moved to a position after the MF, yielding the for declarative sentences expected V2 sequence: "Die Geschäftsführer werden von den gewählten Revisoren kontrolliert [...]".

Coordination. Furthermore, coordination can be described on the topological field layer, resulting in sequences different from the one predicted by the topological field model. Figure 3 shows a tree where the subject "Die neue Addresse", located in the VF of the clause, is shared by the verbs "war" and "wird". Both finite verbs are located in their own LK and are followed by their respective arguments in the MFs. According to the V2 sequence described in Table 1, the clause should end after the MF of the first conjunct, but instead,

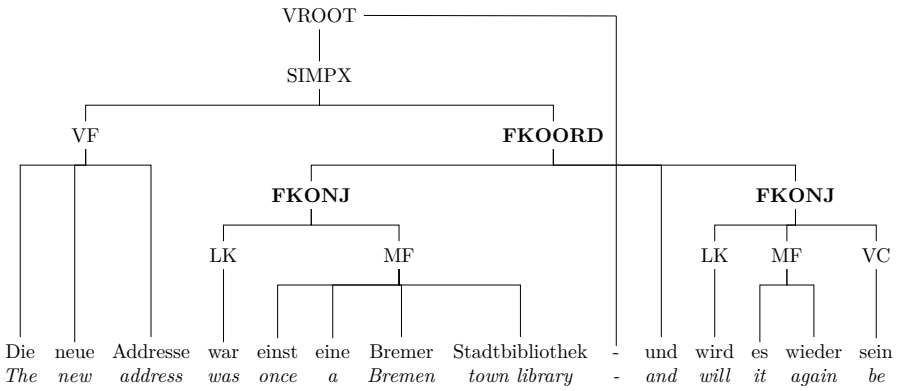

Figure 3: Topological field structure with field coordination.
TüBa-D/Z sentence 3923: "*The new address was once a Bremen town library - and will be it again.*"

the sequence moves back to the LK and even moves one step further to a VC in the final conjunct.

Coordination of elliptical constructions as in the example above are hard to encode in terms of the topological field model as they could be described as two clauses in one. The conjuncts cannot be classified as clauses of their own, as one of the two would then lack its subject.

3 Related Work

In this section, we will discuss fields and methods that are relevant for modeling clause and topological field structure such as constituency parsing, chunking and neural sequence tagging.

3.1 Constituency Parsing

Constituency parsing is the process of constructing a constituency tree from a sentence, representing the underlying syntactic structure according to a phrase structure grammar. There are no inherent assumptions about what kind of categories can be part of such a grammar, as such clause and topological structure can be encoded just as part of the grammar, or the grammar can be composed solely of the clause and topological categories.

Kübler (2005) and Maier (2006) are accounts of treating topological field parsing as an intrinsic task in constituency parsing. Both trained PCFG parsers that learned the topological structure embedded in the constituency structure.

Cheung and Penn (2009) on the other hand frame topological field parsing as a form of shallow parsing where the task is to retrieve the clause and field layers of constituency trees, thus omitting the phrasal nodes. They performed

experiments with the then state-of-the-art Berkeley parser (Petrov et al., 2006), a latent variable-based parser. Latent variable-based parsers see a treebank as a coarse approximation of an underlying, ideal, grammar. Through repeatedly splitting and merging the categories, a better fit grammar is derived.

Recently a number of neural constituency parsers following an encoder-decoder scheme have been proposed. In this scheme, an encoder produces a vectorial, contextual, representation for each word in the sentence. Based on these vectors a decoder then constructs a parse tree.

Stern et al. (2017) and Gaddy et al. (2018) use Long Short-Term Memory networks (LSTM) (Hochreiter and Schmidhuber, 1997) as an encoder architecture, while Kitaev and Klein (2018) apply a self-attentive encoder (Vaswani et al., 2017) to compute contextual representations for the tokens. All of these parsers use the output vectors of the encoder to represents spans in a sentence. Based on those span representations a CYK (Younger, 1967) style chart-decoder finally constructs the constituency trees. The time complexity of the decoder is $O(N^3)$ where N refers to sequence length.

All models described in this section require projective trees that do not include crossing edges. Maier (2010) describe a parser that is capable of predicting non projective constituency trees, but report extensive runtimes, growing exponentially with sequence length.

3.2 Topological Field Chunking

A simple approach to modeling the topological field structure of a sentence is to identify the sentence brackets LK and VC and define the other fields in relation to the brackets. This approach has been referred to as topological field chunking and it has been explored by several researchers.

Veenstra et al. (2002) applied a cascade of Finite State Transducers, a PCFG parser and a memory based learner to identify the sentence brackets and complementizers. Similarly, Liepert (2003) used Support Vector Machines to chunk German sentences into the top-most topological fields. They approach the problem as a sequence tagging task and assign to each token either LK, VC, C or O. While this approach does not produce a tree structure, it allows to draw some conclusions with regard to the sentence structure, namely, after identifying the sentence brackets, it is roughly possible to set the field boundaries in relation to them. This breaks down in the presence of embedded clauses or other phenomena where the field structure is not clear just from the location of the sentence brackets.

3.3 Topological Field Tagging with Neural Networks

Artificial Neural Networks (ANNs) are a family of non-linear mathematical models, the simplest form of such an ANN defines a function:

$$f(\mathbf{x}) = g(\mathbf{x}\mathbf{W} + \mathbf{b}) \tag{1}$$

where $\mathbf{x}$ is a d-dimensional input vector, $\mathbf{W}$ a $d * m$-dimensional matrix and $\mathbf{b}$ a m-dimensional vector, with $\mathbf{W}$ and $\mathbf{b}$ being learned parameters. g is some non-linear activation function such as the Rectified Linear Unit (ReLU, Nair and Hinton, 2010).

Deep feed-forward ANNs are composed of multiple such layers where each layer takes the previous layer's output as input. Usually the parameters of each layer are not shared. For instance, a n-layered feed forward network can be formulated as:

$$FF_n = f_n([...](f_2(f_1(\mathbf{x})))) \tag{2}$$

Feed-forward networks require fixed-width inputs, i.e., following Equation 1, the input $\mathbf{x}$'s dimensionality must match the dimensions of $\mathbf{W}$ in order for the matrix multiplication to succeed. Thus, the only reasonable way to apply such simple feed-forward models to sequence labeling employs a sliding-window technique where a fixed number of following and preceeding representations is concatenated with the representation of the symbol to be classified. For each position, this results in a vector of the same size which can then be classified by the model. While it is feasible to train such a network and some context can be captured, it "forgets" symbols that the windows has moved past and does not take information into account from symbols that are not reached by it yet. For instance, in topological field tagging, it is important to keep track of how far the sequence of fields (see Table 2 has advanced and whether the sentence brackets have been assigned. If a model has classified something as the LK of a clause, it needs to be aware of this later on in the sentence in order to not assign this same label again to another form that also appears to be a finite verb. A feed-forward network has no way to keep track of the greater context within the sentence.

Recurrent Neural Networks (RNN, Elman, 1990), on the other hand, are a class of neural networks that can both process variable length inputs and maintain an unbounded notion of memory throughout the sequence. RNNs have feedback connections that inform the model at later positions about previous outputs. At each position in the sequence, the input to the model includes both the current symbol and the output at the previous position - the previous state $\mathbf{h_{t-1}}$. Hence, its output at any point in the sequence is dependent on its history. The formulation for the output state $\mathbf{h_t}$ at a given timestep t is similar to that at Equation 1 but also includes the state $\mathbf{h_{t-1}}$, which itself is dependent on some earlier state:

$$\mathbf{h_t} = f(\mathbf{h_{t-1}}, \mathbf{x_t}) \tag{3}$$

Given that sequences that should be processed tend to be of finite length, it is possible to repeatedly substitute h_{t-1} for Equation 3 until the initial state h_0 is reached and come up with a formula rather similar to Equation 2:

$$\mathbf{h_t} = f(f(f(\mathbf{h_0}, \mathbf{x_0}), \mathbf{x_{t-1}}), \mathbf{x_t}) \tag{4}$$

The difference here is that a deep feed forward network's layers are independent of each other while there is a temporal dependence for an RNN. The common training routine for RNNs, backpropagation through time (Rumelhart et al., 1985; Werbos, 1990), uses the fact that the RNN can be unrolled in such a manner and reuses the parameters $\mathbf{W}$ and $\mathbf{b}$ at each timestep.

As the virtual depth of the network grows with sequence length, it becomes more and more challenging to actually train such a network due to the exploding and vanishing gradients problems (see Goodfellow et al., 2016, chap. 10.2). These problems were eventually addressed through LSTM networks (Hochreiter and Schmidhuber, 1997), countering vanishing gradients through gating mechanisms.

Unidirectional RNNs, as described above, provide less temporal and structural information for symbols in the beginning of the sequence. Since the output of each timestep is dependent on a previous output, an issue arises for the first symbol in the sequence. There is no previous output of the RNN to be taken into account, thus it is necessary to feed an artificial initial state. This spells out to be problematic if the decision about a symbol in the beginning is dependent on a symbol later on in the sequence. For example by projecting the topological fields in Figure 2 onto the tokens, we get the following sequence:

(1)

Kontrolliert	werden	die	Geschäftsführer	von	[...]
VC	*LK*	*MF*	*MF*	*MF*	
Supervised	are	the	executives	by	[...]

The executives are supervised by [...]

An RNN processing the sentence in a left-to-right manner would encounter the word "Kontrolliert" without any contextual information, apart from the token being at the start of the sentence. "Kontrolliert" at the beginning of a sentence can be the start of a question like:

(2)

Kontrolliert	ihr	das	auch?
LK	*MF*	*MF*	*MF*
Check	you	this	too?

Do you check this, too?

which would place "Kontrolliert" into the LK as it is the finite verb of the clause. To correctly produce the sequence in Example 1, a model has to identify "Kontrolliert" as a participle which would require another token to be the finite verb. In this case "werden" should be identified as such, but since the model is not informed about the tokens later on in the sequence, it cannot come to this conclusion.

A solution to this problem is the use of bidirectional RNNs (BiRNN, Schuster and Paliwal, 1997) that process the sentence not in a single left-to-right pass but also include a reverse, right-to-left pass that is independent of the first iteration. By concatenating the corresponding states of the forward and backward RNNs,

it is possible to get a context sensitive representation for each symbol in the sequence. In sequence tagging these representations can then be projected to the output space to assign an appropriate class to each symbol.

This architecture has been applied to various sequence labeling tasks such as part-of-speech (POS) tagging (Ling et al., 2015) and named entity recognition (Huang et al., 2015) but also to topological field tagging by de Kok and Hinrichs (2016).

de Kok and Hinrichs (2016) developed a topological field tagger in order to provide information about the field structure to a transition-based dependency parser. Transition-based dependency parsers have to make local decisions whether to introduce a dependency relation between two tokens or not. Features therefore need to be available at a token level, or at least associated with the token pair under consideration. To solve this they reduce the topological tree structure to a flat sequence by projecting the hierarchically closest topological field's label onto each token, if a token is not actually dominated by a topological field, a special *unknown* label is assigned to it.

Their model stacks a unidirectional LSTM on top of a bidirectional LSTM (BiLSTM Graves et al., 2005), the BiLSTM's input consists of concatenated word and POS embeddings. The top LSTM is followed by an output layer with a softmax activation function to obtain the probability distribution over the topological fields.

While their model is well fit to provide token level annotations that can be used by a dependency parser, it has a crucial shortcoming: The hierarchical and recursive structure of clauses is entirely lost. It is impossible to determine whether two tokens with the same field tag are actually located in the same topological field or belong to fields with the same label, but in different clauses.

4 Classifying Clause Relations

	Freudenthal	wollte	gestern	nichts	dazu	sagen	,	ob	[...]	etwas	aufgefallen	sei
Freudenthal	—	1	1	1	1	1	0	0	0	0	0	0
,	0	0	0	0	0	0	—	0	0	0	0	0
etwas	0	0	0	0	0	0	0	—	1	1	1	1

Table 2: Simplified clause relation matrix corresponding to the tree in Figure 1. Relations for one token per clause are depicted. 1 indicates the tokens are in the same clause.
Sentence 20 from TüBa-D/Z: *"Freudenthal did not want to say anything yesterday about, whether she found anything [...]"*

Clause structure is commonly treated as an abstract layer within constituency trees. For lexical approaches that require token-level annotations, such abstraction is not necessary. We propose a novel scheme to directly model clause structure by defining binary *is-in-same-clause* relations between token pairs. While these relations are not fit to infer the hierarchical structure of clauses, they allow to disambiguate the predictions of a topological field tagger

w.r.t. whether two tokens belong to the same topological field or are actually located in different clauses and thus different fields. See Table 2 for a simplified relation matrix corresponding to the tree in Figure 1.

Based on the findings of de Kok and Hinrichs (2016), we hypothesize that providing clause information in addition to topological field tags to a dependency parser yields further improvements.

It is straightforward to extract the relations from constituency trees in a treebank or from the predictions of a constituency parser, but e.g. in the previously described case of a transition-based dependency parser, it would be excessive to employ a constituency parser just to provide the information whether two tokens belong to the same clause. Our scheme enables models to directly classify whether a token pair is located in the same clause without building a tree first and inferring the relations from the tree.

Moreover, our formulation has the advantage of not requiring extensive, manual labor in order to fully annotate a constituency treebank. Our scheme merely requires the annotation of the clause-level which is far less complex than annotating full trees. In low-resource languages without large annotated corpora, our scheme is thus far more applicable.

5 Models

In the following section we will introduce our baseline model as well as our proposed new model and some variants thereof. The baseline model is implemented in the Python[1] programming language while all other models are written in Tensorflow (Abadi et al., 2016) and Rust[2] (Matsakis and Klock II, 2014). The departure point for the following models is the topological field tagger by de Kok and Hinrichs (2016).

Topo Model. As described in Section 3.3 the tagger of de Kok and Hinrichs (2016) feeds pretrained embeddings for tokens and part of speeches to a BiLSTM with a unidirectional LSTM stacked on top. We then apply, in contrast to their model, Batch normalization (Ioffe and Szegedy, 2015) to the outputs of the top LSTM. Finally, the hidden states are projected to the output space in order to retrieve the topological field tag for each token.

Baseline. Our *Baseline* model infers the clause relations on the basis of the *Topo* model. After tagging a sentence, we iterate over the tokens while tracking transitions between fields. Initially, all fields are accepted and the first clause is introduced. Subsequent tokens are only added to this clause if they have the same tag as the previous token or advance the sequence of expected topological fields (see Table 1). Finally, we add the constraint that all punctuation signs be placed in the same clause without any other content.

[1] https://www.python.org/
[2] https://www.rust-lang.org/

Joint Model. The *Joint* model builds directly on top of the *Topo* model's batch normalization and output layers. The former layer's output consists of h-dimensional vectors for each timestep, in order to predict the clause relations we concatenate these vectors with t-dimensional embeddings corresponding to the predicted topological field for each token. Since we want to predict the *is-in-same-clause* relation of a specific token with each token following it in the sentence, we create a representation of the token pair by concatenating their $h+t$-dimensional vectors resulting in a $(h+t)*2$ combined vector. Each of these token pairs is associated with a specific distance that can be derived by their position in the sentence. To encode this distance we append a d-dimensional embedding representing it and pass these $(h+t)*2+d$-dimensional vectors to a dense layer with a ReLU activation function (Nair and Hinton, 2010). The output of this dense layer is then fed to the output layer with a logistic function. While training we calculate the categorical cross-entropy losses for both tasks and minimize them jointly with the Adam optimizer (Kingma and Ba, 2014).

The variations $Joint_{noWords}$ and $Joint_{noPOS}$ are restricted to only POS embeddings or word embeddings as input. Further, for the $Joint_{noFields}$ model we do not append any embeddings for topological fields to the output of the batch normalization layer. In the $Joint_{goldTF}$ model the topological field predictions are replaced by gold data to retrieve the field embeddings.

During training the runtime of all *Joint* models grows quadratically with sequence length since we predict a strictly upper triangular matrix with $(N*N-N)/2$ entries. As there is no temporal dependency between the classification of the different token pairs, these operations can be parallellized. At inference time, the models only need the output of the batch normalization layer to classify a token pair, thus requiring runtime in dependence linear to the sentence length. Subsequent token pairs from the processed sentence can be classified in constant time.

Clause Model. Another variant of our model does not predict the topological fields and classify the clause relations at the same time. Similarly to the $Joint_{noFields}$, we only combine the outputs of the batch normalization layer and concatenate these with distance embeddings. Since the model does not predict topological fields, we only optimize for the clause predictions. Apart from these changes, the rest of the model, including runtime complexity, is identical to the *Joint* model.

6 Evaluation

To evaluate our model we compare it to the self-attentive constituency parser by Kitaev and Klein (2018) and to our baseline model which infers clause boundaries in a rule-based manner from the output of the previously described topological field tagger. In the following section, we first describe the data we use and the preprocessing necessary for our experiments. We then intrinsically evaluate

the considered models and finally demonstrate how information about clause relations can improve dependency parsing.

6.1 Data

We use the constituency-annotated version of TüBa-D/Z[3] for all but the dependency parsing experiments. For those we use the dependency-annotated version of the same release. TüBa-D/Z is a relatively large, hand-annotated treebank containing more than 95,000 sentences and roughly 1.8 million tokens. It is composed of articles from the German newspaper TAZ[4]. Both clauses and topological fields are annotated as interior nodes in the constituency structure.

A large fraction of the constituency trees in TüBa-D/Z is non-projective (see Figure 1 or Figure 2 for examples), in fact, more than 40% of the trees contain crossing edges. Since many constituency parsers require projective trees, TüBa D/Z is also distributed in a linearized version, where trees were projectivized by attaching nodes that introduce crossing edges at the highest non-terminal node that allows for a projective structure.

If the non-projective version is taken as gold standard, only 91.74% of the clause relations and 97.07% of the tokens' topological fields are correctly represented by the projectivized treebank. Investigating the misplaced content reveals that by far the biggest contributors to these errors are punctuation signs. Cumulatively, punctuation accounts for 96.38% of incorrect relations and 96.62% of incorrect field tags. Discarding punctuation during evaluation shows that 99.59% of the relations and 99.88% of topological field tags are represented correctly in the linearized format. This is due to the fact that punctuation is virtually always attached to the root node in TüBa-D/Z.

The distribution of tokens over the topological fields is heavily imbalanced. By far the largest part is located in the MF, followed by the VF and VC. Tokens outside of the topological field structure (UNK) make up a sizable partition, too. Within clauses, no punctuation sign is dominated by a topological field. Additionally a fair amount of other tokens also end up without a topological field dominating them. If punctuation is counted, this partition is ranking second behind the MF. If punctuation is excluded, the UNKs remain a relevant size, ranking just behind the LK with about 100,000 remaining tokens. Tokens are very rarely immediately dominated by an NF, we suspect that this is due to the fact that subordinate clauses are often located in the NF and tokens in such a clause are assigned a label from their respective clause. Figure 4 shows the full frequency counts.

The distribution of *is-in-same-clause* relations in the treebank is not as skewed as that of topological fields, but still shows a preference of about 60% towards not holding.

[3] Release 10

[4] https://www.taz.de

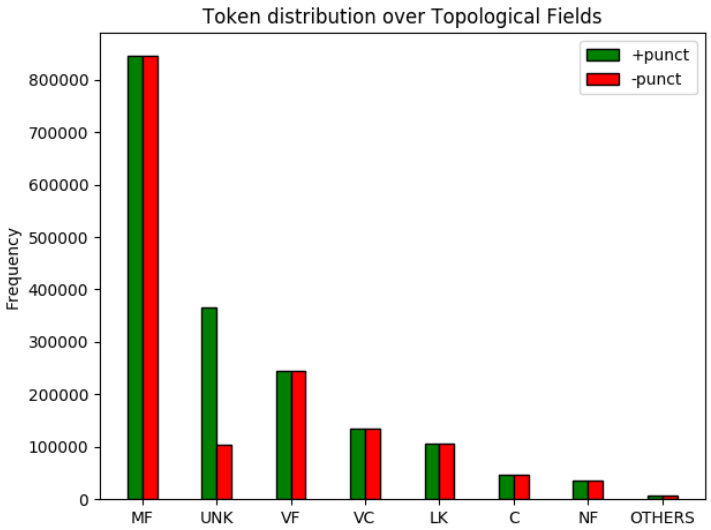

Figure 4: Distribution of tokens topological fields in TüBa-D/Z. *UNK* denotes tokens that were not dominated by a topological field inside their clause. *OTHERS* summarizes KOORD, LV and other infrequent fields. Punctuation exclusively occurs outside of topological fields. About 100,000 tokens do not partake in the topological field structure.

6.2 Preprocessing

All of the preprocessing tools were implemented in Rust and will be made publicly available in the future.

Sequence Tagging and Clause Classification. Since topological fields and clauses are annotated as interior nodes, they are not immediately available on a token level. In order to assign to each token a topological field, we mostly follow the scheme described in de Kok and Hinrichs (2016) but treat the *unknown* label slightly different. They only assign this label to those tokens that were not dominated by a topological field at all. In our projection scheme the *unknown* label is assigned to those tokens that are not dominated by a topological field within their respective clause.

We achieve this by reattaching each node that is not a phrasal node to the closest topological field or clause node. For each node in the tree, we follow the edges up towards the root until the reattachment point is found. After all nodes are attached at the correct position of the tree, we remove all phrasal nodes. Finally, we iterate over the terminal nodes in the shallow tree and the label of the node immediately dominating the terminal to it. If the parent node is not a field node but a clause node or the root, we assign the *unknown* tag.

The clause relations are derived by assigning a unique identifier to each clause

node in a tree and projecting these identifiers onto the tokens. We annotate the clause identifiers through a breadth-first traversal, starting at the root node, keeping track of how many clause nodes have been encountered before. If a new clause is found, we increment the counter and assign it to the new clause. In our implementation each non-terminal node has direct access to the terminal nodes it dominates. To assign the correct identifier to each token, we iterate over the previously encountered clause nodes and assign that clause's identifier to all terminals derived from it. If a terminal is dominated by several clause nodes, the assigned identifier is repeatedly overwritten until the hierarchically closest clause identifier has been projected.

We follow the definitions in Telljohann et al. (2006) to determine whether a node is a clause or field node.

Given the clause identifiers, it is fairly straightforward to annotate the clause relations. We iterate over the tokens in the sentence and retrieve the clause identifier. For each token, we iterate over all subsequent tokens and construct a matrix of clause relations by comparing the clause identifier at the current position with those at the later positions. If they match, the *is-in-same-clause* relation holds. Annotating the full sentence with N tokens in this manner results in a strictly upper triangular matrix with $(N * N - N)/2$ entries, encoding the clause relations for each token with all tokens following it. We do not build a full matrix since determining the clause relation for a token with itself is trivial and the relations are independent of directionality.

Parsing. In order to train the self-attentive parser (Kitaev and Klein, 2018), we have to use the projectivized version of TüBa-D/Z. We conduct experiments with both the fully annotated, unaltered, treebank and with a treebank containing shallow trees, where all phrasal nodes have been removed. The filtering to construct the shallow trees is done in the same manner as described in the previous section, but since assigning a *unknown* label to a token is not possible in the same way, we insert non-terminal nodes with a special label above nodes not dominated by topological fields. If there is a run of tokens outside of topological fields with the same parent node, we collect those nodes under a single node. The examples in Section 2 are shallow trees without the insertion of new nodes.

Setup. We shuffle the preprocessed corpus and split it into three sections, 30% held out for validation, 10% for development and 60% for training. Further, we use a 20% subset of the training set for some of our experiments to examine the impact of corpus size on the different models.

The word embeddings fed as input to our models are a variation of the dependency embeddings proposed in Levy and Goldberg (2014). In contrast to the original dependency embeddings, we include subword information in the same manner as fastText (Bojanowski et al., 2017) and do not collapse certain dependency relations. Our word embeddings are trained with a derivation of

finalfrontier[5] and will be made publicly available in the future. We selected the word embeddings based on the findings of Pütz and Pütz (2019). The hyperparameters for our model and the word embeddings can be found in Appendix A.

All parser models are trained with the best hyperparameters for German according to Kitaev and Klein (2018, Section 6.2). The models build word representations in two ways: through an LSTM over the characters in the word and by training domain-specific word embeddings. We stop the training after a maximum of 30 hours or 100 epochs, whichever is reached first.

We evaluate the per-token accuracy on topological field tagging as well as the per-relation accuracy on the clause classification. Although this puts the constituency parser at a disadvantage, we will use the original, non-projective, format as gold data as this relates to the most truthful analysis of the corpus. As we pointed out in our data analysis, all punctuation is directly attached to the root node in TüBa-D/Z. In order to fairly compare the constituency parser to our tagger, we reattach all punctuation to the root node before deriving the topological field tags and clause relations.

Furthermore, we extrinsically evaluate the clause relations and field predictions on dpar[6] (de Kok and Hinrichs, 2016), a transition-based dependency parser, which we extended to accept clause relations as features.

6.3 Results and Discussion

	Large training set		**Small training set**	
Model	Topo	Clause	Topo	Clause
Baseline	—	82.59	—	—
Parser$_{Full}$	97.78	97.71	96.75	96.98
Parser$_{Shallow}$	97.41	97.11	96.51	96.61
Topo	98.08	—	97.81	—
Clause	—	97.79	—	97.34
Joint	98.12	97.91	97.77	97.39
Joint$_{goldFields}$	98.14	**99.06**	**97.85**	**98.71**
Joint$_{noFields}$	**98.22**	97.87	**97.85**	97.50
Joint$_{noPOS}$	97.58	97.42	—	—
Joint$_{noWords}$	97.82	97.41	—	—

Table 3: Results after training on the large training set (60%) and small training set (20%). High scores in **bold**, those achieved without gold data are underlined. Our joint models outperform both parsers and the non-jointly trained counterparts. The highest accuracy for both training sets is achieved by the *Joint*$_{noFields}$ and Joint$_{goldFields}$ models. The parser models perform substantially worse than our models.

[5] https://github.com/danieldk/finalfrontier
[6] https://github.com/danieldk/dpar/

Main Results. Table 3 shows the results of our main experiments. All considered models, apart from our baseline model, do well on both the clause classification and topological field prediction.

Large Training Set. On topological field tagging, the *Joint* models with unrestricted inputs outperform all other models with *Topo* trailing by a small margin. On clause predictions, we observe a similar pattern: the *Joint* models beat the competition, but do so with a larger improvement over *Clause*. $Joint_{noFields}$, which did not use field embeddings for the clause predictions, achieves the highest score overall on field labeling, but loses out on clause predictions. On this task, $Joint_{goldFields}$ reaches the highest accuracy at more than 99%, which is to be expected as it got information about the correct topological fields for every token during the clause classification. Without the presence of gold field tags, *Joint* shows the best performance on clause predictions, beating $Joint_{noFields}$ by just 0.04%.

The constituency parser trained on full trees achieves 97.78% accuracy on assigning the correct topological field to tokens and 97.71% on modeling clause relations. Training the parser on shallow trees rather than the full constituency trees lowers the scores by 0.37% and 0.60% respectively.

Restricting the inputs of *Joint* to only POS or word embeddings hurts performance on both tasks, but $Joint_{noWords}$ is still able to beat $Parser_{Full}$ on field tagging. $Joint_{noPOS}$ gets worse scores on all of the frequent topological fields than the counterpart without word embeddings. Thus, we cannot pinpoint a specific field to be influenced the most by removing POS information. We conclude from the still decent performance on field tagging by $Joint_{noWords}$ that information about the POS is crucial for this task, while the clause predictions seem rely on word embeddings to some extent. $Joint_{noPOS}$ performs on par with the model without word embeddings on clause relation modeling.

Table 4 shows the performance of some models across the 8 most frequent topological fields (see Figure 4 for an overview). It holds for all of the investigated models in this section that the left sentence bracket LK is almost always identified correctly. Remarkably, $Joint_{noWords}$ achieves together with *Joint* the highest accuracy, pointing at how important POS information is to identify this field and that little to no semantic information, as provided through word embeddings, is required. We observe the same pattern on the right sentence bracket, given that POS embeddings are part of the input, all models reach close to 100% accuracy on labeling the VC. While restricting inputs to word embeddings only had only a slight impact on LK accuracy, the VC is affected more strongly, lowering accuracy by close to 1% when comparing to the unrestricted *Joint* model. For all models, the MF is the largest error contributor as this field is by far the most frequent.

Small Training Set. Using only one third of the training data lowers the accuracy for all of the models (see Table 3). While the parser models suffer the most from reducing the size of the training corpus, $Joint_{goldFields}$ is affected the

Per-field accuracy							
Model	VF	LK	C	MF	VC	NF	UNK
Parser_{Full}	98.26	99.60	98.24	98.45	99.21	**74.04**	97.22
$\text{Parser}_{Shallow}$	97.70	99.46	97.72	97.89	98.80	71.91	96.81
Topo	**98.70**	99.89	**99.08**	99.14	99.80	69.82	96.83
Joint	98.43	**99.92**	98.92	**99.15**	**99.83**	72.57	**96.84**
Joint_{noPOS}	97.97	99.66	97.31	98.83	98.95	64.89	96.63
$\text{Joint}_{noWords}$	98.15	**99.92**	98.66	99.13	**99.83**	68.17	96.36
Frequency	49,365	21,457	9,137	169,157	26,850	6,892	72,774

Table 4: Per-field accuracy on topological field tagging and frequency for the 8 most frequent fields in the validation set. All models were trained on the large training set. Sentence brackets LK and VC are identified with highest accuracy, with $Parser_{Shallow}$ and $Joint_{noPOS}$ as outliers on VC accuracy. The NF is consistently the most difficult class.

least. Training the parser models on the small set yields the only predictions where clause relations were modeled with higher accuracy than the topological field labels. Patterns for our *Joint* models remain similar to those exhibited during training on the large set: the models receiving gold and no field tags for clause predictions perform almost equally on topological field labeling, but, in contrast to the results on the large training set, in the absence of gold field labels, the model not receiving any field embeddings performs the best on clause relations.

Accuracy with non-gold POS		
Model	Topo	Clause
Topo	97.55	—
Clause	—	97.39
Joint	97.65	97.51
$\text{Joint}_{noWords}$	97.24	97.06

Table 5: Results of training our models with predicted POS on the large training set. The accuracy of POS on the corpus is 99.06% Our model falls behind Parser_{Full} without gold POS.

Effect of POS tagging. The evaluation in the previous sections included feeding gold POS tags to our model. To provide a more realistic picture and to investigate the robustness of our model with regard to errors made by a POS tagger, we annotated our corpus using 10-fold jack-knifing (Quenouille, 1956;

TF Accuracy by correct POS		
Model	Correct	Incorrect
Topo	97.78	74.85
Joint	97.88	73.84
$\text{Joint}_{noWords}$	97.49	71.12
$\text{Joint}_{goldPOS}$	98.15	94.34
Joint_{noPOS}	97.73	81.23
Parser_{Full}	97.91	84.17
$\text{Parser}_{Shallow}$	96.67	79.89

Table 6: Accuracy on topological field tagging in relation to correctly predicted POS tags. Joint_{noPOS} and both parser models did not receive POS tags as input while $\text{Joint}_{goldPOS}$ was fed gold tags in addition to word embeddings.

Tukey, 1958) with the publicly available sticker software[7]. The corpus was POS-tagged with an accuracy of 99.06%. We used the predicted POS tags both during training and evaluation as additional experiments showed that training on gold POS tags and evaluating on predicted tags leads to slightly worse results.

We assume that topological field labeling heavily relies on correctly identifying the sentence brackets LK and VC, which in turn should depend on the quality of the provided POS. Thus, we expect performance to drop if we feed predicted POS tags and the error rate to be higher for tokens that were incorrectly POS tagged.

Table 5 confirms our assumption: replacing the gold POS tags with predicted tags lowers the accuracy of our models on both tasks and places it behind $Parser_{Full}$.

In order to understand the losses of providing non-gold POS tags, we examinated the error rate of *Joint* with and without gold tags. We oberserve increasing error rates across all topological fields, except for the very infrequent PARORD and VCE. Among the frequent fields, the NF's accuracy is decreased the most: it drops by 6 points from 72% to 66%. The other fields are more stable, losing between 0.1% and 0.7%, with most of them being around the average 0.5% mark. Albeit seeing a below average increase of 0.2%, the MF still is the largest contributor to the total increase in errors. This is to be expected as the MF dominates by far the most tokens.

Moreover, partitioning the predictions by correct or incorrect POS shows that the accuracy on topological field labeling drops by more than 25% if a token was assigned an incorrect POS. We expected this effect to be more profound with $Joint_{noWords}$ that received POS embeddings as its sole input but only observe a decrease of 4%. $Joint_{noPOS}$ on the other hand was only provided with word embeddings. This improved the accuracy on those words that got

[7] https://git.sr.ht/~danieldk/sticker

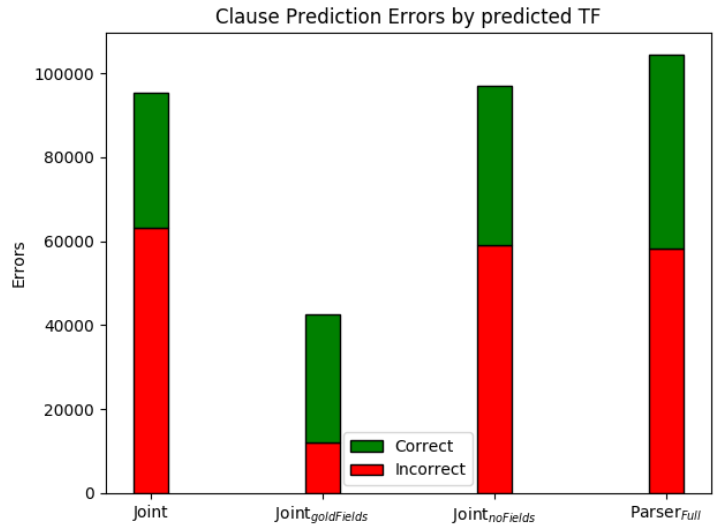

Figure 5: Total number of clause prediction errors partitioned by whether one of the tokens participating in the relation got assigned an incorrect topological field.

mislabeled by the POS tagger, but hurt accuracy overall as the gains on the small set of mistagged tokens cannot offset the losses on the large partition of tokens with correct POS tags. Table 6 shows the full overview.

Interestingly, the error rate on topological field tagging for the partition that got incorrect POS tags is above average for all models, even if the model was fed gold POS tags or none at all, implying that those tokens are particularly challenging to tag.

Relation between field tagging and clause relation accuracy. We suspect that the performance topological field tagging is closely tied to the accuracy on clause relations. As an ablation study, we trained a model where we removed the field embeddings ($Joint_{noFields}$) and another model which was fed gold topological fields for the clause predictions ($Joint_{goldFields}$). While the former model showed the highest accuracy on topological field tagging, it dropped slightly below *Joint* on clause relations. The latter model, on the other hand, showed a massive improvement on predicting clause relations, pushing the accuracy up to 99.06%, which is by far the highest number across all models on this task. We assume that our model heavily relies on the topological field predictions as the vast majority of those are correct. In the few cases where the field predictions are incorrect, the model is misguided by the provided fields.

Furthermore, we investigated the error distribution in clause classification by partitioning the errors of each model into two groups: one where both tokens that are part of a clause prediction got assigned the correct fields and the

Dependency Parsing Scores		
Model	**LAS**	**UAS**
None	91.48	93.30
Parser_{Full}	91.56	93.41
Clause_{pred}	91.69	93.50
Clause_{gold}	91.86	93.65
Topo_{pred}	92.19	93.97
Topo_{gold}	92.49	94.22
Joint_{pred}	<u>92.38</u>	<u>94.14</u>
Joint_{gold}	**92.77**	**94.48**

Table 7: Labeled Attachment Score (LAS) and Unlabeled Attachment Score (UAS) on our validation set. High scores in **bold**, those achieved without gold data are <u>underlined</u>. *None* does not include clause relations or topological field tags. Models with subscript *gold* received gold data, subscript *pred* indicates taken from the predictions of the corresponding model. The *Parser* model derived the clause relations and topological fields from post-processed constituency trees.

other where one or both were mislabeled. Figure 5 shows that all models make fewer mistakes if the predicted field is correct. Decoupling the field information from the field predictions by feeding gold labels to the clause prediction module massively lowers the number of mistakes made on the partition including mislabeled tokens but the errors on pairs from the other partition remain more or less constant among the *Joint* models receiving any kind of field embeddings. $Joint_{noFields}$ is slightly more robust with regard to incorrect topological field labels but loses more than it gained on accuracy for correctly labeled pairs. The parser model's error distribution is more evenly divided, not because it was more robust on clause predictions but rather because it could not benefit from the correctly identified topological field labels.

Dependency Parsing. Table 7 shows that enriching the topological field tags with clause relations yields a sizable improvement on dependency parsing. The baseline model, denoted as *None* in the table, does not receive any information about the topological field structure of the sentence and achieves the lowest scores. Providing the parser with predictions of *Clause* already yields a slight improvement of roughly 0.4% on unlabeled attachment score (UAS) and 0.25% on labeled attachment score (LAS). Passing the output of *Topo* to the parser ends up with a larger gain of about 0.7% on both scores. The best result without feeding gold data comes from incorporating our *Joint*'s predictions, adding 0.9% to LAS and 0.84% to UAS. Replacing the predictions with the gold data for each model pushes the scores even further, reaching 92.77% LAS and 94.48% UAS. The predictions provided by $Parser_{Full}$ prove to be the least helpful for the task.

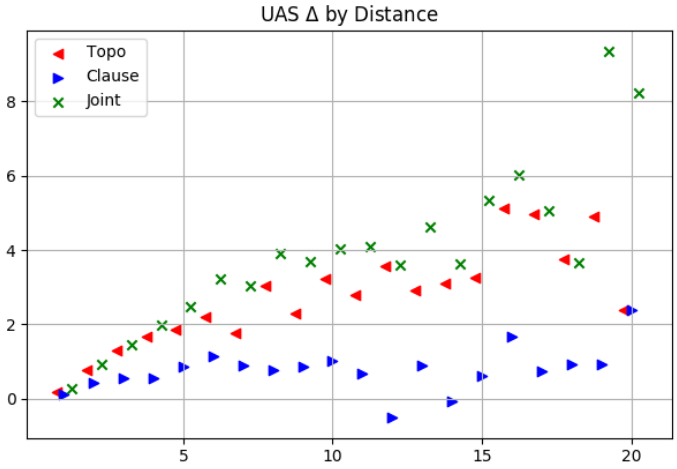

Figure 6: UAS Δ between *None* and $Clause_{gold}$, $Topo_{gold}$ and $Joint_{gold}$. Clause information alone yields only slight improvements. For all three models, the improvements grow over distance.

Although we cannot directly compare our results to de Kok and Hinrichs (2016) because we used a newer release of TüBa-D/Z and larger training sets, we observe similar, but more profound, improvements when we plot the difference in UAS against the linear distance between the head and dependent in each dependency relation (see Figure 6). Adding clause information alone can actually negatively impact UAS at mid-range distances, but is mostly helpful. Enriching the topological fields with clause relations has the biggest impact at higher distances.

Figure 7 shows two plots, the first displays the changes in LAS across the different dependency labels while the second shows absolute changes in errors per category. The highest improvement in LAS for all models is found for parenthetical constructions (PAR), among the frequent labels, adverbial modifiers (ADV), root attachment (ROOT) and some conjunctions (KON) see the largest gains. In absolute numbers, the highest decrease compared to the baseline model *None* comes from the previously mentioned frequent labels ADV, SUBJ, ROOT and KON.

Unlike de Kok and Hinrichs (2016), we see a marked improvement on the prepositional object (OBJP) label. This relation seems to be one of the few that benefit more from clause relations than from topological field information.

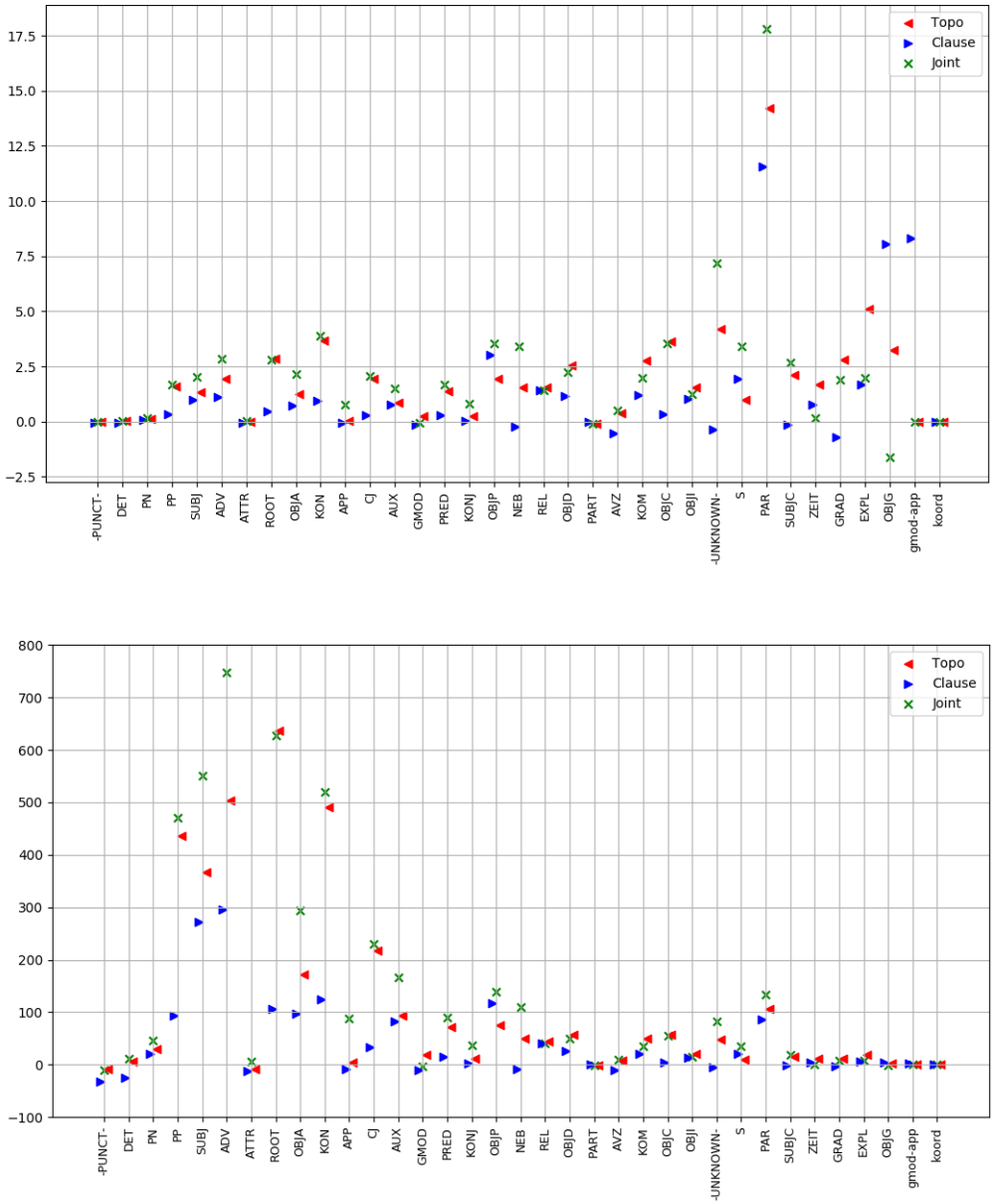

Figure 7: First plot displays LAS Δ between *None* and $Clause_{gold}$, $Topo_{gold}$ and $Joint_{gold}$. The second plot displays absolute error Δ to the baseline *None*. Labels are sorted by frequency in both plots, with the most frequent on the left.

7 Conclusion

In this work, we introduced a new scheme to locally model clause information. Instead of building an abstract tree structure to globally encode the clause structure of a sentence, we define binary relations between tokens, making the information available at the token-level.

We have shown that it is possible to determine whether two tokens are located in the same clause without having to predict a full constituency tree. Our model beats a state-of-the-art constituency parser in predicting such binary relations and even has the advantage of doing so without the overhead of predicting a complex tree structure.

In the model that jointly predicts topological fields and clause relations, we append topological field embeddings based on the predictions of the same model. Here it would be interesting to maintain a continuous representation of the fields, by, for instance, feeding the probability distribution over the fields to the clause module.

As demonstrated by our extrinsic evaluation, a transition-based dependency parser benefits from knowledge about clause relations. Since such a dependency parser has to decide whether to introduce a dependency relation between token pairs or not, it merely needs predictions for the clause relations of the tokens potentially participating in a dependency relation. Our models can provide this on demand, while a constituency parser will always model all relations at once as it requires a full parse to do so. In the future, we want to investigate whether a dependency parser can achieve even better results by providing it with the probability of two tokens belonging to the same clause rather than giving the parser a binary class label.

Preliminary results show that it is possible to predict clause relations jointly with POS tagging, too. This would open opportunities to apply our clause modeling scheme to many other languages that do not have a topological field structure. Moreover, our scheme is not restricted to predicting clause relations, for instance, it could also be applied to determine whether tokens belong to the same phrase or form a multiword unit. Some rule-based parsers, such as the Alpino parser (Van der Beek et al., 2002), can take bracketed input to narrow down the search space. Typically, chunkers were used for such a task, but our scheme has the benefit that it can model discontinuous phrases.

Given a sentence with fully annotated clause relations for every token, we also want to investigate how well the clause boundaries can be inferred.

Further, we want to explore what effect replacing the RNNs in our model with a self-attentive encoder architecture (Vaswani et al., 2017) or a dilated convolutional neural network (Bai et al., 2018) has.

Recently, contextualized word representations (Peters et al., 2018; Devlin et al., 2018) led to large improvements on a variety of tasks. We expect the use of these models to be beneficial for our tasks, too, and want to investigate how they affect topological field labeling and clause relation predictions.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning. In *OSDI*. volume 16, pages 265–283.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* .

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association of Computational Linguistics* 5(1):135–146.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *Proceedings of the workshop on treebanks and linguistic theories*. volume 168.

Jackie Chi Kit Cheung and Gerald Penn. 2009. Topological field parsing of German. 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, pages 64–72.

Daniël de Kok. 2014. Tüba-d/w: a large dependency treebank for german. In *Proceedings of the Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT13)*. page 271.

Daniël de Kok and Erhard Hinrichs. 2016. Transition-based dependency parsing with topological fields. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. volume 2, pages 1–7.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Stefanie Dipper. 2014. Querying topological fields in the TIGER scheme with TIGERSearch. In *Proceedings of the 13th International Workshop on Treebanks and Linguistic Theories (TLT13), Tübingen, Germany*. pages 37–50.

Erich Drach. 1937. 1963. *Grundgedanken der dt. Satzlehre* .

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science* 14(2):179–211.

Oskar Erdmann. 1886. *Grundzüge der deutschen Syntax nach ihrer geschichtlichen Entwicklung*, volume 1. JG Cotta.

Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schäfer. 2003. Integrated shallow and deep parsing: TopP meets HPSG. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics, pages 104–111.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's Going On in Neural Constituency Parsers? An Analysis. *arXiv preprint arXiv:1804.07853* .

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2005. Bidirectional LSTM networks for improved phoneme classification and recognition. In *International Conference on Artificial Neural Networks*. Springer, pages 799–804.

Simon Heinrich Adolf Herling. 1821. *Über die Topik der deutschen Sprache*. Varrentrapp.

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

Tilman Höhle and Albrecht Schöne. 1986. Der Begriff "Mittelfeld", Anmerkungen über die Theorie der topologischen Felder. *Beiträge zur deutschen Grammatik* page 279.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991* .

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* .

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Nikita Kitaev and Dan Klein. 2018. Constituency Parsing with a Self-Attentive Encoder. *arXiv preprint arXiv:1805.01052* .

Sandra Kübler. 2005. How do treebank annotation schemes influence parsing results? Or how not to compare apples and oranges. In *Proceedings of RANLP*. pages 293–300.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. volume 2, pages 302–308.

Martina Liepert. 2003. Topological Fields Chunking for German with SVM's: Optimizing SVM-parameters with GA's. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*. Citeseer.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096* .

Wolfgang Maier. 2006. Annotation schemes and their influence on parsing results. In *Proceedings of the 21st International Conference on computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Association for Computational Linguistics, pages 19–24.

Wolfgang Maier. 2010. Direct parsing of discontinuous constituents in German. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*. Association for Computational Linguistics, pages 58–66.

Nicholas D Matsakis and Felix S Klock II. 2014. The rust language. In *ACM SIGAda Ada Letters*. ACM, volume 34, pages 103–104.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. pages 807–814.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* .

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 433–440.

Sebastian Pütz and Tobias Pütz. 2019. Embeddings for topological field tagging. *Advanced Word Representations for Deep Learning* .

Maurice H. Quenouille. 1956. Notes on bias in estimation. *Biometrika* 43(3/4):353–360. http://www.jstor.org/stable/2332914.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Roland Schäfer. 2015. Processing and querying large web corpora with the cow14 architecture .

M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11):2673–2681. https://doi.org/10.1109/78.650093.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. *arXiv preprint arXiv:1705.03919* .

Heike Telljohann, Erhard Hinrichs, Sandra Kübler, and Ra Kübler. 2004. The TüBa-D/Z treebank: Annotating German with a context-free backbone. In *In Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004*. Citeseer.

Heike Telljohann, Erhard W Hinrichs, Sandra Kübler, Heike Zinsmeister, and Kathrin Beck. 2006. Stylebook for the Tübingen treebank of written German (TüBa-D/Z). In *Seminar fur Sprachwissenschaft, Universitat Tubingen, Tubingen, Germany*.

John W. Tukey. 1958. Abstracts of papers. *Ann. Math. Statist.* 29(2):614–623. https://doi.org/10.1214/aoms/1177706647.

Leonoor Van der Beek, Gosse Bouma, Rob Malouf, and Gertjan Van Noord. 2002. The Alpino dependency treebank. In *Computational linguistics in the netherlands 2001*, Brill Rodopi, pages 8–22.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. pages 5998–6008.

Jorn Veenstra, Frank Henrik Müller, and Tylman Ule. 2002. Topological field chunking for German. In *proceedings of the 6th conference on Natural language learning-Volume 20*. Association for Computational Linguistics, pages 1–7.

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78(10):1550–1560.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and Control* 10(2):189 – 208. https://doi.org/https://doi.org/10.1016/S0019-9958(67)80007-X.

A Hyperparameters

A.1 Word embeddings

- **Context Scheme** Dependencies
- **Dimensions** 300
- **Minimum Token Count** 30
- **Minimum Context Count** 5
- **Negative Sampling Factor** 5
- **Epochs** 5
- **NGram Range** 3-6
- **Dependency Depth** 1

A.2 Topo and Clause models

- **Batchsize** 50
- **Learning Rate**
  - **Initial LR** 0.005
  - **Patience** 15
  - **Adjustment** Half LR after 5 epochs without improvement on development set.
- **Solver** Adam (Kingma and Ba, 2014)
- **Input Dropout** 0.8
- **LSTM Dropout** 0.85
- **LSTM Output Dimensions** 100
- **ReLU Layer Dimensions** 100
- **Topological Field Embedding Dimensions** 25
- **Distance Embedding Dimensions** 10
- **Distance Maximum Value** 25