Constructive Type-Logical Supertagging with Self-Attention Networks

Konstantinos Kogkalidis
Utrecht University
k.kogkalidis@uu.nl

Michael Moortgat
Utrecht University
m.j.moortgat@uu.nl

Tejaswini Deoskar
Utrecht University
t.deoskar@uu.nl

Abstract

We propose a novel application of self-attention networks towards grammar induction. We present an attention-based supertagger for a refined type-logical grammar, trained on constructing types inductively. In addition to achieving a high overall type accuracy, our model is able to learn the syntax of the grammar's type system along with its denotational semantics. This lifts the closed world assumption commonly made by lexicalized grammar supertaggers, greatly enhancing its generalization potential. This is evidenced both by its adequate accuracy over sparse word types and its ability to correctly construct complex types never seen during training, which, to the best of our knowledge, was as of yet unaccomplished.

1 Introduction

Categorial Grammars, in their various incarnations, posit a functional view on parsing: words are assigned simple or complex categories (or: types); their composition is modeled in terms of functor-argument relationships. Complex categories wear their combinatorics on their sleeve, which means that most of the phrasal structure is internalized within the categories themselves; performing the categorial assignment process for a sequence of words, i.e. *supertagging*, amounts to almost parsing (Bangalore and Joshi, 1999).

In machine learning literature, supertagging is commonly viewed as a particular case of sequence labeling (Graves, 2012). This perspective points to the immediate applicability of established, high-performing neural architectures; indeed, recurrent models have successfully been employed (e.g. within the context of Combinatory Categorial Grammars (CCG) (Steedman, 2000)), achieving impressive results (Vaswani et al., 2016). However, this perspective comes at a cost; the supertagger's co-domain, i.e., the different categories it may assign, is considered fixed, as defined by the set of unique categories in the training data. Additionally, some categories have disproportionately low frequencies compared to the more common ones, leading to severe sparsity issues. Since under-represented categories are very hard to learn, in practice models are evaluated and compared based on their accuracy over categories with occurrence counts above a certain threshold, a small subset of the full category set.

This practical concession has two side-effects. The first pertains to the supertagger's inability to capture rare syntactic phenomena. Although the percentage of sentences that may not be correctly analyzed due to the missing categories is usually relatively small, it still places an upper bound on the resulting parser's strength which is hard to ignore. The second, and perhaps more far reaching, consequence is the implicit constraint it places on the grammar itself. Essentially, the grammar must be sufficiently coarse while also allocating most of its probability mass on a small number of unique categories. Grammars enjoying a higher level of analytical sophistication are practically unusable, since the associated supertagger would require prohibitive amounts of data to overcome their inherent sparsity.

We take a different view on the problem, instead treating it as sequence transduction. We propose a novel supertagger based on the Transformer architecture (Vaswani et al., 2017) that is capable of constructing categories inductively, bypassing the aforementioned limitations. We test our model on a highly-refined, automatically extracted type-logical grammar for written Dutch, where it achieves competitive results for high frequency categories, while acquiring the ability to treat rare and even unseen categories adequately.

2 Type-Logical Grammars

The type-logical strand of categorial grammar adopts a proof-theoretic perspective on natural language syntax and semantics: checking whether a phrase is syntactically well-formed amounts to a process of logical deduction deriving its type from the types of its constituent parts (Moot and Retoré, 2012). What counts as a valid deduction depends on the type logic used. The type logic we aim for is a variation on the simply typed fragment of Multiplicative Intuitionistic Linear Logic (MILL), where the type-forming operation of interest is linear implication (for a brief but instructive introduction, refer to Wadler (1993)). Types are inductively defined by the following grammar:

$$\text{T} ::= \text{A} \mid \text{T}_1 \xrightarrow{\text{d}} \text{T}_2 \tag{1}$$

where T, T_1, T_2 are types, A is an atomic type and $\xrightarrow{\text{d}}$ an implication arrow, further subcategorized by the label d.

Atomic types are assigned to phrases that are considered 'complete', e.g. NP for noun phrase, PRON for pronoun, etc. Complex types, on the other hand, are the type signatures of binary functors that compose with a single word or phrase to produce a larger phrase; for instance NP $\xrightarrow{\text{su}}$ S corresponds to a functor that consumes a noun phrase playing the subject role to create a sentence — an intransitive verb.

The logic provides *judgements* of the form $\Gamma \vdash B$, stating that from a multiset of assumptions $\Gamma = A_1, \dots A_n$ one can derive conclusion B. In addition to the axiom $A \vdash A$, there are two rules of inference; implication elimination (2) and implication introduction (3)[1]. Intuitively, the first says that if one has a judgement of the form $\Gamma \vdash A \rightarrow B$ and a judgement of the form $\Delta \vdash A$, one can deduce that assumptions Γ and Δ together derive a proposition B. Similarly, the second says that if one can derive B from assumptions A and Γ together, then from Γ alone one can derive an implication $A \rightarrow B$.

$$\frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash A}{\Gamma, \Delta \vdash B} \rightarrow E \tag{2}$$

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow I \tag{3}$$

[1]For labeled implications $\xrightarrow{\text{d}}$, we make sure that composition is with respect to the d dependency relation.

The view of language as a linear type system offers many meaningful insights. In addition to the mentioned correspondence between parse and proof, the Curry-Howard 'proofs-as-programs' interpretation guarantees a direct translation from proofs to computations. The two rules necessary for proof construction have their computational analogues in function application and abstraction respectively, a link that paves the way to seamlessly move from a syntactic derivation to a program that computes the associated meaning in a compositional manner.

3 Constructive Supertagging

Categorial grammars assign denotational semantics to types, which are in turn defined via a set of inductive rules, as in (1). These, in effect, are the productions a simple, context-free grammar; a *grammar of types* underlying the grammar of sentences. In this light, any type may be viewed as a word of this simple type grammar's language; a regularity which we can try to exploit.

Considering neural networks' established ability of implicitly learning context-free grammars (Gers and Schmidhuber, 2001), it is reasonable to expect that, given enough representational capacity and a robust training process, a network should be able to learn a context-free grammar embedded within a wider sequence labeling task. Jointly acquiring the two amounts to learning a) how to produce types, including novel ones, and b) which types to produce under different contexts, essentially providing all of the necessary building blocks for a supertagger with unrestricted codomain. To that end, we may represent a single type as a sequence of characters over a fixed vocabulary, defined as the union of atomic types and type forming operators (in the case of type-logical grammars, the latter being n-ary logical connectives). A sequence of types is then simply the concatenation of their corresponding representations, where type boundaries can be marked by a special separation symbol.

The problem then boils down to learning how to transduce a sequence of words onto a sequence of unfolded types. This can be pictured as a case of sequence-to-sequence translation, operating on word level input and producing character level output, with the source language now being the natural language and the target language being the language defined by the syntax and semantics of our categorial grammar.

4 Related Work

Supertagging has been standard practice for lexicalized grammars with complex lexical entries since the work of Bangalore and Joshi (1999). In its original formulation, the categorial assignment process is enacted by an N-gram Markov model. Later work utilized Maximum Entropy models that account for word windows of fixed length, while incorporating expanded lexical features and POS tags as inputs (Clark and Curran, 2004). During the last half of the decade, the advent of word embeddings caused a natural shift towards neural architectures, with recurrent neural networks being established as the prime components of recent supertagging models. Xu et al. (2015) first used simple RNNs for CCG supertagging, which were gradually succeeded by LSTMs (Vaswani et al., 2016; Lewis et al., 2016), also in the context of Tree-Adjoining Grammars (Kasai et al., 2017).

Regardless of the particular implementation, the above works all fall in the same category of sequence labeling architectures. As such, the type vocabulary (i.e. the set of candidate categories) is always considered fixed and pre-specified — it is, in fact, hard coded within the architecture itself (e.g. in the network's final classification layer). The inability of such systems to account for unseen types or even consistently predict rare ones has permeated through the training and evaluation process; a frequency cut-off is usually applied on the corpus, keeping only categories that appear at least 10 times throughout the training set (Clark and Curran, 2004). This limitation has been acknowledged in the past; in the case of CCG, certain classes of syntactic constructions pose significant difficulties for parsing due to categories completely missing from the corpus (Clark et al., 2004). An attempt to address the issue was made in the form of an inference algorithm, which iteratively expands upon the lexicon with new categories for unseen words (Thomforde and Steedman, 2011) — its applicability, however, is narrow, as new categories can often be necessary even for words that have been previously encountered.

We differentiate from relevant work in not employing a type lexicon at all, fixed or adaptive. Rather than providing our system with a vocabulary of types, we seek to instead encode the type construction process directly within the network. Type prediction is no longer a discernible part of the architecture, but rather manifested via the network's weights as a dynamic generation process, much like a language model for types that is conditioned on the input sentence.

5 Data

5.1 Corpus

The experiments reported on focus on Dutch, a language with relatively free word order that allows us to highlight the benefits of our non-directional type logic. For our data needs, we utilize the Lassy-Small corpus (van Noord et al., 2006). Lassy-Small contains approximately 65000 annotated sentences of written Dutch, comprised of over 1 million words in total. The annotations are DAGs with syntactic category labels at the nodes, and dependency labels at the edges. The possibility of re-entrancy obviates the need for abstract syntactic elements (gaps, traces, etc.) in the annotation of unbounded dependencies and related phenomena.

5.2 Extracted Grammar

To obtain type assignments from the annotation graphs, we design and apply an adaptation of Moortgat and Moot's (2002) extraction algorithm. Following established practice, we assign phrasal heads a functor (complex) type selecting for its dependents. Atomic types are instantiated by a translation table that maps part-of-speech tags and phrasal categories onto their corresponding types.

As remarked above, we diverge from standard categorial practice by making no distinction between rightward and leftward implication (slash and backslash, respectively), rather collapsing both into the direction-agnostic linear implication. We compensate for the possible loss in word-order sensitivity by subcategorizing the implication arrow into a set of distinct linear functions, the names of which are instantiated by the inventory of dependency labels present in the corpus. This decoration amounts to including the labeled dependency materialized by each head (in the context of a particular phrase) within its corresponding type, vastly increasing its informational content. In practical terms, dependency labeling is no longer treated as a task to be solved by the downstream parser; it is now internal to the grammar's type system. To consistently binarize all of our functor types, we impose an obliqueness or-

$$\dfrac{\dfrac{\dfrac{}{\text{geven} \vdash \text{NP} \xrightarrow{\text{obj}} \text{PRON} \xrightarrow{\text{su}} \text{S}_{\text{main}}}L \quad \dfrac{\dfrac{}{\text{enkele} \vdash \text{NP} \xrightarrow{\text{det}} \text{NP}}L \quad \dfrac{}{\text{voorbeelden} \vdash \text{NP}}L}{\text{enkele, voorbeelden} \vdash \text{NP}}\to E}{\text{geven, enkele, voorbeelden} \vdash \text{PRON} \xrightarrow{\text{su}} \text{S}_{\text{main}}}\to E \quad \dfrac{}{\text{we} \vdash \text{PRON}}L}{\text{we}(\textit{we}), \text{geven}(\textit{give}), \text{enkele}(\textit{some}), \text{voorbeelden}(\textit{examples}) \vdash \text{S}_{\text{main}}}\to E$$

(a) Derivation for “we geven enkele voorbeelden” (*we give some examples)*, showcasing a simple transitive verb derivation.

$$\dfrac{\dfrac{\dfrac{}{\text{welke} \vdash \text{N} \xrightarrow{\text{det}} (\text{N} \xrightarrow{\text{obj}} \text{SV1}) \xrightarrow{\text{body}} \text{WHQ}}L \quad \dfrac{}{\text{rol} \vdash \text{N}}L}{\text{welke, rol} \vdash (\text{N} \xrightarrow{\text{obj}} \text{SV1}) \xrightarrow{\text{body}} \text{WHQ}}\to E \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\text{spelen} \vdash \text{N} \xrightarrow{\text{obj}} \text{NP} \xrightarrow{\text{su}} \text{SV1}}L \quad \dfrac{}{\text{N} \vdash \text{N}}id}{\text{spelen, N} \vdash \text{NP} \xrightarrow{\text{su}} \text{SV1}}\to E \quad \dfrac{}{\text{typen} \vdash \text{NP}}L}{\text{spelen, typen, N} \vdash \text{SV1}}\to E}{\text{spelen, typen} \vdash \text{N} \xrightarrow{\text{obj}} \text{SV1}}\to I}{\text{welke}(\textit{which}), \text{rol}(\textit{role}), \text{spelen}(\textit{play}), \text{typen}(\textit{types}) \vdash \text{WHQ}}\to E$$

(b) Derivation for “welke rol spelen typen” (*which role do types play)*, showcasing object-relativisation via second-order types. Type SV1 stands for verb-initial sentence clause.

$$\dfrac{\dfrac{}{\text{een} \vdash \text{N} \xrightarrow{\text{det}} \text{NP}}L \quad \dfrac{\dfrac{\dfrac{}{\text{en} \vdash \text{ADJ}^{*} \xrightarrow{\text{cnj}} \text{N} \xrightarrow{\text{mod}} \text{N}}L \quad \dfrac{}{\text{eenvoudig} \vdash \text{ADJ}}L \quad \dfrac{}{\text{degelijk} \vdash \text{ADJ}}L}{\text{eenvoudig, en, degelijk} \vdash \text{N} \xrightarrow{\text{mod}} \text{N}} \quad \dfrac{}{\text{idee} \vdash \text{N}}L}{\text{eenvoudig, en, degelijk, idee} \vdash \text{N}}\to E}{\text{een }(\textit{a}), \text{eenvoudig }(\textit{simple}), \text{en }(\textit{and}), \text{degelijk }(\textit{solid}), \text{idee }(\textit{idea}) \vdash \text{NP}}\to E$$

(c) Derivation for “een eenvoudig en degelijk idee” (*a simple and solid idea*), showcasing non-polymorphic conjunction of two adjectives forming a noun-phrase modifier.

Figure 1: Syntactic derivations of example phrases using our extracted grammar. Lexical type assignments are the proofs’ axiom leaves marked L. Identity for non-lexically grounded axioms is marked id. Parentheses are right implicit. Phrasal heads are associated with complex (functor) types. Phrases are composed via function application of functors to their arguments (i.e. implication elimination: $\to E$). Hypothetical reasoning for gaps is accomplished via function abstraction of higher-order types (i.e. implication introduction: $\to I$).

dering (Dowty, 1982) over dependency roles, capturing the degree of coherence between a dependent and the head. Figure 1 presents a few example derivations, indicating how our grammar treats a selection of interesting linguistic phenomena.

The algorithm’s yield is a type-logical treebank, associating a type sequence to each sentence. The treebank counts approximately 5700 unique types, made out of 22 binary connectives (one for each dependency label) and 30 atomic types (one for each part-of-speech tag or phrasal category). As Figure 2 suggests, the comprehensiveness of such a fine-grained grammar comes at the cost of a sparser lexicon. Under this regime, recognizing rare types as first-class citizens becomes imperative.

Finally, given that all our connectives are of a fixed arity, we may represent types unambiguously using polish notation (Hamblin, 1962). Polish notation eliminates the need for brackets, reducing the representation’s length and succinctly encoding a type’s arity in an up-front manner.

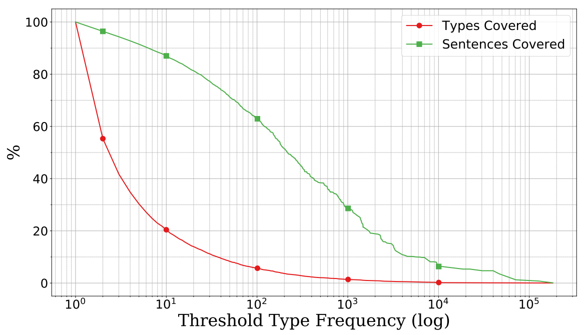

Figure 2: Percentage of types and sentences covered as a function of type frequency. The vast majority of types (80%) are rare (have less than 10 occurrences). At least one such type is present in a non-negligible part of the corpus (12% of the overall sentences). A significant portion of types (45%) appears just once throughout the corpus.

6 Model

Even though prior work suggests that both the supertagging and the CFG-generation problems are learnable (at least to an extent) in isolation, the composition of the two is less straightforward. Predicting the next atomic symbol requires for the network to be able to model local, close-range dependencies as ordained by the type-level syntax. At the same time, it needs a global receptive field in order to correctly infer full types from distant contexts, in accordance with the sentence-level syntax.

Given these two requirements, we choose to employ a variant of the Transformer for the task at hand (Vaswani et al., 2017). Transformers were originally proposed for machine translation; treating syntactic analysis as a translation task is not, however, a new idea (Vinyals et al., 2015). Transformers do away with recurrent architectures, relying only on self-attention instead, and their proven performance testifies to their strength. Self-attention grants networks the ability to selectively shift their focus over their own representations of non-contiguous elements within long sequences, based on the current context, exactly fitting the specifications of our problem formulation.

Empirical evidence points to added benefits from utilizing language models at either side of an encoder-decoder architecture (Ramachandran et al., 2017). Adhering to this, we employ a pretrained Dutch ELMo (Peters et al., 2018; Che et al., 2018) as large part of our encoder.

6.1 Network

Our network follows the standard encoder-decoder paradigm. A high-level overview of the architecture may be seen in Figure 3. The network accepts a sequence of words as input, and as output produces a (longer) sequence of tokens, where each token can be an atomic type, a logical connective or an auxiliary separation symbol that marks type boundaries. An example input/output pair may be seen in Figure 4.

Our encoder consists of a frozen ELMo followed by a single Transformer encoder layer. The employed ELMo was trained as a language model and constructs contextualized, 1024-dimensional word vectors, shown to significantly benefit downstream parsing tasks. To account for domain adaptation without unfreezing the over-parameterized ELMo, we allow for a transformer encoder layer of 3 attention heads to process ELMo's output[2].

Our decoder is a 2-layer Transformer decoder. Since the decoder processes information at a different granularity scale compared to the encoder, we break the usual symmetry by setting its number of attention heads to 8.

At timestep t, the network is tasked with modeling the probability distribution of the next atomic symbol a_t, conditional on all previous predictions $a_0, \dots, a_{t-1}$ and the whole input sentence $w_0, \dots, w_\tau$, and parameterized by its trainable weights θ:

$$p_\theta(a_t | a_0, \dots, a_{t-1}, w_0, \dots, w_\tau)$$

We make a few crucial alterations to the original Transformer formulation.

First, for the separable token transformations we use a two-layer, dimensionality preserving, feed-forward network. We replace the rectifier activation of the intermediate layer with the empirically superior Gaussian Error Linear Unit (Hendrycks and Gimpel, 2016).

Secondly, since there are no pretrained embeddings for the output tokens, we jointly train the Transformer alongside an atomic symbol embedding layer. To make maximal use of the extra parameters, we use the transpose of the embedding matrix to convert the decoder's high-dimensional output back into token class weights. We obtain the final output probability distributions by applying sigsoftmax (Kanai et al., 2018) on these weights.

6.2 Training

We train our network using the adaptive training scheme proposed by Vaswani et al (2017). We apply stricter regularization by increasing both the dropout rate and the redistributed probability mass of the Kullback-Leibler divergence loss to 0.2. The last part is of major importance, as it effectively discourages the network from simply memoizing common type patterns.

[2]Given that no gradient flow is allowed past the transformer encoder layer, in practice we compute the ELMo embeddings of our input sentences in advance, and feed those onto the rest of the network.

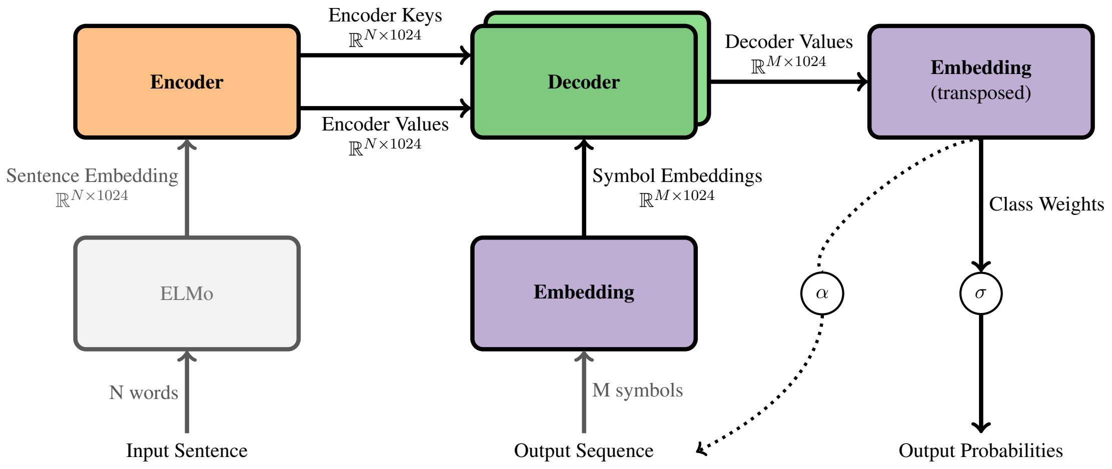
Encoder Keys $\mathbb{R}^{N\times 1024}$
Encoder
Decoder
Decoder Values $\mathbb{R}^{M\times 1024}$
Embedding (transposed)
Encoder Values $\mathbb{R}^{N\times 1024}$
Sentence Embedding $\mathbb{R}^{N\times 1024}$
Symbol Embeddings $\mathbb{R}^{M\times 1024}$
Class Weights
ELMo
Embedding
α
σ
N words
M symbols
Input Sentence
Output Sequence
Output Probabilities

Figure 3: The model architecture, where σ and α denote the *sigsoftmax* and *argmax* functions respectively, grayed out items indicate non-trainable components and the dotted line depicts the information flow during inference.

zijn (*are*)	er (*there*)	toepassingen (*uses*)	voor (*for*)	lineaire (*linear*)	logica (*logic*)
$\mathrm{NP} \xrightarrow{\mathrm{su}} \mathrm{S}_{\mathrm{main}}$	$\mathrm{S}_{\mathrm{main}} \xrightarrow{\mathrm{mod}} \mathrm{S}_{\mathrm{main}}$	NP	$\mathrm{NP} \xrightarrow{\mathrm{obj1}} \mathrm{NP} \xrightarrow{\mathrm{mod}} \mathrm{NP}$	$\mathrm{NP} \xrightarrow{\mathrm{mod}} \mathrm{NP}$	NP

$$\xrightarrow{\mathrm{su}}, \mathrm{NP}, \mathrm{S}_{\mathrm{main}}, \#, \xrightarrow{\mathrm{mod}}, \mathrm{S}_{\mathrm{main}}, \mathrm{S}_{\mathrm{main}}, \# \xrightarrow{\mathrm{obj1}}, \mathrm{NP}, \xrightarrow{\mathrm{mod}}, \mathrm{NP}, \mathrm{NP}, \# \xrightarrow{\mathrm{mod}}, \mathrm{NP}, \mathrm{NP}, \#, \mathrm{NP}, \#$$

Figure 4: Input-output example pair for the sentence "zijn er toepassingen voor lineaire logica?" (*are there applications for linear logic?*). The first two lines present the input sentence and the types that need to be assigned to each word. The third line presents the desired output sequence, with types decomposed to atomic symbol sequences under polish notation, and # used as a type separator.

7 Experiments and Results

In all described experiments, we run the model[3] on the subset of sample sentences that are at most 20 words long. We use a train/val/test split of 80/10/10[4]. We train with a batch size of 128, and pad sentences to the maximum in-batch length. Training to convergence takes, on average, eight hours & 300 epochs for our training set of 45000 sentences on a GTX1080Ti. We report averages over 5 runs.

Accuracy is reported on the type-level; that is, during evaluation, we predict atomic symbol sequences, then collapse subtype sequences into full types and compare the result against the ground truth. Notably, a single mistake within a type is counted as a completely wrong type.

7.1 Main Results

We are interested in exploring the architecture's potential at supertagging, as traditionally formulated, as well as its capacity to learn the grammar beyond the scope of the types seen in the training data. We would like to know whether the latter is at all possible (and, if so, to what degree), but also whether switching to a constructive setting has an impact on overall accuracy.

Digram Encoding Predicting type sequences one atomic symbol or connective at a time provides the vocabulary to construct new types, but results in elongated target output sequence lengths[5]. As a countermeasure, we experiment with *digram encoding*, creating new atomic symbols by iteratively applying pairwise merges of the most frequent intra-type symbol digrams (Gage, 1994), a practice already shown to improve generalization for translation tasks (Sennrich et al., 2016). To evaluate performance, we revert the merges back into their atoms after obtaining the

[3]The code for the model and processing scripts can be found at `https://github.com/konstantinosKokos/Lassy-TLG-Supertagging`.

[4]It is worth pointing out that the training set contains only ∼85% of the overall unique types, the remainder being present only in the validation and/or test sets.

[5]Note that if lexical categories are, on average, made out of c atomic symbols, the overall output length is a constant factor of the sentence length, i.e. there is no change of complexity class with respect to a traditional supertagger.

predictions.

With no merges, the model has to construct types and type sequences using only atomic types and connectives. As more merges are applied, the model gains access to extra short-hands for subsequences within longer types, reducing the target output length, and thus the number of interactions it has to capture. This, however, comes at the cost of a reduced number of full-type constructions effectively seen during training, while also increasing the number of implicit rules of the type-forming context-free grammar. If merging is performed to exhaustion, all types are compressed into single symbols corresponding to the indivisible lexical types present in the treebank. The model then reduces to a traditional supertagger, never having been exposed to the internal type syntax, and loses the potential to generate new types.

We experiment with a fully constructive model employing no merges (M_0), a fully merged one i.e. a traditional supertagger, (M_∞), and three in-between models trained with 50, 100 and 200 merges (M_{50}, M_{100} and M_{200} respectively). Table 1 displays the models' accuracy. In addition to the overall accuracy, we show accuracy over different bins of type frequencies, as measured in the training data: unseen, rare (1-10), medium (10-100) and high-frequency ($>$ 100) types.

	Type Accuracy				
Model	Overall	Unseen Types	Freq 1-10	Freq 10-100	Freq >100
M_0	**88.05**	**19.2**	**45.68**	**65.62**	89.93
M_{50}	88.03	15.97	43.69	64.33	**90.01**
M_{100}	87.87	15.02	41.61	63.71	89.9
M_{200}	87.54	11.7	39.56	62.4	89.64
M_∞	87.2	-	23.91	59.03	89.89

Table 1: Model performance at different merge scales, with respect to training set type frequencies. M_i denotes the model at i merges, where M_∞ means the fully merged model. For the fully merged model there is a 1 to 1 correspondence between input words and output types, so we do away with the separation symbol.

Table 1 shows that all constructive models perform overall better than M_∞, owing to a consistent increase in their accuracy over unseen, rare, and mid-frequency types. This suggests significant benefits to using a representation that is aware of the type syntax. Additionally, the gains are greater the more transparent the view of the type syntax is, i.e. the fewer the merges. The merge-free model M_0 outperforms all other constructive models across all but the most frequent type bins, reaching an overall accuracy of 88.05% and an unseen category accuracy of 19.2%.

Model	New Types Generated	Unique	Correct (%)
M_0	213.6	199.2	**44.39 (20.88)**
M_{50}	186.6	174.2	37.89 (20.3)
M_{100}	187.8	173.4	34.31 (18.27)
M_{200}	190.4	178.8	27.46 (14.42)

Table 2: Repetition-averaged unseen type generation and precision.

We are also interested in quantifying the models' "imaginative" precision, i.e., how often do they generate new types to analyze a given input sentence, and, when they do, how often are they right (Table 2). Although all constructive models are eager to produce types never seen during training, they do so to a reasonable extent. Similar to their accuracy, an upwards trend is also seen in their precision, with M_0 getting the largest percentage of generated types correct.

Together, our results indicate that the type-syntax is not only learnable, but also a representational resource that can be utilized to tangibly improve a supertagger's generalization and overall performance.

7.2 Other Models

Our preliminary experiments involved RNN-based encoder-decoder architectures. We first tried training a single-layer BiGRU encoder over the ELMo representations, connected to a single-layer GRU decoder, following Cho et al. (2014); the model took significantly longer to train and yielded far poorer results (less than 80% overall accuracy and a strong tendency towards memoizing common types). We hypothesize that the encoder's fixed length representation is unable to efficiently capture all of the information required for decoding a full sequence of atomic symbols, inhibiting learning.

As an alternative, we tried a separable LSTM decoder operating individually on the encoder's representations of each word. Even though this model was faster to train and performed marginally better compared to the previous attempt, it still showed no capacity for generalization over rarer types. This is unsurprising, as this approach assumes that the decoding task can be decomposed at the type-level; crucially, the separable decoder's prediction over a word cannot be informed by its predictions spanning other words, an information flow that evidently facilitates learning and generalization.

8 Analysis

8.1 Type Syntax

To assess the models' acquired grasp of the type syntax, we inspect type predictions in isolation. Across all merge scales and consistently over all trained models, all produced types (including unseen ones) are *well-formed*, i.e. they are indeed words of the type-forming grammar. Further, the types constructed are fully complying with our implicit notational conventions such as the obliqueness hierarchy.

Even more interestingly, for models trained on non-zero merges it is often the case that a type is put together using the correct atomic elements that together constitute a merged symbol, rather than the merged shorthand trained on. Judging from the above, it is apparent that the model gains a functionally complete understanding of the type-forming grammar's syntax, i.e. the means through which atomic symbols interact to produce types.

8.2 Sentence Syntax

Beyond the spectrum of single types, we examine type assignments in context.

We first note a remarkable ability to correctly analyze syntactically complex constructions requiring higher-order reasoning, even in the presence of unseen types. An example of such an analysis is shown in Fig 5.

For erroneous analyses, we observe a strong tendency towards self-consistency. In cases where a type construction is wrong, types that interact with that type (as either arguments or functors) tend to also follow along with the mistake. On one hand, this cascading behavior has the effect of increasing error rates as soon as a single error has been made. On the other hand, however, this is a sign of an implicitly acquired notion of phrase-wide well-typedness, and exemplifies the learned long-range interdependencies between types through the decoder's auto-regressive formulation. On a related note, we recognize the most frequent error type as misconstruction of conjunction schemes. This was, to a degree, expected, as coordinators display an extreme level of lexical ambiguity, owing to our extracted grammar's massive type vocabulary.

8.3 Output Embeddings

Our network trains not only the encoder-decoder stacks, but also an embedding layer of atomic symbols. We can extract this layer's outputs to generate vectorial representations of atomic types and binary connectives, which essentially are high-dimensional character-level embeddings of the type language.

Considering that dense supertag representations have been shown to benefit parsing (Kasai et al., 2017), our atomic symbol embeddings may be further utilized by downstream tasks, as a highly refined source of type-level information.

8.4 Comparison

Our model's overall accuracy lies at 88%, which is comparable to the state-of-the-art in TAG supertagging (Kasai et al., 2017) but substantially lower than CCG (Clark et al., 2018). A direct numeric comparison holds little value, however, due to the different corpus, language and formalism used. To begin with, our scores are the result of a more difficult problem, since our target grammar is far more refined. Concretely, we measure accuracy over a set of 5700 types, which is one order of magnitude larger than the CCGBank test bed (425 in most published work; CCGBank itself contains a little over 1100 types) and 20% larger than the set of TAGs in the Penn Treebank. Practically, a portion of the error mass is allotted to mislabeling the implication arrow's name, which is in one-to-one correspondence with a dependency label of the associated parse tree. In that sense, our error rate is already accounting for a portion of the labeled attachment score, a task usually deferred to a parser further down the processing line. Further, the prevalence of entangled dependency structures in Dutch renders its syntax considerably more complicated than English.

9 Conclusion and Future Work

Our paper makes three novel contributions to categorial grammar parsing. We have shown that attention-based frameworks, such as the Trans-

in (*to*)	hoeverre (*what-degree*)	zal (*will*)	het (*the*)
$\mathbf{ADV} \xrightarrow{\mathbf{obj1}} ((\mathbf{INF} \xrightarrow{\mathbf{mod}} \mathbf{INF}) \rightarrow \mathbf{SV1}) \xrightarrow{\mathbf{body}} \mathbf{WHQ}$	ADV	$\mathrm{INF} \xrightarrow{\mathrm{vc}} \mathrm{NP} \xrightarrow{\mathrm{su}} \mathrm{SV1}$	$\mathrm{N} \xrightarrow{\mathrm{det}} \mathrm{NP}$

rapport (*report*)	dan (*then*)	nog (*still*)	een (*a*)	rol (*role*)	spelen (*play*)
N	$\mathrm{INF} \xrightarrow{\mathrm{mod}} \mathrm{INF}$	$\mathrm{INF} \xrightarrow{\mathrm{mod}} \mathrm{INF}$	$\mathrm{N} \xrightarrow{\mathrm{det}} \mathrm{NP}$	N	$\mathrm{NP} \xrightarrow{\mathrm{obj1}} \mathrm{INF}$

Figure 5: Type assignments for the correctly analyzed wh-question "in hoeverre zal het rapport dan nog een rol spelen" (*to what extent will the report still play a role*) involving a particular instance of *pied-piping*. The type of "in" was never seen during training; it consumes an adverb as its prepositional object, to then provide a third-order type that turns a verb-initial clause with a missing infinitive modifier into a wh-question. Such constructions are a common source of errors for supertaggers, as different instantiations require unique category assignments.

former, may act as capable and efficient supertaggers, eliminating the computational costs of recurrence. We have proposed a linear type system that internalizes dependency labels, expanding upon categorial grammar supertags and easing the burden of downstream parsing. Finally, we have demonstrated that a subtle reformulation of the supertagging task can lift the closed world assumption, allowing for unbounded supertagging and stronger grammar learning while incurring only a minimal cost in computational complexity.

Hyper-parameter tuning and network optimization were not the priority of this work; it is entirely possible that different architectures or training algorithms might yield better results under the same, constructive paradigm. This aside, our work raises three questions that we are curious to see answered. First and foremost, we are interested to examine how our approach performs under different datasets, be it different grammar specifications, formalisms or languages, as well as its potential under settings of lesser supervision. A natural continuation is also to consider how our supertags and their variable-length, content-rich vectorial representations may best be integrated with a neural parser architecture. Finally, given the close affinity between syntactic derivations, logical proofs and programs for meaning computation, we plan to investigate how insights on semantic compositionality may be gained from the vectorial representations of types and type-logical derivations.

Acknowledgments

The authors would like to thank the anonymous reviewers for their remarks and suggestions. The first two authors are supported by a NWO grant under the scope of the project "A composition calculus for vector-based semantic modelling with a localization for Dutch" (360-89-070). The first author would like to acknowledge helpful feedback and suggestions from Richard Moot and Vasilis Bountris.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25:237–265.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc V Le. 2018. Semi-supervised sequence modeling with cross-view training. *arXiv preprint arXiv:1809.08370*.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Stephen Clark, Mark Steedman, and James R Curran. 2004. Object-extraction and question-parsing using CCG. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 111–118.

David Dowty. 1982. Grammatical relations and Montague grammar. In P. Jacobson and G. Pullum, editors, *The nature of syntactic representation*, pages 79–130. Reidel.

Philip Gage. 1994. A new algorithm for data compression. *C Users J.*, 12(2):23–38.

F. A. Gers and E. Schmidhuber. 2001. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

Alex Graves. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Charles L Hamblin. 1962. Translation to and from polish notation. *The Computer Journal*, 5(3):210–213.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Sekitoshi Kanai, Yasuhiro Fujiwara, Yuki Yamanaka, and Shuichi Adachi. 2018. Sigsoftmax: Reanalysis of the softmax bottleneck. In *Advances in Neural Information Processing Systems*, pages 284–294.

Jungo Kasai, Bob Frank, Tom McCoy, Owen Rambow, and Alexis Nasr. 2017. Tag parsing with neural networks and vector representations of supertags. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722, Copenhagen, Denmark. Association for Computational Linguistics.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, San Diego, California. Association for Computational Linguistics.

Michael Moortgat and Richard Moot. 2002. Using the spoken Dutch corpus for type-logical grammar induction. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*. European Language Resources Association (ELRA).

Richard Moot and Christian Retoré. 2012. *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics*, volume 6850. Springer.

Gertjan van Noord, Ineke Schuurman, and Vincent Vandeghinste. 2006. Syntactic annotation of large corpora in STEVIN. In *LREC 2006 Proceedings. 5th Edition of the International Conference on Language Resources and Evaluation*. European Language Resources Association (ELRA).

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Prajit Ramachandran, Peter Liu, and Quoc Le. 2017. Unsupervised pretraining for sequence to sequence learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 383–391, Copenhagen, Denmark. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press/Bradford Books.

Emily Thomforde and Mark Steedman. 2011. Semi-supervised CCG lexicon extension. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1246–1256, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237, San Diego, California. Association for Computational Linguistics.

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.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2773–2781. Curran Associates, Inc.

Philip Wadler. 1993. A taste of linear logic. In *International Symposium on Mathematical Foundations of Computer Science*, pages 185–210. Springer.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages

250–255, Beijing, China. Association for Computational Linguistics.