Revisiting Supertagging for Faster HPSG Parsing

Olga Zamaraeva and **Carlos Gómez-Rodríguez**

Universidade da Coruña, CITIC

Departamento de Ciencias de la Computación y Tecnologías de la Información

Campus de Elviña s/n, 15071, A Coruña, Spain

{olga.zamaraeva, carlos.gomez}@udc.es

Abstract

We present new supertaggers trained on English grammar-based treebanks and test the effects of the best tagger on parsing speed and accuracy. The treebanks are produced automatically by large manually built grammars and feature high-quality annotation based on a well-developed linguistic theory (HPSG). The English Resource Grammar treebanks include diverse and challenging test datasets, beyond the usual WSJ section 23 and Wikipedia data. HPSG supertagging has previously relied on MaxEnt-based models. We use SVM and neural CRF- and BERT-based methods and show that both SVM and neural supertaggers achieve considerably higher accuracy compared to the baseline and lead to an increase not only in the parsing speed but also the parser accuracy with respect to gold dependency structures. Our fine-tuned BERT-based tagger achieves 97.26% accuracy on 950 sentences from WSJ23 and 93.88% on the out-of-domain technical essay *The Cathedral and the Bazaar* (*cb*)). We present experiments with integrating the best supertagger into an HPSG parser and observe a speedup of a factor of 3 with respect to the system which uses no tagging at all, as well as large recall gains and an overall precision gain. We also compare our system to an existing integrated tagger and show that although the well-integrated tagger remains the fastest, our experimental system can be more accurate. Finally, we hope that the diverse and difficult datasets we used for evaluation will gain more popularity in the field: we show that results can differ depending on the dataset, even if it is an in-domain one. We contribute the complete datasets reformatted for Huggingface token classification.

1 Introduction

We present new supertaggers for English and use them to improve parsing efficiency for Head-driven Phrase Structure Grammars (HPSG). Grammars have been gaining relevance in the natural language processing (NLP) landscape (Someya et al., 2024), since it is hard to interpret and evaluate the output of NLP systems without robust theories.

Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994, HPSG) is a theory of syntax that has been applied in computational linguistic research (see Bender and Emerson 2021 §3-§4). At the core of such research are precision grammars which encode a strict notion of grammaticality — their purpose is to cover and generate only grammatical structures. They include a relatively small set of phrase-structure rules and a large lexicon where lexical entries contain information about the word's syntactic behavior. HPSG treebanks (and the grammars that produce them) encode not only constituency but also dependency and semantic relations and have proven useful in natural language processing, e.g. in grammar coaching (Flickinger and Yu, 2013; Morgado da Costa et al., 2016, 2020), natural language generation (Hajdik et al., 2019), and as training data for high precision semantic parsers (Lin et al., 2022; Chen et al., 2018; Buys and Blunsom, 2017). Assuming a good parse ranking model, a treebank is produced automatically by parsing text with the grammar, and any updates are encoded systematically in the grammar, with no need of manual treebank annotation.[1]

HPSG parsing, which is typically bottom-up chart parsing, is both relatively slow and RAM-hungry. Often, more than a second is required to parse a sentence (see Table 7), and sometimes the performance is prohibitively bad for long sentences, with a typical user machine requiring unreasonable amounts of RAM to finish parsing with a large parse chart (Marimon et al., 2014; Oepen and Carroll, 2002). It is important to emphasize that **this is the state of the art in HPSG parsing**, and its

---

[1] For a good parse ranking model, it is necessary to select "gold" parses from a potentially large parse forest at least once. This can be done semi-automatically (Packard, 2015).

speed is one of the reasons why the true potential of HPSG parsing in NLP remains not fully realized despite the evidence that it helps create highly precise training data automatically. Approaches to speed up HPSG parsing include local ambiguity packing (Tomita, 1985; Malouf et al., 2000; Oepen and Carroll, 2002), on the one hand, and forgoing exact search and reducing the parser search space, on the other (Dridan et al., 2008; Dridan, 2009, 2013). Here we contribute to the second line of research, aka supertagging, a technique to discard unlikely interpretations of tokens. Dridan et al. (2008) and Dridan (2009, 2013) used maximum entropy-based models trained on a combination of gold and automatically labeled data from English, requiring large-scale computation. They report an efficiency improvement of a factor of 3 for the parser they worked with (Callmeier, 2000) and accuracy improvements with respect to the ParsEval metric.

We present new models for HPSG supertagging, an SVM-based one, a neural CRF-based one, and a fine-tuned-BERT one, and compare their tagging accuracy with a MaxEnt baseline. We now have more English gold training data thanks to the HPSG grammar engineering consortium's treebanking efforts (Flickinger, 2000; Oepen et al., 2004; Flickinger, 2011; Flickinger et al., 2012).[2] It makes sense to train modern models on this wealth of gold data. Then we use the supertags to filter the parse chart at the lexical analysis stage, so that the parser has fewer possibilities to consider. We report the results of parsing all of the test data associated with the English HPSG treebanks (Oepen and Carroll, 2002) in comparison with parsing the same data with the same parsing algorithm but with no tagging at all, as well as with the integrated MEMM-based tagger. If we use the tagger with some exceptions, our system is the most accurate one (using the partial dependency match metric). It is not faster that the MEMM-based tagger integrated into the parser for production mode, although it is of course much faster than parsing without tagging (by a factor of 3).

The paper is organized as follows. In §2, we give the background necessary for understanding the provenance of our training data. §3 presents the methodology, starting from previous work (§3.1). We then describe our training and evaluation data (§3.2), and finally how we trained the new supertag-

gers (§3.3). In §4, we present the results: first for the accuracy of the supertagger (§4.1) and then for the parsing experiments, including parsing speed and parsing accuracy (§4.2).

We trained the neural models with NVIDIA GeForce RTX 2080 GPU, CUDA version 11.2. The SVM model and the MaxEnt baseline were trained using Intel Core i7-9700K 3,60Hz CPU. The parser was run on the same CPU. The code and configurations for the reported results as well as the datasets are online.[3] The original data we used is publicly available.[2] Further details can be found in the Appendix.

2 Background

Below we explain HPSG lexical types (§2.1), which serve as the tags that we predict, and in §2.2, we give the background on the English treebanks which served as our training and evaluation data. §2.3 is a summary for HPSG parsing and the specific parser that we are using for the experiments.

2.1 Lexical types

Any HPSG grammar consists of a hierarchy of types, including phrasal and lexical types, and of a large lexicon which can be used to map surface tokens to lexical types. Each token in the text is recognized by the parser as belonging to one or more of the lexical entries in the lexicon (assuming such an orthographic form is present at all). Lexical entries, in turn, belong to lexical types (Figure 1). Lexical types are similar to POS tags but are
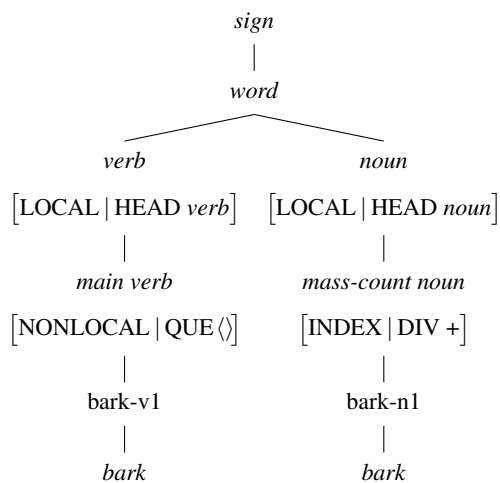

Figure 1: Part of the HPSG type hierarchy (simplified; adapted from ERG). NB: This is not a derivation.

more fine grained (e.g. a precision grammar may distinguish between multiple types of proper nouns or multiple types of *wh*-words, etc). Figure 1 shows the ancestry of two senses of the English word *bark*, a verb (to bark) and a noun (tree bark). The types differ from each other in features and their values. For example, the HEAD feature value is different for nouns and verbs; one of the characteristics of the main verb type is that it is not a question word; the noun subtype denotes divisible entities, etc. The token *bark* will be interpreted as either a verb or a noun during lexical analysis parsing stage. After the lexical analysis, the bottom-up parser runs a constraint unification-based algorithm (Carpenter, 1992) to return a (possibly empty) set of parses. To emphasize, a parser in this context is a separate program implementing a parsing algorithm. The grammar is the type hierarchy which the parser takes as input along with the sentence to parse.

2.2 The ERG treebanks

The English Resource Grammar (ERG; Flickinger, 2000, 2011) is a broad-coverage precision grammar of English implemented in the HPSG formalism. The latest release is from 2023.[4] Its intrinsic evaluation relies on a set of English text corpora. Each release of the ERG includes a treebank of those texts parsed by the current version. The parses are created automatically and the gold structure is verified manually. Treebanking in the ERG context is the process of choosing linguistically (semantically) correct structures from the multiple trees corresponding to one string that the grammar may produce. Fast treebanking is made possible by automatically comparing parse forests and by discriminant-based bulk elimination of unwanted trees (Oepen, 1999; Packard, 2015). The treebanks are stored as databases that can be processed with specialized software e.g. Pydelphin[5].

The 2023 ERG release comes with 30 treebanked corpora containing over 1.5 million tokens and 105,155 sentences. In principle, there are 43,505 different lexical types in the ERG (cf. 48 tags in the Penn Treebank POS tagset (PTB; Marcus et al., 1993)) however only 1299 of them are found in the training portion of the treebank. The genres include well-edited text (news, Wikipedia articles, fiction, travel brochures, and technical essays) as well as customer service emails and transcribed phone conversations. There are also con-

---

structed test suites illustrating linguistic phenomena such as raising and control. The ERG treebanks present more challenging test data compared to the conventional WSJ23 (which is also included). The ERG 2023's average accuracy (correct structure) over all the corpora is 93.77%; the raw coverage (some structure) is 96.96%. The ERG uses PTB-style punctuation tokens and includes PTB POS tags in all tokens, along with a lexical type (§2.1).

2.3 HPSG parsing

Several parsers for different variations of the HPSG formalism exist. We work with the DELPH-IN formalism (Copestake, 2002) which is deliberately restricted for theoretical and performance considerations; it only encodes the unification operation natively (and not e.g. relational constraints). Still, the parsing algorithms' worst-case complexity is intractable (Oepen and Carroll, 2002). Carroll (1993, §3.2.3) (cited in Bender and Emerson 2021, p.1109) states that the worst-case parsing time for HPSG feature structures is proportional to $C^2 n^{\rho+1}$ where ρ is the maximum number of children in a phrase structure rule and C is the (potentially large) maximum number of feature structures. The unification operator takes two feature structures as input and outputs one feature structure which satisfies the constraints encoded in both inputs. Given the complex nature of such structures, implementing a fast unification parser is a hard problem. As it is, the existing parsers may take prohibitively long to parse a long sentence (see e.g. Marimon et al. 2014 as well as §4.2 of this paper).

3 Methodology

Supertagging (Bangalore and Joshi, 1999) reduces the parser search space by discarding the less likely interpretations of an orthography. For example, the word *bark* in English can be a verb or a noun, and in *The dog barks* it is a lot less likely to be a noun than a verb (see also Figure 1). In principle, there are at least two possible interpretations of the sentence *The dog barks*, as can be seen in Figure 2. With supertagging, the pragmatically unlikely second interpretation would be discarded by discarding the noun lexical type (*mass-count noun* in Figure 1) possibility for the word *barks*. In HPSG, there are fine-grained lexical types within the POS class (e.g. subtypes of common nouns or *wh*-words), so the search space can be reduced further.

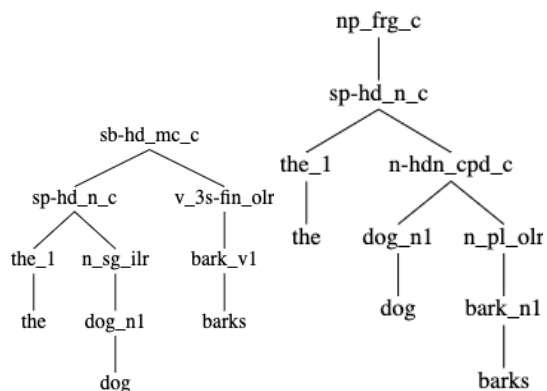In precision grammars, supertagging comes at a

11361

Figure 2: Two interpretations of the sentence *The dog barks.* The second one is an unlikely noun phrase fragment, which would be discarded with the supertagging technique. (Trees provided by the English Resource Grammar Delphin-viz online demo.)

cost to coverage and accuracy; selecting a wrong lexical type even for one word means the entire sentence will likely not be parsed correctly. Thus the accuracy of the tagger is crucial. Related to this is the matter of how many possibilities to consider for supertags: the more are considered, the slower the parsing, but the higher the accuracy. In this paper, we experiment with a single, highest-scored tag for each token. However, we combine this strategy (which prioritizes parsing speed) with a list of tokens exempt from supertagging (which increases accuracy).

3.1 Previous and related work

Bangalore and Joshi (1999) introduced the concept of supertagging. Clark and Curran (2003) showed mathematically that supertagging improves parsing efficiency for a lexicalized formalism (CCG). They used a maximum entropy model; Xu et al. (2015) introduced a neural supertagger for CCG. Vaswani et al. (2016) and Tian et al. (2020) further improved the accuracy of neural-based CCG supertagging achieving an accuracy of 96.25% on WSJ23. Liu et al. (2021) use finer categories within the CCG tagset and report 95.5% accuracy on in-domain test data and 81% and 92.4% accuracy on two out-of-domain datasets (Bioinfer and Wikipedia). Prange et al. (2021) have started exploring the long-tail phenomena related to supertagging and strategies to not discard rare tags. Kogkalidis and Moortgat (2023) have shown how supertagging, through its relation to underlying grammar principles, improves neural networks' abilities to deal with rare

("out-of-vocabulary") words.[6]

Supertagging experiments with HPSG parsing speed using hand-engineered grammars are summarized in Table 1. In addition, there were experiments on the use of supertagging for parse ranking with statistically derived HPSG-like grammars (Ninomiya et al., 2007; Matsuzaki et al., 2007; Miyao and Tsujii, 2008; Zhang et al., 2009, 2010; Zhang and Krieger, 2011; Zhang et al., 2012). These statistically derived systems are principally different from the ERG as they do not represent HPSG theory as understood by syntacticians. In the context of the ERG, Dridan et al. 2008 represents our baseline SOTA for the tagger accuracy. Dridan 2013 is a related work on "ubertagging", which includes multi-word expressions. Specifically, an ubertagger considers various multi-word spans, whereas a supertagger relies on a standard tokenizer. We use the ubertagger that was implemented for the ACE parser for the parsing speed experiments, as the baseline (§4.2). Dridan's (2013) parsing accuracy results, however, are not comparable to ours; she used a different dataset, a different parser, and a different accuracy metric.

3.2 Data

We train and evaluate our taggers, both for the baseline (§4.1.1) and for the experiment (§3.3), on gold lexical types from the ERG 2023 release (§2.2). We use the train-dev-test split recommended in the release.[7] There are 84,894 sentences in the training data, 2,045 in dev, and 7,918 in test. WSJ section 23 is used as test data, as is traditional, but so are a number of other corpora, notably *The Cathedral and the Bazaar* (Raymond, 1999), a technical essay which serves as the out-of-domain test data. See Table 2 for the details about the test data. The column titled "training tokens" shows the number of tokens for the training dataset which is from the same domain as the test dataset in the row. For example, WSJ23 has 23K tokens and WSJ1-22 have 960K tokens in the ERG treebanks.

3.3 SVM, LSTM+CRF, and fine-tuned BERT

We train a liblinear SVM model with default parameters (L2 Squared Hinge loss, C=1, one-v-rest, up to 1,000 training iterations) using the scikit-learn

---

[6]These works do not report experiments on parsing speed; they are concerned with tagging accuracy issues only.

[7]Download redwoods.xls from the ERG repository for details and see https://github.com/delph-in/docs/wiki/RedwoodsTop. This split is different than in Dridan 2009.

model	grammar	training tok	tagset size	speed-up factor
N-gram (Prins and van Noord, 2004)	Alpino (Dutch)	24 mln	1,365	2
HMM (Blunsom, 2007, p. 167)	ERG (English)	113K	615	8.5
MEMM (Dridan, 2009, p. 169)	ERG (English)	158K	676	12

Table 1: Supertagging effects on HPSG parsing speed.

dataset	description	sent	tok	train tok	MaxEnt	SVM	NCRF++	BERT	D2009
cb	technical essay	713	17,244	0	88.96	89.53	91.94	**93.88**	74.61
ecpr	e-commerce	1,088	11,550	24,934	91.80	91.99	95.09	**96.09**	
jh*,tg*,ps*, ron*	travel brochures	2,116	34,098	147,166	90.45	91.21	95.44	**96.11**	91.47
petet	textual entailment	581	7,135	1,578	92.88	95.31	96.93	**97.71**	
vm32	phone conv.	1,000	8,730	86,630	93.57	94.29	95.62	**96.64**	
ws213-214	Wikipedia	1,470	29,697	161,623	91.31	92.02	93.66	**95.59**	
wsj23	Wall Street J.	950	22,987	959,709	94.27	94.72	96.05	**97.26**	
all	all test sets as one	7,918	131,441	1,381,645	91.57	92.28	94.46	**96.02**	
all	average	7,918	131,441	1,381,645	91.89	92.72	94.96	**96.18**	
speed (sen/sec)	average	7,918	131,441	1,381,645	1,024	**7,414**	125	346	

Table 2: Baseline (MaxEnt) and experimental supertaggers' accuracy and speed on test data; tagset size is 1,299.

model	top mistaken token	top underpredicted		top overpredicted	
		all	not closely rel	all	not closely rel
BERT	*i*	n-pn	adj-i	n-pn-gen	d-poss-my
NCRF++	*to*	n-c	adj-i	v-np	adj-i
SVM	*to*	n-pn	v-np*	n-pn-gen	adj-i
MaxEnt	*have*	n-pn	v-np*	n-pn-gen	adj-i

Table 3: A summary of taggers' errors

library (Pedregosa et al., 2011). To train an LSTM sequence labeling model, we use the NCRF++ library (Yang and Zhang, 2018). We choose the model by training and validating 31 models up to 100 iterations with the starting learning rate of 0.009 and the batch size of 3 (the latter parameters are the largest that are feasible for the combination of our data and the library code). The best NCRF++ model is described in the Appendix in Table 10. To fine-tune BERT, we use the Huggingface transformers library (Wolf et al., 2019) and Pytorch (Paszke et al., 2017). We try both 'base-bert-cased' and 'base-bert-uncased' pretrained models which we fine-tune for up to 50 epochs (stopping once there is no improvement for 5 epochs) with weight decay=0.01. The 'cased' model with learning rate 2e-5 achieves the best dev accuracy (Table 14).

We construct feature vectors similarly to what is described in Dridan 2009 and ultimately in Ratnaparkhi et al. 1996. The training vector consists of the word orthography itself, the two previous and the two subsequent words, the word's POS tag, and, for autoregressive models, the two gold lexical type labels for the two previous words. Nonautoregressive models simply do not have the previous tag features. The test vector is the same except, for autoregressive models, instead of the gold labels for the two previous tokens, it has labels assigned to the two previous tokens by the model itself in the previous evaluation steps (an autoregressive model). The word orthographic forms come from the treebank derivation terminals obtained using the Pydelphin library.[8] The PTB-style POS tags come from the treebanks and they were automatically assigned by an HMM-based tagger that is part of the ACE parser code. The POS tags provided by the parser are per token, not per terminal, so for terminals which consist of more than one token, we map the combination of more than one tag to a single PTB-style tag using a mapping constructed manually by the first author for the training data. Any combination of tags not in the training data are at test time mapped to the first tag based on that being the most frequently correct prediction in the training data.[9] We only saw 15 unknown combinations of tags in the entire dev and test data.

3.4 The ACE HPSG Parser

We work with ACE (Crysmann and Packard, 2012), which has seen regular releases since the publica-

---
[8] https://pydelphin.readthedocs.io/
[9] The first tag is the correct tag in about 1/3 of the cases.

tion date and remains the state-of-the-art HPSG parser. It is intended for settings which include individual use, including with limited RAM. This parser has default RAM settings[10] which can be modified, and also an in-built "ubertagger". While the ubertagger is based on Dridan 2013, it is not the same thing and its performance has never been published before. In particular, its tagging accuracy is unknown and we did not seek to evaluate it (evaluating a different MaxEnt model instead). The ubertagger was integrated into the ACE parser code with great care, optimizing for performance. We also do not seek to compete with such optimizations in our experiments. For our experiments, we provide ACE with the tags predicted by the best supertagger (the BERT-based supertagger) along with the character spans corresponding to the token for which the tag was predicted.[11] We then prune all lexical chart edges which correspond to this token span but do not have the predicted lexical type. As such, we follow the general idea of using supertagging for reducing the lexical chart size but we do not use the same code that the integrated ubertagger uses for this procedure. We assume that our code could be further optimized for production.

3.5 Exceptions for supertagging

As already mentioned, mistakes in supertagging are very costly for precision grammar parsing; one wrongly predicted lexical type means the entire sentence will not be parsed correctly. After the maxent-based supertaggers were trained by Dridan 2009 and Dridan 2013, the developer of the English Resource Grammar Flickinger experimented with them and has come up with a list of lexical types which the supertagger tended to predict wrong. The list included fine-grained lexical types representing words such as *do*, *many*, *less*, *hard* (among many others).[12] Using such exception lists counteracts the effects of supertagging and slows down the parsing, while increasing accuracy. We include this exception list methodology into our experiments, but we compile our own list based on the top mistakes our supertaggers made on the dev data.

4 Results

4.1 Tagger accuracy and tagging speed

4.1.1 Tagging accuracy baseline

For our baseline, we use a MaxEnt model similar to Dridan 2009. While Dridan (2009) used off-the-shelf TnT (Brants, 2000) and C&C (Clark and Curran, 2003) taggers, we use the off-the-shelf logistic regression library from scikit-learn (Pedregosa et al., 2011) which is a popular off-the-shelf tool for classic machine learning algorithms. The baseline tagger accuracy is included in Table 2. The details on how the best baseline model was chosen are in Appendix A. The results are presented in Table 2.

4.1.2 Tagger accuracy results

Table 2 shows that the baseline models achieve similar performance to Dridan 2009 (D2009 in Table 2) on in-domain data and are better on out-of-domain data. This may indicate that these models are close to their maximum performance on in-domain data on this task but adding more training data still helps for out-of-domain data. Dridan's (2009) models were trained on a subset of our data. Dridan (2009, p.84) reports getting 91.47% accuracy on the in-domain data (which loosely corresponds to row 'jh*, tg*, ps*') using the TnT tagger (Brants, 2000).

The SVM and the neural models are better than the baseline models on all test datasets, and fine-tuned BERT is the best overall. On the portion of WSJ23 for which we have gold data, fine-tuned BERT achieves 97.26%. The neural models are slower than the baseline models (using GPU for decoding); on the other hand, SVM is remarkably fast (at over 7000 sen/sec).

All models make roughly the same mistakes (Table 3), with prepositions, pronouns, and auxiliary verbs being the most misclassified tokens, and the proper noun being the least accurate tag.[13]

4.2 Results: Parsing Speed and Accuracy

We measure the effect of supertagging on parsing speed and accuracy using the ACE parser (§3.4). Recall that HPSG parsing is chart parsing, and for a large grammar, the charts can be huge. The goal

---

[10]1.2GB for chart building plus 1.5 for "unpacking", which is a lexical disambiguation procedure.

[11]The speed of the tagging itself is negligible because the tagger tags 346 sentences per second (0.003 sec/sen) while HPSG parsing is an order of magnitude slower.

[12]The full list can be found in the release of the ERG in the folder titled 'ut' (ubertagging).

[13]In Table 3, the "not closely related" column represents mistakes where the true label and the predicted label differ in their general subcategory; in this column, we did not count nouns mistaken for other types of nouns, etc. We use the ERG lexical type naming convention to filter the errors. The "n-c" type is a subtype of common noun; the "n-pn" and "n-pn-gen" types are subtypes of proper nouns; "v-np*" is a subtype of verbs that take clausal complements; "adj-i" is a subtype of intersective adjectives, "d-poss" is a possessive determiner.

of supertagging is to reduce the size of the lexical chart. This can make parsing faster, however if a good lexical analysis is thrown out by mistake (due to a wrong tag), the entire sentence is likely to be lost (not parsed or parsed in a meaningless way). The parser speed and the parser accuracy are therefore in tension: the more time we give the parser the more chances it will have to build the correct structure in a bigger chart. For accuracy, we report two metrics: exact match with the gold semantic structure (MRS) and partial match Elementary Dependency Match metric (EDM; Dridan and Oepen, 2011). The exact match is less important because it usually can only be achieved on short, easy sentences. The EDM (and similar) is the usual practice. The results are presented in Tables 4-9, which are also summarized in Figure 3.
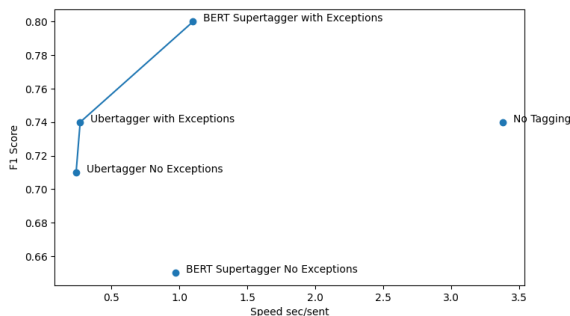

Figure 3: Pareto Frontier (Speed and F-score)

4.2.1 Baseline

We compare our system with two systems: ACE with no tagging at all and ACE with the in-built "ubertagger". The system with no tagging at all is the baseline for parsing speed and, theoretically, the upper boundary for the parsing accuracy (as the parser could have access to the full lexical chart). However, in practice it is difficult to obtain this upper bound because it requires at least 54GB of RAM (see §A.5) and the parsing takes unreasonably long (up to several minutes per sentence). With realistic settings, the system with no tagging fails to parse some of the longer sentences because the lexical chart exceeds the RAM limit. It is precisely the problem that ubertagging/supertagging is supposed to solve: reduce the size of the lexical chart so that the parsing can be done with realistic RAM allocation and in reasonable time.

The ubertagger is a MEMM tagger based on Dridan 2013. It was trained on millions of sentences using large computational resources (the Titan system at University of Oslo) and as such is not easily reproducible. In contrast, our BERT-based model is fairly easy to fine-tune and reproduce on an individual machine. For the purposes of parsing accuracy and speed, rather than comparing our system to other experimental taggers presented in §4.1, we compare it to the ubertagger because the ubertagger is integrated into the ACE parser for production and as such is a more challenging baseline.

Below we present the results in two settings: (1) default settings, and (2) default RAM with tag exceptions. In Tables 4 and 7, the best result is bolded, and the experimental result is italicized in the cases where it is not the best but much closer to the ubertagger than to the no-tagging baseline.

4.2.2 Default parsing

Tables 4, 5, and 6 present the results for the ACE parser default RAM limit setting (1200MB). On the ubertagger and the supertagger side, we use all the predictions and do not exclude any tags from the pruning process.

The results show that while we can parse faster with tagging (the ubertagger being the fastest), both the ubertagger and the supertagger suffer from the high cost of each tagging mistake: while the new BERT-based supertagger is more accurate, its accuracy is still not 100%, and even at 99% tagger accuracy, the likelihood of losing an entire sentence due to one incorrect tag is high. Dridan (2013) comments on this, too, and suggests taking into account the top mistakes that the tagger makes to achieve higher recall. This is what we do below.

4.2.3 Parsing with exceptions lists

Tables 7-9 present the results for parsing with ubertagging and supertagging with exceptions. The no-tagging system's results are the same as before; we repeat them for convenience.

We have looked at the most common mistakes in the supertags in the training data and have compiled a list of 15 tags which BERT tends to predict wrong.[14] On the ubertagger side, there was already a list of exceptions. The ubertagger's exception list is a list of 1715 lexical entries (words, e.g. "my"), whereas ours is a list of 15 lexical types (tags, e.g. "d-poss-my", which is a supertype for "my" in the grammar). The ubertagger's list includes some of

---

[14]The list includes: some punctuation/quotation marks, the tags for out-of-vocabulary proper names, the verb *is*, the pronouns *my* and *me*, and types for denoting times and dates. Cf. Table 3 which shows similar findings on the test data, which we did not take into account.

dataset	description	sent	tok	sec/sen		
				No tagging	Ubertagging	BERT-based supertags
cb	technical essay	713	17,244	6.15	**0.42**	*0.76*
ecpr	e-commerce	1,088	11,550	0.55	**0.05**	0.52
jh*,tg*,ps*, ron*	travel brochures	2,116	34,098	2.40	**0.13**	0.32
petet	textual entailment	581	7,135	1.93	**0.10**	*0.23*
vm32	phone conv.	1,000	8,730	0.73	**0.04**	*0.06*
ws214	Wikipedia	598	12,395	5.68	**0.42**	*1.24*
wsj23	Wall Street J.	950	22,987	6.27	**0.46**	3.60
all	average	7,918	131,441	3.38	**0.24**	0.97

Table 4: Effects of supertagging on DEFAULT parsing speed (ACE Parser)

dataset	description	sent	tok	No tagging			Ubertagging			BERT-based supertags		
				Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
cb	technical essay	713	17,244	0.89	**0.40**	**0.55**	0.86	0.39	0.53	**0.91**	0.27	0.41
ecpr	e-commerce	1,088	11,550	**0.95**	**0.93**	**0.94**	0.93	0.67	0.78	0.94	0.88	0.91
jh*,tg*,ps*, ron*	travel brochures	2,116	34,098	**0.91**	**0.74**	**0.81**	0.89	0.61	0.71	0.91	0.69	0.78
petet	textual entailment	581	7,135	0.93	**0.78**	**0.85**	0.93	0.67	0.78	**0.94**	0.47	0.63
vm32	phone conv.	1,000	8,730	0.92	**0.88**	**0.90**	0.93	0.75	0.83	0.94	0.52	0.67
ws214	Wikipedia	598	12,395	0.89	0.45	0.60	0.88	**0.54**	**0.66**	**0.92**	0.42	0.58
wsj23	Wall Street J.	950	22,987	**0.92**	0.38	0.54	0.90	**0.49**	**0.64**	0.91	0.44	0.60
all	average	7,918	131,441	0.92	**0.65**	**0.74**	0.90	0.59	0.71	**0.93**	0.53	0.65

Table 5: Effects of supertagging on DEFAULT parsing accuracy (EDM metric)

dataset	description	sent	tok	exact match		
				No tagging	Ubertagging	BERT-based supertags
cb	technical essay	713	17,244	0.13	**0.15**	0.10
ecpr	e-commerce	1,088	11,550	**0.50**	0.47	0.47
jh*,tg*,ps*, ron*	travel brochures	2,116	34,098	**0.33**	0.31	0.29
petet	textual entailment	581	7,135	0.46	**0.52**	0.23
vm32	phone conv.	1,000	8,730	0.55	**0.58**	0.49
ws214	Wikipedia	598	12,395	0.23	**0.25**	0.14
wsj23	Wall Street J.	950	22,987	0.13	**0.15**	0.14
all	average	7,918	131,441	0.33	**0.35**	0.26

Table 6: Effects of supertagging on DEFAULT parsing accuracy (exact match over MRS)

dataset	description	sent	tok	sec/sen		
				No tagging	Ubertagging	BERT-based supertags
cb	technical essay	713	17,244	6.15	**0.46**	*0.91*
ecpr	e-commerce	1,088	11,550	0.55	**0.05**	0.52
jh*,tg*,ps*, ron*	travel brochures	2,116	34,098	2.36	**0.16**	*0.40*
petet	textual entailment	581	7,135	1.93	**0.11**	*0.27*
vm32	phone conv.	1,000	8,730	0.73	**0.05**	*0.08*
ws214	Wikipedia	598	12,395	5.68	**0.46**	*1.48*
wsj23	Wall Street J.	950	22,987	6.27	**0.55**	4.04
all	average	7,918	131,441	3,38	**0.27**	1.10

Table 7: Effects of supertagging WITH EXCEPTIONS on parsing speed (ACE Parser)

dataset	description	sent	tok	No tagging			Ubertagging			BERT-based supertags		
				Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
cb	technical essay	713	17,244	0.89	0.40	0.55	0.87	0.43	0.58	**0.92**	**0.49**	**0.65**
ecpr	e-commerce	1,088	11,550	**0.95**	**0.93**	**0.94**	0.93	0.81	0.87	0.95	0.90	0.92
jh*,tg*,ps*, ron*	travel brochures	2,116	34,098	0.91	0.74	0.81	0.89	0.65	0.75	**0.93**	**0.80**	**0.86**
petet	textual entailment	581	7,135	0.93	**0.78**	**0.85**	0.92	0.69	0.79	**0.97**	0.69	0.81
vm32	phone conv.	1,000	8,730	0.92	**0.88**	**0.90**	0.93	0.79	0.86	**0.94**	0.85	0.89
ws214	Wikipedia	598	12,395	0.89	0.45	0.60	0.88	0.54	0.67	**0.93**	**0.56**	**0.70**
wsj23	Wall Street J.	950	22,987	0.92	0.38	0.54	0.89	0.53	0.67	**0.94**	**0.69**	**0.80**
all	average	7,918	131,441	0.92	0.65	0.74	0.90	0.63	0.74	**0.94**	**0.71**	**0.80**

Table 8: Effects of supertagging WITH EXCEPTIONS on parsing accuracy (EDM metric)

the words that we expect would be tagged with some of our excluded types, although in principle, the two models may of course make different mistakes. We did not modify the existing ubertagger nor consulted its exceptions for our list. From the speeds that we are seeing, we conclude that our supertagger is less aggressive than the ubertagger and excludes more words from pruning, losing more in speed but winning considerably in accuracy as a result. This is what we would expect since we exclude entire lexical types and not just individual lexical items. The goal is a balanced tradeoff be-

dataset	description	sent	tok	sec/sen		
				No tagging	Ubertagging	BERT-based supertags
cb	technical essay	713	17,244	0.13	0.17	**0.21**
ecpr	e-commerce	1088	11,550	0.55	**0.56**	0.48
jh*,tg*,ps*, ron*	travel brochures	2,116	34,098	0.33	0.32	**0.38**
petet	textual entailment	581	7,135	0.46	0.54	**0.66**
vm32	phone conv.	1,000	8,730	0.55	0.61	**0.67**
ws214	Wikipedia	598	12,395	0.23	**0.26**	0.22
wsj23	Wall Street J.	950	22,987	0.13	0.17	**0.27**
all	average	7,918	131,441	0.33	0.37	**0.42**

Table 9: Effects of supertagging WITH EXCEPTIONS on parsing accuracy (exact match over MRS)

tween accuracy and speed. We want the supertagger to be noticeably faster than the baseline and much more accurate than the ubertagger. This is what we observe in Tables 7-9.

Because pruning the lexical chart may and often will result in wrongly sacrificing the correct lexical type for a word, we expect the recall for the tagging systems to be lower compared to the no-tagging system. On the other hand, the no-tagging system will often run out of resources and so its overall accuracy may be lower for that reason. What we see in Table 8 is that our supertagging system is the most precise one on most datasets and shows large recall gains on Wikipedia, Wall Street Journal Section 23, and the technical essay data. It is strictly better than the no-tagging system on WSJ23 as well as on Wikipedia and *The Cathedral and the Bazaar*, and it is strictly better than the ubertagger across the board on the partial match EDM metric. While the recall difference is partially explained by the supertagger being less aggressive in pruning, the precision has to be due to the higher accuracy of the tagging model (BERT). On the exact match metric, the ubertagger wins on two datasets: e-commerce and Wikipedia. The supertagger wins on the rest.

Our system is strictly faster than the baseline, by a factor of 3, although on two datasets (e-commerce and WSJ) it fails to achieve a speedup factor of 2. The ubertagger is still the fastest overall, remarkably by a factor of 12, on average across all datasets. This is not too surprising because the supertagger is experimental and it is hard for it to compete with the ubertagger which was integrated into the parser for production, with the focus on performance. We believe that the supertagger could be integrated better into the parser's C code in the future. In other words, its current speed is in part a purely C engineering problem. On the other hand, clearly the exceptions list would have an effect. Since we are excluding 15 types of words from pruning, the supertagger's lexical chart is likely to be bigger than

the ubertagger's. This is the expected tension between speed and accuracy that we expected to see, and our supertagger system shows overall benefits in both speed and accuracy. The only dataset on which our system is not the best in accuracy is the e-commerce (ecpr). It appears that for this type of data, tagging is the least effective; we gain a 6% speed increase with the supertagger at the cost of 3% F-score, while the more aggressive ubertagger parses this data very fast but at the cost of 16% F-score. We note particularly large recall gains on the WSJ data, but this may be related to the fact that statistical systems have been overtrained on WSJ so much that the effects are seen throughout the field (Hovy and Søgaard, 2015).

5 Conclusion and future work

We used the advancements in HPSG treebanking to train more accurate supertaggers. The ERG is a major project in syntactic theory and an important resource for creating high quality semantic treebanks. It has the potential to contribute to NLP tasks that require high precision and/or interpretability including probing of the LLMs, and thus making HPSG parsing faster is strategic for NLP. We tested the new supertagging models with the state-of-the-art HPSG parser and saw improvements in parsing speed as well as accuracy. We consider the results on multiple domains, well beyond the WSJ Section 23. We show promising results but also confirm that domain remains important, and purely statistical systems are brittle and often require rule-based additions in real-life scenarios. We contribute the ERG datasets converted to huggingface transformers format intended for token classification, along with the code which can be adapted for other purposes.

6 Limitations

Our paper is concerned with training supertagging models on an English HPSG treebank. The lim-

itations therefore are associated mainly with the training of the models including neural networks, and with the building of broad-coverage grammars such as the English Resource Grammar. Crucially, while our method does not require industry-scale computational resources, training a neural classifier such as ours still requires a certain amount of training data, and this means that our method assumes that a large HPSG treebank is available for training. The availability of such a treebank, in turn, depends directly on the availability of a broad-coverage grammar. While choosing the gold trees for the treebank can be done relatively fast using treebanking tools once the grammar parsed the corpus, building a broad-coverage grammar itself requires an investment of years of expert work. At the moment, such an investment was made only for a few languages (English, Spanish, Japanese, Chinese), English being the largest one. Furthermore, the coverage of a precision grammar is never perfect and regular grammar updates are needed. A limitation related to using neural networks is that while the NCRF++ library can in principle be very efficient on some tasks (e.g. POS tagging), with our data and large label set it proved relatively slow, and so ideally a more efficient neural architecture may be required for future work in this direction.

Acknowledgments

References

Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. 2020. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems. ArXiv:2007.03051.

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265.

Emily M Bender and Guy Emerson. 2021. Computational linguistics and grammar engineering. In Stephan Müller, Anne Abeillé, Robert D. Borsley, and Jean-Pierre Koenig, editors, *Head-Driven Phrase Structure Grammar: The handbook*.

Philip Blunsom. 2007. *Structured classification for multilingual natural language processing*. Ph.D. thesis, University of Melbourne.

Thorsten Brants. 2000. TnT-a statistical part-of-speech tagger. *arXiv preprint cs/0003055*.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226.

Ulrich Callmeier. 2000. PET–a platform for experimentation with efficient hpsg processing techniques. *Natural Language Engineering*, 6(1):99–107.

Robert Carpenter. 1992. *The logic of typed feature structures: with applications to unification grammars, logic programs and constraint resolution*, volume 32. Cambridge University Press.

John Carroll. 1993. *Practical unification-based parsing of natural language*. Ph.D. thesis, University of Cambridge.

Yufei Chen, Weiwei Sun, and Xiaojun Wan. 2018. Accurate SHRG-based semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 408–418, Melbourne, Australia. Association for Computational Linguistics.

Stephen Clark and James R Curran. 2003. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 97–104.

Ann Copestake. 2002. Definitions of typed feature structures. In Stephan Oepen, Dan Flickinger, Jun-ichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering*, pages 227–230. CSLI Publications, Stanford, CA.

Berthold Crysmann and Woodley Packard. 2012. Towards efficient HPSG generation for German, a non-configurational language. In *COLING*, pages 695–710.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Rebecca Dridan. 2009. *Using lexical statistics to improve HPSG parsing*. Ph.D. thesis, University of Saarland.

Rebecca Dridan. 2013. Ubertagging: Joint segmentation and supertagging for english. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1201–1212.

Rebecca Dridan, Valia Kordoni, and Jeremy Nicholson. 2008. Enhancing performance of lexicalised grammars. In *Proceedings of ACL-08: HLT*, pages 613–621.

Rebecca Dridan and Stephan Oepen. 2011. Parser evaluation using elementary dependency matching. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 225–230, Dublin, Ireland. Association for Computational Linguistics.

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(01):15–28.

Dan Flickinger. 2011. Accuracy v. robustness in grammar engineering. In Emily M. Bender and Jennifer E. Arnold, editors, *Language from a Cognitive Perspective: Grammar, Usage and Processing*, pages 31–50. CSLI Publications, Stanford, CA.

Dan Flickinger and Jiye Yu. 2013. Toward more precision in correction of grammatical errors. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 68–73.

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank. a dynamically annotated treebank of the wall street journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, pages 85–96.

Valerie Hajdik, Jan Buys, Michael W Goodman, and Emily M Bender. 2019. Neural text generation from rich semantic representations. In *Proceedings of NAACL-HLT*, pages 2259–2266.

Dirk Hovy and Anders Søgaard. 2015. Tagging performance correlates with author age. 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 483–488.

Konstantinos Kogkalidis and Michael Moortgat. 2023. Geometry-aware supertagging with heterogeneous dynamic convolutions. In *Proceedings of the 2023 CLASP Conference on Learning with Small Data (LSD)*, pages 107–119.

Zi Lin, Jeremiah Zhe Liu, and Jingbo Shang. 2022. Towards collaborative neural-symbolic graph semantic parsing via uncertainty. *Findings of the Association for Computational Linguistics: ACL 2022*.

Yufang Liu, Tao Ji, Yuanbin Wu, and Man Lan. 2021. Generating CCG categories. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13443–13451.

Robert Malouf, John Carroll, and Ann Copestake. 2000. Efficient feature structure operations without compilation. *Natural Language Engineering*, 6(1):29–46.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-93-87.

Montserrat Marimon, Núria Bel, and Lluís Padró. 2014. Automatic selection of hpsg-parsed sentences for treebank construction. *Computational Linguistics*, 40(3):523–531.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Efficient hpsg parsing with supertagging and cfg-filtering. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1671–1676.

Yusuke Miyao and Jun'ichi Tsujii. 2008. Feature forest models for probabilistic HPSG parsing. *Computational linguistics*, 34(1):35–80.

Luis Morgado da Costa, Francis Bond, and Xiaoling He. 2016. Syntactic well-formedness diagnosis and error-based coaching in computer assisted language learning using machine translation. In *Proceedings of the 3rd Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA2016)*, pages 107–116.

Luis Morgado da Costa, Roger VP Winder, Shu Yun Li, Benedict Christopher Lin Tzer Liang, Joseph Mackinnon, and Francis Bond. 2020. Automated writing support using deep linguistic parsers. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 369–377.

Takashi Ninomiya, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2007. A log-linear model with an n-gram reference distribution for accurate hpsg

parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 60–68. Citeseer.

Stephan Oepen. 1999. [incr tsdb ()] competence and performance laboratory. User and reference manual.

Stephan Oepen and John Carroll. 2002. Efficient parsing for unification-based grammars. In Stephan Oepen, Dan Flickinger, Jun-ichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering*. CSLI Press.

Stephan Oepen, Dan Flickinger, Kristina Toutanova, and Christopher D Manning. 2004. LinGO Redwoods. *Research on Language and Computation*, 2(4):575–596.

Woodley Packard. 2015. Full-forest treebanking. Master's thesis, University of Washington.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. The University of Chicago Press and CSLI Publications, Chicago, IL and Stanford, CA.

Jakob Prange, Nathan Schneider, and Vivek Srikumar. 2021. Supertagging the long tail with tree-structured decoding of complex categories. *Transactions of the Association for Computational Linguistics*, 9:243–260.

RP Prins and GJM van Noord. 2004. Reinforcing parser preferences through tagging. *Traitement Automatique des Langues*, 3:121–139.

Adwait Ratnaparkhi et al. 1996. A maximum entropy model for part-of-speech tagging. In *EMNLP*, volume 1, pages 133–142. Citeseer.

Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49.

Taiga Someya, Ryo Yoshida, and Yohei Oseki. 2024. Targeted syntactic evaluation on the chomsky hierarchy. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 15595–15605.

Yuanhe Tian, Yan Song, and Fei Xia. 2020. Supertagging combinatory categorial grammar with attentive graph convolutional networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6037–6044.

Masaru Tomita. 1985. An efficient context-free parsing algorithm for natural languages. In *IJCAI*, volume 2, pages 756–764.

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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

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.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Deniz Yuret, Aydin Han, and Zehra Turgut. 2010. Semeval-2010 task 12: Parser evaluation using textual entailments. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 51–56.

Yao-Zhong Zhang, Takuya Matsuzaki, and Jun'ichi Tsujii. 2012. Structure-guided supertagger learning. *Natural Language Engineering*, 18(2):205–234.

Yao-Zhong Zhang, Takuya Matsuzaki, and Jun'ichi Tsujii. 2009. HPSG supertagging: A sequence labeling view. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 210–213.

Yao-Zhong Zhang, Takuya Matsuzaki, and Jun'ichi Tsujii. 2010. A simple approach for HPSG supertagging using dependency information. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 645–648.

Yi Zhang and Hans-Ulrich Krieger. 2011. Large-scale corpus-driven PCFG approximation of an HPSG. In *Proceedings of the 12th international conference on parsing technologies*, pages 198–208.

A Appendix A

A.1 Tuning ranges

BERT (Devlin et al., 2019) was fine-tuned using transformers (Wolf et al., 2019) and pytorch (Paszke et al., 2017) using 4 learning rates: 1e-5,

Parameter	value	default/tuned	range
lstm layers	2	tuned	1–4
hidden dim.	800	tuned	100–1200
word embeddings	glove840B	pretrained	
word emb. dim.	300	N/A	
char emb. dim.	50	tuned	30–50
momentum	0	default	
dropout	0.5	default	
l2	1^{-8}	default	

Table 10: NCRF++ model parameters

2e-5, 3e-5, and 5e-6. Cased and uncased pretrained BERT models were tried.

A.2 Computational resources

We trained the neural models with a single NVIDIA GeForce RTX 2080 GPU, CUDA version 11.2. The SVM model and the MaxEnt baseline were trained using Intel Core i7-9700K 3,60Hz CPU (using single core processing for each model). We have experimented with Stochastic Gradient Descent (SGD) optimizer along with AdaGrad, Adam, and AdaDelta. The ranges for parameter values can be found in Table 10. The decoding time (sentences per second) for the models can be found in Table 2. The training times are presented in this Appendix in Table 11. The energy costs as estimated by the Python library carbontracker (Anthony et al., 2020)[15] are in Table 12.

A.3 Development accuracies

The development (validation set) accuracies are presented in Tables 13, 14, and 15. The best models are bolded. NCRF++ has nondeterministic components, and the average dev accuracy of the best (bolded) model in Table 14; the average accuracy is 95.15%; standard deviation 0.07088.

A.4 MaxEnt model

Below we describe in detail how we trained the baseline MaxEnt models.

A.4.1 MaxEnt model selection

Rather than comparing our experimental numbers with numbers obtained by Dridan (2009),[16] we

create our own baseline because we want to be able to compare classic models with neural models with the same amount of training data.

We experimented with autoregressive and non-autoregressive MaxEnt models and in the end chose one MaxEnt as the baseline.

A.4.2 MaxEnt classifiers

We use scikit learn Python library (Pedregosa et al., 2011) to train the baseline MaxEnt classifiers.[17] The scikit learn classifiers are optimized for processing a large number of observations. For that reason, we organized our evaluation data (dev and test) so as to maximize the number of observations passed to the classifier at each step. Dridan's (2009) models were autoregressive; we also implemented autoregressive baseline models, and in order to make them faster at test time, we organized the evaluation data by the word's position in the sentence. So the classifier would first process all the first words in all sentences, then all the second words, etc. For nonautoregressive models, which we also tried in order to find the best-performing baseline model, we just pass the classifier the entire list of observations in their original order.

We choose the single baseline MaxEnt model from the following types of models, by validation on the dev set: (1) MaxEnt autoregressive models which at test time, if more than one sentence is passed to the classifier, first classify all first tokens in all sentences, then all second tokens, etc; (2) MaxEnt nonautoregressive models where the observation tokens are organized in the same way as in (1); (3) MaxEnt nonautoregressive models where tokens are not reordered in any way and are stored consecutively. The best model happens to be of type (3).

All models achieve above 91% accuracy on the dev set. The validation (dev) data consists of one Wikipedia section and one e-commerce corpus (2267 sentences and 25,076 tokens total), with both domains represented also in the training data. Our best performing MaxEnt baseline model is a nonautoregressive 'One-versus-rest' (OVR) model with L1 regularization and SAGA optimizer (92.21% accuracy on the dev set).[18]

---

[15]https://pypi.org/project/carbontracker/

[16]Dridan (2009, p.84) reports getting 91.47% accuracy on in-domain data using the TnT off-the-shelf tagger (Brants, 2000); as a sanity check, we obtain 91.94% using an autoregressive one-versus-rest L1 SAGA MaxEnt model trained with the scikit-learn library (Pedregosa et al., 2011) on training and test datasets very similar to the ones used by Dridan (2009). On *The Cathedral and the Bazaar* with the same setup, we obtain 73.85% compared to Dridan's (2009) 74.61%. We attribute the slight differences to the differences between TnT

and scikit-learn.

[17]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[18]We tried multinomial and OVR models, L1 and L2 regularization, and SAG and SAGA solvers on the dev set. The autoregressive models were not strictly better than the nonau-

Model type	models trained for tuning	total time for all models in this row (sec)
SVM Scikit-learn	1	3664
MaxEnt Scikit-learn	14	106,922
NCRF++	31	955,500 (approx.)
BERT	5	100,000 (approx.)

Table 11: Training times for models used to choose the best baseline and best experimental models

Measurement	Value NCRF++	Value BERT
Process used	5.55 kWh	3.5 kWh
Carbon emissions	1.63 kg CO2	5.5 kg CO2
Equivalent km driven	13 km	5.5 km

Table 12: Energy cost estimate for training the final NCRF++ model in 38 epochs (31 were trained in total, number of epochs varied) and for BERT 50 epochs

Model	dev accuracy(%)
multinomial L2 SAG	91.59
multinomial L2 SAG autoreg	91.41
OVR L2 SAG	91.18
OVR L2 SAG autoreg	91.27
multinomial L2 SAGA	91.53
multinomial L2 SAGA autoreg	88.56
OVR L2 SAGA	91.17
OVR L2 SAGA autoreg	91.26
OVR L1 SAGA	**92.17**
OVR L1 SAGA autoreg	92.12
multinomial L1 SAGA	92.12
multinomial L1 SAGA autoreg	91.17
liblinear SVM	92.88

Table 13: Development (validation) set accuracies for MaxEnt and SVM

A.5 Parsing with more RAM

To give the baseline system an opportunity to build the full lexical chart, more than 50GB RAM is required (24+30), according to our experiments with a subset of the WSJ training data that includes 25 sentences some of which are very long and ambiguous (Yuret et al., 2010), presented below in Table 16. On this dataset, even with 54GB RAM, 100% coverage is not achieved, and the parsing speed becomes intractable (77 sec/sen).

Since spending 77 sec/sen is not viable, we did not run the full experiments with 54GB RAM. We present below a subset of experiments, showing the baseline F-score gain due to higher recall. The ubertagger and the supertagger do not end up with

such large lexical charts and thus do not benefit from more RAM, so we do not repeat the results from Tables 4.2-8 in Table 18.

The 'Verbmobil' (phone conversations) and the 'ecpr' (e-commerce) datasets are easy to parse fast (as we see from Table 7) and on such data, using more RAM may be justified with the baseline system, however other types of data lead to the parsing time increasing noticeably. On the travel brochures data ('jhk'), the baseline system achieves an F-score of 87% at the cost of spending 8.68 seconds per sentence, while our supertagger achieves 86% with only 0.78 seconds/sentence and with only 2.7GB of RAM. Figure 3 summarizes the results presented in Tables 4-8, showing that if we optimize for both speed and F-score, the best models include our model and the ubertagger models.

---

toregressive ones, achieving very similar accuracies on the dev set. They were also much slower, being able to process only 2 sentences per second.

optimizer	lstm layers	char embed	hid dim	embed	epochs (up to 100)	dev accuracy(%)
SGD	1	30	200	pretrained (NCRF++)	17	92.14
SGD	4	30	200	pretrained (NCRF++)	23	93.20
SGD	1	30	200	random	19	92.82
SGD	1	30	200	glove840	17	93.69
SGD	4	30	200	glove840	23	94.27
SGD	1	30	250	glove840	20	93.86
SGD	4	30	400	glove840	18	94.68
SGD	4	30	100	glove840	19	93.07
SGD	2	30	200	glove840	20	94.04
SGD	4	30	1200	glove840	18	94.74
SGD	4	30	600	glove840	19	94.62
SGD	2	30	400	glove840	13	94.56
SGD	4	30	1400	glove840	16	94.68
SGD	5	30	1200	glove840	12	94.35
SGD	4	50	1200	glove840	21	94.66
SGD	3	30	1200	glove840	21	94.92
SGD	2	30	1200	glove840	19	95.01
SGD	1	30	1200	glove840	16	94.60
SGD	3	30	1400	glove840	14	94.65
SGD	3	30	600	glove840	15	94.86
SGD	2	50	1200	glove840	19	95.00
SGD	2	30	1400	glove840	8	94.57
SGD+0.3 momentum	3	30	1200	glove840	21	94.67
SGD	2	30	1200	glove840	12	94.69
SGD	2	30	1000	glove840	17	94.95
SGD	**2**	**30**	**800**	**glove840**	17	**95.12**
adagrad	1	30	200	glove840	17	92.00
adagrad	4	30	200	glove840	42	92.42
adam	1	30	200	glove840	1	86.85
adadelta	4	30	200	glove840	stuck on 73	did not finish

Table 14: Development (validation) set accuracies for neural models

Model	dev accuracy(%)
BERT cased LR 2e-5	**96.46**
BERT cased LR 1e-5	96.37
BERT cased LR 3e-5	96.31
BERT cased LR 5e-6	96.34
BERT uncased LR 2e-5	95.97

Table 15: Development (validation) set accuracies fine-tuned BERT models

	2.7GB RAM (default)	54GB RAM
baseline coverage	11/25 (44%)	19/25 (76%)
baseline speed (sec/sen)	0.77	77

Table 16: Baseline (no tagging) coverage gain with more RAM on the PETE dataset (ERG version)

	2.7GB RAM (default)			54GB RAM		
dataset	coverage	F-score	speed	coverage	F-score	speed
ecpr	0.99	0.94	0.54	0.99	0.95	0.68
jhk	0.87	0.78	2.61	0.98	0.87	8.68
petet	0.92	0.85	1.96	0.99	0.92	4.34
vm32	0.98	0.90	0.74	0.99	0.92	0.99

Table 17: Baseline (no tagging) recall gains and speed loss with generous RAM

dataset	No tagging			Ubertagging			BERT supertagging		
	coverage	F-score	speed	coverage	F-score	speed	coverage	F-score	speed
cb	0.86	0.77	59.3	0.58	0.58	0.66	0.63	0.64	8.58
ecpr	0.99	0.95	0.68	0.96	0.87	0.06	0.97	0.93	0.68
jhk	0.98	0.88	8.89	0.81	0.75	0.22	0.91	0.87	0.35
petet	0.99	0.92	4.34	0.79	0.79	0.12	0.85	0.85	0.32
vm32	0.99	0.92	0.99	0.87	0.86	0.05	0.94	0.90	0.10
wsj23	0.85	0.79	52.2	0.64	0.69	0.81	-	-	-

Table 18: Parsing with 54GB RAM