Automatic Generation of High Quality CCGbanks for Parser Domain Adaptation

Masashi Yoshikawa[1]
yoshikawa.masashi.yh8@is.naist.jp

Hiroshi Noji[2]
hiroshi.noji@aist.go.jp

Koji Mineshima[3]
mineshima.koji@ocha.ac.jp

Daisuke Bekki[3]
bekki@is.ocha.ac.jp

[1]Nara Institute of Science and Technology, Nara, Japan
[2]Artificial Intelligence Research Center, AIST, Tokyo, Japan
[3]Ochanomizu University, Tokyo, Japan

Abstract

We propose a new domain adaptation method for Combinatory Categorial Grammar (CCG) parsing, based on the idea of automatic generation of CCG corpora exploiting cheaper resources of dependency trees. Our solution is conceptually simple, and not relying on a specific parser architecture, making it applicable to the current best-performing parsers. We conduct extensive parsing experiments with detailed discussion; on top of existing benchmark datasets on (1) biomedical texts and (2) question sentences, we create experimental datasets of (3) speech conversation and (4) math problems. When applied to the proposed method, an off-the-shelf CCG parser shows significant performance gains, improving from 90.7% to 96.6% on speech conversation, and from 88.5% to 96.8% on math problems.

1 Introduction

The recent advancement of Combinatory Categorial Grammar (CCG; Steedman (2000)) parsing (Lee et al., 2016; Yoshikawa et al., 2017), combined with formal semantics, has enabled high-performing natural language inference systems (Abzianidze, 2017; Martínez-Gómez et al., 2017). We are interested in transferring the success to a range of applications, e.g., inference systems on scientific papers and speech conversation.

To achieve the goal, it is urgent to enhance the CCG parsing accuracy on new domains, i.e., solving a notorious problem of *domain adaptation* of a statistical parser, which has long been addressed in the literature. Especially in CCG parsing, prior work (Rimell and Clark, 2008; Lewis et al., 2016) has taken advantage of highly informative categories, which determine the most part of sentence structure once correctly assigned to words. It is demonstrated that the annotation of only preterminal categories is sufficient to adapt a CCG parser to new domains. However, the solution is limited to a specific parser's architecture, making non-trivial the application of the method to the current state-of-the-art parsers (Lee et al., 2016; Yoshikawa et al., 2017; Stanojević and Steedman, 2019), which require full parse annotation. Additionally, some ambiguities remain unresolved with mere supertags, especially in languages other than English (as discussed in Yoshikawa et al. (2017)), to which the method is not portable.

Distributional embeddings are proven to be powerful tools for solving the issue of domain adaption, with their unlimited applications in NLP, not to mention syntactic parsing (Lewis and Steedman, 2014b; Mitchell and Steedman, 2015; Peters et al., 2018). Among others, Joshi et al. (2018) reports huge performance boosts in constituency parsing using contextualized word embeddings (Peters et al., 2018), which is orthogonal to our work, and the combination shows huge gains. Including Joshi et al. (2018), there are studies to learn from partially annotated trees (Mirroshandel and Nasr, 2011; Li et al., 2016; Joshi et al., 2018), again, most of which exploit specific parser architecture.

In this work, we propose a conceptually simpler approach to the issue, which is agnostic on any parser architecture, namely, *automatic generation of CCGbanks* (i.e., CCG treebanks)[1] for new domains, by exploiting cheaper resources of dependency trees. Specifically, we train a deep conversion model to map a dependency tree to a CCG tree, on aligned annotations of the Penn Treebank (Marcus et al., 1993) and the English CCGbank (Hockenmaier and Steedman, 2007) (Figure 1a). When we need a CCG parser tailored for

[1]In this paper, we call a treebank based on CCG grammar *a CCGbank*, and refer to the specific one constructed in Hockenmaier and Steedman (2007) as *the English CCGbank*.

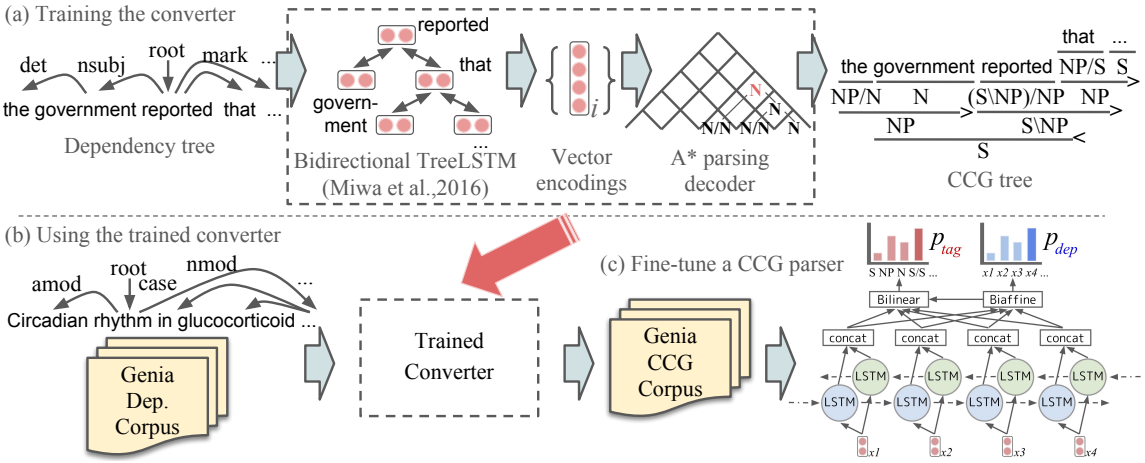

Figure 1: Overview of the proposed method. (a) A neural network-based model is trained to convert a dependency tree to a CCG one using aligned annotations on WSJ part of the Penn Treebank and the English CCGbank. (b) The trained converter is applied to an existing dependency corpus (e.g., the Genia corpus) to generate a CCGbank, (c) which is then used to fine-tune the parameters of an off-the-shelf CCG parser.

a new domain, the trained converter is applied to a dependency corpus in that domain to obtain a new CCGbank (1b), which is then used to fine-tune an off-the-shelf CCG parser (1c). The assumption that we have a dependency corpus in that target domain is not demanding given the abundance of existing dependency resources along with its developed annotation procedure, e.g., Universal Dependencies (UD) project (Nivre et al., 2016), and the cheaper cost to train an annotator.

One of the biggest bottlenecks of syntactic parsing is handling of countless *unknown words*. It is also true that there exist such unfamiliar input data types to our converter, e.g., disfluencies in speech and symbols in math problems. We address these issues by *constrained decoding* (§4), enabled by incorporating a parsing technique into our converter. Nevertheless, syntactic structures exhibit less variance across textual domains than words do; our proposed converter suffers less from such unseen events, and expectedly produces high-quality CCGbanks.

The work closest to ours is Jiang et al. (2018), where a conversion model is trained to map dependency treebanks of different annotation principles, which is used to increase the amount of labeled data in the target-side treebank. Our work extends theirs and solves a more challenging task; the mapping to learn is to more complex CCG trees, and it is applied to datasets coming from plainly different natures (i.e., domains). Some prior studies design conversion algorithms to induce CCGbanks for languages other than English from dependency treebanks (Bos et al., 2009; Ambati et al., 2013). Though the methods may be applied to our problem, they usually cannot cover the entire dataset, consequently discarding sentences with characteristic features. On top of that, unavoidable information gaps between the two syntactic formalisms may at most be addressed probabilistically.

To verify the generalizability of our approach, on top of the existing benchmarks on (1) **biomedical texts** and (2) **question sentences** (Rimell and Clark, 2008), we conduct parsing experiments on (3) **speech conversation texts**, which exhibit other challenges such as handling informal expressions and lengthy sentences. We create a CCG version of the Switchboard corpus (Godfrey et al., 1992), consisting of full train/dev/test sets of automatically generated trees and manually annotated 100 sentences for a detailed evaluation. Additionally, we manually construct experimental data for parsing (4) **math problems** (Seo et al., 2015), for which the importance of domain adaptation is previously demonstrated (Joshi et al., 2018). We observe huge additive gains in the performance of the `depccg` parser (Yoshikawa et al., 2017), by combining contextualized word embeddings (Peters et al., 2018) and our domain adaptation method: in terms of unlabeled F1 scores, 90.68% to 95.63% on speech conversation, and 88.49% to 95.83% on math problems, respectively.[2]

[2]All the programs and resources used in this work are available at: https://github.com/masashi-y/depccg.

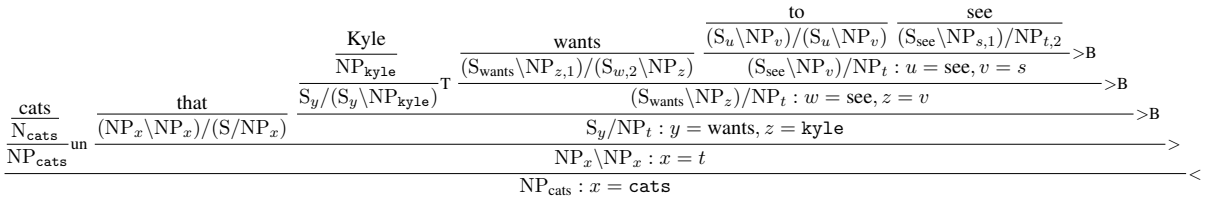

Figure 2: Example CCG derivation tree for phrase *cats that Kyle wants to see*. Categories are combined using rules such as an application rule (marked with ">", $\mathrm{X/Y}\ \ \mathrm{Y} \Rightarrow \mathrm{X}$) and a composition rule (">B": $\mathrm{X/Y}\ \ \mathrm{Y/Z} \Rightarrow \mathrm{X/Z}$). See Steedman (2000) for the detail.

2 Combinatory Categorial Grammar

CCG is a lexicalized grammatical formalism, where words and phrases are assigned categories with complex internal structures. A category $\mathrm{X/Y}$ (or $\mathrm{X\backslash Y}$) represents a phrase that combines with a Y phrase on its right (or left), and becomes an X phrase. As such, a category $(\mathrm{S\backslash NP})/\mathrm{NP}$ represents an English transitive verb which takes NPs on both sides and becomes a sentence (S).

The semantic structure of a sentence can be extracted using the functional nature of CCG categories. Figure 2 shows an example CCG derivation of a phrase *cats that Kyle wants to see*, where categories are marked with variables and constants (e.g., `kyle` in $\mathrm{NP}_{\texttt{kyle}}$), and argument ids in the case of verbs (subscripts in $(\mathrm{S}_{\text{see}}\backslash\mathrm{NP}_{s,1})/\mathrm{NP}_{t,2}$). Unification is performed on these variables and constants in the course of derivation, resulting in chains of equations $s = v = z = \texttt{kyle}$, and $t = x = \texttt{cats}$, successfully recovering the first and second argument of *see*: *Kyle* and *cats* (i.e., capturing *long-range dependencies*). What is demonstrated here is performed in the standard evaluation of CCG parsing, where the number of such correctly predicted predicate-argument relations is calculated (for the detail, see Clark et al. (2002)). Remarkably, it is also the basis of CCG-based semantic parsing (Abzianidze, 2017; Martínez-Gómez et al., 2017; Matsuzaki et al., 2017), where the above simple unification rule is replaced with more sophisticated techniques such as λ-calculus.

There are two major resources in CCG: the English CCGbank (Hockenmaier and Steedman, 2007) for news texts, and the Groningen Meaning Bank (Bos et al., 2017) for wider domains, including Aesop's fables. However, when one wants a CCG parser tuned for a specific domain, he or she faces the issue of its high annotation cost:

- The annotation requires linguistic expertise, being able to keep track of semantic composition performed during a derivation.
- An annotated tree must strictly conform to the grammar, e.g., inconsistencies such as combining N and $\mathrm{S\backslash NP}$ result in ill-formed trees and hence must be disallowed.

We relax these assumptions by using *dependency tree*, which is a simpler representation of the syntactic structure, i.e., it lacks information of long-range dependencies and conjunct spans of a coordination structure. However, due to its simplicity and flexibility, it is easier to train an annotator, and there exist plenty of accessible dependency-based resources, which we exploit in this work.

3 Dependency-to-CCG Converter

We propose a domain adaptation method based on the automatic generation of a CCGbank out of a dependency treebank in the target domain. This is achieved by our dependency-to-CCG converter, a neural network model consisting of a dependency tree encoder and a CCG tree decoder.

In the encoder, higher-order interactions among dependency edges are modeled with a bidirectional TreeLSTM (Miwa and Bansal, 2016), which is important to facilitate mapping from a dependency tree to a more complex CCG tree. Due to the strict nature of CCG grammar, we model the output space of CCG trees explicitly[3]; our decoder is inspired by the recent success of A* CCG parsing (Lewis and Steedman, 2014a; Yoshikawa et al., 2017), where the most probable valid tree is found using A* parsing (Klein and D. Manning, 2003). In the following, we describe the details of the proposed converter.

[3]The strictness and the large number of categories make it still hard to leave everything to neural networks to learn. We trained constituency-based RSP parser (Joshi et al., 2018) on the English CCGbank by disguising the trees as constituency ones, whose performance could not be evaluated since most of the output trees violated the grammar.

Firstly, we define a probabilistic model of the dependency-to-CCG conversion process. According to Yoshikawa et al. (2017), the structure of a CCG tree $\boldsymbol{y}$ for sentence $\boldsymbol{x} = (x_1, ..., x_N)$ is almost uniquely determined[4] if a sequence of the pre-terminal CCG categories (supertags) $\boldsymbol{c} = (c_1, ..., c_N)$ and a dependency structure $\boldsymbol{d} = (d_1, ..., d_N)$, where $d_i \in \{0, ..., N\}$ is an index of dependency parent of x_i (0 represents a root node), are provided. Note that the dependency structure $\boldsymbol{d}$ is generally different from an input dependency tree.[5] While supertags are highly informative about the syntactic structure (Bangalore and Joshi, 1999), remaining ambiguities such as attachment ambiguities need to be modeled using dependencies. Let the input dependency tree of sentence $\boldsymbol{x}$ be $\boldsymbol{z} = (\boldsymbol{p}, \boldsymbol{d}', \boldsymbol{\ell})$, where p_i is a part-of-speech tag of x_i, d'_i an index of its dependency parent, ℓ_i is the label of the corresponding dependency edge, then the conversion process is expressed as follows:[6]

$$P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{N} p_{tag}(c_i|\boldsymbol{x}, \boldsymbol{z}) \prod_{i=1}^{N} p_{dep}(d_i|\boldsymbol{x}, \boldsymbol{z}).$$

Based on this formulation, we model c_i and d_i conditioned on a dependency tree $\boldsymbol{z}$, and search for $\boldsymbol{y}$ that maximizes $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ using A* parsing.

Encoder A bidirectional TreeLSTM consists of two distinct TreeLSTMs (Tai et al., 2015). A *bottom-up* TreeLSTM recursively computes a hidden vector $\boldsymbol{h}_i^{\uparrow}$ for each x_i, from vector representation $\boldsymbol{e}_i$ of the word and hidden vectors of its dependency children $\{\boldsymbol{h}_j^{\uparrow} | d'_j = i\}$. A *top-down* TreeLSTM, in turn, computes $\boldsymbol{h}_i^{\downarrow}$ using $\boldsymbol{e}_i$ and a hidden vector of the dependency parent $\boldsymbol{h}_{d'_i}^{\downarrow}$. In total, a bidirectional TreeLSTM returns concatenations of hidden vectors for all words: $\boldsymbol{h}_i = \boldsymbol{h}_i^{\uparrow} \oplus \boldsymbol{h}_i^{\downarrow}$.

We encode a dependency tree as follows, where $\boldsymbol{e}_v$ denotes the vector representation of variable v, and Ω and $\Xi_{\boldsymbol{d}'}$ are shorthand notations of the series of operations of sequential and tree bidirectional LSTMs, respectively:

$$\boldsymbol{e}_1, ..., \boldsymbol{e}_N = \Omega(\boldsymbol{e}_{p_1} \oplus \boldsymbol{e}_{x_1}, ..., \boldsymbol{e}_{p_N} \oplus \boldsymbol{e}_{x_N}),$$
$$\boldsymbol{h}_1, ..., \boldsymbol{h}_N = \Xi_{\boldsymbol{d}'}(\boldsymbol{e}_1 \oplus \boldsymbol{e}_{\ell_1}, ..., \boldsymbol{e}_N \oplus \boldsymbol{e}_{\ell_N}).$$

[4]The uniqueness is broken if a tree contains a unary node.

[5]In this work, input dependency tree is based on Universal Dependencies (Nivre et al., 2016), while dependency structure $\boldsymbol{d}$ of a CCG tree is Head First dependency tree introduced in Yoshikawa et al. (2017). See § 5 for the detail.

[6]Here, the independence of each c_is and d_is is assumed.

Decoder The decoder part adopts the same architecture as in Yoshikawa et al. (2017), where $p_{dep|tag}$ probabilities are computed on top of $\{\boldsymbol{h}_i\}_{i \in [0,N]}$, using a *biaffine* layer (Dozat and Manning, 2017) and a bilinear layer, respectively, which are then used in A* parsing to find the most probable CCG tree.

Firstly a biaffine layer is used to compute unigram head probabilities p_{dep} as follows:

$$\boldsymbol{r}_i = \psi_{child}^{dep}(\boldsymbol{h}_i), \quad \boldsymbol{r}_j = \psi_{head}^{dep}(\boldsymbol{h}_j),$$
$$s_{i,j} = \boldsymbol{r}_i^{\mathrm{T}} W \boldsymbol{r}_j + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{r}_j,$$
$$p_{dep}(d_i = j|\boldsymbol{x}, \boldsymbol{z}) \propto \exp(s_{i,j}),$$

where ψ denotes a multi-layer perceptron. The probabilities p_{tag} are computed by a bilinear transformation of vector encodings x_i and $x_{\hat{d}_i}$, where $\hat{d}_i$ is the most probable dependency head of x_i with respect to p_{dep}: $\hat{d}_i = \arg\max_j p_{dep}(d_i = j|\boldsymbol{x}, \boldsymbol{z})$.

$$\boldsymbol{q}_i = \psi_{child}^{tag}(\boldsymbol{h}_i), \quad \boldsymbol{q}_{\hat{d}_i} = \psi_{head}^{tag}(\boldsymbol{h}_{\hat{d}_i}),$$
$$s_{i,c} = \boldsymbol{q}_i^{\mathrm{T}} W_c \boldsymbol{q}_{\hat{d}_i} + \boldsymbol{v}_c^{\mathrm{T}} \boldsymbol{q}_i + \boldsymbol{u}_c^{\mathrm{T}} \boldsymbol{q}_{\hat{d}_i} + b_c,$$
$$p_{tag}(c_i = c|\boldsymbol{x}, \boldsymbol{z}) \propto \exp(s_{i,c}).$$

A* Parsing Since the probability $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ of a CCG tree $\boldsymbol{y}$ is simply decomposable into probabilities of subtrees, the problem of finding the most probable tree can be solved with a chart-based algorithm. In this work, we use one of such algorithms, A* parsing (Klein and D. Manning, 2003). A* parsing is a generalization of A* search for shortest path problem on a graph, and it controls subtrees (corresponding to a node in a graph case) to visit next using a priority queue. We follow Yoshikawa et al. (2017) exactly in formulating our A* parsing, and adopt an admissible heuristic by taking the sum of the max $p_{tag|dep}$ probabilities outside a subtree. The advantage of employing an A* parsing-based decoder is not limited to the optimality guarantee of the decoded tree; it enables constrained decoding, which is described next.

4 Constrained Decoding

While our method is a fully automated treebank generation method, there are often cases where we want to control the form of output trees by using external language resources. For example, when generating a CCGbank for biomedical domain, it will be convenient if a disease dictionary is utilized to ensure that a complex disease name in a text is always assigned the category NP. In our

decoder based on A* parsing, it is possible to perform such a controlled generation of a CCG tree by imposing *constraints* on the space of trees.

A constraint is a triplet (c, i, j) representing a constituent of category c spanning over words $x_i, ..., x_j$. The constrained decoding is achieved by refusing to add a subtree (denoted as (c', k, l), likewise, with its category and span) to the priority queue when it meets one of the conditions:

- The spans overlap: $i < k \leq j < l$ or $k < i \leq l < j$.
- The spans are identical ($i = k$ and $j = l$), while the categories are different ($c \neq c'$) and no category c'' exists such that $c' \Rightarrow c''$ is a valid unary rule.

The last condition on unary rule is necessary to prevent structures such as (NP (N dog)) from being accidentally discarded, when using a constraint to make a noun phrase to be NP. A set of multiple constraints are imposed by checking the above conditions for each of the constraints when adding a new item to the priority queue. When one wants to constrain a terminal category to be c, that is achieved by manipulating p_{tag}: $p_{tag}(c|\boldsymbol{x}, \boldsymbol{z}) = 1$ and for all categories $c' \neq c$, $p_{tag}(c'|\boldsymbol{x}, \boldsymbol{z}) = 0$.

5 Experiments

5.1 Experimental Settings

We evaluate our method in terms of performance gain obtained by fine-tuning an off-the-shelf CCG parser `depccg` (Yoshikawa et al., 2017), on a variety of CCGbanks obtained by converting existing dependency resources using the method.

In short, the method of `depccg` is equivalent to omitting the dependence on a dependency tree $\boldsymbol{z}$ from $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ of our converter model, and running an A* parsing-based decoder on $p_{tag|dep}$ calculated on $\boldsymbol{h}_1, ..., \boldsymbol{h}_N = \Omega(\boldsymbol{e}_{x_1}, ..., \boldsymbol{e}_{x_N})$, as in our method. In the plain `depccg`, the word representation $\boldsymbol{e}_{x_i}$ is a concatenation of GloVe[7] vectors and vector representations of affixes. As in the previous work, the parser is trained on both the English CCGbank (Hockenmaier and Steedman, 2007) and the tri-training dataset by Yoshikawa et al. (2017). In this work, on top of that, we include as a baseline a setting where the affix vectors are replaced by contextualized word representation (ELMo; Peters et al. (2018)) ($\boldsymbol{e}_{x_i} = \boldsymbol{x}_{x_i}^{GloVe} \oplus \boldsymbol{x}_{x_i}^{ELMo}$),[8] which we find marks the current best scores in the English CCGbank parsing (Table 1).

Method	UF1	LF1
`depccg`	94.0	88.8
+ ELMo	94.98	90.51
Converter	96.48	92.68

Table 1: The performance of baseline CCG parsers and the proposed converter on WSJ23, where UF1 and LF1 represents unlabeled and labeled F1, respectively.

The evaluation is based on the standard evaluation metric, where the number of correctly predicted predicate argument relations is calculated (§2), where *labeled* metrics take into account the category through which the dependency is constructed, while *unlabeled* ones do not.

Implementation Details The input word representations to the converter are the concatenation of GloVe and ELMo representations. Each of $\boldsymbol{e}_{p_i}$ and $\boldsymbol{e}_{\ell_i}$ is randomly initialized 50-dimensional vectors, and the two-layer sequential LSTMs Ω outputs 300 dimensional vectors, as well as bidirectional TreeLSTM $\Xi_{\boldsymbol{d}'}$, whose outputs are then fed into 1-layer 100-dimensional MLPs with ELU non-linearity (Clevert et al., 2016). The training is done by minimizing the sum of negative log likelihood of $p_{tag|dep}$ using the Adam optimizer (with $\beta_1 = \beta_2 = 0.9$), on a dataset detailed below.

Data Processing In this work, the input tree to the converter follows Universal Dependencies (UD) v1 (Nivre et al., 2016). Constituency-based treebanks are converted using the Stanford Converter[9] to obtain UD trees. The output dependency structure $\boldsymbol{d}$ follows Head First dependency tree (Yoshikawa et al., 2017), where a dependency arc is always from left to right. The conversion model is trained to map UD trees in the Wall Street Journal (WSJ) portion 2-21 of the Penn Treebank (Marcus et al., 1993) to its corresponding CCG trees in the English CCGbank (Hockenmaier and Steedman, 2007).

[7] https://nlp.stanford.edu/projects/glove/

[8] We used the "original" ELMo model, with 1,024-dimensional word vector outputs (https://allennlp.org/elmo).

[9] https://nlp.stanford.edu/software/stanford-dependencies.shtml. We used the version 3.9.1.

Relation	Parser	Converter	#
(a) *PPs attaching to NP / VP*			
$(NP\backslash\underline{\mathbf{NP}})/NP$	90.62	97.46	2,561
$(S\backslash NP)\backslash(\underline{\mathbf{S}\backslash\mathbf{NP}})/NP$	81.15	88.63	1,074
(b) *Subject / object relative clauses*			
$(NP\backslash\underline{\mathbf{NP}})/(S_{dcl}\backslash NP)$	93.44	98.71	307
$(NP\backslash\underline{\mathbf{NP}})/(S_{dcl}/NP)$	90.48	93.02	20

Table 2: Per-relation F1 scores of the proposed converter and depccg + ELMo (Parser). "#" column shows the number of occurrence of the phenomenon.

Fine-tuning the CCG Parser In each of the following domain adaptation experiments, newly obtained CCGbanks are used to fine-tune the parameters of the baseline parser described above, by re-training it on the mixture of labeled examples from the new target-domain CCGbank, the English CCGbank, and the tri-training dataset.

5.2 Evaluating Converter's Performance

First, we examine whether the trained converter can produce high-quality CCG trees, by applying it to dependency trees in the test portion (WSJ23) of Penn Treebank and then calculating the standard evaluation metrics between the resulting trees and the corresponding gold trees (Table 1). This can be regarded as evaluating the upper bound of the conversion quality, since the evaluated data comes from the same domain as the converter's training data. Our converter shows much higher scores compared to the current best-performing depccg combined with ELMo (1.5% and 2.17% up in unlabeled/labeled F1 scores), suggesting that, using the proposed converter, we can obtain CCGbanks of high quality.

Inspecting the details, the improvement is observed across the board (Table 2); the converter precisely handles PP-attachment (2a), notoriously hard parsing problem, by utilizing input's pobj dependency edges, as well as relative clauses (2b), one of well-known sources of long-range dependencies, for which the converter has to learn from the non-local combinations of edges, their labels and part-of-speech tags surrounding the phenomenon.

5.3 Biomedical Domain and Questions

Previous work (Rimell and Clark, 2008) provides CCG parsing benchmark datasets in biomedical texts and question sentences, each representing two contrasting challenges for a newswire-trained parser, i.e., a large amount of out-of-vocabulary words (biomedical texts), and rare or even unseen grammatical constructions (questions).

Method	P	R	F1
C&C	77.8	71.4	74.5
EasySRL	81.8	82.6	82.2
depccg	83.11	82.63	82.87
+ ELMo	85.87	85.34	85.61
+ GENIA1000	85.45	84.49	84.97
+ Proposed	**86.90**	**86.14**	**86.52**

Table 3: Results on the biomedical domain dataset (§5.3). P and R represent precision and recall, respectively. The scores of C&C and EasySRL fine-tuned on the GENIA1000 is included for comparison (excerpted from Lewis et al. (2016)).

Method	P	R	F1
C&C	-	-	86.8
EasySRL	88.2	87.9	88.0
depccg	90.42	**90.15**	**90.29**
+ ELMo	**90.55**	89.86	90.21
+ Proposed	90.27	89.97	90.12

Table 4: Results on question sentences (§5.3). All of baseline C&C, EasySRL and depccg parsers are re-trained on Questions data.

Since the work also provides small training datasets for each domain, we utilize them as well: GENIA1000 with 1,000 sentences and Questions with 1,328 sentences, both annotated with pre-terminal CCG categories. Since pre-terminal categories are not sufficient to train depccg, we automatically annotate Head First dependencies using RBG parser (Lei et al., 2014), trained to produce this type of trees (We follow Yoshikawa et al. (2017)'s tri-training setup).

Following the previous work, the evaluation is based on the Stanford grammatical relations (GR; Marneffe et al. (2006)), a deep syntactic representation that can be recovered from a CCG tree.[10]

Biomedical Domain By converting the Genia corpus (Tateisi et al., 2005), we obtain a new CCGbank of 4,432 sentences from biomedical papers annotated with CCG trees. During the process, we have successfully assigned the category NP to all the occurrences of complex biomedical terms by imposing constraints (§4) that NP spans in the original corpus be assigned the category NP in the resulting CCG trees as well.

[10]We used their public script (https://www.cl.cam.ac.uk/~sc609/candc-1.00.html).

Table 3 shows the results of the parsing experiment, where the scores of previous work (C&C (Clark and Curran, 2007) and EasySRL (Lewis et al., 2016)) are included for reference. The plain depccg already achieves higher scores than these methods, and boosts when combined with ELMo (improvement of 2.73 points in terms of F1). Fine-tuning the parser on GENIA1000 results in a mixed result, with slightly lower scores. This is presumably because the automatically annotated Head First dependencies are not accurate. Finally, by fine-tuning on the Genia CCGbank, we observe another improvement, resulting in the highest 86.52 F1 score.

Questions In this experiment, we obtain a CCG version of the QuestionBank (Judge et al., 2006), consisting of 3,622 question sentences, excluding ones contained in the evaluation data.

Table 4 compares the performance of depccg fine-tuned on the QuestionBank, along with other baselines. Contrary to our expectation, the plain depccg retrained on Questions data performs the best, with neither ELMo nor the proposed method taking any effect. We hypothesize that, since the evaluation set contains sentences with similar constructions, the contributions of the latter two methods are less observable on top of Questions data. Inspection of the output trees reveals that this is actually the case; the majority of differences among parser's configurations are irrelevant to question constructions, suggesting that the models capture well the syntax of question in the data.[11]

5.4 Speech Conversation

Setup We apply the proposed method to a new domain, transcription texts of speech conversation, with new applications of CCG parsing in mind. We create the CCG version of the Switchboard corpus (Godfrey et al., 1992), by which, as far as we are aware of, we conduct the first CCG parsing experiments on speech conversation.[12] We obtain a new CCGbank of 59,029/3,799/7,681 sentences for each of the train/test/development set, where the data split follows prior work on dependency parsing on this dataset (Honnibal and Johnson, 2014).

a.	*we should cause it does help*
b.	*the only problem i see with term limitations is that i think that the bureaucracy in our government as is with most governments is just so complex that there is a learning curve and that you ca n't just send someone off to washington and expect his first day to be an effective congress precision*

Table 5: Example sentences from the manually annotated subset of Switchboard test set.

Error type	#
PP-attachment	3
Adverbs attaching wrong place	11
Predicate-argument	5
Imperative	2
Informal functional words	2
Others	11

Table 6: Error types observed in the manually annotated Switchboard subset data.

In the conversion, we have to handle one of the characteristics of speech transcription texts, *disfluencies*. In real application, it is ideal to remove disfluencies such as interjection and repairs (e.g., *I want a flight to Boston um to Denver*), prior to performing CCG-based semantic composition. Since this corpus contains a layer of annotation that labels their occurrences, we perform constrained decoding to mark the gold disfluencies in a tree with a dummy category X, which can combine with any category from both sides (i.e., for all category C, C X $\Rightarrow$ C and X C $\Rightarrow$ C are allowed). In this work, we perform parsing experiments on texts that are clean of disfluencies, by removing X-marked words from sentences (i.e., a pipeline system setting with an oracle disfluency detection preprocessor).[13]

Another issue in conducting experiments on this dataset is evaluation. Since there exists no evaluation protocol for CCG parsing on speech texts, we evaluate the quality of output trees by two procedures; in the first experiment, we parse the entire test set, and convert them to constituency trees us-

[11] Due to many-to-many nature of mapping to GRs, the evaluation set contains relations not recoverable from the gold supertags using the provided script; for example, we find that from the annotated supertags of sentence *How many battles did she win ?*, the (amod battle many) relation is obtained instead of the gold det relation. This implies one of the difficulties to obtain further improvement on this set.

[12] Since the annotated part-of-speech tags are noisy, we automatically reannotate them using the core_web_sm model of spaCy (https://spacy.io/), version 2.0.16.

[13] We regard developing joint disfluency detection and syntactic parsing method based on CCG as future work.

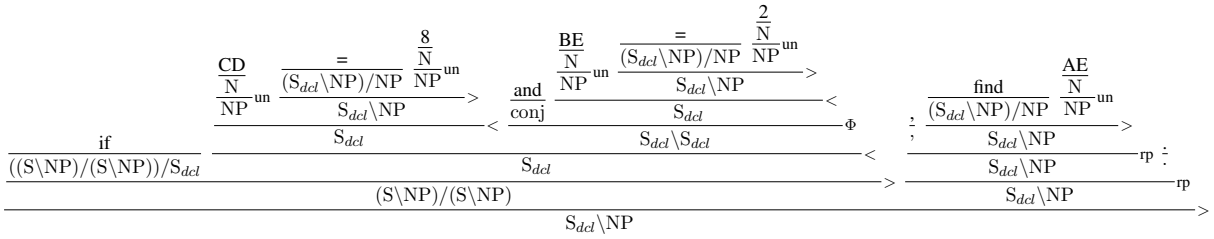

Figure 3: Parse output by the re-trained parser for sentence *if CD = 8 and BE = 2, find AE.* from math problems.

Method	Whole			Subset	
	P	R	F1	UF1	LF1
depccg	74.73	73.91	74.32	90.68	82.46
+ ELMo	75.76	76.62	76.19	93.23	86.46
+ Proposed	**78.03**	**77.06**	**77.54**	**95.63**	**92.65**

Table 7: Results on speech conversation texts (§5.4), on the whole test set and the manually annotated subset.

Method	UF1	LF1
depccg	88.49	66.15
+ ELMo	89.32	70.74
+ Proposed	**95.83**	**80.53**

Table 8: Results on math problems (§5.5).

ing a method by Kummerfeld et al. (2012).[14] We report labeled bracket F1 scores between the resulting trees and the gold trees in the true Switchboard corpus, using the EVALB script.[15] However, the reported scores suffer from the compound effect of failures in CCG parsing as well as ones occurred in the conversion to the constituency trees. To evaluate the parsing performance in detail, the first author manually annotated a subset of randomly sampled 100 sentences from the test set. Sentences with less than four words are not contained, to exclude short phrases such as nodding. Using this test set, we report the standard CCG parsing metrics. Sentences from this domain exhibit other challenging aspects (Table 5), such as less formal expressions (e.g., use of *cause* instead of *because*) (5a), and lengthy sentences with many embedded phrases (5b).[16]

Results On the whole test set, depccg shows consistent improvements when combined with ELMo and the proposed method, in the constituency-based metrics (**Whole** columns in Table 7). Though the entire scores are relatively lower, the result suggests that the proposed method is effective to this domain on the whole. By directly evaluating the parser's performance in terms of predicate argument relations (**Subset** columns), we observe that it actually recovers the most of the dependencies, with the fine-tuned depccg achieving as high as 95.63% unlabeled F1 score.

We further investigate error cases of the fine-tuned depccg in the subset dataset (Table 6). The tendency of error types is in accordance with other domains, with frequent errors in PP-attachment and predicate-argument structure, and seemingly more cases of attachment errors of adverbial phrases (11 cases), which occur in lengthy sentences such as in Table 5b. Other types of error are failures to recognize that the sentence is in imperative form (2 cases), and ones in handling informal functional words such as *cause* (Table 5a). We conclude that the performance on this domain is as high as it is usable in application. Since the remaining errors are general ones, they will be solved by improving general parsing techniques.

5.5 Math Problems

Setup Finally, we conduct another experiment on parsing math problems. Following previous work of constituency parsing on math problem (Joshi et al., 2018), we use the same train/test sets by Seo et al. (2015), consisting of 63/62 sentences respectively, and see if a CCG parser can be adapted with the small training samples. Again, the first author annotated both train/test sets, dependency trees on the train set, and CCG trees on the test set, respectively. In the annotation, we follow the manuals of the English CCGbank and the UD. We regard as an important future work extending the annotation to include fine-grained feature values in categories, e.g., marking a distinction between integers and real numbers (Matsuzaki et al., 2017). Figure 3 shows an example

[14] https://github.com/jkkummerfeld/berkeley-ccg2pst

[15] https://nlp.cs.nyu.edu/evalb/

[16] Following Honnibal and Johnson (2014), sentences in this data are fully lower-cased and contain no punctuation.

CCG tree from this domain, successfully parsed by fine-tuned depccg.

Results Table 8 shows the F1 scores of depccg in the respective settings. Remarkably, we observe huge additive performance improvement. While, in terms of labeled F1, ELMo contributes about 4 points on top of the plain depccg, adding the new training set (converted from dependency trees) improves more than 10 points.[17] Examining the resulting trees, we observe that the huge gain is primarily involved with expressions unique to math. Figure 3 is one of such cases, which the plain depccg falsely analyzes as one huge NP phrase. However, after fine-tuning, it successfully produces the correct "If S_1 and S_2, S_3" structure, recognizing that the equal sign is a predicate.

6 Conclusion

In this work, we have proposed a domain adaptation method for CCG parsing, based on the automatic generation of new CCG treebanks from dependency resources. We have conducted experiments to verify the effectiveness of the proposed method on diverse domains: on top of existing benchmarks on biomedical texts and question sentences, we newly conduct parsing experiments on speech conversation and math problems. Remarkably, when applied to our domain adaptation method, the improvements in the latter two domains are significant, with the achievement of more than 5 points in the unlabeled metric.

Acknowledgments

We thank the three anonymous reviewers for their insightful comments. This work was in part supported by JSPS KAKENHI Grant Number JP18J12945, and also by JST AIP-PRISM Grant Number JPMJCR18Y1, Japan.

References

Lasha Abzianidze. 2017. LangPro: Natural Language Theorem Prover. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 115–120. Association for Computational Linguistics.

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2013. Using CCG categories to improve Hindi dependency parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 604–609. Association for Computational Linguistics.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.

Johan Bos, Valerio Basile, Kilian Evang, Noortje J. Venhuizen, and Johannes Bjerva. 2017. The Groningen Meaning Bank. In *Handbook of Linguistic Annotation*, pages 463–496. Springer Netherlands.

Johan Bos, Bosco Cristina, and Mazzei Alessandro. 2009. Converting a Dependency Treebank to a Categorial Grammar Treebank for Italian. In *In Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories*, pages 27–38.

Stephen Clark and James R. Curran. 2007. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building Deep Dependency Structures with a Wide-coverage CCG Parser. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 327–334. Association for Computational Linguistics.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ICLR*.

Timothy Dozat and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. *ICLR*.

John J. Godfrey, Edward C. Holliman, and Jane McDaniel. 1992. SWITCHBOARD: Telephone Speech Corpus for Research and Development. In *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 517–520. IEEE Computer Society.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Matthew Honnibal and Mark Johnson. 2014. Joint Incremental Disfluency Detection and Dependency Parsing. *Transactions of the Association for Computational Linguistics*, 2:131–142.

Xinzhou Jiang, Zhenghua Li, Bo Zhang, Min Zhang, Sheng Li, and Luo Si. 2018. Supervised Treebank Conversion: Data and Approaches. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 2706–2716. Association for Computational Linguistics.

[17] Note that, while in the experiment on this dataset in the previous constituency parsing work (Joshi et al., 2018), they evaluate on partially annotated (unlabeled) trees, we perform the "full" CCG parsing evaluation, employing the standard evaluation metrics. Given that, the improvement is even more significant.

Vidur Joshi, Matthew Peters, and Mark Hopkins. 2018. Extending a Parser to Distant Domains Using a Few Dozen Partially Annotated Examples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1190–1199. Association for Computational Linguistics.

John Judge, Aoife Cahill, and Josef van Genabith. 2006. QuestionBank: Creating a Corpus of Parse-Annotated Questions. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 497–504. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003. A* Parsing: Fast Exact Viterbi Parse Selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 40–47. Association for Computational Linguistics.

Jonathan K. Kummerfeld, Dan Klein, and James R. Curran. 2012. Robust Conversion of CCG Derivations to Phrase Structure Trees. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 105–109. Association for Computational Linguistics.

Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2016. Global Neural CCG Parsing with Optimality Guarantees. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2366–2376. Association for Computational Linguistics.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-Rank Tensors for Scoring Dependency Structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1381–1391. 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. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014a. A* CCG Parsing with a Supertag-factored Model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 990–1000. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014b. Improved CCG Parsing with Semi-supervised Supertagging. *Transactions of the Association for Computational Linguistics*, 2:327–338.

Zhenghua Li, Min Zhang, Yue Zhang, Zhanyi Liu, Wenliang Chen, Hua Wu, and Haifeng Wang. 2016. Active Learning for Dependency Parsing with Partial Annotation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 344–354. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):314–330.

M. Marneffe, B. Maccartney, and C. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, pages 449–454. European Language Resources Association.

Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. 2017. On-demand Injection of Lexical Knowledge for Recognising Textual Entailment. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 710–720. Association for Computational Linguistics.

Takuya Matsuzaki, Takumi Ito, Hidenao Iwane, Hirokazu Anai, and Noriko H. Arai. 2017. Semantic Parsing of Pre-university Math Problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 2131–2141. Association for Computational Linguistics.

Seyed Abolghasem Mirroshandel and Alexis Nasr. 2011. Active Learning for Dependency Parsing Using Partially Annotated Sentences. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 140–149. Association for Computational Linguistics.

Jeff Mitchell and Mark Steedman. 2015. Parser Adaptation to the Biomedical Domain without Re-Training. In *Proceedings of the Sixth International Workshop on Health Text Mining and Information Analysis*, pages 79–89. Association for Computational Linguistics.

Makoto Miwa and Mohit Bansal. 2016. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1105–1116. Association for Computational Linguistics.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation*, pages 1659–1666. European Language Resources Association.

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,*, pages 2227–2237. Association for Computational Linguistics.

Laura Rimell and Stephen Clark. 2008. Adapting a Lexicalized-Grammar Parser to Contrasting Domains. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 475–484. Association for Computational Linguistics.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1476. Association for Computational Linguistics.

Miloš Stanojević and Mark Steedman. 2019. CCG Parsing Algorithm with Incremental Tree Rotation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 228–239. Association for Computational Linguistics.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1556–1566. Association for Computational Linguistics.

Yuka Tateisi, Akane Yakushiji, Tomoko Ohta, and Jun'ichi Tsujii. 2005. Syntax Annotation for the GENIA Corpus. In *Companion Volume to the Proceedings of Conference including Posters/Demos and tutorial abstracts*, pages 220–225. Association for Computational Linguistics.

Masashi Yoshikawa, Hiroshi Noji, and Yuji Matsumoto. 2017. A* CCG Parsing with a Supertag and Dependency Factored Model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 277–287. Association for Computational Linguistics.