Unsupervised Formal Grammar Induction with Confidence

Jacob Collard
Cornell University
jacob@thorsonlinguistics.com
To Appear in the Proceedings of SCiL 2020

Abstract

I present a novel algorithm for minimally supervised formal grammar induction using a linguistically-motivated grammar formalism. This algorithm, called the Missing Link algorithm (ML), is built off of classic chart parsing methods, but makes use of a probabilistic confidence measure to keep track of potentially ambiguous lexical items. Because ML uses a structured grammar formalism, each step of the algorithm can be easily understood by linguists, making it ideal for studying the learnability of different linguistic phenomena. The algorithm requires minimal annotation in its training data, but is capable of learning nuanced data from relatively small training sets and can be applied to a variety of grammar formalisms. Though evaluating an unsupervised syntactic model is difficult, I present an evaluation using the Corpus of Linguistic Acceptability and show state-of-the-art performance.

1 Introduction

Most research on learning algorithms for natural language syntax has focused on supervised parsing, in which the parser learns from sentences in the target language paired with a corresponding, hand-constructed parse tree. Major natural language corpora, such as the Penn Treebank (Marcus et al., 1994) and the Universal Dependencies framework (Nivre et al., 2016) exemplify this tendency. This has allowed for highly performant models for dependency parsing such as ClearNLP (Choi and McCallum, 2013), CoreNLP (Manning et al., 2014), Mate (Bohnet, 2010), and Turbo (Martins et al., 2013), all of which have achieved an accuracy of over 89% on standard evaluation tasks (Choi et al., 2015).

Unsupervised learning for natural language processing is a much more difficult task, as the algorithm must explore the entire search space with minimal confirmation of its hypotheses. Nevertheless, a number of algorithms have attempted to solve the problem of unsupervised parsing. Most of these rely on gold standard part of speech tags (Headden III et al., 2009; Spitkovsky et al., 2010), though there are some exceptions, such as Spitkovsky (2011). Almost all unsupervised algorithms for natural language syntactic processing are based on dependency parsing; most of the published literature on other grammar formalisms, such as tree-adjoining grammar (TAG) and combinatory categorial grammars (CCG) is either supervised or hand-engineered. Again, there are some exceptions, such as (Bisk et al., 2015), which learns CCGs using a small amount of initial part-of-speech data. Edelman et al. (2003) also present a model of unsupervised learning which blends properties of construction grammars with tree-adjoining grammars; however, their model has not, as yet, been evaluated empirically.

Other models are only indirectly supervised; the syntax of the target language is learned without any syntactic annotations, but annotations may be present representing other facts about the sentence, such as its logical form. Notable examples of this include work by Kwiatkowski et al. (2010; 2011) and Artzi and Zettlemoyer (2013).

Another recent innovation in unsupervised learning is the introduction of pre-trained language models for deep learning algorithms, such as BERT (Devlin et al., 2018) and its relatives. These algorithms can be pre-trained on raw text in order to produce a language model which can then be used to bootstrap learning for a wide variety of additional tasks. Though supervision may be required by these downstream tasks, the unsupervised component has been shown to greatly improve learning. The representations that these models produce are somewhat opaque; though they have been shown to represent syntactic infor-

mation for some tasks (Goldberg, 2019), an exact description of what the model is representing is difficult to produce.

Though most of the above systems do not rely on a strict notion of grammar formalism, in this paper, I will argue that a well-defined grammar formalism can produce strong results when used as the basis for an unsupervised learning algorithm. Dependence on a grammar formalism has a number of benefits. First, it means that each step of the algorithm can be (relatively) easily understood by humans. Each processing step either produces a novel derivation for a sentence or reinforces an old one, and each derivation conforms to the rules of the given formalism. Thus, as long as the rules of the formalism are understood, the meaning behind each processing step can also be understood. Second, using a grammar formalism ensures that certain facts about the resulting grammar will always hold. For example, using CCG or TAG will guarantee that the resulting grammar is in the class of mildly context-sensitive languages. Third, using a grammar formalism means that the properties of the grammar formalism can be studied as well. Though formalisms such as CCG and TAG are weakly equivalent (Joshi et al., 1990), there may be differences between the two formalisms with respect to learning. Similarly, different variants of a particular formalism can be studied as well. For example, different combinators can be added or removed from CCG to produce different learning results. By using the grammar formalism as a core parameter in learning, the formalism becomes an independent variable that can be explored.

In this paper I introduce an algorithm, called the Missing Link algorithm (ML), which has several interesting properties which, I argue, are beneficial to the study of linguistics, grammar formalisms, and natural language processing. These properties include:

- **Minimal supervision.** The Missing Link algorithm learns from raw, tokenized text. The only annotation required is assurance of the sentencial category, which is trivial for most training sets and grammar formalisms.

- **Formalism Dependence.** The grammar formalism is the core motivator for learning and parsing in Missing Link. This means that the formalism can easily be replaced with another and that the formalism can be studied as a parameter.

- **Interpretability.** Due partly to formalism dependence, the Missing Link algorithm is highly interpretable. Each step of the algorithm can be viewed as a derivation using the input formalism.

- **Performance.** The Missing Link algorithm performs well on an evaluation using linguistic acceptability judgments. The results of the evaluation are competitive with supervised algorithms such as BERT for the specific task used. Missing Link supplements the input formalism with a model of confidence for lexical entries that allows it to robustly handle potential ambiguity.

1.1 Related Work

The Missing Link algorithm builds off of relatively simple models for grammar induction and parsing. Parsing is done via a simple bottom-up chart-based method (Younger, 1967; Kasami, 1965). Learning is done in a top-down fashion using the same chart, with some extensions described in Section 2.2.

The Missing Link algorithm is closely related to the Unification-Based Learning (UBL) algorithm described in Kwiatkowski et al. (2010). UBL is an algorithm for semantic parsing, but the decomposition operations used in Missing Link are essentially the same as the higher-order unification used in UBL, albeit applied directly to syntactic categories instead of logical forms. Unlike UBL, Missing Link ensures that every stop of processing is interpretable; the probabilistic grammar used in UBL can potentially obscure why individual parses are excluded.

2 The Missing Link Algorithm

There are two main stages to the Missing Link algorithm: parsing and learning, which are performed in order for every sentence in the training set. As the algorithm processes more sentences, it updates a lexicon, mapping words to their syntactic categories and a probability representing how confident the algorithm is that the given category is valid in the target grammar.

2.1 Inputs

The core inputs to the Missing Link algorithm are:

- A grammar formalism, which defines a (possibly infinite) set of grammatical units E and two functions: COMPOSE : $E \times E \rightarrow E*$ and DECOMPOSE : $E \times E \times E \rightarrow (E \times E)*$. E always contains a special null element 0 indicating that the grammatical category is not known.

- A collection of training examples. Each training example consists of a tokenized sentence and an annotation describing the possible grammatical categories of the sentence.

The two functions of the grammar formalism determine the behavior of the parser and learner. The COMPOSE function returns a collection of elements in E that can be produced by combining the two input elements. This typically represents the basic structure-building operation of the given formalism. In Minimalism, for example, it corresponds to MERGE; in CCG, it corresponds to the various combinators; in TAG to substitution and adjunction, etc.

The DECOMPOSE function is essentially the inverse of COMPOSE. It returns the set of pairs of elements that can be composed to produce a given input. Though it takes three elements, only the first, representing the root, is necessary. The second arguments are supplied to force one or both of the elements in the results to take a particular value. This will become important to avoid exploring the entirety of the search space; when a value is already known, the DECOMPOSE function will maintain that value whenever possible.

Note that the input of Missing Link places some restrictions on the types of formalisms that can be used. In particular, the formalism does not carry any language-specific information outside of the lexicon – the formalism must be strongly lexicalized. Many major formalisms, including CCG and TAG, adhere to this rule, though some formalisms, such as standard context-free grammars, cannot be represented by Missing Link. In addition, the results of DECOMPOSE and COMPOSE must be finite sets. Though this seems problematic for formalisms like CCG, where there are infinite ways to compose two arbitrary elements to produce a third, this can usually be avoided by schematization. That is, instead of returning every possible result, the results can be summarized using variables.

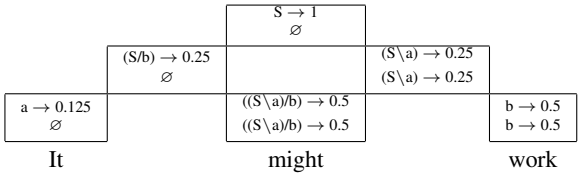


Figure 1: A chart showing the parse of the sentence *It might work*. Parse values are given at the bottom of each node, while learn values are given at the top.

2.2 The Chart

The Missing Link algorithm is built around a CKY-style chart. The chart consists of cells representing potential parses for each substring of the sentence. In Missing Link, each cell stores two separate analyses: one for the potential bottom-up parses of the sentence, and one for the potential top-down decompositions of the sentence, based on the category assigned to the sentence. These will be referred to as the "parse value" and the "learn value" for each cell.

Both the parse value and the learn value are represented using the same data structure, which is also used in the lexicon. This data structure maps potential categories (elements of E according to the target formalism) to probabilities, which represent the algorithm's confidence that the given category is valid for the corresponding substring. Note that the sum of the probabilities for the different categories is not necessarily 1: in the case where the algorithm is certain that a substring is ambiguous, the probabilities will sum to at least 2. Missing Link does not assign probabilities based on frequency relative to other substrings, sentences, or categories and makes no distinction between alternative parses other than their probability of being grammatical.

An example chart is shown in Figure 1.

2.3 Parsing

For each sentence in the training data, the Missing Link algorithm begins by looking up each word in the lexicon. The results of the lookup are assigned to the parse value for the bottom cells of the chart, which represent the length-1 substrings. The algorithm then attempts to parse as much of the sentence as possible. Initially, it will not be possible to parse any sentences, as no words have been introduced to the lexicon. However, as the algorithm sees more sentences in the training set, the lexicon will expand and the parses will become more complete.

The parsing step is fairly typical for a probabilistic CKY parser, proceeding in a bottom-up direction by calling COMPOSE for each pair of adjacent substrings. The only major difference does not come from the parsing strategy *per se*, but from necessary constraints on the formalisms compatible with Missing Link.

The confidence values for subsequent cells in the chart are determined under the assumption that all assignments are independent. Thus, in most cases, $P(\text{COMPOSE}(A, B)) = P(A)P(B)$. If the same category is found multiple times, these values are also treated as independent; thus, $P(A) = P(A_1) + P(A_2) - P(A_1)P(A_2)$, where A_1 and A_2 are separate instances of the category A.

Once as much of the chart has been filled by the parser as possible, the parse stage ends for that sentence. The parse values of the cells are retained when the chart is passed to the learning stage; incomplete parses are used to inform the learning stage. Even if the parse was successful, the learning stage still occurs, in order to update the system's confidence in the values used to produce the successful parse.

2.4 Learning

The learning stage is similar to the parsing stage, although it is somewhat more involved as it is able to take advantage of the results of the parse to minimize its search space.

First, the learn value of the root of the chart is initialized with the annotated category for the sentence. Then, the algorithm proceeds in a top-down manner calling DECOMPOSE on each cell and two corresponding substrings. For the most part, this proceeds in the same manner as the parsing stage, except in a top-down direction. There are, however, a few differences.

The base probability for a learned category is based on the probability of the root and the probability of any known sub-constituents.

When assigning probabilities in the learning stage, the learner must contend with the fact that there are multiple possibilities and no guarantee that they are all valid. The learner must also contend with the fact that there are multiple possible tree structures. Since chart parsing deals only with binary-branching trees, it is possible to calculate the number of possible trees for a sentence of a given length. The probability must then also be modulated by the $n-1$st Catalan number, where n is the length of the substring corresponding to the current cell.

Thus, the total probability for a given result is equal to $\frac{p}{C_{n-1}l}$ where p is the base probability, C_n is the nth Catalan number, n is the length of the substring, and l is the number of results produced by DECOMPOSE.

The results are also dependent on the parse values of the corresponding sub-constituents. If both of the sub-constituents have known parse values, then learning does not necessarily need to occur. If these sub-constituents can be composed to produce at least one value in the current cell's learn value, then those sub-constituents will be added to the learn values of their cells, with their original probabilities. In other words, if the values are known and can produce the target value, then no additional learning needs to occur. If the target value cannot be produced, then learning occurs as normal, as if neither value were known.

A similar situation occurs when only one of the sub-constituents is known. In this case, DECOMPOSE is applied as normal, with the restriction that the known value remains constant. If DECOMPOSE is successful, then the probability remains constant as well. On the other hand, if DECOMPOSE cannot learn any values, then the decomposition is attempted again, as though neither value were known.

2.5 Lexical Update

Once both parsing and learning have occurred for a given sentence, the algorithm updates the lexicon based on the learned values for the length-1 substring cells in the chart. Since values in the lexicon are represented the same way as cells in the chart, this is a fairly straightforward process. If the category for a word in the current sentence is the same as a category already in the lexicon, the probability of the word is updated according to the assumption that the two probabilities are independent, as described above.

3 Implementing Combinatorial Categorial Grammar

For the purposes of this paper, Combinatory Categorial Grammar (CCG) will be used as an example formalism with Missing Link. CCG is an efficiently parseable grammar formalism in which all language-specific rules are stored in the lexicon (Steedman and Baldridge, 2002).

To implement a formalism for Missing Link, it is only necessary to define the set E, and the functions COMPOSE and DECOMPOSE. In CCG, the set E will be the set of categories, which is defined as follows:

- Given a set A of atoms, if $a \in A$, then a is a category.
- If a and b are categories, then (a\b) and (a/b) are categories. These are referred to as complex or functional categories.

For the purposes of Missing Link, it is also necessary to define one additional type of category: the variable. In this paper, variables are represented using lowercase letters while atoms are represented using capital letters.

In this paper, I start with the variant of CCG defined in Eisner (1996). In this variant, the COMPOSE operator can be defined fairly straightforwardly using two combinators:

$$\frac{\text{(X/Y)} \qquad X|_n \cdots |_2 Z_2 |_1 Z_1}{X|_n Z_n \cdots |_2 Z_2 |_1 Z_1} > Bn$$

$$\frac{X|_n \cdots |_2 Z_2 |_1 Z_1 \qquad \text{(X\Y)}}{X|_n Z_n \cdots |_2 Z_2 |_1 Z_1} < Bn$$

These combinators, generalized forward composition and generalized backward composition, respectively, result in a TAG-equivalent formalism. I use Steedman's result-first notation for CCG categories and assume left-associativity unless disambiguated by parentheses.

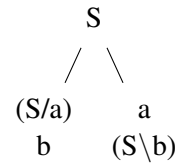
The COMPOSE function for CCG can be defined as simply taking two categories and returning the results of either of these combinators. The exception is when one of the categories contains a variable; in this case, if the variable can be assigned to a value in the other category to produce a valid combination, then it will be. For more details on variables, see Section 3.2.

The DECOMPOSE, as the inverse of COMPOSE, can be derived from the same combinators. In effect, the algorithm attempts to match as much of the pattern as is available in order to construct the rest of the proof. Any values which cannot be determined concretely are replaced with variables. For example, given the root X, this matches both combinators where $n = 0$, allowing the function to select ((X/a), a) and (b, (X\b)) as potential results.

3.1 Modalities

As written above, this formalism is performs poorly with Missing Link. Though COMPOSE and DECOMPOSE always produce finite results, the generalized combinators prove problematic for learning. This is because crossed composition in CCG can result in permutation, which makes it impossible for Missing Link to distinguish between certain alternatives involving functional categories.

For example, consider the first learning instance involving a two-word sentence with no words known. This will produce a derivation such as the following, where multiple categories on a node correspond to alternative hypotheses proposed by Missing Link.



If the algorithm then attempts to parse this same sentence (or any similar one), it will run into a problem. Since the algorithm cannot tell which hypothesis for the left node was originally paired with which hypothesis for the right node, it is forced to try all possibilities during composition. This will result in the algorithm attempting to compose (S/a) and (S\b). If $a = S$, then this can compose according to the generalized composition rules above, resulting in (S\b). However, the algorithm has made a crucial mistake in assigning S to the input type b. In fact, because these situations crop up whenever an atom or variable is decomposed, this sort of assignment is exceedingly common. In addition, because S is usually the only concrete atomic category (other atoms must be represented by clusters of variables, since Missing Link is unsupervised!), eventually *all* variables tend to converge to S. This results in an incomprehensible grammar in which the only atom is S.

One possible solution would be to prevent hypotheses from composing with competing alternatives (i.e., alternatives generated at the same time). However, this solution would not be sufficient, as the same situation could still occur with other words – the pattern that created the first set of hypotheses will apply to other words as well. Thus, convergence to S will still occur in these situations.

The solution that I used was to take advantage of

the slash modalities used in many CCG variants, such as Baldridge (2002). These variants place modalities on the slash categories that restrict their application to certain combinators. I use four basic modalities, taken from Steedman and Baldridge (2002).

- Star ($*$): Categories with this slash can only apply in simple applicative contexts (function composition of all types is forbidden).
- Diamond ($\diamond$): Categories with this slash can compose only with categories of the same slash direction.
- Cross ($\times$): Categories with this slash can compose only with categories of the opposite slash direction.
- Dot ($\cdot$): Categories with this slash can compose with any other category.

Using these modalities restricts the cases where certain compositions may occur. During learning, the most restrictive category that applies to the given decomposition is always used. For example, the decomposition of S results in ((S/$*$a), a) and (b, (S\$*$b)). Due to the restrictions on the modalities, it is no longer possible to compose the results using crossed composition: (S/$*$a) and (S\$*$b) are incompatible! The problem of convergence to S no longer exists, and the computational properties of the formalism are maintained: crossed composition can still occur as a last resort, in cases where it is clear that it can be derived.

As an aside, it is no accident that crossed composition was the cause of this issue. The permutations caused by crossed composition also make it necessary in mildly context-sensitive CCGs to account for data in languages such as Swiss German and Dutch. It is interesting, though not surprising, that this comes with its own difficulties in learning.

3.2 Variables and State

As described in previous sections, this implementation of CCG for Missing Link uses variables to represent categories whose exact value cannot be determined. Though variables are necessary, they also introduce additional complexity into the algorithm. There are a few special notes that relate to the treatment of variables in CCG.

The values of variables are stored in a global state, which maps variable IDs to their values (if a value is known). When a variable is evaluated in parsing or learning, it is first resolved according to the state. In most cases, a variable cannot be resolved completely (the final representation will still contain one or more variables); this is expected, as the algorithm is not able to induce new atomic categories, it must instead make use of variables that are designated to represent new categories.

The value of a variable may be a complex category, an atomic category, or another variable (the latter case being used primarily to set two variables equal to one another). If a variable is set equal to a complex category, it may be that the complex category itself contains variables. This creates the possibility of reference cycles, which would produce undefined values. To avoid this, every time an assignment is made, the algorithm performs an occurs check to ensure that the assignment will not produce any reference cycles: the new value is checked for any instances, direct or indirect, of the variable. If there are any, the assignment cannot be completed and the algorithm must try another alternative.

Once a variable is assigned a value, it is permanent. However, the algorithm is still free to introduce a new variable by re-decomposing categories in future training samples, according to the rules given in Section 2. Furthermore, if a variable assignment fails within a combinator (due to an occurs check), the state is rolled back to the way it was before the combinator began processing. This prevents known inconsistencies from filling the state; though the state may still contain inconsistencies, all values in the state are part of a potential analysis. Any remaining inconsistencies typically do not achieve high probability during learning, as they cannot be used to successfully parse many sentences.

4 Evaluation

Evaluating an unsupervised learning algorithm is a difficult prospect. Though many unsupervised syntactic learning algorithms are evaluated by comparing the resulting dependency structures to a gold standard, typically by ensuring that each predicted dependency is directed in the same way between the same two lexical units as the gold standard dependency. However, comparison by dependency structures is not always a good choice for unsupervised learning algorithms. In particu-

lar, because one of the goals of Missing Link is to provide a framework for analyzing grammar formalisms in an otherwise theory-independent manner, it is undesirable to make use of any theory-dependent analysis. Though dependencies may be largely independent of theory, they still make conventional decisions. For example, the Universal Dependencies project does not usually allow functional categories to be heads, while alternatives, such as Stanford Dependencies, do allow functional heads. If the learning algorithm is free to choose from the alternatives on its own, then it cannot be accurately evaluated against such a standard.

To evaluate Missing Link, I therefore use a secondary task. Similar to BERT (Devlin et al., 2018), I use Missing Link as an unsupervised pre-training algorithm for a downstream task. In this case, I use linguistic acceptability (grammaticality) judgments as the downstream task. This is an ideal task for Missing Link, since Missing Link's confidence values essentially capture the notion of probability that a sentence (or substring) is grammatical. I use the Corpus of Linguistic Acceptability (Warstadt et al., 2018) to provide the gold standard and training data. The Corpus of Linguistic Acceptability (CoLA) provides a basic classification task in which sentences are annotated with boolean grammaticality judgments – that is, each sentence is either considered grammatical or not. Missing Link will provide the pre-trained linguistic model, and a simple logistic regression will use Missing Link confidence values to classify the test data.

4.1 Pre-Training

Before training the model, Missing Link is used to pre-train a linguistic model of the target language, in this case English. In order to keep the conditions between the training set and the testing set as close as possible, it is necessary to pre-train Missing Link on a different dataset than the annotated linguistic acceptability data. In addition, the corpus of linguistic acceptability is relatively small. To this end, I pre-train Missing Link using the much larger Billion Words corpus (Chelba et al., 2013).

For pre-training, I sorted the sentences of the Billion Words corpus (BWB) in ascending order by length. Missing Link is able to assign higher confidence to words in shorter sentences. This allows it to have a relatively small set of hypotheses for many words that occur in shorter sentences, which it can then use to better learn nearby words in longer sentences. Sentences of length less than 3 were excluded, since most short sentences in BWB are simple noun phrases or noisy punctuation, which can confuse Missing Link. Sentences of length greater than 10 were excluded as well, since they tend to contribute little to Missing Link's confidence.

For practical reasons, I also restricted the number of categories learned for each cell in the chart and lexical entry to 50, to improve efficiency while still allowing for multiple simultaneous hypotheses.

4.2 Training and Testing

Once the linguistic model is pre-trained, then it can be used to train a logistic regression. To train the logistic regression, Missing Link processes each sentence in the CoLA training set, producing a confidence value for each potential sentential category. If no parse is produced, or if S is not in the results, then Missing Link is allowed to learn from the sentence. By learning from sentences where no valid parse was produced, Missing Link becomes robust to sentences where some words were not in the original pre-training set. After learning, Missing Link attempts to parse the sentence again.

The confidence of the category S is used as one independent variable for training the logistic regression. The other independent variables are the length of the sentence, whether the category S was found (including after re-training), and whether retraining was required. Longer sentences are inherently associated with lower probabilities due to the independence assumptions used in composition. Sentences where the category S was never found are also distinguished from sentences where the probability of S was negligible; although both are likely to indicate an ungrammatical sentence, for long sentences, the distinction may be necessary. Lastly, whether retraining was necessary is a plausible predictor as well, since retraining indicates that either some words are out of vocabulary or the sentence could not be parsed with the previously learned categories.

Once the logistic regression has been fit to the training data, the same process is applied to the testing data in order to predict whether each sen-

Model	MCC
MT-DNN	68.4
RoBERTa	67.8
XLNet-Large	67.8
GLUE Human Baseline	66.4
Missing Link	**63.0**
XLM	62.9
BERT-24	60.5

Table 1: CoLA Benchmark[1]

tence is grammatical or not.

5 Results and Conclusion

The Corpus of Linguistic Acceptability is evaluated using Matthews Correlation Coefficients, since there are far more grammatical sentences in the data than ungrammatical ones. A number of other systems have been tested against CoLA and can be used as benchmarks for Missing Link, since the same training-testing split is used by all systems.

The results of the evaluation of Missing Link as well as some of the top performing competitors are given in Table 1. With an MCC of 63.0, Missing Link does not advance the state-of-the-art compared to deep learning models, but it does perform competitively. Given that Missing Link uses only a basic logistic regression on top of the pretrained model, this presents evidence that Missing Link is producing a reasonable grammar for the data.

Given that Missing Link produces a reasonable grammar, it can then be used for further study in the fields of grammar formalisms and theoretical linguistics. Different grammar formalisms can be compared using the same core algorithm, allowing for any variation in performance to be attributed to properties of the grammar formalism. The algorithm can also be used to explore specific linguistic phenomena from a learning perspective. Given two alternatives to a linguistic phenomenon, it is possible to use Missing Link as one potential way of distinguishing between the two. This presents a new paradigm in linguistic research as a means of exploring generative linguistics through a formal, but nuanced, model of learning and learnability. Missing Link is not necessarily the only algorithm that supports this paradigm, but presents evidence that such a paradigm is feasible for linguistic theory.

[1]These baselines are taken from the GLUE leaderboard at the time of writing (Wang et al., 2019).

References

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Jason Baldridge. 2002. *Lexically specified control in combinatory categorial grammar*. Ph.d. dissertation, University of Edinburgh.

Yonatan Bisk, Christos Christodoulopoulos, and Julia Hockenmaier. 2015. Labeled grammar induction with minimal supervision. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 870–876, Beijing, China. Association for Computational Linguistics.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of COLING*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorston Brants, and Phillip Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st annual meeting of the Association for Computational Linguistics*, pages 1052–1062.

Jinho D. Choi, Joel Tetreault, and Amanda Stent. 2015. It depends: Dependency parser comparison using a web-based evaluation tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 387–396.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Shimon Edelman, Zach Solan, David Horn, and Eytan Ruppin. 2003. Rich syntax from a raw corpus: Unsupervised does it. In *Syntax, Semantics and Statstics Workshop at NIPS '03*, Whistler, British Columbia, Canada.

Jason Eisner. 1996. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86. Santa Cruz.

Yoav Goldberg. 2019. Assessing BERT's syntactic abilities. Unpublished manuscript.

William P. Headden III, Mark Johnson, and David McClosky. 2009. Improving unsupervised dependency parsing with richer contexts and smoothing.

In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 101–109, Stroudsburg, PA, USA. Association for Computational Linguistics.

Aravind K. Joshi, K. Vijay Shankar, and David Weir. 1990. The convergence of mildly context-sensitive grammar formalisms. Technical Report MS-CIS-90-01, Department of Computer and Information Science, University of Pennsylvania.

Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, AFCRL.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1512–1523, Stroudsburg, PA, USA. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karne Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the workshops on Human Language Technology*, pages 114–119. Association for Computational Linguistics.

André F.T. Martins, Miguel B. Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the ACL*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zema. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of LREC*, pages 1659–1666.

Valentin I. Spitkovsky, Hiyan Alshawi, Angel X. Chang, and Daniel Jurafsky. 2011. Unsupervised dependency parsing without gold part-of-speech tags. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1281–1290, Stroudsburg, PA, USA. Association for Computational Linguistics.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2010. From baby steps to leapfrog: How "Less is More" in unsupervised dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 751–759, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mark Steedman and Jason Baldridge. 2002. Combinatory categorial grammar. In Robert D. Borsley and Kersti Börjars, editors, *Non-Transformational Syntax*, pages 181–224. Blackwell.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2018. Neural network acceptability judgments. *CoRR*, abs/1805.12471.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208.