Tetra-Tagging: Word-Synchronous Parsing with Linear-Time Inference

Nikita Kitaev and **Dan Klein**
Computer Science Division
University of California, Berkeley
{kitaev, klein}@cs.berkeley.edu

Abstract

We present a constituency parsing algorithm that maps from word-aligned contextualized feature vectors to parse trees. Our algorithm proceeds strictly left-to-right, processing one word at a time by assigning it a label from a small vocabulary. We show that, with mild assumptions, our inference procedure requires constant computation time per word. Our method gets 95.4 F1 on the WSJ test set.

1 Introduction

The initial stages of modern NLP models typically deal with token-aligned vector representations. These may begin as word vectors, and can later be modified to incorporate contextual information using neural architectures such as RNNs or the Transformer. Architectures that produce these representations are general-purpose, can be shared across tasks, and can be effectively pre-trained on large amounts of data.

Modern parsers make use of such representations, but require leaving the word-synchronous domain when producing their output. For example, our previous parser (Kitaev and Klein, 2018) constructs and operates over representations for each *span* in the sentence. In this paper, we present an approach to parsing that fully operates in the word-synchronous paradigm. Our method has the following properties:

- It is **word-synchronous**. The entire trainable portion of the parser consists of producing word-aligned feature vectors and predicting labels using these features. As a result, the method can immediately leverage any advances in building general-purpose word representations.

- It uses **constant bit rate**: each position in the sentence is assigned a label from a fixed vocabulary. The set of labels is fully determined by the grammar of the language and does not depend on the specific inputs presented to the parser.

- **Inference time per word is constant**, subject to mild assumptions based on the observation that certain syntactic configurations are difficult for humans to understand and are unattested in treebank data.

2 Related Work

Chart parsing Chart parsers fundamentally operate over *span-aligned* rather than *word-aligned* representations. This is true for both classical methods (Klein and Manning, 2003) and more recent neural approaches (Durrett and Klein, 2015; Stern et al., 2017). The size of a chart is quadratic in the length of the sentence, and the unoptimized CKY algorithm has cubic running time. Because the bit rate and inference time per word are not constant, additional steps must be taken to scale these systems to the paragraph or document level.

Label-based parsing A variety of approaches have been proposed to mostly or entirely reduce parsing to a sequence labeling task. One family of these approaches is *supertagging* (Bangalore and Joshi, 1999), which is particularly common for CCG parsing. CCG imposes constraints on which supertags may form a valid derivation, necessitating complex search procedures for finding a high-scoring sequence of supertags that is self-consistent. An example of how such a search procedure can be implemented is the system of Lee et al. (2016), which uses A* search. The required inference time per word is not constant with this method, and in fact the worst-case running time is exponential in the sentence length. Gómez-Rodríguez and Vilares (2018) proposed a different approach that fully reduces parsing to sequence labeling, but the label vocabulary is unbounded: it expands with tree depth and related properties of the input, rather than being fixed for any given language. There have been attempts to address this by adding redundant labels, where each word has multiple correct labels (Vilares et al., 2019), but that only increases the label vocabulary rather than restricting it to a finite set. Our approach, on the other hand, uses just 4 labels in its simplest formulation (hence the name *tetra-tagging*).

Shift-reduce transition systems A number of parsers proposed in the literature fall into the broad category of *shift-reduce* parsers (Henderson, 2003; Sagae and Lavie, 2005; Zhang and Clark, 2009; Zhu et al., 2013). These systems rely on generating sequences of actions, but the actions need not be evenly distributed throughout the sentence. For example, the construction of a deep right-branching tree might involve a series of *shift* actions (one per word in the sentence), followed by equally many consecutive *reduce* actions that all cluster at the end of the derivation. Due to the uneven alignment between actions and locations in a sentence, neural network architectures in recent shift-reduce systems (Vinyals et al., 2015; Dyer et al., 2016; Liu and Zhang, 2017) broadly follow an encoder-decoder approach rather than directly assigning labels to positions in the input. Our proposed parser is also transition-based, but there are guaranteed to be exactly two decisions to make after shifting one word and before shifting the next. As a result, the amount of computation required per word is uniform as the algorithm proceeds left-to-right through the sentence.

Left-corner parsing Our parsing algorithm is inspired by and shares several key properties with left-corner parsing; see Section 3.5 for a discussion of related work in this area.

3 Method

To introduce our method, we first restrict ourselves to only consider unlabeled full binary trees (no labels, no unary chains, and no nodes with more than two children). We defer the discussion of labeling and non-binary structure to Section 3.6.

3.1 Preliminaries

Before we present our word-synchronous parse tree representation, let's first answer the question: *what is the minimal bit rate (per word) required to represent a parse tree?* Absent any assumptions about the linguistic infeasibility of certain syntactic configurations, a parser would need to be capable of producing all possible trees over a given set of words. The number of full binary trees over $n + 1$ words is the n^{th} Catalan number, C_n. Asymptotically, the Catalan numbers scale as:

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

From this we can conclude that a parser that selects between 4 possible options per word is *asymptotically optimal*, in the sense that selecting from a smaller inventory of options is insufficient to encode all possible trees. We exhibit such a method in the next subsection.

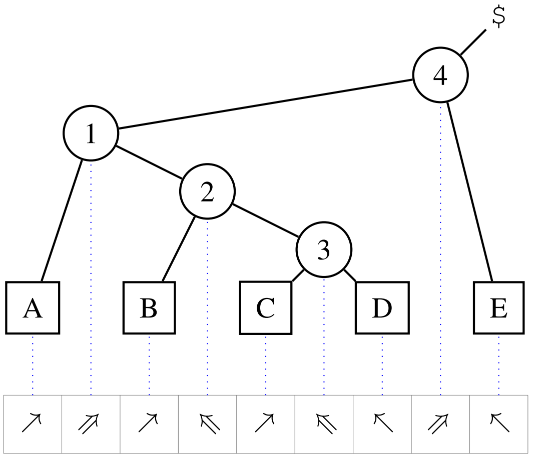

Figure 1: An example tree with the corresponding labels. The nonterminal nodes have been numbered based on an in-order traversal.

3.2 Labels

Consider the example tree shown in Figure 1. The tree is fully binarized (i.e. every node has either 0 or 2 children) and consists of 5 terminal symbols (A,B,C,D,E) and 4 nonterminal nodes (1,2,3,4). For any full binary parse tree, the number of nonterminals will always be one less than the number of words, so we can construct a one-to-one mapping between nonterminals and fenceposts (i.e. positions between words): each fencepost is matched with the smallest span that crosses it. Equivalently, we can number the nonterminals based on the in-order traversal of the tree and match node 1 with the first fencepost (between words A and B), node 2 with the second fencepost (between words B and C), etc.

For each node, we calculate the *direction of its parent*, i.e. whether the node is a left-child or a right-child. Although the root node in the tree does not have a parent, by convention we treat it as though it were a left-child (in Figure 1, this is denoted by the dummy parent labeled $).

Our scheme associates each word and fencepost in the sentence with one of four labels:

- "↗": This terminal node is a left-child
- "↖": This terminal node is a right-child
- "⇗": The shortest span crossing this fencepost is a left-child
- "⇖": The shortest span crossing this fencepost is a right-child

We refer to our method as **tetra-tagging** because it uses only these four labels.

Given a sentence with $n + 1$ words, there are altogether $2n + 1$ decisions (each with two options). The label representation of a tree is unique by construction. However, a fully one-to-one mapping between trees and label sequences is not possible because a sentence with $n + 1$ words admits $2 \cdot 4^n$ possible label sequences but only C_n distinct trees.

In the next subsection we show how the four labels above can be interpreted as actions in a transition-based parser, whereby some label sequences are *valid* and can be mapped back to trees, while others are *invalid*. In the following subsection, we describe an efficient dynamic program for finding the highest-scoring valid sequence under a probabilistic model.

3.3 Transition System

In this section, we re-interpret the four labels ("↗", "↖", "⇗", "⇖") as actions in a transition system can map from label sequences back to trees. Our transition system maintains a *stack* of partially-constructed trees, where each element of the stack is one of the following: (a) a terminal symbol, i.e. a word; (b) a complete tree; or (c) a tree with a single empty slot, denoted by the special element $\varnothing$. An empty slot must be the rightmost leaf node in its tree, but may occur at any depth.

The tree operations used are:

- MAKE-NODE(*left-child*, *right-child*): creates a new tree node.
- COMBINE(*parent-tree*, *child-tree*): replaces the empty slot $\varnothing$ in the parent tree with the child tree.

Algorithm 1 Decoding algorithm

```
Input: A list of words (words) and a corresponding list of
       tetra-tags (actions)
Output: A parse tree
stack ← []
buffer ← words
for action in actions do
    switch action do
        case "↗"
            leaf ← POP-FIRST(buffer)
            stack ← PUSH-LAST(stack, leaf)
        end case
        case "↖"
            leaf ← POP-FIRST(buffer)
            stack[−1] ← COMBINE(stack[−1], leaf)
        end case
        case "⇗"
            stack[−1] ← MAKE-NODE(stack[−1], ∅)
        end case
        case "⇖"
            tree ← POP-LAST(stack)
            tree ← MAKE-NODE(tree, ∅)
            stack[−1] ← COMBINE(stack[−1], tree)
        end case
    end switch
end for          ▷ The stack should only have one element
return stack[0]
```

The decoding system is shown in Algorithm 1, and an example derivation in shown in Figure 2.

Each action in the transition system is responsible for adding a single tree node onto the stack: the actions "↗" and "↖" do this by shifting in a leaf node, while the actions "⇗" and "⇖" construct a new non-terminal node. The transition system maintains the invariant that the topmost stack element is a complete tree if and only if a leaf node was just shifted (i.e. the last action was either "↗" or "↖"), and all other stack elements have a single empty slot.

The actions "↖" and "⇖" both make use of the COMBINE operation to fill an empty slot on the stack with a newly-introduced node, which makes the new node a right-child. New nodes from the actions "↗" and "⇗", on the other hand, are introduced directly onto the stack and can become left-children via a later MAKE-NODE operation. As a result, the behavior of the four actions ("↗", "↖", "⇗", "⇖") matches the label definitions from the previous section.

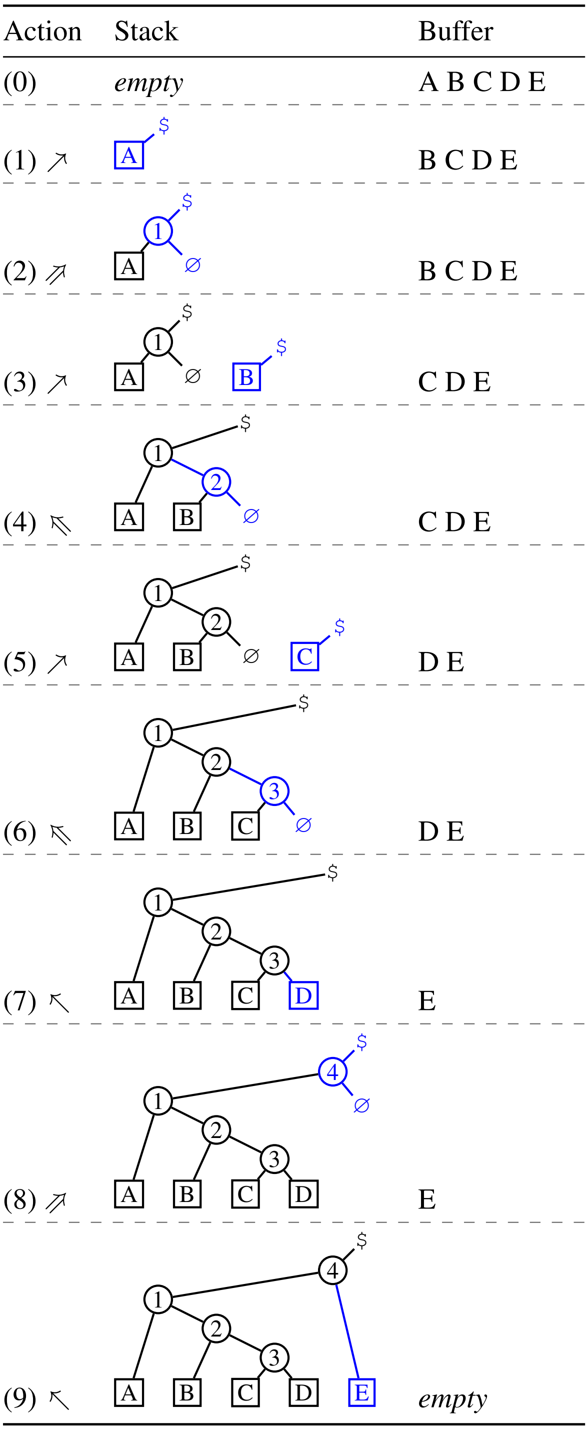

Figure 2: An example derivation under our transition system.

3.4 Inference

It should be noted that not all sequences of labels are valid under our transition system; in particular:

- The stack is initially empty and the only valid initial action is "↗", which shifts the first word in the sentence from the buffer onto the stack.

- The action "⇖" relies on there being more than one element on the stack (lines 16-18 of Algorithm 1).

- After executing all actions, the stack should contain a single element. Due to the invariant that the top stack element after a "↗" or "↖" action is always a tree with no empty slots, this single stack element is guaranteed to be a complete tree that spans the full sentence.

A tagging model that directly outputs an arbitrary sequence of labels is not guaranteed to produce a valid tree. Instead, we will work with models that output a *probability distribution* over sequences of labels; in particular, we will assume that label probabilities are predicted independently for each position in the sentence (conditioned on the input):

$$p(l_{0:T}) = \prod_{t=0}^{T} p(l_t)$$

We observe that the validity constraints for our transition system can be expressed entirely in terms of the *number* of stack elements at each point in the derivation, and do not depend on the precise structure of those elements. This property enables an optimal and efficient dynamic program for finding the valid sequence of labels that has the highest probability under the model.

The dynamic program maintains a table of the highest-scoring parser state for each combination of *number of actions taken* and *stack depth*. Prior to taking any actions, the stack must be empty. The algorithm then proceeds left-to-right to fill in highest-scoring stack configurations after action 1, 2, etc.

This same inference algorithm can also be seen as a type of beam search, where at each timestep the set of hypotheses under consideration is updated based on the label scores at that timestep. For each possible stack depth, only the highest-scoring hypothesis needs to be retained on the beam to achieve an optimal decode.

3.5 Connection to left-corner parsing

The design of our transition system borrows from past work on left-corner parsing, in that it preserves key properties that are motivated by considerations regarding human syntactic processing.

It has been observed that humans have no particular difficulty in processing deep left- and right-branching constructions, but that even a small amount of center-embedding greatly hurts comprehension (Miller and Chomsky, 1963). This disparity has been attributed to cognitive limitations with respect to working memory and processing capabilities.

An analysis of the processing and space requirements of top-down and bottom-up parsing strategies reveals that neither strategy has equivalent treatment of left- and right-branching structure. On the other hand, left-corner parsing has the property that space utilization is constant when processing fully left- or right-branching structures, but increases whenever a center-embedded construct is encountered (Abney and Johnson, 1991; Resnik, 1992).

Past work has operationalized these considerations by defining a class of grammars and the order in which grammar rules are applied (Rosenkrantz and Lewis, 1970), or by applying a left-corner transform (or the closely-related right-corner transform) to syntactic trees (Johnson, 1998; Schuler et al., 2010). Our method is instead formulated as a transformation from trees to label sequences, and uses word-synchronous feature vectors to drive the derivation rather than requiring an explicit grammar to fulfill that role.

In the context of our tetra-tag parser, a left-branching tree is characterized by the action sequence "↗ ⇗ ↖ ⇗ ↖ ⇗ ↖ . . .", where neither "⇗" nor "↖" change the size of the stack (see Algorithm 1). A right-branching tree has the action sequence ". . . ⇖ ↗ ⇖ ↗ ⇖ ↗ ↖", where each "↗" action grows the stack by one element and each "⇖" action shrinks the stack by one element. As a result, the depth of the stack remains effectively constant when parsing either left- or right-branching structures. Larger stack sizes are needed only when center-embedded constructs are encountered.

In practice, the largest stack depth observed at any point in the derivation for any tree in the Penn Treebank is 8. By comparison, the median sentence length in the data is 23, and the largest sentence is over 100 words long.

These observations directly impact how efficiently we can perform inference in our parsing framework. The time complexity of the dynamic program described in the previous section is $O(nd^2)$, where n is the length of the sentence and d is the maximum possible depth of the stack. If our parser were required to produce arbitrary trees, we would have $d = O(n)$ and an overall time complexity of $O(n^3)$. When dealing with natural-language parse trees, however, we can cap the maximum stack depth allowed in our inference procedure, for example by setting $d = 8$. If we assume that this cap is a constant (as would be the case if it corresponds directly to human cognitive limitations), the $O(nd^2)$ time complexity effectively becomes $O(n)$. In other words, our inference procedure will, in practice, require a constant amount of time per word in the sentence.

3.6 Handling of labels and non-binary trees

Thus far, we have only dealt with binary unlabeled trees, but our method can be readily extended to the labeled and non-binary settings.

To incorporate labels, we note that each of our four actions corresponds to a single node in the binary tree. The label to assign to a node can therefore be incorporated into the corresponding action; for example, the action "⇗S" will construct an S node that is a left-child in the tree, and the action "↖NP" will construct a single-word Noun Phrase that is a right-child. We do not impose any constraints on valid label configurations, so our inference procedure essentially remains unchanged.

To handle non-binary trees, we follow past work by binarizing the trees and collapsing unary chains. We use fully right-branching binarization, where a dummy label is introduced and assigned to nodes generated as a result of binarization.

4 Results

We evaluate our proposed method by training a parser that directly predicts action sequences from pre-trained BERT (Devlin et al., 2018) word representations. Two independent projection matrices are applied to the feature vector for the last sub-word unit within each word: one projection produces scores for actions corresponding to that word, and the other for actions at the following fencepost. The model is trained to maximize the likelihood of the correct action sequence, where BERT parameters are fine-tuned as part of training. We compare our model with our previous chart parser (Kitaev and Klein, 2018), which was fine-tuned from the same initial BERT representations. Unlike the tetra-tagging approach, the chart parser constructs feature vectors for each span in the sentence and uses the cubic-time CKY algorithm for inference.

	F1
Chart (Kitaev and Klein, 2018)	95.59
Tetra-Tagging (ours)	95.44

Table 1: Comparison of F1 scores on the WSJ test set. Both parsers use BERT$_{\texttt{LARGE}}$ (Devlin et al., 2018) word representations fine-tuned from the same initial parameters.

We evaluate both models on the Penn Treebank (Marcus et al., 1993). Results are shown in Table 1. The tetra-tagging approach comes close to matching the accuracy of the chart parser (which, to our knowledge, achieves the best-reported numbers on this dataset.)

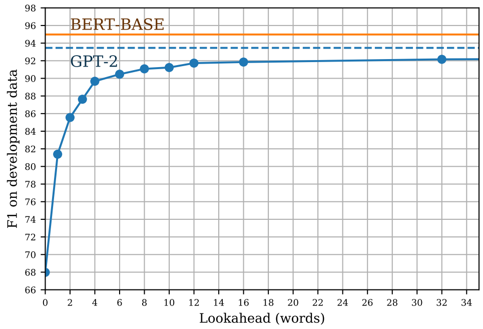

Figure 3: Incremental parsing accuracy is low when using GPT-2 with no lookahead, but adding a modest amount of lookahead substantially improves F1 (solid blue curve). Allowing bi-directional attention across multiple layers following GPT-2 helps further (dashed blue line). We also compare with a similarly-sized BERT model (solid orange line).

5 Word-Synchronous vs. Incremental

Thus far we've discussed *word-synchronous* syntactic analysis. A closely related question, arising from a number of perspectives including computational efficiency and cognitive modeling, is whether we can build a fully *incremental* parser. While our parsing algorithm itself is incremental (meaning that it operates in a strictly left-to-right manner) the BERT word representations underneath are not. Indeed, the BERT model is specifically constructed to be *deeply bi-directional*.

We experiment with incremental parsing instead by using the GPT-2 (Radford et al., 2019) architecture to construct token-aligned vector representations. The publicly-available GPT-2 model is configured to be roughly comparable to BERT$_{\texttt{BASE}}$: both models use a 12-layer architecture with 768-dimensional hidden states and 12 self-attention heads per layer. However, in GPT-2 a given position in the sentence is only allowed to attend to what came before it (and not any of the later words).[1]

We further augment GPT-2 with a notion of *lookahead* by allowing the parser to incorporate information from a fixed context window following a word. We opt to add lookahead by introducing extra layers on top of GPT-2, rather than modifying the pre-initialized GPT-2 architecture in a way that deviates from the pre-training conditions. To achieve a lookahead of k words, we first add 8 self-attention layers on top of the GPT-2 architecture. Attention in the first of these extra layers is constrained to look no more than k words into the future. The following 7 layers use fully causal self-attention, meaning that attention to subsequent words is disallowed.

We find that using GPT-2 with no lookahead performs poorly, but that even a modest amount of lookahead leads to substantial quality improvements (Figure 3). We also include in our comparison two architectures that allow arbitrary amounts of lookahead. The first uses GPT-2 but allows unrestricted attention patterns in all 8 layers that follow it. This deeply bi-directional lookahead outperforms lookahead at only a single layer (93.5 F1 vs. 92.3 F1). The second point of comparison is a model that admits no lookahead in the parser-specific self-attention layers, but uses BERT$_{\texttt{BASE}}$ rather than GPT-2. The model with BERT achieves 95 F1 vs. 93.5 F1 for the best application of GPT-2, which suggests that some aspect of the BERT architecture (perhaps its deep bi-directionality during pre-training) makes it more suitable for use in our parser than GPT-2.

6 Conclusion

We present a word-synchronous linearized tree representation that uses a set of four actions. The actions are word-aligned and can be directly predicted from contextualized word vectors using an approach we call *tetra-tagging*. We present an optimal dynamic programming algorithm for finding the highest-scoring tree from a sequence of tag probabilities and show that, with mild assumptions, the inference algorithm runs in linear time.

[1] BERT and GPT-2 also use different pre-training data and sub-word tokenization.

References

Steven P. Abney and Mark Johnson. 1991. Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*.

Greg Durrett and Dan Klein. 2015. Neural CRF Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent Neural Network Grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent Parsing as Sequence Labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324. Association for Computational Linguistics.

James Henderson. 2003. Inducing History Representations for Broad Coverage Statistical Parsing. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 103–110.

Mark Johnson. 1998. Finite-state Approximation of Constraint-based Grammars using Left-corner Grammar Transforms. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*.

Nikita Kitaev and Dan Klein. 2018. Multilingual Constituency Parsing with Self-Attention and Pre-Training. *arXiv:1812.11760 [cs]*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. 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.

Jiangming Liu and Yue Zhang. 2017. In-Order Transition-based Constituent Parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424.

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

George A Miller and Noam Chomsky. 1963. Finitary models of language users. *Handbook of mathematical psychology*, II.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. page 24.

Philip Resnik. 1992. Left-corner parsing and psychological plausibility. In *Proceedings of the 14th Conference on Computational Linguistics-Volume 1*, pages 191–197. Association for Computational Linguistics.

Daniel J. Rosenkrantz and Philip M. Lewis. 1970. Deterministic left corner parsing. In *Switching and Automata Theory, 1970., IEEE Conference Record of 11th Annual Symposium On*, pages 139–152. IEEE.

Kenji Sagae and Alon Lavie. 2005. A Classifier-Based Parser with Linear Run-Time Complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132. Association for Computational Linguistics.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-Coverage Parsing Using Human-Like Memory Constraints. *Computational Linguistics*, 36(1):1–30.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A Minimal Span-Based Neural Constituency Parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827. Association for Computational Linguistics.

David Vilares, Mostafa Abdou, and Anders Søgaard. 2019. Better, Faster, Stronger Sequence Tagging Constituent Parsers. *arXiv:1902.10985 [cs]*.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems 28*, pages 2755–2763. Curran Associates, Inc.

Yue Zhang and Stephen Clark. 2009. Transition-Based Parsing of the Chinese Treebank using a Global Discriminative Model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171. Association for Computational Linguistics.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and Accurate Shift-Reduce Constituent Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443. Association for Computational Linguistics.