Learned Incremental Representations for Parsing

Nikita Kitaev **Thomas Lu** **Dan Klein**
Computer Science Division
University of California, Berkeley
{kitaev,tlu2000,klein}@berkeley.edu

Abstract

We present an incremental syntactic representation that consists of assigning a single discrete label to each word in a sentence, where the label is predicted using strictly incremental processing of a prefix of the sentence, and the sequence of labels for a sentence fully determines a parse tree. Our goal is to induce a syntactic representation that commits to syntactic choices *only as they are incrementally revealed by the input*, in contrast with standard representations that must make output choices such as attachments speculatively and later throw out conflicting analyses. Our learned representations achieve 93.72 F1 on the Penn Treebank with as few as 5 bits per word, and at 8 bits per word they achieve 94.97 F1, which is comparable with other state of the art parsing models when using the same pre-trained embeddings. We also provide an analysis of the representations learned by our system, investigating properties such as the interpretable syntactic features captured by the system and mechanisms for deferred resolution of syntactic ambiguities.

1 Introduction

Language comprehension in humans is, to a non-trivial extent, an incremental process. Human speech is heard word by word, and, while the precise nature of the incrementality is not a settled question, a listener does not wait for a full sentence to end before any processing or understanding can begin. In contrast, some of the highest-performing machine models for syntactic parsing operate precisely in this manner: they require a full sentence as input, and perform deeply bidirectional processing to produce their outputs. Human capabilities suggest that we should also be able to build accurate parsers that instead operate incrementally.

Incrementality in NLP has often been equated with left-to-right processing. For example, incremental transition-based parsers receive their input one word at a time, and — after each word — output any number of actions such as *shift* or *reduce*, where the full sequence of actions represents a syntactic analysis of the input. However, in this paper we are interested in a stronger notion of incrementality, which we refer to as *non-speculative incrementality*. We say that a representation is *speculative* when a symbol in the representation encodes a commitment to a certain syntactic decision, but the evidence for that decision is not present in the corresponding prefix of the input. Transition-based systems are frequently speculative; we give an example sentence in Figure 1, where a decision must be made regarding whether the preposition "on" attaches to noun "proposal" or the verb "approved." Transition-based approaches such as shift-reduce or attach-juxtapose (Yang and Deng, 2020) place the action that determines the preposition attachment earlier in the left-to-right processing pattern than the disambiguating word ("Monday" or "taxes") that reveals the correct analysis. Similarly in CCG parsing, the representation of a CCG analysis in the form of a sequence of supertags is likewise speculative — including for this same example, where the

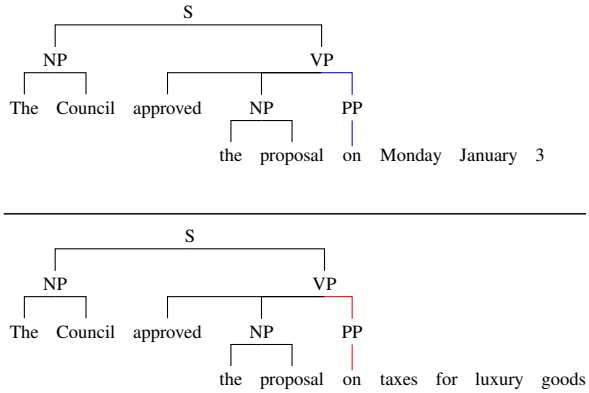

Figure 1: A case of ambiguity where speculatively committing to an attachment decision can lead an incremental parsing system into a dead end.

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 3086 - 3095

correct supertag for each word cannot be predicted based on only that word and its preceding context.

The speculative nature of incremental transition systems or CCG supertags makes it impractical to recover an accurate parse by simply committing to the highest-scoring option at each each point where a decision must be made. An incremental parser in the speculative paradigms must instead consider multiple analyses in parallel, and later throw out analyses that are inconsistent with the sentence; this can be done through a procedure like beam search. In other words, the true representation of syntactic information at each point in the sentence is not a single sequence of actions (or supertags), but rather a *belief state* (beam state) that contains multiple candidate analyses. In the limit of infinite beam size, the parser ceases to be incremental: its belief state can contain all reasonable analyses, deferring all choices to the very end of a sentence.

Our goal in this work is to design a representation for parsing that is maximally *speculation free*. In other words, it should record commitments to syntactic choices only as they are incrementally revealed by the input. We additionally want our representation to relate to constituency trees in a similar way to how transition-based actions relate to them: that is, through a deterministic transformation function. A sequence of shift-reduce or attach-juxtapose actions is not identical to a parse tree, but it can be mapped to a tree using a deterministic automaton that interprets the discrete actions as operations on a tree fragment or stack of tree fragments. A sequence of supertags is likewise not the same as a tree, and mapping it to a tree requires further processing in the form of finding and applying an appropriate series of combinators. These mappings are non-trivial, especially in the case of CCG, so we should not expect our mapping to be trivial either — our only requirement is that it be deterministic and operate entirely from our representation, having already discarded the raw text in the sentence. Finally, we would like our representation to take the familiar form of a sequence of discrete symbols.

We propose to arrive at such a representation through *end-to-end learning*, rather than manual construction. The model can then make its own decisions about when syntactic decisions take place, how to handle cases of ambiguity, and how to represent belief states within the learned system itself. This system will learn to encode linguistic and structural features to allow effective incremental parsing. Our end-to-end approach is a model that proceeds in two stages. The first stage maps from individual words to syntactic decisions, which are represented as discrete tokens drawn from a small, bounded vocabulary. The second component of our system is a read-out network that takes a sequence of discrete tags as input and produces a conventional parse tree as output. Both stages are trained jointly in an end-to-end manner. Crucially, we do not a priori assign meaning to the discrete tokens (e.g. actions like *shift*, or supertags like in CCG); we only specify the total number of symbols available to the model to control the complexity of the representation. Unlike a speculative system, our representation can be used by finding the single highest-scoring tag at each position in a sentence, and then converting the resulting sequence of tags into a tree.

Important properties that we evaluate for our proposed approach are its quality (as measured by F1 score on the Penn Treebank), as well as compactness (how many bits per word are required to encode syntactic information). At 5 bits per word, a parser using our representations achieves 93.72 F1, and at 8 bits per word it achieves 94.97 F1 — comparable to the method of Kitaev et al. (2019) trained with the same pre-trained embeddings.

We further provide an analysis of the symbols learned by our model, including explorations of the linguistic features captured by the symbol set, the information content of our incremental representation for prefixes of a full utterance, and the system's ability to defer resolution of attachment ambiguities.

Our models and code are publicly available.[1]

2 Related Work

This work is inspired by the concept of incremental parsing implemented in works such as Larchevêque (1995) and Lane and Henderson (2001). With regards to neural parsers, recent strides in incremental parsing include the attach-juxtapose parsers from Yang and Deng (2020). However, these neural models often have incremental tree construction mechanisms, but are not incremental from the raw input level due to reliance on pretrained bidirectional models such as the works of Devlin et al. (2019) and Yang et al. (2019).

[1] https://github.com/thomaslu2000/Incremental-Parsing-Representations

The placement of an information bottleneck on token representations has also been studied in the bidirectional case by Li and Eisner (2019), who reported many similar findings about the syntactic features learned by discrete tags. However, our model differs in that it explores the incremental, non-speculative case, as well as in the implementation of the parsing model and its constraints on representation size.

Our incremental parsing system can be compared to manually formulated representations such as shift-reduce or CCG supertagging. However, for purely incremental parsing, limitations of shift-reduce and CCG supertagging may necessitate the use of beam search to produce more accurate, viable parse trees, as in the works of Zhu et al. (2013) and Bhargava and Penn (2020).

Other works have also analyzed the discrete features useful for syntactic parsing. Some researchers augmented parsing models by adding discrete, hand-coded indicator features based on the raw sentence as in Hall et al. (2014). Similar hand-coded, discrete features have been shown to improve other tasks such as NMT (Sennrich and Haddow, 2016). Previous experiments by Gaddy et al. (2018) have analyzed whether neural parsers based on bidirectional LSTMs capture other hand-made indicator functions from earlier hypotheses by Petrov and Klein (2007). By contrast, our model seeks to directly learn new features, and in fact, many of the hand-made indicators from previous works arise naturally in the learned symbols of our model.

There also exists work examining the learned grammatical rules of a stack-based recurrent neural network via analysis of an attention mechanism (Kuncoro et al., 2017). By contrast, our analysis has a lesser focus on the attention distribution between tokens, and a greater focus on the features and syntactic decisions captured by each individual symbol.

3 Approach

Our model is based on a parsing architecture that contains an encoder layer that uses a pretrained network and a chart-based decoder, as detailed in Kitaev and Klein (2018). To ensure incrementality, the encoder for this incremental model uses GPT-2 as a base, which disallows a backwards flow of information from future tokens (Radford et al., 2019).

At the interface between the pre-trained encoder and subsequent parts of the model (which we refer to as the *read-out network*), we introduce a discretization step that collapses the continuous, high-dimensional vectors from the encoder network to a small inventory of discrete symbols. The read-out network has access only to these discrete symbols and not to the original text of the input; in other words, the sequence of discrete symbols must encode all information necessary to represent the syntactic structure of the sentence. We introduce an information bottleneck that limits the size of the discrete token vocabulary to as few as 32 distinct symbols per raw input token. The decision to label each token with a single symbol is partially rooted in prior research providing evidence that syntactic decisions among human speakers adhere to the uniform information density hypothesis, thus each token may convey similar amounts of syntactic information (Levy and Jaeger, 2006).

Concretely, a learned projection matrix is first applied to the token-level representation vectors of GPT-2. Each projected vector is then converted into a single discrete symbol via vector quantization (van den Oord et al., 2017). The number of symbols is kept small; as such, only a few bits are needed to encode all symbols. In comparison, the base architecture uses a 512-dimensional vector of 32-bit floating point numbers for each token. We can obtain high parsing accuracy sending 5 bits per token, which is only 0.03% of the bits of the base architecture's token representations. At around 8 bits per token, parsing performance approximately matches that of the base architecture.

After discretization, each symbol from the sequence is associated with a learned embedding, as specified in the vector quantization codebook. These vectors are fed as an input to the bidirectional read-out network, which consists of Transformer layers and an MLP-based span classification layer that otherwise match the base architecture. The output of the network is a chart of scores representing each possible constituent span of the sentence. A tree is then efficiently generated through the CKY algorithm following the span scoring methods of Stern et al. (2017).

It should be noted that while the encoder is unidirectional, our read-out network is bidirectional. The bidirectionality allows the network enough slack to learn a flexible mapping between the induced representation and standard constituency

trees. For example, the discrete symbol associated with a word may help determine a syntactic attachment that concerns previous words that have already been assigned their own symbols. In practice, the behavior of the read-out network exhibits consistent patterns that we interpret in Section 5. Moreover, the main product of our method — and the principle object of analysis in this paper — is not the network itself but rather the sequence of discrete symbols, each of which encodes no knowledge of future context.

We train our models using a learning rate of 3e-5 for weights carried over from pre-training, a learning rate of 1e-4 for randomly initialized weights, and a batch size of 32. In order to facilitate training, the first two epochs of training proceed without the use of vector quantization. During this time, a streaming k-means algorithm (Ackermann et al., 2012) calculates the initial centroids to use for vector quantization. Over the course of the third epoch, the model linearly interpolates between continuous and quantized representations, and uses only the quantized version from the fourth epoch until the end of training. We found that cold-starting with randomly-initialized centroids performs worse, in the sense that some centroids would never be used or updated at any point during training. We attribute this degradation to the fact that randomly sampled code vectors are a poor distributional fit for outputs arising from a pre-trained GPT-2 model.

4 Results

We apply our approach to the labeled constituent trees of the English Penn Treebank (Marcus et al., 1993). The final incremental model generated using this setup achieves a score of 94.97 F1 on the Penn Treebank WSJ test set. This model uses only 8 bits per token (256 symbols) to define the discrete symbol set using a unidirectional pretrained model (GPT2-medium). A comparable model (Kitaev et al., 2019) that combines the same pre-trained encoder with deep bidirectional processing achieves 95.10 F1. This shows that our representation can induce parse trees with competitive accuracy.

In Table 1, we present an F1 score comparison that highlights the behavior of different syntactic representations with different choices of encoder. When directly predicting either per-span label probabilities (following the span classification approach of Stern et al., 2017), or actions in the attach-juxtapose transition system (Yang and Deng, 2020), failing to include bidirectional layers on top of a unidirectional GPT-2 incurs a strong accuracy penalty. This is despite the fact that both systems can discard speculative attachment decisions. In the case of the chart parser with representations that consist of label probabilities for each span, adding an additional word can cause a switch to a new analysis by way of the CKY decoding procedure. In the case of the attach-juxtapose parser, the same can be achieved via the use of beam search. Nevertheless, incrementally predicting either of these representations fails to leverage the full power of the pre-trained encoder.

	Encoder Type		
	Bi (↔)		Uni (→)
Representation	BERT	GPT-2	GPT-2
Span Classification (Kitaev et al., 2019)	95.59	**95.10**†	93.95†
Attach-Juxtapose (Yang and Deng, 2020)	**95.79**	94.53†	87.66†
Learned (This work)	95.55	–	**94.97**

Table 1: F1 on the WSJ test set for parsers using different syntactic representations and pre-trained encoders. Our representation results in an F1 score that is close to the maximum achievable with the same pre-trained GPT-2-medium model. †Results based on integrating GPT-2 with publicly available code for prior methods, with and without bidirectional layers on top of GPT-2.

The choice of GPT-2 rather than a stronger bidirectional model has a large effect on the performance on the Penn Treebank. To give a more accurate comparison with other models, Table 1 also shows F1 scores for models based on BERT, with the recognition that no model with such a deeply bidirectional encoder can truly be referred to as incremental. Our approach of inducing learned representations with vector quantization also performs well in this setting, validating the method.

Even higher scores are achievable by using stronger pre-trained models, different forms of bidirectional processing, and additional supervision in the form of dependency trees; Mrini et al. (2020) combine all of these elements to achieve 96.38 F1. However, many of these techniques are either orthogonal to our work, or they cannot be borrowed into an incremental setting due to their focus on deeply bidirectional neural processing.

	Context Type		
Set Size	Uni (→)	Bi (↔)	Clustered Lexicon
2	45.71	38.42	27.92
4	68.93	69.74	52.36
8	86.28	87.41	65.88
16	92.13	92.78	71.53
32	93.50	94.82	79.51
64	94.01	95.24	83.17
128	94.13	95.33	84.65
256	94.49	95.41	85.86

Table 2: Parsing performance on WSJ development set using different model contexts. Each number is an average F1 score across 3 runs. Our unidirectional model surpasses the baseline performance of a clustered lexicon and approaches the performance of a bidirectional context.

We further evaluate our approach in terms of the compactness of the produced representations. To do this, we trained a number of models while varying the size of the symbol set. For added comparison, we also trained models using a bidirectional pretrained encoder (BERT). As a baseline, we also produced a model that assigns symbols through simple k-means clustering of single-word embeddings (Mikolov et al., 2013) rather than fine-tuned contextual models. The average F1 score for each model across a range of tag set sizes is shown in Table 2. Note that while the numbers of possible tags are all powers of two, this is not a strict requirement of the model, and any positive integer may be used as the tag set size.

While our best-performing unidirectional model uses 8 bits per token, using as few as 5 bits per token (32 symbols) retains a performance of 93.72 F1 on the test set. As a point of comparison, gold CCG supertags in the CCGbank (Hockenmaier and Steedman, 2007) training data have an entropy of 5.14 bits per word. However, CCG decoding typically requires multiple analyses to be considered in parallel. A better comparison, then, might be the entropy of top-k supertag predictions from a supertagging model. We find that the trained model of Tian et al. (2020) has an entropy of 5.98 bits/word for its ranked top-2 predictions, 7.57 for top-3, and 9.03 for top-4. Our method's best-performing setting of 8 bits per word is therefore at an entropy level similar to top-3 or top-4 predictions for a recent CCG supertagger.

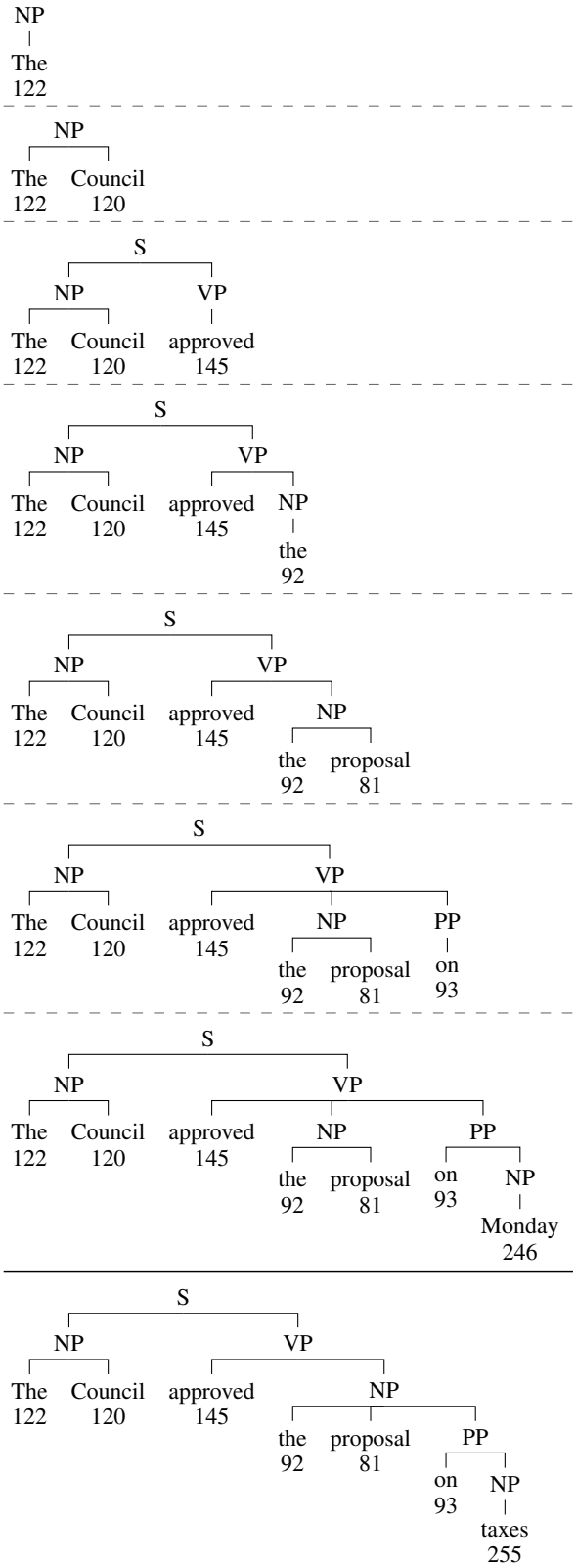

Figure 2: Applying our read-out network to prefixes of the syntactic tag sequence demonstrates that syntactic decisions are committed to incrementally, and are not all deferred to the end of the sentence. Nevertheless, the *same* tag sequence may decode to *different* trees, depending on a future tag. Numbers below each word denote the discrete tags assigned by our model.

5 Discussion

5.1 Incremental Behavior

Having achieved high F1 scores, we must next demonstrate that our representation is, in fact, incremental. An incremental representation has meaningful syntactic information in each of its prefixes, and we can probe this by running our read-out network after each word in the sentence, as shown in Figure 2. The resulting trees involve mostly local changes from word to word, which shows that important information is not being deferred to the very end of a sentence.

It should be noted that our read-out network was *never trained* on anything but complete sentences. Applying it to fragments will produce individual trees that may not be representative of the ambiguity present in the underlying representation. For example, after the word "on" the read-out network outputs a prepositional phrase that initially appears to attach to the verb. Depending on the label chosen for the next word, however, the final attachment can be to either the verb or the noun phrase.

Nevertheless, this approach allows us to probe the degree to which the representation encodes syntactic decisions immediately, versus deferring them to some later point. For each span in the final tree, we can walk backwards through the partial read-outs to find the furthest point when the span still appears in a readout; we call this the point in time that a span is *finalized*. In Figure 2, the noun phrase "The Council" is finalized after the word "Council," and the verb phrase is finalized after the word "approved." For the purposes of identifying whether a span is the same across two different trees, we assume that a span is uniquely identified by its label, its starting position, and the position of the last word in the leftmost child of the span (or the position of the single word in the span, if it only covers one word). The last of these we also refer to as the *split point* of the span.

Figure 3 shows that approximately half of all spans are finalized either immediately at or immediately after their split point. The distribution has a tail that falls off roughly exponentially, as shown by the loosely straight-line decay on the log-linear plot. The presence of this tail stands in contrast with the attach-juxtapose representation, where all attachments are determined immediately after a split point, and the only way to defer a decision past that point is to retain multiple analyses on something like a beam. An extremely frequent phenomenon within the tail is when a phrase expands to be nested inside another phrase of the same type: sometimes this happens due to the nature of the constituency representation we're converting to, and sometimes it reflects actual syntactic ambiguity. One example in the latter is shown in Figure 4, where either the NP or the S node must expand due to coordination. Note how our representation can handle this situation without considering multiple candidate labelings, while speculative transition-based systems would not.

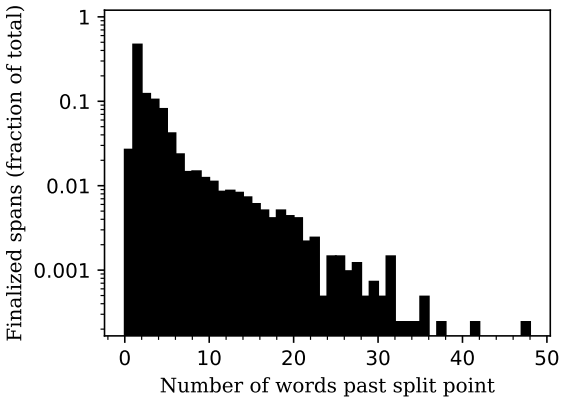

Figure 3: Our representation commits to (finalizes) the majority of spans within just a few words of their split point. The tail of the histogram reflects cases where it may not commit to a decision (e.g. resolving an ambiguity) until many words past the split point of the relevant span. Note the log scale for the y-axis.

5.2 Entropy of Symbol Distribution

In our parsing scheme, each new token is assigned a single syntactic symbol based on all tokens up to the current. The subsequent sequence of symbols then fully determines a constituency tree.

For different random initializations of our approach with the same set size, similar features are typically captured by the system. Models using smaller sets of symbols tend to have the most variability in terms of feature distribution. The entropy of several random initializations of these sets is shown in Figure 5.

Entropy appears to roughly stabilize after a small number of training iterations. At this point, the characteristics of each symbol also roughly stabilize. The entropy of the distribution of symbols seems to increase linearly with the number of bits per representation, but does not reach a level that corresponds to uniform usage frequency for all symbols in the discrete inventory.

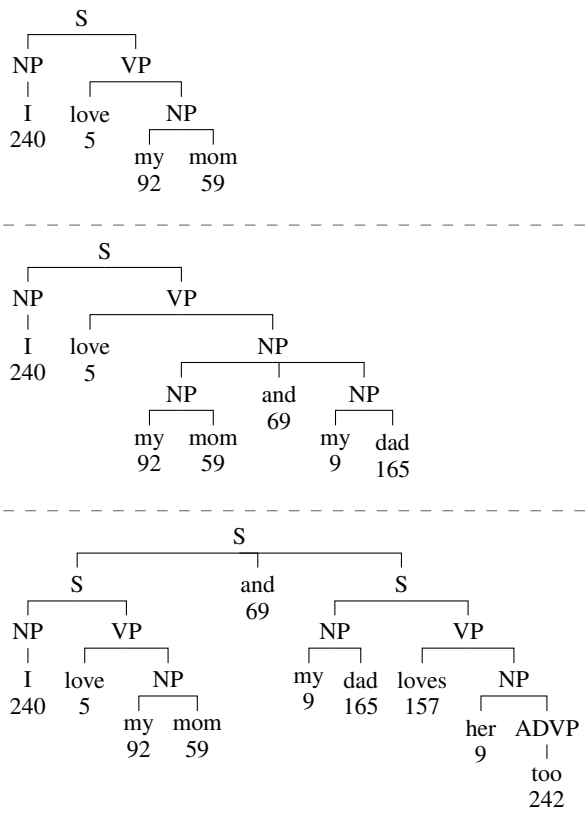

Figure 4: Our representation is a strictly append-only sequence of tags. Nevertheless, later tags can resolve ambiguities (in this case, coordination scope) introduced at an earlier point in the sentence.

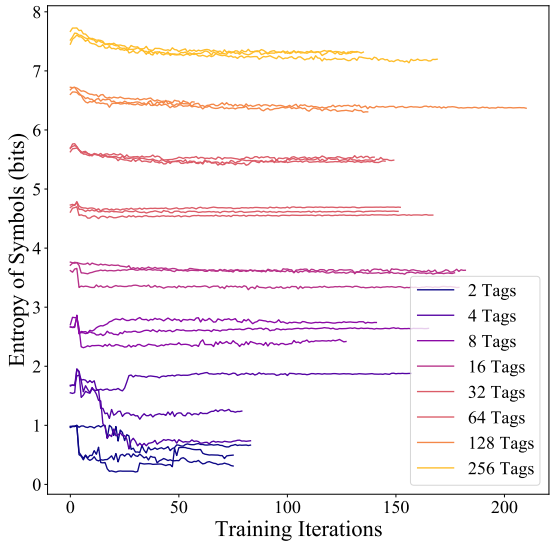

Figure 5: Entropy of derived symbol sets over the training period (in bits). Multiple training runs are shown for each tag set size.

5.3 Learned Token-Level Features

Due to the small size of our information bottleneck, we hypothesize that our symbols encode the most powerful features needed to produce an accurate constituent tree representable by the given bitrate. Thus, by analyzing the features captured by differently sized symbol sets, we can deduce a rough hierarchy of distinct features that are relevant to the incremental parsing task.

Starting with a system using only 2 discrete symbols, we steadily increase the bit rate of the dictionary and manually inspect the representation to find interpretable token-level features. Many of these are similarly found in other works investigating the linguistic features captured by the token representations of neural parsers (Gaddy et al., 2018; Li and Eisner, 2019). What follows is the rough order in which several of these features appear:

1. Separation between noun phrases and verb phrases
2. Symbols representing a new determiner or noun phrase, and ending ongoing noun phrases
3. Special symbols for other simple parts of speech (adjectives, adverbs, questions, punctuation, etc.)
4. Indication of a token being in subordinate clauses or relative clauses
5. Multiple symbols per part of speech (often nouns, verbs, and prepositions) signifying different attachments
6. Indication of other specific and specialized structures, such as clauses which are the object of a verb phrase, or a noun within a relative clause
7. Other specific common language features, such as possessive markings, gerunds, tokens introducing new clauses, or adverbs that modify adjectives rather than verbs

5.4 Clause Separation

To demonstrate the features learned and captured by these tags, consider a model using only 32 symbols. Main, subordinate, and relative clauses are typically associated with different discrete symbols for the same parts of speech.

The sentences in Figure 6 display the predicted tag sequences and parses involving many of the

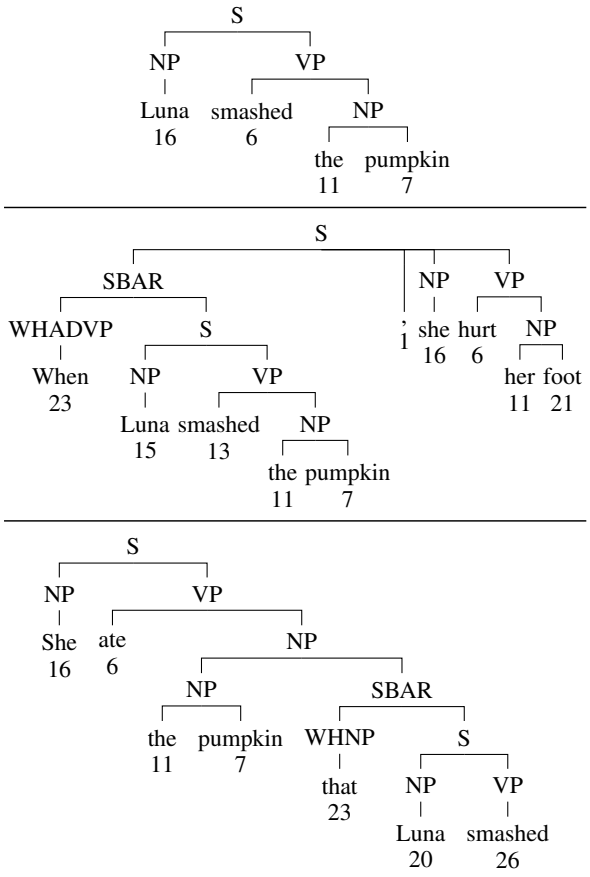

Figure 6: Predicted parses of the clause "Luna smashed the pumpkin," where the subject and verb are assigned different symbols depending on the clause type. Tags shown below each word were predicted by a model that may use up to 32 distinct symbols.

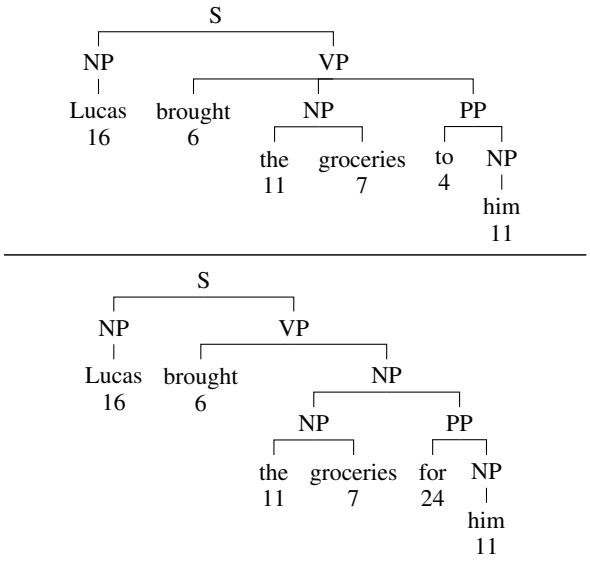

Figure 7: Derivation of the sentences "Lucas brought the groceries to him" and "Lucas brought the groceries for him." Tags shown below each word were predicted by a model that may use up to 32 distinct symbols.

same words, but within different clause types. In main clauses, subjects and verbs are assigned symbols 16 and 6. Subordinate clauses, however, tend to use alternate symbols 15 and 13 for subject nouns and verbs respectively, while relative clauses use 20 and 26.

This feature of our symbol set suggests that our tags capture structural context beyond the current word, and the features learned by these tags can have human-interpretable meanings upon analysis.

5.5 Ambiguity Resolution

The structure of the final parse tree is interpolated from the series of discrete symbols resulting from the encoder network. To analyze how syntactic decisions are encoded in our representation, we first attempted to train a modified PCFG based on Klein and Manning (2003), with the goal of replicating the behavior of our read-out network. However, this system could only reach a performance around 76.18 F1 towards the reconstruction task, suggesting that the PCFG's assumptions of locality and sub-tree independence are not valid for our learned representations.

To better understand the mechanism by which our representations are capable of representing a wide range of syntactic structures, we focus specifically on cases with potential syntactic ambiguities. Consider the minimal pair shown in Figure 7, where the predicted syntactic structure differs by only a single prepositional attachment. This pair uses the same encoder model as the previous example, which has a maximum of 32 discrete symbols. Due to the different symbols assigned to the prepositions, the read-out network attaches the prepositional phrase at a different height.

Not all prepositional attachments can be reliably determined based on only the words up to and including the preposition. To avoid speculative behavior, the tag sequences must contain mechanisms for recording instances of ambiguity and then resolving them based on tokens further down in the string. Figure 8 shows an example of how our representation handles such situations. Running the read-out network for the prefix "Lucas brought the groceries for" produces a partial parse that attaches the preposition to "the groceries." However, the final token offers additional information that may influence the attachment location, suggesting that the symbol sequence up to the preposition does not eliminate either possible structure, but rather

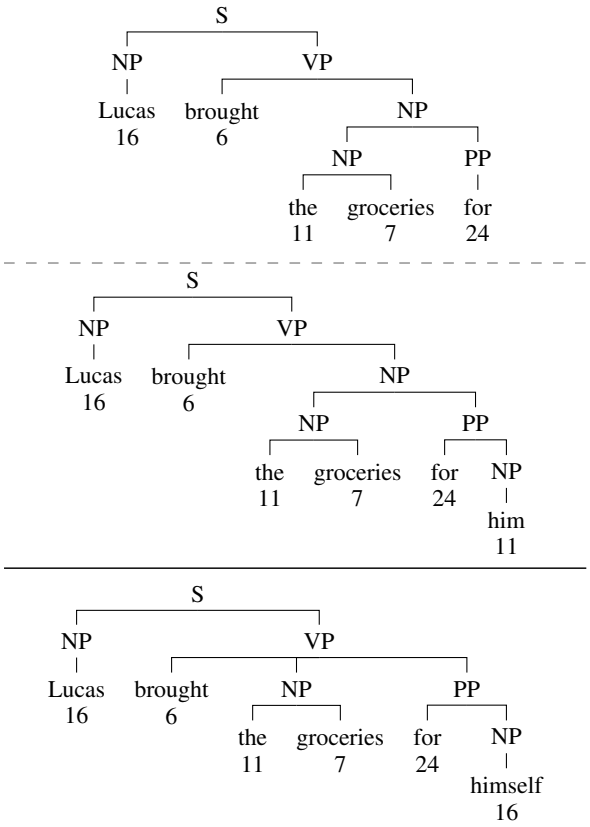

Figure 8: Two possible sentences continuing from the prefix "Lucas brought the groceries for" where the final attachment height for the prepositional phrase is determined by the discrete symbol for the word following the preposition. Tags shown below each word were predicted by a model that may use up to 32 distinct symbols.

encodes the locations of other likely attachments. The encoder's decision over whether to mark the final token as symbol 11 or 16 allows the final tree to have an attachment to the verb phrase, rather than adhering to the partial interpretation of targeting the noun phrase.

6 Conclusion

In this paper, we present an approach to inducing syntactic representations that associate each token in the input with a discrete symbol from an arbitrarily-sized vocabulary, where the representations can be predicted incrementally in a strictly append-only manner. Our models achieve high F1 on the WSJ test set despite a steep information bottleneck limiting the information that can be associated with each token. The token-level tags produced by our model encode relevant syntactic information suitable for the given bit rate, while the locations of these tags serve to concretely define the location at which syntactic decisions can be committed to in a speculation-free manner. These systems can serve to improve our understanding of incremental parsing and sequential decision making, and the underlying computational methods may be useful in the analysis of other incremental contexts.

Acknowledgments

This research was supported by DARPA under the LwLL program / Grant No. FA8750-19-1-0504.

References

Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. 2012. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–1.

Aditya Bhargava and Gerald Penn. 2020. Supertagging with CCG primitives. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 194–204, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's going on in neural constituency parsers? an analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010, New Orleans, Louisiana. Association for Computational Linguistics.

David Hall, Greg Durrett, and Dan Klein. 2014. Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–237, Baltimore, Maryland. Association for Computational Linguistics.

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.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual*

Meeting of the Association for Computational Linguistics, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

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

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia, Spain. Association for Computational Linguistics.

P. R. Lane and J. B. Henderson. 2001. Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(02):219–231.

J.-M. Larchevêque. 1995. Optimal incremental parsing. *ACM Trans. Program. Lang. Syst.*, 17(1):1–15.

R. Levy and T. Florian Jaeger. 2006. Speakers optimize information density through syntactic reduction. In *NIPS*.

Xiang Lisa Li and Jason Eisner. 2019. Specializing word embeddings (for parsing) by information bottleneck. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2744–2754, Hong Kong, China. 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):313–330.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR Workshops Track*.

Khalil Mrini, Franck Dernoncourt, Quan Hung Tran, Trung Bui, Walter Chang, and Ndapa Nakashole. 2020. Rethinking self-attention: Towards interpretability in neural parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 731–742, Online. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Rico Sennrich and Barry Haddow. 2016. Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 1, Research Papers*, pages 83–91, Berlin, Germany. Association for Computational Linguistics.

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, Vancouver, Canada. Association for Computational Linguistics.

Yuanhe Tian, Yan Song, and Fei Xia. 2020. Supertagging Combinatory Categorial Grammar with attentive graph convolutional networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6037–6044, Online. Association for Computational Linguistics.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. In *NIPS*.

Kaiyu Yang and Jia Deng. 2020. Strongly incremental constituency parsing with graph neural networks. In *NeurIPS*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

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, Sofia, Bulgaria. Association for Computational Linguistics.