Beam Tree Recursive Cells

Anonymous [1]

Abstract

We propose Beam Tree Recursive Cell (or BT-Cell) - a backpropagation-friendly framework to extend Recursive Neural Networks (RvNNs) with beam search for latent structure induction. We further extend this framework by proposing a relaxation of the hard top-k operators in beam search for better propagation of gradient signals. We evaluate our proposed models in different out-of-distribution splits in both synthetic and realistic data. Our experiments show that BT-Cell achieves near-perfect performance on several challenging structure-sensitive synthetic tasks like ListOps and logical inference while maintaining comparable performance in realistic data against other RvNN-based models. Additionally, we identify a previously unknown failure case for neural models in generalization to unseen number of arguments in ListOps. The code is in the supplementary and will be made available online.

1. Introduction

In the space of sequence encoders, Recursive Neural Networks (RvNNs) can be said to lie somewhere in-between Recurrent Neural Networks (RNNs) and Transformers in terms of flexibility. While vanilla Transformers show phenomenal performance and efficient scalability on a variety of tasks, they can often struggle in length generalization and systematicity in syntax-sensitive tasks (Tran et al., 2018; Shen et al., 2019a; Lakretz et al., 2021; Csordás et al., 2022). RvNN-based models, on the other hand, can often excel on some of the latter kind of tasks (Shen et al., 2019a; Chowdhury & Caragea, 2021; Liu et al., 2021; Bogin et al., 2021) making them worthy of further study although they may suffer from limited scalability in their current formulations.

Given an input text, RvNNs (Pollack, 1990; Socher et al., 2010) are designed to build up the representation of the whole text by recursively building up the representations of their constituents starting from the most elementary representations (tokens) in a bottom-up fashion. As such, RvNNs can model the hierarchical part-whole structures underlying texts. However, originally RvNNs required access to pre-defined hierarchical constituency-tree structures. Several works (Socher et al., 2011; Havrylov et al., 2019; Choi et al., 2018; Maillard et al., 2019; Chowdhury & Caragea, 2021) introduced latent-tree RvNNs that sought to move beyond this limitation by making RvNNs able to learn to automatically determine the structure of composition from any arbitrary downstream task objective.

Among these approaches, Gumbel-Tree models (Choi et al., 2018) are particularly attractive for their simplicity. However, they not only suffer from biased gradients (due to the use of Straight-Through Estimation (STE)), but they also perform poorly on synthetic tasks like ListOps (Nangia & Bowman, 2018) that were specifically designed to diagnose the capacity of neural models for automatically inducing underlying hierarchical structures. To tackle these issues, we propose the Beam Tree Cell (BT-Cell) framework that incorporates beam-search on RvNNs replacing the Gumbel Softmax in Gumbel-Tree models. Instead of greedily selecting the highest scored sub-tree representations like Gumbel-Tree models, BT-Cell chooses and maintains top-k highest scored sub-tree representations. We show that BT-Cell outperforms Gumbel-Tree models in challenging structure sensitive tasks by several folds. For example, in ListOps, when testing for samples of length 900-1000, BT-Cell increases the performance of a comparable Gumbel-Tree model from 37.9% to 86.7% (see: Table 1). We further extend BT-Cell by replacing its non-differentiable top-k operators with a novel operator called OneSoft Top-K. Our proposed operator, combined with BT-Cell, achieves a new state-of-the-art in length generalization and depth-generalization in structure-sensitive synthetic tasks like ListOps and performs comparably in realistic data against other strong models.

A few recently proposed latent-tree models simulating RvNNs including LSTM-RL (reinforcement learning) (Havrylov et al., 2019), Ordered Memory (OM) (Shen et al., 2019a) and Continuous RvNNs (CRvNNs) (Chowdhury & Caragea, 2021) are also strong contenders to BT-Cell on synthetic data. However, unlike BT-Cell, LSTM-RL relies on expensive reinforcement learning and several sophisti-

[*]Equal contribution [1]Anonymous Affiliation. Correspondence to: Anonymous <Anonymous>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

cated techniques to stabilize training. Moreover, compared to OM and CRvNN, one distinct advantage of BT-Cell is that it not just provides the final sequence encoding (representing the whole input text) but also the intermediate constituent representations at different levels of hierarchy (representations of all nodes of the underlying induced trees). Such tree-structured node representations can be useful as inputs to further downstream modules like a Transformer (Vaswani et al., 2017) or Graph Neural Network (Scarselli et al., 2009) in a full end-to-end setting.[1] While CYK-based RvNNs (Maillard et al., 2019) are also promising and similarly can provide multiple span representations they tend to be much more expensive than BT-Cell (see §5.3).

As a further contribution, we also identify a previously unknown failure case for even the best performing neural models when it comes to argument generalization in ListOps (Nangia & Bowman, 2018)—opening up a new challenge for future research.

2. Preliminaries

Problem Formulation: Similar to Choi et al. (2018), throughout this paper, we explore the use of RvNNs as a sentence encoder. Formally, given a sequence of token embeddings $\mathcal{X} = (e_1, e_2, \ldots, e_n)$ (where $\mathcal{X} \in \mathbb{R}^{n \times d_e}, e_i \in \mathbb{R}^{d_e}$, and d_e is the embedding size), the task of a sentence encoding function $\mathcal{E} : \mathbb{R}^{n \times d_e} \rightarrow \mathbb{R}^{d_h}$ is to encode the whole sequence of vectors into a single vector $o = \mathcal{E}(\mathcal{X})$ (where $o \in \mathbb{R}^{d_h}$ and d_h is the size of the encoded vector). We can use a sentence encoder for sentence-pair comparison tasks like logical inference or for text classification.

2.1. RNNs and RvNNs

A core component of both RNNs and RvNNs is a recursive cell R. In our contexts, R takes as arguments two vectors ($a_1 \in \mathbb{R}^{d_{a_1}}$ and $a_2 \in \mathbb{R}^{d_{a_2}}$) and returns a single vector $v = R(a_1, a_2)$. $R : \mathbb{R}^{d_{a_1}} \times \mathbb{R}^{d_{a_2}} \rightarrow \mathbb{R}^{d_v}$. In our settings, we generally set $d_{a_1} = d_{a_2} = d_v = d_h$. Given a sequence $\mathcal{X}$, both RNNs and RvNNs sequentially process it through a recursive application of the cell function. For a concrete example, consider a sequence of token embeddings such as $(2 + 4 \times 4 + 3)$ (assume the symbols 2, 4, + etc. represent transformations of corresponding embedding vectors $\in \mathbb{R}^{d_h}$). Given any such sequence, RNNs can only follow a fixed left-to-right order of composition. For the particular aforementioned sequence, an RNN-like application of the cell function can be expressed as:

$$o = R(R(R(R(R(R(h0, 2), +), 4), \times), 4), +), 3) \quad (1)$$

Here $h0$ is some input-independent initial state ("initial hidden state") set in the model. In contrast to RNNs, RvNNs can compose the sequence in more flexible orders. For example, one way (among many) that RvNNs could apply the cell function is as follows:

$$o = R(R(R(R(2, +), R(R(4, \times), 4)), +), 3) \quad (2)$$

Thus, RvNNs can be considered as a generalization of RNNs where a strict left-to-right order of composition is not anymore enforced. As we can see, by these strategies of recursively reducing two vectors into a single vector, both RNNs and RvNNs can implement the sentence encoding function in the form of $\mathcal{E}$. Moreover, the form of application of cell function exhibited by RNNs and RvNNs can also be said to reflect a tree-structure. For any application of the cell function in the form $v = R(a_1, a_2)$, v can be treated as the representation of the immediate parent node of child nodes a_1 and a_2 in an underlying tree.

In Eqn. 2, we find that RvNNs can align the order of composition to PEMDAS whereas RNNs cannot. Nevertheless, RNNs can still learn to simulate RvNNs by modeling tree-structures implicitly in their hidden state dimensions (Bowman et al., 2015b). For example, RNNs can learn to hold off the information related to "2+" until "4×4" is processed. Their abilities to handle tree-structures is analogous to how we can use pushdown automation in a recurrent manner through an infinite stack to detect tree-structured grammar. Still, RNNs can struggle to effectively learn to appropriately organize information in practice for large sequences. Special inductive biases can be incorporated to enhance their abilities to handle their internal memory structures (Shen et al., 2019b;a). However, even then, memories remain bounded in practice and there is a limit to what depth of nested structures they can model.

More direct approaches to RvNNs, in contrast, can alleviate the above problems and mitigate the need of sophisticated memory operations to arrange information corresponding to a tree-structure because they can directly compose according to the underlying structure (Eqn. 2). However, in the case of RvNNs, we have the problem of first determining the underlying structure to even start composition. One approach to handle the issue can be to train a separate parser to induce a tree structure from sequences using gold tree parses. Then we can use the trained parser in RvNNs. However, this is not ideal. Not all tasks or languages would come with gold trees for training a parser and a parser trained in one domain may not translate well to another. A potentially better approach is to jointly learn both the cell function and structure induction from a downstream objective. We focus

[1] There are several works that have used intermediate span representations for better compositional generalization in generalization tasks (Liu et al., 2020; Herzig & Berant, 2021; Bogin et al., 2021; Liu et al., 2021; Mao et al., 2021). We keep it as a future task to explore whether the span representations returned by BT-Cell can be used in relevant ways.

on this latter approach. Below we discuss one framework for this approach.

2.2. Easy-First Parsing and Gumbel Tree Models

Here we describe an adaptation (Choi et al., 2018) of easy-first parsing (Goldberg & Elhadad, 2010) for RvNN-based sentence-encoding. The algorithm relies on a scorer function $score : \mathbb{R}^{d_h} \rightarrow \mathbb{R}^1$ that scores parsing decisions. Particularly, if we have $v = R(a_1, a_2)$, then $score(v)$ represents the plausibility of a_1 and a_2 belonging to the same immediate parent constituent. Similar to (Choi et al., 2018), we keep the scorer as a simple linear transformation: $score(v) = W_v v$ (where $W_v \in \mathbb{R}^{1 \times d_h}$ and $v \in \mathbb{R}^{d_h}$).

Recursive Loop: In this algorithm, at every iteration in a recursive loop, given a sequence of hidden states $(h_1, h_2, \ldots, h_n)$ we consider all possible immediate candidate parent compositions taking the current states as children: $(R(h_1, h_2), R(h_2, h_3), \ldots, R(h_{n-1}, h_n))$.[2] We then score each of the candidates with the score function and greedily select the highest scoring candidate (i.e. we commit to the "easiest" decision first). For the sake of illustration, assume $score(R(h_i, h_{i+1})) \geq score(R(h_j, h_{j+1})) \; \forall j \in \{1, 2, \ldots, n\}$. Thus, following the algorithm the parent candidate $R(h_i, h_{i+1})$ can be chosen. The parent representation $R(h_i, h_{i+1})$ would then replace its immediate children h_i and h_{i+1}. Thus, the resulting sequence will become: $(h_1, \ldots, h_{i-1}, R(h_i, h_{i+1}), h_{i+2}, \ldots, h_n)$. Like this, the sequence will be iteratively reduced to a single element representing the final sentence encoding. The full algorithm is presented in the Appendix (see Algorithm 1).

One issue here is to decide how to choose the highest scoring candidate. One way to do it is to simply use an argmax operator but it will not be differentiable. Gumbel-Tree models (Choi et al., 2018) address this by using Straight Through Estimation (STE) (Bengio et al., 2013) with Gumbel Softmax (Jang et al., 2017; Maddison et al., 2017) instead of argmax. However, STE is known to cause high bias in gradient estimation. Moreover, as it was previously discovered (Nangia & Bowman, 2018), and as we independently verify, STE Gumbel-based strategies perform poorly when tested in structure-sensitive tasks. Instead, to overcome these issues, we propose an alternative of extending argmax with a top-k operator under a beam search strategy.

3. Beam Tree Cell

Motivation: Gumbel-Tree models, as described, are relatively fast and simple but they are fundamentally based on a greedy algorithm for a task where the greedy solution is not guaranteed to be optimal. On the other hand, adaptation of dynamic programming-based CYK-models (Maillard et al., 2019) leads to high computational complexity (see §5.3). A "middle way" between the two extremes is then to simply extend Gumbel-Tree models with beam-search to make it less greedy while still being less costly than CYK-parsers. Moreover, using beam-search also provides additional opportunity to recover from local errors whereas a greedy single-path approach (like Gumbel Tree models) will be stuck with any errors made. All these factors motivate the framework of Beam Tree Cells (BT-Cell).

Implementation: The beam search extension to Gumbel-Tree models is straight-forward and similar to standard beam search. The method is described more precisely in Appendix A.1 and Algorithm 2. In summary, in BT-Cell, given a beam size k, we maintain a maximum of k hypotheses (or beams) at each recursion. In any given iteration, each beam constitutes a sequence of hidden states representing a particular path of composition and an associated score for that beam based on the addition of log-softmaxed outputs of the $score$ function (as defined in §2.2) over each chosen composition for that sequence. At the end of the recursion, we will have k sentence encodings ($(o_1, o_2, \ldots, o_k)$ where $o_i \in \mathbb{R}^{d_h}$) and their corresponding scores ($(s_1, s_2, \ldots, s_k)$ where $s_i \in \mathbb{R}^1$). The final sequence encoding can be then represented as: $\sum_{i=1}^{k} \left(\frac{exp(s_i) \cdot o_i}{\sum_{i=1}^{k} exp(s_i)} \right)$. This aims at computing the expectation over the k sequence encodings.

3.1. Top k Variants

As in standard beam search, BT-Cell requires two top-k operators. The first top-k replaces the straight-through Gumbel Softmax (simulating top-1) in Gumbel-Tree models. However, selecting and maintaining k possible choices for every beam in every iteration leads to an exponential increase in the number of total beams. Thus, a second top-k operator is used for pruning the beams to maintain only a maximum of k beams at the end of each iteration. Here, we focus on variations of the second top-k operator that is involved in truncating beams:

Plain Top-k: The simplest variant is just the vanilla top-k operator. However, the vanilla top-k operator is discrete and non-differentiable preventing gradient propagation to non-selected paths. Despite that, this can still work for the following reasons: (1) gradients can still pass through the final top k beams and scores. The scorer function can thus learn to increase the scores of better beams and lower the scores of the worse ones among the final k beams; (2) a rich enough cell function can be robust to local errors in the structure and learn to adjust for it by organizing information better in its hidden states. We believe that as a combination of these two factors, plain BT-Cell even with non-differentiable top-k operators can learn to perform well

[2] We focus only on the class of binary projective tree structures. Thus all the candidates are compositions of two contiguous elements.

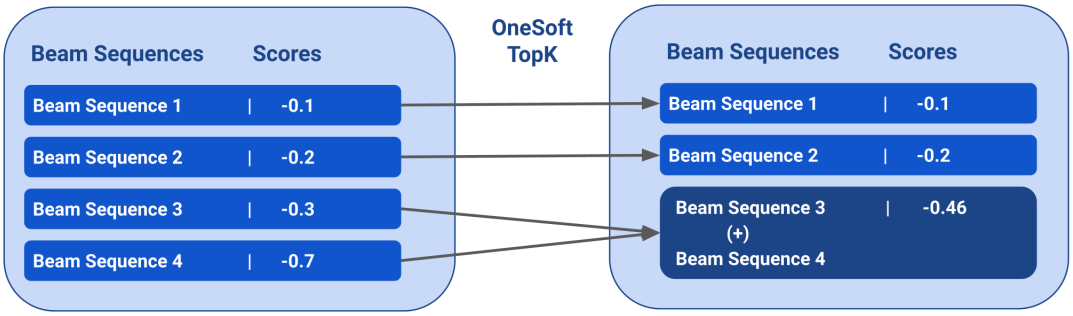

Figure 1. Visualization of Top-k OneSoft selection from $m = 4$ beams to top $k = 3$ beams. (+) represents interpolation.

for structure-sensitive tasks (as we will empirically observe).

OneSoft Top-k: While non-differentiable top-k operators can work, they still can be a bottleneck because gradient signals will be received only for k beams in a space of exponential possibilities. To address this issue, we consider if we can make the truncation or deletion of beams "softer". To that end, we develop a new Top-k operator that we call OneSoft Top-k. We motivate and describe it below.

As a concrete case, assume we have m beams (sequences and their corresponding scores). The target for a top-k operator is to keep only the top scoring k beams (where $k \leq m$). Ideally we want to keep the beam representations "sharp" and avoid washed out representations owing to interpolation (weighted vector averaging) of distinct paths (Drozdov et al., 2020). This can be achieved by plain top-k. However, it prevents propagation of gradient signals through the bottom $m - k$ beams. Another line of approach is to create a soft permutation matrix $P \in \mathbb{R}^{m \times m}$ through a differentiable sorting algorithm such that P_{ij} represents the probability of the i^{th} beam being the j^{th} highest scoring beam. P can then be used to softly select the top k beams. However, running differentiable sorting in a recursive loop can significantly increase computation overheads and also create more "washed out" representations leading to higher error accumulation (also see §5.1 and §5.3). We tackle all these challenges by instead proposing OneSoft as a simple hybrid strategy to approach top-k selection. We provide a formal description of our proposed strategy below and a visualization of the process in Figure 1.

Assume we have m beams consisting of m sequences: $H = (\mathcal{H}_1, \ldots, \mathcal{H}_m)$ ($\mathcal{H}_i \in \mathbb{R}^{n \times d_h}$ where n is the sequence length) and m corresponding scalar scores: $S = (s_1, \ldots, s_m)$. First, we simply use the plain top-k operator to discretely select the top $k - 1$ beams (instead of k). This allows us to keep the most promising beams "sharp":

$$idx = topk(S, K = k - 1), \quad Top = \{(\mathcal{H}_i, s_i) \mid i \in idx\} \tag{3}$$

Second, for the k^{th} beam we instead perform a softmax-based marginalization of the bottom $m-(k-1)$ beams. This allows us to still propagate gradients through the bottom scoring beams (unlike in the pure plain top-k operator):

$$B = \{(\mathcal{H}_i, s_i) \mid (i \notin idx) \wedge (i \in \{1, 2, \ldots, m\})\} \tag{4}$$

$$Z = \sum_{(\mathcal{H},s) \in B} \exp(s) \tag{5}$$

$$SP = \left(\sum_{(\mathcal{H},s) \in B} \left(\frac{\exp(s)}{Z} \cdot \mathcal{H} \right), \sum_{(\mathcal{H},s) \in B} \left(\frac{exp(s)}{Z} \cdot s \right) \right) \tag{6}$$

Here B represents the bottom scoring $m - (k - 1)$ beams and SP represents the softmax-based marginalization. Finally, we add the SP to the top $k - 1$ discretely selected beams to get the final set of k beams: $Top \cup \{SP\}$. Thus, we get to achieve a "middle way" between plain top-k and differentiable sorting: partially getting the benefit of sharp representations of the former through discrete top $k - 1$ selection, and partially getting the benefit of gradient propagation of the latter through soft-selection of the k^{th} beam. In practice, we switch to plain top-k during inference. This makes tree extraction convenient during inference if needed.

4. Experiments and Results

We present the main models below. Hyperparameters and other architectural details are in Appendix F.

1. RecurrentGRC: RecurrentGRC is an RNN implemented with the Gated Recursive Cell (GRC) (Shen et al., 2019a) as the cell function (see Appendix B for description of GRC). **2. GoldTreeGRC:** GoldTreeGRC is a GRC-based RvNN with gold tree structures. **3. GumbelTreeGRC:** This is the same as GumbelTreeLSTM (Choi et al., 2018) but with GRC instead of LSTM. **4. CYK-GRC:** This is the CYK-based model proposed by Maillard et al. (2019) but with GRC. **5. Ordered Memory (OM):** This is a memory-

Model	near-IID	Length Gen.			Argument Gen.		LRA
(Lengths)	≤ 1000	200-300	500-600	900-1000	100-1000	100-1000	2000
(Arguments)	≤ 5	≤ 5	≤ 5	≤ 5	10	15	10
With gold trees							
GoldTreeGRC	99.95	99.88	99.85	100	80.5	79	78.1
Baselines without gold trees							
RecurrentGRC	84.05	33.85	20.2	15.1	37.35	30.10	20.7
GumbelTreeGRC	74.89	47.6	43.85	37.9	51.35	50.5	46.1
CYK-GRC	97.87	93.75	—	—	60.75	42.45	—
Ordered Memory	<u>99.88</u>	**99.55**	92.7	76.9	**84.15**	**75.05**	**80.1**
CRvNN†	99.6_3	98.51_{11}	97.95_{11}	96.78_{19}	—	—	—
CRvNN	98.86	95.89	93.15	89.4	57.8	24.35	45.1
Ours							
BT-GRC	99.39	96.15	92.55	86.7	<u>77.1</u>	63.7	67.3
BT-GRC + OneSoft	**99.92**	<u>99.5</u>	**99**	**97.2**	76.05	<u>67.9</u>	<u>71.8</u>

Table 1. Accuracy on ListOps. † indicates results from (Chowdhury & Caragea, 2021). For our models we report the median of 3 runs. Our models were trained on lengths ≤ 100, depth ≤ 20, and arguments ≤ 5. We bold the best results and underline the second-best among models that do not use gold trees. Subscript represents standard deviation. As an example, $90_1 = 90 \pm 0.1$

.

augmented RNN simulating certain classes of RvNN functions as proposed by (Shen et al., 2019a). OM also uses GRC. **6. CRvNN:** CRvNN is a variant of RvNN with a continuous relaxation over its structural operations as proposed by (Chowdhury & Caragea, 2021). CRvNN also uses GRC. **7. BT-GRC:** BT-Cell with GRC cell and plain top-k. **8. BT-GRC + OneSoft:** BT-GRC with OneSoft top-k.

For experiments with BT-Cell models, we set beam size as 5 as a practical choice (neither too big nor too small).

4.1. ListOps Length Generalization Results

Dataset Settings: ListOps (Nangia & Bowman, 2018) is a challenging synthetic task that requires solving of nested mathematical operations over lists of arguments. We present our results on ListOps in Table 1. To test for length-generalization performance, we train the models only on sequences with ≤ 100 lengths (we filter the rest) and test on splits of much larger lengths (eg. $200-300$ or $900-1000$) taken from (Havrylov et al., 2019). "Near-IID" is the original test set of ListOps (it is "near" IID and not fully IID because a percentage of the split has > 100 length sequences whereas such lengths are absent in the training split).

Results: RecurrentGRC: As discussed before in §2.1, RNNs have to model tree structures implicitly in their bounded hidden states and thus can struggle generalizing to unseen structural depths. This is reflected in the sharp degradation in its length generalization performance. **GumbelTreeGRCs:** Consistent with prior work (Nangia & Bowman, 2018), Gumbel-Tree models fail to perform well in this task; likely, due to their biased gradient estimation. **CYK-GRC:** CYK-GRC shows some promise to length generalization but it was too slow to run in higher lengths. **Ordered Memory (OM):** Here, we find that OM struggles to generalize to higher unseen lengths. OM's reliance on soft sequential updates in a nested loop can lead to higher error accumulation over larger unseen lengths or depths. **CRvNN:** Consistent with Chowdhury & Caragea (2021), CRvNN performs relatively well at higher lengths. **BT-GRC:** Here, we find a massive boost over Gumbel-tree baselines even when using the base model. In the 900-1000 length generalization split, BT-GRC increases the performance of GumbelTree-GRC from 37.9% to 86.7% - by incorporating beam search with plain top-k. **BT-GRC+OneSoft:** As discussed in §3.1, BT-GRC+OneSoft counteracts the bottleneck of gradient propagation being limited through only k beams and achieves near perfect length generalization as we can observe from Table 1.

4.2. ListOps Argument Generalization Results

Dataset Settings: While length generalization (Havrylov et al., 2019; Chowdhury & Caragea, 2021) and depth generalization (Csordás et al., 2022) have been tested before for ListOps, the performance on argument generalization is yet to be considered. In this paper, we ask what would happen if we increase the number of arguments in the test set beyond the maximum number encountered during training. The training set of the original ListOps data only has ≤ 5 arguments for each operator. To test for argument generalization we created two new splits - one with 10 arguments per operator and another with 15 arguments per operator. In addition, we also consider the test set of ListOps from Long Range Arena (LRA) dataset (Tay et al., 2021) which serves as a check for both length generalization (it has sequences of length 2000) and argument generalization (it has

Model	SST5	IMDB		MNLI					
	IID	Con. OOD	Count. OOD	M	MM	Len M	Len MM	Neg M	Neg MM
RecurrentGRC	$52.19_{1.5}$	74.86_{28}	82.72_{19}	71.2_{3}	71.4_{4}	49_{25}	49.5_{24}	49.3_{6}	50.1_{6}
GumbelTreeGRC	$51.67_{8.8}$	70.63_{21}	81.97_{5}	71.2_{7}	71.2_{6}	57.5_{17}	59.6_{12}	50.5_{20}	51.8_{20}
Ordered Memory	$\underline{52.30_{2.7}}$	$\underline{76.98_{5.8}}$	$83.68_{7.8}$	$\mathbf{72.5_{3}}$	$\mathbf{73_{2}}$	56.5_{33}	57.1_{31}	50.9_{7}	51.7_{13}
CRvNN	51.75_{11}	$\mathbf{77.80_{15}}$	$\underline{85.38_{3.5}}$	72.2_{4}	72.6_{5}	62_{44}	63.3_{47}	52.8_{6}	53.8_{4}
Ours									
BT-GRC	$\mathbf{52.32_{4.7}}$	75.07_{29}	82.86_{23}	71.6_{2}	72.3_{1}	64.7_{6}	66.4_{5}	$\mathbf{53.7_{37}}$	$\mathbf{54.8_{43}}$
BT-GRC + OneSoft	$51.92_{7.2}$	75.68_{21}	84.77_{11}	71.7_{1}	71.9_{2}	$\mathbf{65.6_{13}}$	$\mathbf{66.7_{9}}$	53.2_{2}	54.2_{5}

Table 2. Mean accuracy and standard deviaton on SST5, IMDB, and MNLI. Con. represents Contrast set and Count. represents Countefactuals. Our models were run 3 times on different seeds. Subscript represents standard deviation. As an example, $90_1 = 90 \pm 0.1$
.

10 arguments) simultaneously[3]. The results are in Table 1.

Results: Interestingly, we find that all the models do not perform well ($< 90\%$) on argument generalization. However, with the exception of Ordered Memory (OM), BT-Cell models perform much better than any other models (including otherwise strong contenders like CRvNN). Interestingly, OM performs quite well in this split. Some potential reasons could be the following; First, we know OM's performance is not simply due to better parsing because it even surpasses GoldTreeGRC at times. Second, we also know OM's performance is not just due to a better recursive cell, since its cell (GRC) is shared by many other models that do not perform as well. This may suggest that the memory-augmented RNN style setup in OM is more amenable for argument generalization. Note that Transformer-based architectures tend to get $\leq 40\%$ on LRA test set for ListOps (Tay et al., 2021) despite training on in-distribution data whereas we can still generalize to a performance ranging in between 70-80$\%$ by using RvNN-based models trained on data with much limited sequence lengths and fewer arguments.

4.3. Semantic Analysis (SST and IMDB) Results

Dataset Settings: SST5 (Socher et al., 2013) and IMDB (Maas et al., 2011) are natural language classification datasets (for sentiment classification). For IMDB, to focus on OOD performance, we also test our models on the contrast set (Con.) from (Gardner et al., 2020) and the counterfactual test set (Count.) from (Kaushik et al., 2020). We present our results on these datasets in Table 2.

Results: The results in these natural language tasks are rather mixed. There are, however, some interesting highlights. CRvNN and OM do particularly well in the OOD splits (contrast set and counterfactual split) of IMDB, correlating with their better OOD generalization in synthetic data. BT-GRC + OneSoft remains competitive in those splits with OM and CRvNN and is better than any other models besides CRvNN and OM. STE Gumbel-based models tend to perform particularly worse on IMDB.

[3]Note that the LRA test set is in-domain for the LRA training set and thus, does not test for argument generalization originally

4.4. Natural Language Inference Experiments

Dataset Settings: We ran our models on MNLI (Williams et al., 2018) which is a natural language inference task. We tested our models on the development set of MNLI and used a randomly sampled subset of $10,000$ data points from the original training set as the development set. Our training setup is different from (Chowdhury & Caragea, 2021) and other prior latent tree models which combine SNLI (Bowman et al., 2015a) and MNLI training sets (we do not add SNLI data). We filter sequences ≥ 150 from the training set for efficiency. We also test our models in various stress tests (Naik et al., 2018). We report the results in Table 2. In the table, M denotes matched development set (used as test set) of MNLI. MM denotes mismatched development set (used as test set) of MNLI. LenM denotes length matched stress set from (Naik et al., 2018). LenMM denotes length mismatched stress set from (Naik et al., 2018). NegM denotes negation matched stress set from (Naik et al., 2018). NegMM denotes negation mismatched stress set from (Naik et al., 2018). Len M/Len MM stress sets add to the length of the premise by adding tautologies. Neg M/Neg MM stress sets add tautologies containing "not" terms which can bias the model to falsely predict contradictions.

Results: The results in Table 2 show that BT-GRC models perform comparably with the other models in the standard matched/mismatched sets (M and MM). However, they outperform all the other models on length matched/mismatched stress test (Len M and Len MM). Also, BT-GRC models tend to do better than the other models in negation matched/mismatched test sets (Neg M and Neg MM). Overall BT-Cell shows better robustness to stress tests.

5. Analysis

5.1. Analysis of Neighbor Models

We also analyze some other models that are similar to BT-GRC in Table 3 as a form of ablation and show that BT-GRC

Model	**near-IID**	**Length Gen.**			**Argument Gen.**		**LRA**
(Lengths)	≤ 1000	200-300	500-600	900-1000	100-1000	100-1000	2000
(Arguments)	≤ 5	≤ 5	≤ 5	≤ 5	10	15	10
Alternative models in the Vicinity of BT-GRC							
BT-LSTM	94.1	85.1	83.5	78.8	67.9	44.3	57.9
BSRP-GRC	70.3	42.4	33.2	26.3	40.2	35.8	29.7
MC-GumbelTreeGRC	89.3	36.8	28.2	25.1	39.5	34	30.1
BT-GRC+SOFT	69	44	37.1	29.4	39.5	38.6	31.6
Robustness of OneSoft Top-K to lower Beam Size (size 2)							
BT-GRC	94.18	68.2	56.85	50.2	64.45	56.95	55.85
BT-GRC + OneSoft	99.69	97.55	95.40	91	75.75	62	66.1

Table 3. Accuracy of different models on ListOps (same setting as in Table 1). We report the median of 3 runs.

is still superior to them. We describe these models below and discuss their performance on ListOps:

BT-LSTM: This is just BT-Cell with an LSTM cell (Hochreiter & Schmidhuber, 1997) instead of GRC. In Table 3, We find that BT-LSTM can still perform moderately well (showing the robustness of BT-Cell as a framework) but worse than what it can do with GRC. This is consistent with prior works showing superiority of GRC as a cell function (Shen et al., 2019a; Chowdhury & Caragea, 2021)

BSRP-GRC: This is an implementation of Beam Shift Reduce Parser (Maillard & Clark, 2018) with a GRC cell. Similar to us, this approach applies beam search but to a shift-reduce parsing model as elaborated in Appendix C. Surprisingly, despite using a similar framework to BT-Cell, BSRPC-GRC performs quite poorly in Table 3. We suspect this is because of the limited gradient signals from its top-k operators coupled with the high recurrent depth for backpropagation (twice the sequence length) encountered in BSRP-GRC compared to that in BT-Cell (the recurrent depth is the tree depth). Moreover, BSRP-GRC, unlike BT-Cell, also lacks the global competition among all parent compositions when making shift/reduce choices.

MC-GambelTreeGRC: Here we propose a Monte Carlo approach towards Gumbel Tree GRC. This model runs k gumbel-tree models with shared parameters in parallel. Since the models are stochastic, they can sample different latent structures. In the end we can average the final k sentence encodings treating this as a Monte-Carlo approximation. We set $k = 5$ to be comparable with BT-Cell. MC-GumbelTreeGRC is similar to BT-Cell because it can model different structural interpretations. However, it fails to do as effectively as BT-Cell in ListOps. We suspect this is because beam-search based structure selection allows more competition between structure candidates when using top-k for truncation and thus enables better structure induction.

BT-GRC+SOFT: This model incorporates another potential alternative to OneSoft within BT-GRC. It uses a differentiable sorting algorithm, SOFT Top-k, that was previously used in beam search for language generation (Xie et al., 2020), to implement the top-k operator replacing OneSoft. However, it performs poorly. Its poor performance supports our prior conjecture (§3.1) that using a soft permutation matrix in all recursive iterations is not ideal because of increased chances of error accumulation and more "washing out" through weighted averaging of distinct beams.

5.2. OneSoft Top-k with Lower Beam

We motivated (§3.1) the proposal of OneSoft top-k to specifically counteract the bottleneck of gradient propagation being limited through only k beams in the base BT-Cell model (with plain top-k). While we validate this bottleneck through our experiments in Table 1 for beam size 5, the bottleneck should be even worse when k (beam size) is low (e.g., 2). Based on our motivation, OneSoft should perform much better than plain top-k when beam size is low. We perform experiments with beam size 2 on ListOps to understand if that is true and show the results in Table 3. As we can see, OneSoft indeed performs much better than plain top-k with lower beam size of 2 where BT-GRC gets only 50.2% in the 900-1000 split of ListOps, and BT-GRC+OneSoft gets 91%. As we would expect beam size 5 (from Table 1) still outperforms beam size 2 in a comparable setting.

5.3. Efficiency Analysis

Settings: In Table 4, we compare the empirical performance of various models in terms of time and memory. We train each model on ListOps splits of different sequence lengths (200-250, 500-600, and 900-1000). Each split contains 100 samples. Batch size is set as 1. Other hyperparameters are same as those used for ListOps. For CRvNN, we show the worst case performance (without early halt).

Discussion: RecurrentGRC and GumbelTreeGRC can be relatively efficient in terms of both runtime and memory consumption. BSRP-GRC and OM, being recurrent models, can be highly efficient in terms of memory but their com-

Model	Sequence Lengths					
	$200-250$		$500-600$		$900-1000$	
	Time	Memory	Time	Memory	Time	Memory
RecurrentGRC	0.2 min	0.02 GB	0.5 min	0.02 GB	1.3 min	0.03 GB
GumbelTreeGRC	0.5 min	0.35 GB	2.1 min	1.95 GB	3.5 min	5.45 GB
CYK-GRC	9.3 min	32.4 GB	OOM	OOM	OOM	OOM
BSRP-GRC	2.3 min	0.06 GB	6.1 min	0.19 GB	10.5 min	0.42 GB
Ordered Memory	8.0 min	0.09 GB	20.6 min	0.21 GB	38.2 min	0.35 GB
CRvNN	1.5 min	1.57 GB	4.3 min	12.2 GB	8.0 min	42.79 GB
MC-GumbelTreeGRC	1.1 min	1.71 GB	2.4 min	9.85 GB	4.3 min	27.33 GB
BT-GRC	1.1 min	1.71 GB	2.6 min	9.82 GB	5.1 min	27.27 GB
BT-GRC + OneSoft	1.4 min	2.74 GB	4.0 min	15.5 GB	7.1 min	42.95 GB
BT-GRC + SOFT	5.1 min	2.67 GB	12.6 min	15.4 GB	23.1 min	42.78 GB

Table 4. Empirical time and memory consumption for various models on an RTX A6000. Ran on 100 ListOps data with batch size 1

.

plex recurrent operations make them slow. CYK-GRC is the worst in terms of efficiency because of its expensive chart-based operation. CRvNN is faster than OM/BSRP-GRC but its memory consumption can scale worse than BT-GRC because of Transformer-like attention matrices for neighbor retrieval. MC-GumbelTreeGRC is similar to a batched version of GumbelTreeGRC. BT-GRC performs similarly to MC-GumbelTreeGRC showing that the cost of BT-GRC is similar to increasing batch size of GumbelTreeGRC. BT-GRC + OneSoft perform similarly to CRvNN. BT-GRC + SOFT is much slower due to using a more expensive optimal transport based differentiable sorting mechanism (SOFT top-k) in every iteration. This shows another advantage of using OneSoft over other more sophisticated alternatives.

5.4. Additional Analysis and Experiments

Heuristics Tree Models: We analyze heuristics-based tree models (Random tree, balanced tree) in Appendix D.4.

Synthetic Logical Inference: We present our results on a challenging synthetic logical inference task (Bowman et al., 2015b) in Appendix D.3. We find that most variants of BT-Cell can perform on par with prior SOTA models.

Depth Generalization: We also run experiments to test depth-generalization performance on ListOps (Appendix D.1). We find that BT-Cell can easily generalize to much higher depths and it does so more stably than OM.

Transformers: We experiment briefly with Neural Data Routers (Csordás et al., 2022) which is a Transformer-based model proven to do well in tasks like ListOps. However, we find that Neural Data Routers (NDRs), despite their careful inductive biases, still struggle with sample efficiency and length generalization compared to strong RvNN-based models. We discuss more in Appendix D.2.

Parse Tree Analysis: We analyze parsed trees and score distributions in Appendix D.5.

6. Related Works

Goldberg & Elhadad (2010) proposed the easy-first algorithm for dependency parsing. Ma et al. (2013) extended it with beam search for parsing tasks. Choi et al. (2018) integrated easy-first-parsing with an RvNN. Similar to us, Maillard & Clark (2018) used beam search to extend shift-reduce parsing whereas Drozdov et al. (2020) used beam search to extend CYK-based algorithms. However, BT-Cell-based models achieve higher accuracy than the former style of models (e.g., BSRP-GRC) and are computationally more efficient than the latter style of models (e.g., CYK-GRC). Similar to us, Collobert et al. (2019) also use beam search in an end-to-end fashion during training but in the context of sequence generation. However, none of the above approaches explored beyond hard top-k operators in beam search. One exception is Xie et al. (2020) where a differentiable top-k operator is used in beam search for language generation but as we show in §5.1 it does not work as well. We provide an extended related work survey in Appendix E.

7. Conclusion

We present BT-Cell as an intuitive way to extend RvNN that is nevertheless highly competitive with more sophisticated models like Ordered Memory (OM) and CRvNN. Infact, BT-Cell is the only model that gets moderate performance in argument generalization while also excelling in length generalization in ListOps. It also shows more robustness in MNLI, and overall it is much faster than OM or CYK-GRC. The ideal future direction would be to focus on argument generalization and systematicity while maintaining computational efficiency. We also aim for added flexibility for handling more relaxed structures like non-projective trees or directed acyclic graphs as well as richer classes of languages (DuSell & Chiang, 2022; Del'etang et al., 2022). Another direction to explore would be to linearize the recursion (Gu et al., 2022) to make these models more scalable.

References

Adams, R. P. and Zemel, R. S. Fast differentiable sorting and ranking. In *ArXiv*, 2011. URL https://arxiv.org/abs/1106.1925.

Bengio, Y., Léonard, N., and Courville, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Blondel, M., Teboul, O., Berthet, Q., and Djolonga, J. Fast differentiable sorting and ranking. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 950–959. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/blondel20a.html.

Bogin, B., Subramanian, S., Gardner, M., and Berant, J. Latent compositional representations improve systematic generalization in grounded question answering. *Transactions of the Association for Computational Linguistics*, 9: 195–210, 2021. doi: 10.1162/tacl_a_00361. URL https://aclanthology.org/2021.tacl-1.12.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 632–642, Lisbon, Portugal, September 2015a. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL https://aclanthology.org/D15-1075.

Bowman, S. R., Manning, C. D., and Potts, C. Tree-structured composition in neural networks without tree-structured architectures. In *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches - Volume 1583*, COCO'15, pp. 37–42, Aachen, DEU, 2015b. CEUR-WS.org.

Bowman, S. R., Gauthier, J., Rastogi, A., Gupta, R., Manning, C. D., and Potts, C. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1466–1477, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1139. URL https://aclanthology.org/P16-1139.

Choi, J., Yoo, K. M., and Lee, S. Learning to compose task-specific tree structures. In McIlraith, S. A. and Weinberger, K. Q. (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 5094–5101. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16682.

Chowdhury, J. R. and Caragea, C. Modeling hierarchical structures with continuous recursive neural networks. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1975–1988. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/chowdhury21a.html.

Collobert, R., Hannun, A., and Synnaeve, G. A fully differentiable beam search decoder. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1341–1350. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/collobert19a.html.

Csordás, R., Irie, K., and Schmidhuber, J. The neural data router: Adaptive control flow in transformers improves systematic generalization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=KBQP4A_J1K.

Cuturi, M., Teboul, O., and Vert, J.-P. Differentiable ranking and sorting using optimal transport. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/d8c24ca8f23c562a5600876ca2a550ce-Paper.pdf.

Del'etang, G., Ruoss, A., Grau-Moya, J., Genewein, T., Wenliang, L. K., Catt, E., Hutter, M., Legg, S., and Ortega, P. A. Neural networks and the chomsky hierarchy. *ArXiv*, abs/2207.02098, 2022.

Drozdov, A., Verga, P., Yadav, M., Iyyer, M., and McCallum, A. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. 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)*, pp. 1129–1141, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1116. URL https://aclanthology.org/N19-1116.

Drozdov, A., Rongali, S., Chen, Y.-P., O'Gorman, T., Iyyer, M., and McCallum, A. Unsupervised parsing with S-DIORA: Single tree encoding for deep inside-outside recursive autoencoders. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4832–4845, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.392. URL https://aclanthology.org/2020.emnlp-main.392.

DuSell, B. and Chiang, D. Learning context-free languages with nondeterministic stack RNNs. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pp. 507–519, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.conll-1.41. URL https://aclanthology.org/2020.conll-1.41.

DuSell, B. and Chiang, D. Learning hierarchical structures with differentiable nondeterministic stacks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=5LXw_QplBiF.

Fei, H., Ren, Y., and Ji, D. Retrofitting structure-aware transformer language model for end tasks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2151–2161, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.168. URL https://aclanthology.org/2020.emnlp-main.168.

Gardner, M., Artzi, Y., Basmov, V., Berant, J., Bogin, B., Chen, S., Dasigi, P., Dua, D., Elazar, Y., Gottumukkala, A., Gupta, N., Hajishirzi, H., Ilharco, G., Khashabi, D., Lin, K., Liu, J., Liu, N. F., Mulcaire, P., Ning, Q., Singh, S., Smith, N. A., Subramanian, S., Tsarfaty, R., Wallace, E., Zhang, A., and Zhou, B. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1307–1323, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.117. URL https://aclanthology.org/2020.findings-emnlp.117.

Goldberg, Y. and Elhadad, M. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 742–750, Los Angeles, California, June 2010. Association for Computational Linguistics. URL https://aclanthology.org/N10-1115.

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. Learning to transduce with unbounded memory. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pp. 1828–1836, Cambridge, MA, USA, 2015. MIT Press.

Grover, A., Wang, E., Zweig, A., and Ermon, S. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1eSS3CcKX.

Gu, A., Goel, K., and Re, C. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Havrylov, S., Kruszewski, G., and Joulin, A. Cooperative learning of disjoint syntax and semantics. 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)*, pp. 1118–1128, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1115. URL https://aclanthology.org/N19-1115.

Hendrycks, D. and Gimpel, K. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *ArXiv*, abs/1606.08415, 2016. URL http://arxiv.org/abs/1606.08415.

Herzig, J. and Berant, J. Span-based semantic parsing for compositional generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 908–921, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.74. URL https://aclanthology.org/2021.acl-long.74.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Hu, X., Mi, H., Wen, Z., Wang, Y., Su, Y., Zheng, J., and de Melo, G. R2D2: Recursive transformer based on differentiable tree for interpretable hierarchical language modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4897–4908, Online,

August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.379. URL https://aclanthology.org/2021.acl-long.379.

Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. Deep unordered composition rivals syntactic methods for text classification. 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)*, pp. 1681–1691, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1162. URL https://aclanthology.org/P15-1162.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Kaushik, D., Hovy, E., and Lipton, Z. Learning the difference that makes a difference with counterfactually-augmented data. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Sklgs0NFvr.

Kool, W., Van Hoof, H., and Welling, M. Stochastic beams and where to find them: The Gumbel-top-k trick for sampling sequences without replacement. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3499–3508. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/kool19a.html.

Lakretz, Y., Desbordes, T., Hupkes, D., and Dehaene, S. Causal transformers perform below chance on recursive nested constructions, unlike humans. *ArXiv*, abs/2110.07240, 2021.

Le, P. and Zuidema, W. The forest convolutional network: Compositional distributional semantics with a neural chart and without binarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1155–1164, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1137. URL https://aclanthology.org/D15-1137.

Liu, C., An, S., Lin, Z., Liu, Q., Chen, B., Lou, J.-G., Wen, L., Zheng, N., and Zhang, D. Learning algebraic recombination for compositional generalization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 1129–1144, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.97. URL https://aclanthology.org/2021.findings-acl.97.

Liu, Q., An, S., Lou, J.-G., Chen, B., Lin, Z., Gao, Y., Zhou, B., Zheng, N., and Zhang, D. Compositional generalization by learning analytical expressions. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Ma, J., Zhu, J., Xiao, T., and Yang, N. Easy-first POS tagging and dependency parsing with beam search. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 110–114, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://aclanthology.org/P13-2020.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL https://aclanthology.org/P11-1015.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1jE5L5gl.

Maillard, J. and Clark, S. Latent tree learning with differentiable parsers: Shift-reduce parsing and chart parsing. In *Proceedings of the Workshop on the Relevance of Linguistic Structure in Neural Architectures for NLP*, pp. 13–18, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2903. URL https://aclanthology.org/W18-2903.

Maillard, J., Clark, S., and Yogatama, D. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *Natural Language Engineering*, 25(4):433–449, 2019. doi: 10.1017/S1351324919000184.

Mao, J., Shi, F. H., Wu, J., Levy, R. P., and Tenenbaum, J. B. Grammar-based grounded lexicon learning. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=iI6nkEZkOl.

Munkhdalai, T. and Yu, H. Neural tree indexers for text understanding. In *Proceedings of the 15th Conference*

of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pp. 11–21, Valencia, Spain, April 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-1002.

Naik, A., Ravichander, A., Sadeh, N., Rose, C., and Neubig, G. Stress test evaluation for natural language inference. In *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 2340–2353, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/C18-1198.

Nangia, N. and Bowman, S. ListOps: A diagnostic dataset for latent tree learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pp. 92–99, New Orleans, Louisiana, USA, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-4013. URL https://aclanthology.org/N18-4013.

Nguyen, X.-P., Joty, S., Hoi, S., and Socher, R. Tree-structured attention with hierarchical accumulation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxK5pEYvr.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. Differentiable sorting networks for scalable sorting and ranking supervision. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8546–8555. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/petersen21a.html.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. Monotonic differentiable sorting networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=IcUWShptD7d.

Pollack, J. B. Recursive distributed representations. *Artificial Intelligence*, 46(1):77 – 105, 1990. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(90)90005-K. URL http://www.sciencedirect.com/science/article/pii/000437029090005K.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80, jan 2009. ISSN 1045-9227. doi: 10.1109/TNN.2008.2005605. URL https://doi.org/10.1109/TNN.2008.2005605.

Shen, Y., Tan, S., Hosseini, A., Lin, Z., Sordoni, A., and Courville, A. C. Ordered memory. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 5037–5048. Curran Associates, Inc., 2019a. URL http://papers.nips.cc/paper/8748-ordered-memory.pdf.

Shen, Y., Tan, S., Sordoni, A., and Courville, A. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=B1l6qiR5F7.

Shen, Y., Tay, Y., Zheng, C., Bahri, D., Metzler, D., and Courville, A. StructFormer: Joint unsupervised induction of dependency and constituency structure from masked language modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7196–7209, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.559. URL https://aclanthology.org/2021.acl-long.559.

Shi, H., Zhou, H., Chen, J., and Li, L. On tree-based neural sentence modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4631–4641, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1492. URL https://aclanthology.org/D18-1492.

Socher, R., Manning, C. D., and Ng, A. Y. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *In Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 151–161, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL https://aclanthology.org/D11-1014.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

Tran, K., Bisazza, A., and Monz, C. The importance of being recurrent for modeling hierarchical structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4731–4736, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1503. URL https://aclanthology.org/D18-1503.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wang, Y., Lee, H.-Y., and Chen, Y.-N. Tree transformer: Integrating tree structures into self-attention. 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)*, pp. 1061–1070, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1098. URL https://aclanthology.org/D19-1098.

Williams, A., Nangia, N., and Bowman, S. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-1101.

Xie, Y., Dai, H., Chen, M., Dai, B., Zhao, T., Zha, H., Wei, W., and Pfister, T. Differentiable top-k with optimal transport. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 20520–20531. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/ec24a54d62ce57ba93a531b460fa8d18-Paper.pdf.

Yogatama, D., Blunsom, P., Dyer, C., Grefenstette, E., and Ling, W. Learning to compose words into sentences with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skvgqgqxe.

Zhang, A., Tay, Y., Shen, Y., Chan, A., and ZHANG, S. Self-instantiated recurrent units with dynamic soft recursion. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 6503–6514. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/3341f6f048384ec73a7ba2e77d2db48b-Paper.pdf.

Algorithm 1 Easy First Composition

Input: data $X = [x_1, x_2, ...x_n]$
while True **do**
  if $len(X) == 1$ **then**
    return $X[0]$
  end if
  if $len(X) == 2$ **then**
    return $cell(X[0], X[1])$
  end if
  $Children_L, Children_R \leftarrow X[: len(X) - 1], X[1 :]$
  $Parents \leftarrow [cell(child_L, child_R)$ for $child_L$, $child_R$ in $zip(Children_L, Children_R]$
  $Scores \leftarrow [scorer(parent)$ for $parent$ in $Parents]$
  $index \leftarrow argmax(Scores)$
  $X[index] \leftarrow Parents[index]$
  Delete $X[index + 1]$
end while

A. Pseudocodes

We present the pseudocode of the easy first composition in Algorithm 1 and the pseudocode of BT-cell in Algorithm 2. Note that the algorithms are written as they are for the sake of illustration: in practice, many of the nested loops are made parallel through batched operations in GPU.

A.1. Beam Tree Cell Algorithm

Here, we briefly describe the algorithm of BT-cell (Algorithm 2) in words. In BT-Cell, instead of maintaining a single sequence per sample, we maintain some k (initially 1) number of sequences and their corresponding scores (initialized to 0). k is a hyperparameter defining the beam size. Each sequence (henceforth, interchangeably referred to as "beam") is a hypothesis representing a particular sequence of choices of parents. Thus, each beam represents a different path of composition (for visualization see Figure 1). At any moment the score represents the log-probability for its corresponding beam. Now, we describe the steps in each iteration in the recursion of BT-Cell. **Step 1:** similar to gumbel-tree models, we create all candidate parent compositions for each of the k beams. **Step 2:** we score the candidates with the $score$ function (defined in §2.2). **Step 3:** we choose top-k highest scoring candidates. We treat the top-k choices as mutually exclusive. Thus, each of the k beams encounters k branching choices, and are updated into k distinct beams (similar to before, the children are replaced by the chosen parent). Thus, we get $k \times k$ beams. **Step 4:** we update the beam scores. The sub-steps involved in the update are described next. **Step 4.1:** we apply a log-softmax to the scores of the latest candidates to put the scores into the log-probability space. **Step 4.2:** we add the log-softmaxed scores of the latest chosen candidate to the existing beam score for the corresponding beam where the candidate is chosen. As a result, we will have $k \times k$ beam scores. **Step 5:** we truncate the $k \times k$ beams and beam scores into k beams and their corresponding k scores to prevent exponential increase of the number of beams. For that, we again simply use a top-k operator to keep only the highest scored beams.

At the end of the recursion, instead of a single item representing the sequence-encoding, we will have k beams of items with their k scores. At this point, to get a single item, we do a weighted summation with the softmaxed scores as the weights as described in §3.

B. Gated Recursive Cell (GRC)

The Gated Recursive Cell (GRC) was originally introduced by (Shen et al., 2019a) drawing inspiration from the Transformer's feed-forward networks. In our implementation, we use the same variant of GRC as was used in (Chowdhury & Caragea, 2021) where a GELU (Hendrycks & Gimpel, 2016) activation function was used. We present the equations of GRC here:

$$\begin{bmatrix} z_i \\ h_i \\ c_i \\ u_i \end{bmatrix} = W_2 \text{ GeLU} \left(W_1^{Cell} \begin{bmatrix} child_{left} \\ child_{right} \end{bmatrix} + b_1 \right) + b_2 \tag{7}$$

Algorithm 2 Beam Tree Cell

Input: data $X = [x_1, x_2, ...x_n], k$ (beam size)
$BeamX \leftarrow [X]$
$BeamScores \leftarrow$ [0]
while True **do**
  if $len(BeamX[0]) == 1$ **then**
    $BeamX \leftarrow [beam[0]$ for $beam$ in $BeamX]$
    break
  end if
  if $len(BeamX[0]) == 2$ **then**
    $BeamX \leftarrow [cell(beam[0], beam[1])$ for $beam$ in $BeamX]$
    break
  end if
  $NewBeamX \leftarrow []$
  $NewBeamScores \leftarrow []$

  for $Beam$,$BeamScore$ in $zip(BeamX, BeamScores)$ **do**
    $Parents \leftarrow [cell(beam[i], beam[i+1])$ for i in $range(0, len(beam) - 1)]$
    $Scores \leftarrow log \circ softmax([scorer(parent)$ for $parent$ in $Parents])$
    $Indices \leftarrow topk(Scores, k)$

    for i in $range(K)$ **do**
      $newBeam \leftarrow deepcopy(Beam)$
      $newBeam[Indices[i]] \leftarrow Parents[Indices[i]]$
      Delete $newBeam[Indices[i] + 1]$
      $NewBeamX.append(newBeam)$
      $newScore \leftarrow BeamScore + Scores[indices[i]]$
      $newBeamScores.append(newScore)$
    end for
  end for
  $Indices \leftarrow topk(newBeamScores, k)$
  $BeamScores \leftarrow [newBeamScores[i]$ for i in Indices]
  $BeamX \leftarrow [newBeamX[i]$ for i in Indices]
end while
$BeamScores \leftarrow Softmax(BeamScores)$
Return $sum([score * X$ for $score, X$ in $zip(BeanScores, BeamX)])$

$$o_i = LN(\sigma(z_i) \odot child_{left} + \sigma(h_i) \odot child_{right} + \sigma(c_i) \odot u_i) \tag{8}$$

σ is $sigmoid$; o_i is the parent composition $\in \mathbb{R}^{d_h \times 1}$; $child_{left}, child_{right} \in \mathbb{R}^{d_h \times 1}$; $W_1^{cell} \in \mathbb{R}^{d_{cell} \times 2 \cdot d_h}$; $b_1 \in \mathbb{R}^{d_{cell} \times 1}$; $W_2 \in \mathbb{R}^{d_h \times d_{cell}}$; $b_1 \in \mathbb{R}^{d_h \times 1}$. We use this same GRC function for any recursive model (including our implementation of Ordered Memory) that constitutes GRC.

C. BSRP-GRC Details

For the decisions about whether to shift or reduce, we use a scorer function similar to that used in (Chowdhury & Caragea, 2021). Where (Chowdhury & Caragea, 2021) use the decision function on concatenation of local hidden states (n-gram window), we use the decision function on the concatenation of last two items in the stack and the next item in the queue. The output is a scalar sigmoid activated logit score s. We then treat $log(s)$ as the score for reducing in that step, and $log(1 - s)$ as the score for shifting in that step. The scores are manipulated appropriately for edge cases (when there are no next item to shift, or when there are no two items in the stack to reduce). Besides that, we use the familiar beam search strategy over standard shift-reduce parsing. Finally the beams of final states are merged through the weighted summation of the states based on the softmaxed scores of each beam similar to BT-Cell models as described in §3.

Model	**DG**	**Length Gen.**			**Argument Gen.**		**LRA**
(Lengths)	≤ 100	200-300	500-600	900-1k	100-1k	100-1k	2K
(Arguments)	≤ 5	≤ 5	≤ 5	≤ 5	10	15	≤ 10
(Depths)	8-10	≤ 20	≤ 20	≤ 20	≤ 10	≤ 10	≤ 10
With gold trees							
GoldTreeGRC	99.95	99.95	99.9	99.8	76.95	77.1	74.55
Baselines without gold trees							
CYK-GRC	99.45	99.0	—	—	67.8	35.15	—
Ordered Memory	**99.95**	<u>99.8</u>	99.25	96.4	**79.95**	**77.55**	**77**
CRvNN	<u>99.9</u>	99.4	99.45	98.9	65.7	43.4	65.1
Ours							
BT-GRC	**99.95**	**99.95**	**99.95**	**99.9**	75.35	72.05	68.1
BT-GRC + OneSoft	<u>99.9</u>	99.6	98.1	97.1	<u>78.1</u>	71.25	<u>75.45</u>

Table 5. Accuracy on ListOps-DG. We report the median of 3 runs except in the last block where we report the mean/std of 10 runs as mentioned. Our models were trained on lengths ≤ 100, depth ≤ 6, and arguments ≤ 5. We bold the best results and underline the second-best among models that do not use gold trees.

.

Model	**DG**	**Length Gen.**			**Argument Gen.**		**LRA**
(Lengths)	≤ 100	200-300	500-600	900-1k	100-1k	100-1k	2K
(Arguments)	≤ 5	≤ 5	≤ 5	≤ 5	10	15	≤ 10
(Depths)	8-10	≤ 20	≤ 20	≤ 20	≤ 10	≤ 10	≤ 10
Stability Test: Mean/Std with 10 runs. Beam size 5 for BT-GRC							
Ordered Memory	$99.94_{0.6}$	97.58_{32}	78.785_{197}	61.85_{291}	77.66_{30}	69.03_{107}	67.35_{125}
BT-GRC	$99.84_{1.5}$	$99.58_{5.8}$	98.8_{21}	97.85_{39}	73.82_{57}	66.21_{107}	66.975_{102}

Table 6. Accuracy on ListOps-DG (Stability test). We report the mean and standard deviation of of 10. Our models were trained on lengths ≤ 100, depth ≤ 6, and arguments ≤ 5. Subscript represents standard deviation. As an example, $90_1 = 90 \pm 0.1$

.

D. Additional Experiments and Analysis

D.1. ListOps-DG Experiment

Dataset Settings: The length generalization experiments in ListOps do not give us an exact perspective in depth generalization[4] capacities. So there is a question of how models will perform in unseen depths. To check for this, we create a new ListOps split which we call "ListOps-DG". For this split, we create $100,000$ training data with arguments ≤ 5, lengths ≤ 100, and depths ≤ 6. We create 2000 development data with arguments ≤ 5, lengths ≤ 100, and depths 7. We create 2000 test data with arguments ≤ 5, lengths ≤ 100, and depths 8-10. In addition, we also still tested on the same length-generalization splits (which now simultaneously have much higher depths too: ≤ 20), argument generalization splits, and LRA. The results are presented in Table 5. We only evaluate the models that were promising ($\geq 90\%$ in near IID settings) in the original ListOps split. We report the median of 3 runs for each model (except in the last block of the table).

Results: Interestingly, we find that base BT-GRC, CRvNN, and Ordered Memory now does much better in length generalization compared to the original listops split. We think this is because of the increased data (the training data in the original ListOps is ~ 75000 after filtering data of length > 100 whereas here we generated $100,000$ training data). However, while the median of 3 runs in Ordered Memory is decent, we found one run to have very poor length generalization performance. To investigate more deeply if Ordered Memory has a particular stability issue, we ran Ordered Memory for 10 times with diferent seeds, and we find that it frequently fails to learn to generalize over length. As a baseline, we also ran BT-GRC similarly for 10 runs and found it to be much more stable. We report the mean and standard deviation of 10 runs of Ordered Memory and BT-GRC in Table 6. As can be seen, the mean of BT-GRC is much higher than that of Ordered Memory in length generalization splits.

[4]By depth, we simply mean the maximum number of nested operators in a given sequence in case of ListOps

Model	**DG1**	**Length Gen.**			**Argument Gen.**		**LRA**
(Lengths)	≤ 50	200-300	500-600	900-1000	100-1000	100-1000	2000
(Arguments)	≤ 5	≤ 5	≤ 5	≤ 5	10	15	≤ 10
(Depths)	8-10	≤ 20	≤ 20	≤ 20	≤ 10	≤ 10	≤ 10
After Training on ListOps-DG1							
NDR (layer 24)	96.7	48.9	32.85	22.1	65.65	64.6	42.6
NDR (layer 48)	91.75	34.60	24.05	19.7	54.65	52.45	39.95
Model	**DG2**	**Length Gen.**			**Argument Gen.**		**LRA**
(Lengths)	≤ 100	200-300	500-600	900-1000	100-1000	100-1000	2000
(Arguments)	≤ 5	≤ 5	≤ 5	≤ 5	10	15	≤ 10
(Depths)	8-10	≤ 20	≤ 20	≤ 20	≤ 10	≤ 10	≤ 10
After Training on ListOps-DG2							
NDR (layer 24)	95.6	44.15	30.7	20.3	67.85	58.05	46
NDR (layer 48)	92.65	38.6	29.15	22.1	73.1	64.4	50.4

Table 7. Accuracy on ListOps-DG1 and ListOps-DG2. We report the max of 3 runs. In ListOps-DG1, NDR was trained on lengths ≤ 50, depth ≤ 6, and arguments ≤ 5. In ListOps-DG1, NDR was trained on lengths ≤ 100, depth ≤ 6, and arguments ≤ 5. Lyaers denote the layers used during inference.

.

D.2. NDR Experiments

Dataset Settings: Neural Data Routers (NDR) is a Transformer-based model that was shown to perform well in algorithmic tasks including Listops (Csordás et al., 2022). We tried some experiments with it too. We found NDR to be struggling in the original ListOps splits or the ListOps-DG split. We noticed that in the paper (Csordás et al., 2022), NDR was trained in a much larger sample size (~ 10 times more data than in ListOps-DG) and also on lower sequence lengths (~ 50). To better check for the capabilities of NDR, we created two new ListOps split - DG1 and DG2. In DG1, we set the sequence length to 10-50 in training, development, and testing set. We created 1 million data for training, and 2000 data for development and testing. Other parameters (number of arguments, depths etc.) are same as in ListOps-DG split. Split DG2 is the same as ListOps-DG split in terms of data-generation parameters (i.e it includes length sizes ≤ 100) but with much larger sample size for the training split (again, 1 million samples same as DG1). We present the results in Table 7.

Results: We find that even when we focus on the best of 3 runs in the table, although NDR generalizes to slightly higher depths (8-10 from ≤ 6) (as reported in (Csordás et al., 2022)), it still struggles with splits with orders of magnitude higher depths, lengths, and unseen arguments. Following the suggestions of (Csordás et al., 2022), we also increase the number of layers during inference (eg. upto 48) to handle higher depth sequences but that did not help substantially. Thus, even after experiencing more data, NDR generalizes worse than Ordered Memory, CRvNN, or BT-GRC. Moreover, NDR requires some prior estimation of the true computation depth of the task for its hyperparameter setup unlike the other latent-tree models.

D.3. Synthetic Logical Inference Results

Dataset Settings: We also consider the synthetic testbed for detecting logical relations between sequence pairs as provided by (Bowman et al., 2015b). Following (Tran et al., 2018), we train the models on sequences with ≤ 6 operators and test on data with greater number of operators (here, we check for cases with ≥ 8 operators) to check for capacity to generalize to unseen number of operators. Similar to (Shen et al., 2019a; Chowdhury & Caragea, 2021), we also train the model on the systematicity split C. In this split we remove any sequence matching the pattern $*(and(not*))*$ from the training set and put them in the test set to check for systematic generalization.

Results: In Table 8, in terms of operation generalization, our proposed BT-Cell models perform similarly to prior SOTA models like Ordered Memory (OM) and CRvNN while approximating GoldTreeGRC for both beam sizes. In terms of systematicity (split C), OM and BT-GRC perform similarly (both above 94%) and much better than the other models. Surprisingly, however, OneSoft extension also hurt systematicity in this context. CYK-GRC shows promise in operator generalization but shows poor systematicity as well. Gumbel-GRC performs better than RecurrentGRC but still far from SOTA which is not unexpected given its poor results in ListOps.

Model	Number of Operations					
	8	9	10	11	12	C
With gold trees						
GoldTreeGRC	97.14_{1}	96.5_{2}	$95.29_{2.5}$	$94.21_{9.9}$	$93.67_{7.7}$	$97.41_{1.6}$
Baselines without gold trees						
Transformer*	52	51	51	51	48	51
Universal Transformer*	52	51	51	51	48	51
ON-LSTM*	87	85	81	78	75	60
Self-IRU†	95	93	92	90	88	—
RecurrentGRC	93.04_{6}	$90.43_{4.9}$	88.48_{6}	$86.57_{5.8}$	$80.58_{1.5}$	$83.17_{5.1}$
GumbelTreeGRC	93.46_{14}	91.89_{19}	90.33_{22}	88.43_{18}	85.70_{24}	89.34_{29}
CYK-GRC	$96.62_{2.3}$	$96.07_{4.6}$	94.67_{11}	$93.44_{8.8}$	$92.54_{9.3}$	77.08_{27}
CRvNN	$96.9_{3.7}$	$95.99_{2.8}$	$94.51_{2.9}$	$\mathbf{94.48_{5.6}}$	92.73_{15}	89.79_{58}
Ordered Memory	$\mathbf{97.5_{1.6}}$	$\mathbf{96.74_{1.4}}$	94.95_{2}	$93.9_{2.2}$	$93.36_{6.2}$	$\mathbf{94.88_{7}}$
Ours						
BT-GRC	96.83_{1}	$95.99_{2.4}$	$95.04_{2.3}$	$\underline{94.29_{3.8}}$	$93.36_{2.4}$	$\underline{94.17_{14}}$
BT-GRC + OneSoft	$\underline{97.03_{1.4}}$	$\underline{96.49_{1.9}}$	$\mathbf{95.43_{4.5}}$	$94.21_{6.6}$	$\mathbf{93.39_{1.5}}$	78.04_{43}

Table 8. Mean accuracy and standard deviaton on the Logical Inference for ≥ 8 number of operations after training on samples with ≤ 6 operations. We also report results of the systematicity split C. We bold the best results and underline the second-best for all models without gold trees. * indicates that the results were taken from (Shen et al., 2019a) and † indicates results from (Zhang et al., 2021). Our models were run 3 times on different seeds. Subscript represents standard deviation. As an example, $90_{1} = 90 \pm 0.1$

.

Model	near-IID	Length Gen.			Argument Gen.		LRA
(Lengths)	≤ 1000	200-300	500-600	900-1000	100-1000	100-1000	2000
(Arguments)	≤ 5	≤ 5	≤ 5	≤ 5	10	15	10
RandomTreeGRC	70.56	48.70	45.35	37.53	54.8	55.6	49.8
BalancedTreeGRC	59.4	44.85	43.35	35.70	45.88	45.25	41.95

Table 9. Accuracy of different models on ListOps (same setting as in Table 1). We report the median of 3 runs.

D.4. Heuristics Tree Models

We consider two heuristics-based tree model - RandomTreeGRC (uses random tree strcutures) and BalancedTreeCell (follows a balanaced binary tree structure (Shi et al., 2018)). We run them on ListOps and show the results in Table 9. As we would expect the heuristics-based model perform very poorly. We also run them on natural language data and show the results in Table 10. It performs relatively more competitively against other models in realistic data but they still generally fall behind the OM, CRvNN, and BT-Cell.

D.5. Parse Tree Analysis

In this section, we analyze the induced structures of BT-Cell models. Note, however, although induced structures can provide some insights to the model, we can draw limited conclusions from them. First, if we take a stance similar to (Choi et al., 2018) in considering it suitable to allow different kinds of structures to be induced as appropriate for a specific task then it's not clear how structures should be evaluated by themselves (besides just the donwstream task evaluations). Second, the extracted structures may not completely reflect what the models may implicitly induce because the recursive cell can override some of the parser decisions (given how there is evidence that even simple RNNs (Bowman et al., 2015b) can implicitly model different tree structures within its hidden states to an extent even when its explicit structure always conform to the left-to-right order of composition). Third, even if the extracted structure perfectly reflects what the model induces, another side of the story is the recursive cell itself and how it utilizes the structure for language understanding. This part of the story can still remain unclear because of the blackbox-nature of neural nets. Nevertheless, extractive structures may still provide some rough idea of the inner workings of BT-Cell variants.

In Table 11, we show the parsed structures of some iconic sentences by BT-GRC after it is trained on MNLI. We report all beams and their corresponding scores. Note, although beam search ensures that the sequence of parsing actions for each

Model	SST5	IMDB		MNLI					
	IID	Con.	Count.	M	MM	Len M	Len MM	Neg M	Neg MM
RandomTreeGRC	$51.78_{1.2}$	74.93_{14}	$82.38_{9.3}$	72.2_{3}	72.3_{5}	61.4_{23}	62.3_{23}	51.7_{3}	52.7_{7}
BalancedTreeGRC	$52.35_{6.2}$	74.93_{22}	83.61_{15}	71.1_{5}	71.4_{1}	59_{8}	60.7_{5}	50.2_{4}	50.4_{6}

Table 10. Mean accuracy and standard deviaton on SST5, IMDB, and MNLI. Con. represents Contrast set and Count. represents Countefactuals. Our models were run 3 times on different seeds. Subscript represents standard deviation. As an example, $90_1 = 90 \pm 0.1$.

Score	Parsed Structures
	BT-GRC (beam size 5)
0.42	((i (did not)) (((like a) (single minute)) ((of this) film)))
0.40	(((i (did not)) ((like a) (single minute))) ((of this) film))
0.20	((i (did not)) (((like a) ((single minute) of)) (this film)))
0.40	((i (shot an)) ((elephant in) (my pajamas)))
0.21	(((i shot) (an elephant)) ((in my) pajamas))
0.19	(((i shot) (an elephant)) (in (my pajamas)))
0.19	((i shot) ((an elephant) ((in my) pajamas)))
0.40	((john saw) ((a man) (with binoculars)))
0.40	(((john saw) (a man)) (with binoculars))
0.20	((john (saw a)) ((man with) binoculars))
0.61	(((roger (dodger is)) (one (of the))) (((most compelling) (variations of)) (this theme)))
0.40	(((roger (dodger is)) ((one (of the)) (most compelling))) ((variations of) (this theme)))
	BT-GRC (beam size 2)
0.50	((i ((did not) like)) (((a single) minute) ((of this) film)))
0.50	((i (((did not) like) (a single))) ((minute of) (this film)))
0.50	((i (shot an)) ((elephant in) (my pajamas)))
0.50	((i ((shot an) elephant)) ((in my) pajamas))
$0,51$	((john (saw a)) ((man with) binoculars))
0.49	(john (((saw a) man) (with binoculars)))
1.0	((roger ((dodger is) one)) ((((of the) most) (compelling variations)) ((of this) theme)))

Table 11. Parsed Structures of BT-GRC trained on MNLI. Each block represents different beams.
.

beam is unique, different sequences of parsing action can still lead to the same structure. Thus, some beams end up being duplicates. In such cases, for the sake of more concise presentation, we collapse the duplicates into a single beam and add up their corresponding scores. This is why we can note in Table 11 that we sometimes have fewer induced structures than the beam size.

At a rough glance, we can see that the different induced structures roughly correspond to human intuitions. One interesting appeal for beam search is that it can more explicitly account for ambiguous interpretations corresponding to ambiguous structures. For example, *"i shot an elephant in my pajamas"* is ambiguous with respect to whether it is the elephant who is in the shooter's pajamas, or if it is the shooter who is in the pajamas. The induced structure (beam size 5 model in Table 11) *(((i shot) (an elephant)) ((in my) pajamas))* corresponds better to the latter interpretation whereas *((i shot) ((an elephant) ((in my) pajamas)))* corresponds better to the former interpretation (because "an elephant" is first composed with "in my pajamas").

Similar to above, *"john saw a man with binoculars"* is also ambiguous. Its interpretation is ambiguous with respect to whether it is John who is seeing through binoculars, or whether it is the man who just possesses the binoculars. Here, again, we can find (beam size 5 model in Table 11) that the induced structure *(((john saw) (a man)) (with binoculars)* corresponds better to the former interpretation whereas *((john saw) ((a man) (with binoculars)))* corresponds better to the latter. Generally, we find the score distributions to have a high entropy. A future consideration would be whether we should add an auxiliary objective to minimize entropy.

In Table 12, we show the parsed structures of the same sentences by BT-GRC+OneSoft after it is trained on MNLI. Most of the points above applies here for OneSoft as well. Interestingly, OneSoft seems to have a relatively lower entropy distribution - that is most evident in beam size 2.

Score	Parsed Structures
BT-GRC + OneSoft (beam size 5)	
0.42	(((i did) (not like)) (((a single) minute) ((of this) film)))
0.20	((((i did) not) ((like a) single)) ((minute of) (this film)))
0.19	((((i did) (not like)) ((a single) minute)) ((of this) film))
0.19	(((i (did not)) ((like a) single)) ((minute of) (this film)))
0.41	(((i shot) an) ((elephant in) (my pajamas)))
0.21	(((i shot) (an elephant)) ((in my) pajamas))
0.19	((i (shot an)) ((elephant in) (my pajamas)))
0.19	((((i shot) an) (elephant in)) (my pajamas))
0.21	((john (saw a)) ((man with) binoculars))
0.20	(((john saw) (a man)) (with binoculars))
0.20	((john saw) ((a man) (with binoculars)))
0.19	((john ((saw a) man)) (with binoculars))
0.40	(((roger dodger) (is one)) ((((of the) most) (compelling variations)) ((of this) theme)))
0.21	(((roger (dodger is)) ((one of) the)) (((most compelling) variations) ((of this) theme)))
0.20	((((roger dodger) (is one)) ((of the) most)) ((compelling variations) ((of this) theme)))
0.19	((roger (dodger is)) ((((one of) the) ((most compelling) variations)) ((of this) theme)))
BT-GRC + OneSoft (beam size 2)	
0.57	((i ((did not) like)) (((a single) minute) ((of this) film)))
0.43	((i ((did not) like)) (((a single) (minute of)) (this film)))
0.54	((i ((shot an) elephant)) ((in my) pajamas))
0.46	((i (shot an)) ((elephant in) (my pajamas)))
0.55	((john (saw a)) ((man with) binoculars))
0.45	((john ((saw a) man)) (with binoculars))
0.53	((roger ((dodger is) one)) ((((of the) most) (compelling variations)) ((of this) theme)))
0.47	(((roger ((dodger is) one)) ((of the) most)) ((compelling variations) ((of this) theme)))

Table 12. Parsed Structures of BT-GRC + OneSoft trained on MNLI. Each block represents different beams.

.

We found the structures induced by BT-Cell variants after training on SST5 or IMDB to be more ill-formed. This may indicate that sentiment classification does not provide a strong enough signal for parsing or rather, exact induction of structures are not as necessary (Iyyer et al., 2015). We show the parsings of these models after training on IMDB and SST datasets in a text file included in the supplementary.

E. Extended Related Works

Initially RvNN (Pollack, 1990; Socher et al., 2010) was used with user-annotated tree-structured data. Some explored use of heuristic trees such as balanced trees for RvNN-like settings (Munkhdalai & Yu, 2017; Shi et al., 2018). In due time, several approaches were introduced for dynamically inducing structures from data for RvNN-style processing. This includes the greedy easy-first framework using children-reconstruction loss (Socher et al., 2011) or gumbel softmax (Choi et al., 2018), RL-based frameworks (Havrylov et al., 2019), CYK-based framework (Le & Zuidema, 2015; Maillard et al., 2019; Drozdov et al., 2019; Hu et al., 2021), shift-reduce parsing or memory-augmented or stack-augmented RNN frameworks (Grefenstette et al., 2015; Bowman et al., 2016; Yogatama et al., 2017; Maillard & Clark, 2018; Shen et al., 2019a; DuSell & Chiang, 2020; 2022), and soft-recursion-based frameworks (Chowdhury & Caragea, 2021; Zhang et al., 2021). Besides RvNNs, other approaches range from adding information-ordering biases to hidden states in RNNs (Shen et al., 2019b) or even adding additional structural or recursive constraints to Transformers (Wang et al., 2019; Nguyen et al., 2020; Fei et al., 2020; Shen et al., 2021; Csordás et al., 2022).

There are multiple versions of differentiable top-k operators or sorting functions (Adams & Zemel, 2011; Grover et al., 2019; Cuturi et al., 2019; Xie et al., 2020; Blondel et al., 2020; Petersen et al., 2021; 2022). We leave a more exhaustive analysis of them as future work. However, note that many of them would suffer from the same systematic issues as SOFT top-k (Xie et al., 2020) - that is they can significantly slow down the model and they can lead to "washed out" representations.

F. Hyperparameters

For all recursive/recurrent models, we use a linear layer followed by layer normalization for initial leaf transformation before starting the recursive loop (similar to (Shen et al., 2019a; Chowdhury & Caragea, 2021)). Overall we use the same boilerplate classifier architecture for classification and the same boilerplate sentnece-pair siamese architecture for logical inference as (Chowdhury & Caragea, 2021) over our different encoders. In practice, for BT-Cell, we use a stochastic top-k through gumbel perturbation similar to Kool et al. (2019). However, we find deterministic selection to work similarly. In our implementation of CRvNN, we ignore some extraneous elements from CRvNN such as transition features and halt penalty which were deemed to have little effect during ablation in Chowdhury & Caragea (2021).

In terms of the optimizer, hidden size, and other hyperparameters besides dropout, we use the same ones as used by (Chowdhury & Caragea, 2021) for all models for corresponding datasets; for number of memory slots and other ordered memory specific parameters we use the same ones as used by (Shen et al., 2019a). For BSRP-Cell we use a beam size of 8 (we also tried with 5 but results were similar or slightly worse). We use a dropout rate of 0.1 for logical inference for all models (tuned on the validation set using grid search among $[0.1, 0.2, 0.3, 0.4]$ with 5 epochs per run using BalancedTreeCell for GRC-based models and GumbelTreeLSTM for LSTM based models). We use dropouts in the same places as used in (Chowdhury & Caragea, 2021). We then use the same chosen dropouts for ListOps. We tune the dropouts for SST in the same way (but with a maximum epoch of 20 per trial) on SST5 using RecurrentGRC for GRC-models, and Gumbel-Tree-LSTM for LSTM models. After tuning, for GRC-based models in SST5, we found a dropout rate of 0.4 for input/output dropout layers, and 0.2 for the dropout layer in the cell function. We found a dropout of 0.3 for LSTM-based models in SST5. and We share the hyperparameters of SST5 with IMDB. For MNLI, we used similar settings as Chowdhury & Caragea (2021).

For NDR experiments, we use the same hyperparameters as used for ListOps by (Csordás et al., 2022). The hyperparameters will also be available in code.

All codes are run in a single RTX A6000 GPU.