BEAM TREE RECURSIVE CELLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive Neural Networks (RvNNs) generalize Recurrent Neural Networks (RNNs) by allowing sequential composition in a more flexible order, typically, based on some tree structure. While initially user-annotated tree structures were used, in due time, several approaches were proposed to automatically induce tree-structures from raw text to guide the recursive compositions in RvNNs. In this paper, we present an approach called Beam Tree Recursive Cell (or BT-Cell) based on a simple yet overlooked backpropagation-friendly framework. BT-Cell applies beam search on easy-first parsing for simulating RvNNs with automatic structure-induction. Our results show that BT-Cell achieves near-perfect performance on several aspects of challenging structure-sensitive synthetic tasks like ListOps and also comparable performance in realistic data to other RvNN-based models. We further introduce and analyze several extensions of BT-Cell based on relaxations of the hard top-k operators in beam search. We evaluate the models in different out of distribution splits in both synthetic and realistic data. Additionally, we identify a previously unknown failure case for neural models in generalization to unseen number of arguments in ListOps. We will release our code.

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, it 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.

RvNNs (Pollack, 1990; Socher et al., 2010) belong to one among multiple neural approaches that have been proposed to model hypothesized syntactical tree-structures underlying languages by adding more explicit structure-inducing inductive biases. Besides RvNNs, other approaches range from adding information-ordering biases to hidden states in RNNs (Shen et al., 2019b), simulating shift-reduce parsing or pushdown automata through memory augmented RNNs (Shen et al., 2019a; Yogatama et al., 2017a), 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).

RvNNs take a rather straight-forward approach to tree-oriented sequence composition. Effectively, it recursively composes child nodes into parent nodes starting from the leafs and ending at the root. However, in the original conception, RvNN required users themselves to provide the input tree-structures (typically, in case of texts, they are dependency or constituency parses). Several works (Socher et al., 2011; Havrylov et al., 2019; Choi et al., 2018; Maillard et al., 2019; Chowdhury & Caragea, 2021) sought to move beyond this limitation by extending the original RvNN framework to automatically induce the underlying structure of the input sequence and process it accordingly. Our proposal belongs to this class of approaches.

In our proposal, we introduce a simple yet overlooked framework, Beam Tree Cell (BT-Cell), for simulating RvNNs with automatic structure-induction. Particularly, we use beam search with an easy first parsing strategy (Goldberg & Elhadad, 2010). We show that this simple framework is enough to get state-of-the-art length generalization and depth-generalization in structure-sensitive

synthetic tasks like ListOps and comparable performances in realistic data. BT-Cell has several architectural advantages over prior approaches. Unlike Gumbel-Tree-Models (Choi et al., 2018), as we shall show, BT-Cell can actually perform in structure-sensitive tasks; unlike Ordered Memory (OM) (Shen et al., 2019a), BT-Cell avoids nested sequential tensor operations over memory cells; unlike Havrylov et al. (2019), BT-Cell avoids expensive reinforcement learning, remaining backpropagation-friendly, and unlike Chowdhury & Caragea (2021), BT-Cell avoids multiple attention-like matrix operations for neighbor retrieval per iteration. BT-Cell also provides direct access to span representations or representations for each node in the induced tree-structure. This can be useful to some downstream applications Herzig & Berant (2021); Bogin et al. (2021); Liu et al. (2021) and it can induce multiple beams of tree-structures with different associated probabilities which can be more suitable in ambiguous contexts.

We also explore several variants of BT-Cell. Particularly, we explore ways to replace the non-differentiable top-k operators involved in beam search with different alternatives such as straight-through gumbel softmax (Jang et al., 2017) and a novel strategy of maintaining a convex combination of multiple paths.

Besides proposal and evaluation of BT-Cell, our paper also serves as a survey of how well prior proposed techniques (e.g., CYK-based models (Maillard et al., 2019) and Straight-Through Gumbel-Tree models (Choi et al., 2018)) work in structure-sensitive synthetic tasks and out-of-distribution-splits in natural language classification tasks when they are adapted with a better recursive cell function. Additionally, as a further contribution, we 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

2.1 RECCURENT NEURAL NETWORKS AND RECURSIVE NEURAL NETWORKS

A core component of both RNNs and RvNNs is a recursive cell function which can be expressed as $cell(.,.)$. The cell function takes two items as arguments ($a1$ and $a2$) and returns an item (v) representing the composition of the two arguments: $v = cell(a1, a2)$ (we can also treat $a1$ and $a2$ as forming children of v as a parent in an underlying tree structure)[1]. The data structure of the items can depend on particular implementations; usually, the only constraint for typical RNNs is that the data-structure of v should be the same as at least one of the arguments, whereas, for typical RvNNs, $a1, a2$, and v should all share the same data structure. In the simple case, $a1, a2$, and v are vectors.

Given a sequence of items, both RNNs and RvNNs sequentially process the sequence through recursive application of the cell function. For a concrete example, consider a sequence of tokens such as $(2 + 4 \times 4 + 3)$[2]. 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:

$$h = cell(cell(cell(cell(cell(cell(cell(h0, 2), +), 4), \times), 4), +), 3) \tag{1}$$

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:

$$h = cell(cell(cell(cell(2, +), cell(cell(4, \times), 4)), +), 3) \tag{2}$$

Thus, RNNs can be considered as, roughly, a special case of RvNNs where a strict left-to-right order of composition is enforced. As we can see in Eqn. 2, in RvNNs, we can align the order of composition to PEMDAS, whereas in RNNs we cannot. Nevertheless, RNNs can still learn to simulate RvNNs by modeling tree-structures implicitly in their intermediate states (hidden states or memory) resulting from the $cell(.,.)$ (Bowman et al., 2015). For example, RNNs can learn to

[1] In some implementations there could be multiple arguments but we are focusing on more typical cases

[2] Before, application of the cell function each of the tokens "2", "4" etc. would be mapped to some data-structure by application of some function $e(.)$ (typically an embedding layer) but we avoid the extra notations here for the sake of brevity.

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. Special inductive biases can be incorporated to enhance their abilities to handle their internal memory structures (Shen et al., 2019b;a). Still, their memories are bounded and there can be a limit to what level of nested structures they can handle.

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. 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 on the latter approach for sentence-encoding. Below we discuss one framework for this approach.

2.2 EASY-FIRST PARSING

Here we describe an adaptation (inspired from Choi et al. (2018)) of the easy-first parsing Goldberg & Elhadad (2010) algorithm for RvNN-based sentence-encoding. In this adapted algorithm, at every iteration in a recursive loop, given a sequence of items $(x_1, x_2, \ldots, x_n)$ we consider all possible immediate candidate parent compositions taking the current items as children: $(cell(x_1, x_2), cell(x_2, x_3), \ldots, cell(x_{n-1}, x_n)$[3]. We then score (with some scalar value) each of the candidates with some scorer function (which can be a neural network). We commit the highest scoring candidate (committing to the "easiest" decision first) and ignore the other candidates. For the sake of illustration, if $cell(x_i, x_{i+1})$ is chosen, it would replace its children x_i and x_{i+1}; thus, the resulting sequence will become: $(x_1, \ldots, x_{i-1}, cell(x_i, x_{i+1}), x_{i+2}, \ldots, x_n)$. 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 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-Cell models (Choi et al., 2018) address this difficulty by using straight-through (Bengio et al., 2013) gumbel softmax (Jang et al., 2017) instead of argmax. However, as it was previously discovered (Nangia & Bowman, 2018), and as we independently verify, gumbel-based strategies fail to perform as well when tested in structure-sensitive tasks. Instead, in this paper, we present an alternative of extending argmax with a top-k operator under a beam search strategy.

3 BEAM TREE CELL

We propose Beam Tree Cell (BT-Cell) as an extension of the easy-first composition strategy by the application of beam search. In BT-Cell, instead of maintaining a single sequence per sample, we maintain some B (initially 1) number of sequences and their corresponding scores (initialized to 0). Each sequence (henceforth, interchangeably referred to as "beam") is a hypothesis representing a particular sequence of choices of parents (thus, each represents a different path of composition). At any moment the score represents the log-probability for its corresponding beam. Now, we describe the steps in each iteration in the recursion. **Step 1:** similar to before, we create all candidate parent compositions for each of the B beams. **Step 2:** we score the candidates with some scorer function. **Step 3:** we choose top-k (k is the beam size) highest scoring candidates. We treat the top-k choices as mutually exclusive. Thus, each of the B 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 $B \times k$ 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

[3]We restrict our attention only to the class of binary projective tree structures. Thus all the candidates are compositions of two contiguous elements. We leave generalization to non-projective structures as a future work.

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 $B \times k$ beam scores. **Step 5:** we truncate the $B \times k$ beams and beam scores into B beams and their corresponding B 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 B beams of items with their B scores. At this point, to get a single item (usually, the items are in vector format with the same dimension), we do a weighted summation with the normalized scores as the weights. Formally, let us say P represents the set of B pairs of beams and their score after the recursive loop. For $(b, s) \in P$, where b is the beam and s is its corresponding score, the final sentence-encoding representation h will be:

$$h = \sum_{(b,s)\in P} \left(\frac{exp(s)}{\sum_{(b,s)\in P} exp(s)} \right) \cdot b \tag{3}$$

This can approximate the expectation of the final hidden state over different possible structures. The algorithm of BT-Cell is described in more details in the appendix (see Algorithm 2)

Note that in BT-Cell we have two non-differentiable top-k operators to return the indices of top-k scores. As such, they will prevent any gradient signals through paths that are not selected. Despite this limitation, BT-Cell 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 perform well for structure-sensitive tasks.

There are two components (scorer and the cell function) whose specific implementations we have not discussed yet. We discuss them below.

Scorer Function: Similar to Choi et al. (2018), we use a linear layer of shape $D \times 1$ (where D is the dimensions of the hidden states) as a scorer.

Recursive Cells: Shen et al. (2019a) proposed a new Transformer-inspired recursive cell (Gated Recursive Cell or GRC for short) that has been crucial for recent state-of-the-art performance in structure-sensitive tasks (Chowdhury & Caragea, 2021). Thus, we also mainly use a variant of GRC as our main recursive cells (we refer to BT-Cell with GRC as BT-GRC). Particularly, we use the GRC function as used by Chowdhury & Caragea (2021) (described in Appendix A.2). We still conduct some experiments with LSTM (Hochreiter & Schmidhuber, 1997) as a cell function (this also leads to the creation of BT-LSTM which is BT-Cell with an LSTM cell).

3.1 EXTENSIONS

While non-differentiable top-k operators can work, they still can be a bottleneck because gradient signals will be recieved only for k beams in a space of combinatorial possibilities. Here, we seek some modifications of the top-k operators for better gradient propagation to non-selected parent compositions and truncated paths. We consider three extensions for that purpose [4]:

Gumbel-BT-GRC: Among the two top-k operators in BT-Cell, here, we focus on the first one (involved in selecting parent composition). In this extension, we replace the plain top-k operator with a straight-through estimation (Bengio et al., 2013) through gumbel top-k (Kool et al., 2019)[5].

BT-GRC + Softpath: Here, we focus on the second top-k operator that is involved in truncating beams. As a concrete case, assume we have some $B \times k$ number of beams of sequences in a list (X) and their corresponding list of scores (S). The target is to keep only the top scoring k beams among

[4] In Appendix A.3, we also discuss about differentiable sorting models

[5] Replacing the second top-k operator with straight-through gumbel top-k tends to lead to instability or worse performance.

them. In this extension, we first choose only the top $k-1$ beams through plain top-k:

$$idx = topk(S, K = k-1). \quad \textit{Top} = \{(X[i], S[i]) \mid i \in idx\} \tag{4}$$

Next, instead of forgetting about the rest of the bottom $B \times k - (k-1)$ beams we construct a convex combination of them based on their normalized scores:

$$\textit{Bottom} = \{(X[i], S[i]) \mid (i \notin idx) \wedge (0 \leq i < |X|) \wedge (i \in \mathbb{N} \cup \{0\})\} \tag{5}$$

$$\textit{softpath} = \left(\sum_{(b,s) \in Bottom} \left(\frac{exp(s)}{\sum_{(b,s) \in Bottom} exp(s)} \right) \cdot b, \sum_{(b,s) \in Bottom} \left(\frac{exp(s)}{\sum_{(b,s) \in Bottom} exp(s)} \right) \cdot s \right) \tag{6}$$

Thus, here, softpath acts as an interpolation of all the bottom scoring paths. Since it is a convex combination of beams based on scores generated by a differentiable layer, it is possible to back-propagate gradients through all the bottom beams which would have been otherwise ignored in the vanilla BT-Cell framework. Finally we add the softpath to the top $k-1$ discretely selected beams, to get the final set of k beam-score pairs (P):

$$P = \textit{Top} \cup \{\textit{softpath}\} \tag{7}$$

In practice, we find it beneficial (in terms of downstream performance) to switch to base BT-Cell (discarding the softpath extention) during inference even if the extension is used during training.

BT-GRC + Gumbelpath: This approach is mostly the same as the softpath extension. The only difference is that we replace the softmax in the softpath extension with a straight-through gumbel softmax making the induced structure discrete during training as well.

4 BASELINES

Here, we discuss the main baselines that we implement.

RecurrentGRC: RecurrentGRC is simply an RNN implemented with GRC as the cell function.

RandomTreeGRC: RandomTreeGRC randomly chooses parent compositions at each iteration (based on uniform probabilities).

BalancedTreeGRC: BalancedTreeGRC is an implementation of RvNN with GRC cell and an enforced binary balanced tree structure. Some prior experiments (Shi et al., 2018) have shown strong results just by using heuristics-based trees like this.

GoldTreeGRC: GoldTreeGRC is an implementation of RvNN with gold tree structures.

GumbelTreeLSTM: This is the straight-through gumbel-tree implementation by Choi et al. (2018).

GumbelTreeGRC: This is same as GumbelTreeLSTM but with GRC as a recursive cell.

BSRP-GRC: BSRP (Beam Shift-Reduce Parser) is based on the model proposed by Maillard & Clark (2018). Maillard & Clark (2018) proposed a similar approach to ours where they apply beam search into a shift-reduce parsing framework instead of easy-first-parsing. We adapt that framework with the GRC cell and call it as BSRP-GRC. More details on this are provided in Appendix A.4.

CYK-GRC: This is the CYK-parsing based model proposed by Maillard et al. (2019) but with a GRC cell.

Ordered Memory: This is a form of memory-augmented RNN simulating certain classes of RvNN functions as proposed by (Shen et al., 2019a). In our implementation, we use the same GELU-activation function based GRC as we use in other models and as was used in Chowdhury & Caragea (2021) for the sake of better comparison.

CRvNN: CRvNN is a special variant of RvNN with a continuoux relaxation over its structural operations as proposed by Chowdhury & Caragea (2021). In our implementation, we ignore some extraneous elements from CRvNN such as transition features and halt penalty which were deemed to have little effect during ablation (Chowdhury & Caragea, 2021).

Model	**Number of Operations**						
	7	8	9	10	11	12	C
With gold trees							
GoldTreeGRC	$98.01_{1.9}$	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*	51	52	51	51	51	48	51
Universal Transformer*	51	52	51	51	51	48	51
ON-LSTM*	91	87	85	81	78	75	60
Self-IRU†	97	95	93	92	90	88	—
RecurrentGRC	$95.35_{.7}$	93.04_{6}	$90.43_{4.9}$	88.48_{6}	$86.57_{5.8}$	$80.58_{1.5}$	$83.17_{5.1}$
BalancedTreeGRC	$83.27_{2.9}$	$77.81_{3.6}$	$72.56_{6.6}$	$67.54_{6.4}$	$63.66_{6.6}$	$57.44_{7.6}$	74.45_{10}
RandomTreeGRC	90.87_{2}	$86.67_{5.1}$	$84.78_{8.2}$	$80.45_{8.2}$	$76.62_{8.5}$	$71.71_{4.7}$	$78.06_{9.7}$
GumbelTreeLSTM	82.25_{12}	77.03_{12}	$74.62_{6.1}$	$69.55_{1.7}$	$67.94_{8.1}$	59.95_{10}	78.20_{11}
GumbelTreeGRC	$96_{8.5}$	93.46_{14}	91.89_{19}	90.33_{22}	88.43_{18}	85.70_{24}	89.34_{29}
CYK-GRC	97.95_{10}	$96.62_{2.3}$	$96.07_{4.6}$	94.67_{11}	$93.44_{8.8}$	$92.54_{9.3}$	77.08_{27}
BSRP-GRC	92.44_{21}	89.37_{18}	85.92_{30}	83.06_{35}	80.63_{33}	74.91_{42}	81.88_{31}
CRvNN	$97.81_{.4}$	$96.9_{3.7}$	$95.99_{2.8}$	$94.51_{2.9}$	$\underline{94.48_{5.6}}$	92.73_{15}	89.79_{58}
Ordered Memory	97.94_{1}	$\mathbf{97.5_{1.6}}$	$\mathbf{96.74_{1.4}}$	94.95_{2}	$93.9_{2.2}$	$93.36_{6.2}$	$\mathbf{94.88_{7}}$
Beam Tree Models with beam size 5 (also without gold trees)							
BT-LSTM	$95.43_{4.8}$	$93.27_{2.5}$	$92.63_{5.3}$	88.55_{10}	$87.85_{8.2}$	84.56_{12}	73.02_{12}
BT-GRC	$97.79_{1.6}$	96.83_{1}	$95.99_{2.4}$	$95.04_{2.3}$	$94.29_{3.8}$	$93.36_{2.4}$	$\underline{94.17_{14}}$
Gumbel-BT-GRC	$97.08_{6.9}$	95.56_{8}	94.14_{11}	92.77_{16}	91.71_{21}	89.84_{18}	86.55_{35}
BT-GRC + Softpath	$\mathbf{98.11_{1.8}}$	$\underline{97.03_{1.4}}$	$\underline{96.49_{1.9}}$	$\underline{95.43_{4.5}}$	$94.21_{6.6}$	$\underline{93.39_{1.5}}$	78.04_{43}
BT-GRC + Gumbelpath	$97.87_{1.9}$	$96.93_{1.2}$	$95.82_{1.5}$	$94.67_{0.6}$	$93.48_{2.9}$	92.58_{9}	83.63_{47}
Beam Tree Models with beam size 2 (also without gold trees)							
BT-GRC	97.93_{2}	$96.63_{2.2}$	$95.95_{3.7}$	$\mathbf{95.45_{3.1}}$	$93.71_{5.5}$	$93.28_{3.9}$	86.97_{38}
BT-GRC + Softpath	$98.00_{.9}$	$96.96_{1.1}$	$\underline{96.49_{1.2}}$	$95.38_{1.8}$	$\mathbf{94.64_{3.4}}$	$\mathbf{93.55_{3.3}}$	81.39_{65}
BT-GRC + Gumbelpath	$97.76_{2.6}$	$96.48_{1.5}$	$96.02_{4.7}$	$94.71_{7.2}$	93.94_{7}	$92.46_{1.1}$	90.87_{50}

Table 1: Mean accuracy and standard deviaton on the Logical Inference for ≥ 7 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, $90_1 = 90 \pm 0.1$

.

5 Experiments and Results

Hyperparameters and other architectural details are in the Appendix A.7.

5.1 Logical Inference

We consider the synthetic testbed for detecting logical relations between sequence pairs as provided by Bowman et al. (2015). Following Tran et al. (2018), we train the models on sequences with $\leq$ 6 operators and test on data with $\geq$ 7 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. We mainly train our BT-Cell models with beam search 5 but we also explore the most promising ones with beam search 2. In Table 1, 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. GRC-based models perform better than comparative LSTM-based models. Interestingly, Gumbel-BT-GRC does not perform as well, possibly due to bias associated with straight-through gradients. In terms of systematicity (split C), OM and BT-GRC (with beam size 5) perform similarly (both above 94%) and much better than the other

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							
LSTM*	71.5_{15}	—	—	—	—	—	—
LSTM-RL*	99.2_{5}	—	—	—	—	—	—
RecurrentGRC	84.05	33.85	20.2	15.1	37.35	30.10	20.7
BalancedTreeGRC	59.4	44.85	43.35	35.70	45.88	45.25	41.95
RandomTreeGRC	70.56	48.70	45.35	37.53	54.8	55.6	49.8
GumbelTreeLSTM	63.08	45.8	42.75	36.8	48.95	49.2	45.35
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	—
BSRP-GRC	70.34	42.4	33.15	26.3	40.15	35.75	29.65
Ordered Memory	99.88	**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
Beam Tree Models with beam size 5 (also without gold trees)							
BT-LSTM	94.11	85.1	83.5	78.8	67.9	44.25	57.85
BT-GRC	99.39	96.15	92.55	86.7	77.1	63.7	67.3
Gumbel-BT-GRC	96.15	75.35	64.25	55.1	58.65	50.4	50.2
BT-GRC + Softpath	**99.92**	99.5	99	97.2	76.05	67.9	71.8
BT-GRC + Gumbelpath	99.84	99.45	**99.2**	**99.4**	79.25	63	72.85
Beam Tree Models with beam size 2 (also without gold trees)							
BT-GRC	94.18	68.2	56.85	50.2	64.45	56.95	55.85
BT-GRC + Softpath	99.69	97.55	95.40	91	75.75	62	66.1
BT-GRC + Gumbelpath	88.56	55.4	51.3	46.1	54.65	52.95	49.55

Table 2: Accuracy on ListOps. * indicates results from Havrylov et al. (2019). † indicates results from Chowdhury & Caragea (2021). LSTM-RL is the model proposed in Havrylov et al. (2019). 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, $90_1 = 90 \pm 0.1$

.

models. Lower beam size or the softpath/gumbelpath extensions hurts systematicity performance for BT-GRC. CYK-GRC shows promise in operator generalization but shows poor systematicity. Gumbel-GRC performs better than fixed tree models (reccurentGRC or balanced trees) or random trees but still far from SOTA.

5.2 LISTOPS

ListOps Nangia & Bowman (2018) is a challenging synthetic tasks requiring solving of nested mathematical operations over list of arguments. We present our results on ListOps in Table 2. 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, because some > 100 length sequences are present in it).

We find that most of the models fail to get good performance even in the near-IID settings. OM is near-perfect in near-IID settings but struggles in higher lengths. Among BT-GRC models, we find that Softpath and Gumbelpath based extensions with beam size 5, are able to get near perfect performance in length generalization. We also find that Softpath extension performs well even with lower beam sizes (beam size 2). This is because Softpath can allow gradient signals to (softly)-truncated paths or beams (which would otherwise be completely truncated).

Model	**SST2**	**SST5**		**IMDB**		
	IID	IID	LG	IID	Con.	Count.
RecurrentGRC	$89.44_{0.5}$	$52.19_{1.5}$	$47.45_{3.9}$	$90.94_{1.2}$	74.86_{28}	82.72_{19}
BalancedTreeGRC	$87.83_{3.7}$	$\underline{52.35_{6.2}}$	$46.37_{3.9}$	$90.71_{0.9}$	74.93_{22}	83.61_{15}
RandomTreeGRC	89.33_{1}	$51.78_{1.2}$	$49.83_{1.2}$	$\underline{91.68_{1.1}}$	74.93_{14}	$82.38_{9.3}$
GumbelTreeLSTM	$88.65_{2.1}$	$\mathbf{52.44_{8}}$	$48.43_{6.3}$	$88.39_{6.9}$	72.27_{15}	80.67_{13}
GumbelTreeGRC	$89.22_{5.6}$	$51.67_{8.8}$	$\mathbf{50.3_{6.3}}$	85.11_{11}	70.63_{21}	81.97_{5}
CYK-GRC	$\mathbf{89.66_{5.8}}$	51.99_{13}	$49.14_{3.3}$	OOM	OOM	OOM
Ordered Memory	$\underline{89.46_{1.6}}$	$52.30_{2.7}$	49.68_{8}	$\mathbf{91.69_{0.5}}$	$\underline{76.98_{5.8}}$	$83.68_{7.8}$
CRvNN	88.58_{12}	51.75_{11}	49.5_{3}	$91.47_{1.2}$	$\mathbf{77.80_{15}}$	$\underline{85.38_{3.5}}$
Beam Tree Models with beam size 5						
BT-LSTM	$88.50_{1.1}$	$50.8_{3.5}$	47.14_{17}	$90.77_{1.3}$	$74.25_{7.7}$	$82.24_{3.5}$
BT-GRC	$88.52_{2.9}$	$52.32_{4.7}$	48.45_{10}	$91.29_{1.2}$	75.07_{29}	82.86_{23}
Gumbel-BT-GRC	$88.82_{.9}$	$51.73_{7.1}$	$49.67_{1.9}$	$86.62_{11.3}$	72.20_{29}	83.27_{28}
BT-GRC + Softpath	$88.34_{.7}$	$51.92_{7.2}$	48.01_{6}	$90.86_{9.3}$	75.68_{21}	84.77_{11}
BT-GRC + Gumbelpath	$88.67_{3.4}$	$51.77_{4.8}$	48.15_{16}	88.39_{18}	72_{39}	82.65_{57}
Beam Tree Models with beam size 2						
BT-GRC	88.94_{6}	$52.14_{2.8}$	$\underline{50.08_{7.4}}$	$91.51_{1.5}$	75.21_{23}	82.51_{23}
BT-GRC + Softpath	$88.52_{4.3}$	$52.2_{4.4}$	$47_{2.5}$	$90.41_{1.8}$	75.89_{22}	$\mathbf{85.45_{15}}$
BT-GRC + Gumbelpath	$89.27_{3.4}$	52.1_{4}	$49.43_{7.2}$	83.24_{19}	63.8_{17}	$70.42_{4.2}$

Table 3: Accuracy on SST2, SST5, and IMDB. Con. refers to Contrast test set, and Count. refers to Counterfactual test set. We report the mean/std of 3. We bold the best resuls and underline the second-best. Subscript represents standard deviation. As an example, $90_1 = 90 \pm 0.1$

.

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 was yet to be considered. In this paper, we also consider what would happen if we increase the number of arguments in the test set beyond the maximum number encountered in the training set. 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 check for both length generalization (it has sequences of length 2000) and argument generalization (it has 10 arguments per operators) simultaneously.

Interestingly, we find all the models including GoldTreeCell to perform poorly on argument generalization. This indicates that the issue goes beyond having the right tree structure. Ordered Memory perform comparatively better than other models (even GoldTreeCell) which may suggest that the memory-augmented RNN style setup in it may be more amenable for argument generalization (although seemingly at the cost of length generalization). BT-GRC-based models (particularly the Softpath or Gumbelpath extensions) still perform generally much better than any other models in argument generalization although it lags behind Ordered Memory. Overall, BT-GRC extensions do better in length generalization whereas Ordered Memory does better in 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.

We find CYK-GRC to be rather expensive to run. Its inference on longer sequences were too slow. We only report its results on lower length sequences. It shows some promise in length generalization but does poorly on argument generalization.

We also run experiments to test depth-generalization performance (see Appendix A.5) alongside some experiments with a Transformer-based architecture (Neural Data Routers (Csordás et al., 2022)). We find that Neural Data Routers (NDRs), despite their careful inductive biases, still struggle with sample efficiency and length generalization. We discuss more on NDRs in Appendix A.6.

5.3 SST AND IMDB

SST Socher et al. (2013) and IMDB Maas et al. (2011) are natural language classification datasets. In the IID splits (standard split) of SST2 and SST5 all the models perform similarly. To better check for OOD performance, we create a length-generalization (LG) split in SST5 (we call this the LG split). Particularly, we only keep sequences of length ≤ 15 in the training set, we keep sequences of length $\leq$ 16-29 in the validation set, and we keep all sequences of length ≥ 30 in the test set. In the LG-split, BalancedTreeCell does particularly worse whereas GumbelTreeGRC and BT-GRC (beam 2) does better than most. Interestingly, RandomTreeGRC is also competitive in this split (however relatively worse in IID split). For IMDB, besides the IID test set, we also test our models on the contrast set from Gardner et al. (2020) and the counterfactual test set from Kaushik et al. (2020). CRvNN gets the best on average in IMDB; however Ordered Memory and BT-GRC with Softpath remains competitive on the OOD sets as well. They generally perform better then the rest. Note that Choi et al. (2018) often found the use of an initial token contextualization layer before using the gumbel-tree-based recursion to be beneficial. We do not explore that modification for the sake of better controlled comparison but that could be a way to further enhance the performance of BT-Cell models in realistic settings.

6 RELATED WORKS

Goldberg & Elhadad (2010) proposed the easy-first algorithm for dependency parsing (which was later adapted to be used as a part of RvNNs (Choi et al., 2018)) and also originally suggested the beam search extension. However, to the best of our knowledge, a concrete formalization of that extension was never made in the context of creating an RvNN-based sequence encoder. Similar to us, Maillard & Clark (2018) use beam search to extend shift-reduce parsing whereas Drozdov et al. (2020) uses beam search to extend CYK-based algorithms. However, in our experiments, we find that the former (BSRP-GRC) is not as effective whereas CYK-based encoding algorithms (with or without beam search) tend to be computationally quite expensive (both temporally and spatially). Instead, BT-Cell based models prove to be more efficient and effective in practice. 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 decoding. The above approaches do not explore extensions over the hard top-k operators in beam search. 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., 2017b; Maillard & Clark, 2018; Shen et al., 2019a; DuSell & Chiang, 2020; 2022), and soft-recursion-based frameworks (Zhang et al., 2021; Chowdhury & Caragea, 2021).

7 CONCLUSION

Overall, we find all three of Ordered Memory, CRvNN, and BT-Cell are competitive with each other; neither being completely superior in all aspects. BT-Cell with Softpath excels in length generalization at ListOps with moderate performance on argument generalization, whereas Ordered Memory excels (comparatively) in argument generalization while struggling in length generalization. CRvNN generally maintains a balanced performance throughout with some edge on IMDB but poor in argument generalization. Softpath and Gumbelpath helps in ListOps and IMDB over base BT-Cell however hurts in systematicity in logical inference. CYK-GRC shows some promise too but is several times more expensive to run while having poor performance in systematicity and argument generalization. The ideal future direction would be to extend to methods which generalize better in all aspects while maintaining computational efficiency and at the same time having a more flexible architecture for handling more free structures like non-projective trees or directed acyclic graphs and also richer classes of languages (DuSell & Chiang, 2022; Del'etang et al., 2022). Another direction to explore would be the linearize the recursion particularly taking inspiration from state space models (Gu et al., 2022) to make these models more scalable.

REFERENCES

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

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

Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In Hal Daumé III and Aarti Singh (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.

Ben Bogin, Sanjay Subramanian, Matt Gardner, and Jonathan Berant. 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.

Samuel R. Bowman, Christopher D. Manning, and Christopher Potts. 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, 2015. CEUR-WS.org.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 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.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. Learning to compose task-specific tree structures. In Sheila A. McIlraith and Kilian Q. Weinberger (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.

Jishnu Ray Chowdhury and Cornelia Caragea. Modeling hierarchical structures with continuous recursive neural networks. In Marina Meila and Tong Zhang (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.

Ronan Collobert, Awni Hannun, and Gabriel Synnaeve. A fully differentiable beam search decoder. In Kamalika Chaudhuri and Ruslan Salakhutdinov (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.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. 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.

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

Gr'egoire Del'etang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Marcus Hutter, Shane Legg, and Pedro A. Ortega. Neural networks and the chomsky hierarchy. *ArXiv*, abs/2207.02098, 2022.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 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.

Andrew Drozdov, Subendhu Rongali, Yi-Pei Chen, Tim O'Gorman, Mohit Iyyer, and Andrew McCallum. 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.

Brian DuSell and David Chiang. 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.

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

Hao Fei, Yafeng Ren, and Donghong Ji. 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.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. 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.

Yoav Goldberg and Michael Elhadad. 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.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 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.

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

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

Serhii Havrylov, Germán Kruszewski, and Armand Joulin. 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.

Jonathan Herzig and Jonathan Berant. 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.

Sepp Hochreiter and Jürgen Schmidhuber. 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.

Xiang Hu, Haitao Mi, Zujie Wen, Yafang Wang, Yi Su, Jing Zheng, and Gerard de Melo. 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.

Eric Jang, Shixiang Gu, and Ben Poole. 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.

Divyansh Kaushik, Eduard Hovy, and Zachary Lipton. 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.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The Gumbel-top-k trick for sampling sequences without replacement. In Kamalika Chaudhuri and Ruslan Salakhutdinov (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.

Yair Lakretz, Theo Desbordes, Dieuwke Hupkes, and Stanislas Dehaene. Causal transformers perform below chance on recursive nested constructions, unlike humans. *ArXiv*, abs/2110.07240, 2021.

Phong Le and Willem Zuidema. 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.

Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. 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.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 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.

Jean Maillard and Stephen Clark. 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.

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

Tsendsuren Munkhdalai and Hong Yu. 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.

Nikita Nangia and Samuel Bowman. 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.

Xuan-Phi Nguyen, Shafiq Joty, Steven Hoi, and Richard Socher. Tree-structured attention with hierarchical accumulation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxK5pEYvr.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Differentiable sorting networks for scalable sorting and ranking supervision. In Marina Meila and Tong Zhang (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.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Monotonic differentiable sorting networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=IcUWShptD7d.

Jordan B. Pollack. 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.

Yikang Shen, Shawn Tan, Arian Hosseini, Zhouhan Lin, Alessandro Sordoni, and Aaron C Courville. Ordered memory. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (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.

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

Yikang Shen, Yi Tay, Che Zheng, Dara Bahri, Donald Metzler, and Aaron Courville. 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.

Haoyue Shi, Hao Zhou, Jiaze Chen, and Lei Li. 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.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 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.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 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.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 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.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

Ke Tran, Arianna Bisazza, and Christof Monz. 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.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. 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.

Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (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.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. In *International Conference on Learning Representations*, 2017a. URL https://openreview.net/forum?id=Skvgqgqxe.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. In *International Conference on Learning Representations*, 2017b. URL https://openreview.net/forum?id=Skvgqgqxe.

Aston Zhang, Yi Tay, Yikang Shen, Alvin Chan, and SHUAI ZHANG. Self-instantiated recurrent units with dynamic soft recursion. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (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.