Recursive Top-Down Production for Sentence Generation with Latent Trees

Shawn Tan[*]
Mila / University of Montreal
tanjings@mila.quebec

Yikang Shen[*]
Mila / University of Montreal
yi-kang.shen@umontreal.ca

Timothy J. O'Donnell
Dept. of Linguistics
Mila / McGill University
Canada CIFAR AI Chair

Alessandro Sordoni
Microsoft Research

Aaron Courville
Mila / University of Montreal
Canada CIFAR AI Chair

Abstract

We model the recursive production property of context-free grammars for natural and synthetic languages. To this end, we present a dynamic programming algorithm that marginalises over latent binary tree structures with N leaves, allowing us to compute the likelihood of a sequence of N tokens under a latent tree model, which we maximise to train a recursive neural function. We demonstrate performance on two synthetic tasks: SCAN (Lake and Baroni, 2017), where it outperforms previous models on the LENGTH split, and English question formation (McCoy et al., 2020), where it performs comparably to decoders with the ground-truth tree structure. We also present experimental results on German-English translation on the Multi30k dataset (Elliott et al., 2016), and qualitatively analyse the induced tree structures our model learns for the SCAN tasks and the German-English translation task.

1 Introduction

Given the hierarchical nature of natural language, tree structures have long been considered a fundamental part of natural language understanding. In recent years, a number of studies have shown that incorporating these structures into deep learning systems can be beneficial for various natural language tasks (Socher et al., 2013; Bowman et al., 2015; Eriguchi et al., 2016).

Various work has explored the introduction of syntactic structures into recursive encoders, either with explicit syntactic information (Du et al., 2020; Socher et al., 2010; Dyer et al., 2016) or by means of unsupervised latent tree learning (Williams et al., 2018; Shen et al., 2019; Kim et al., 2019b). Some attempts at formulating structured decoders are Zhang et al. (2015a) and Alvarez-Melis and Jaakkola (2016) which propose binary top-down tree LSTM architectures for natural language. Chen et al. (2018) proposes a tree-structured decoder for code generation. These methods require ground-truth trees from an external source, and this extra input may not be available for all languages or data sources.

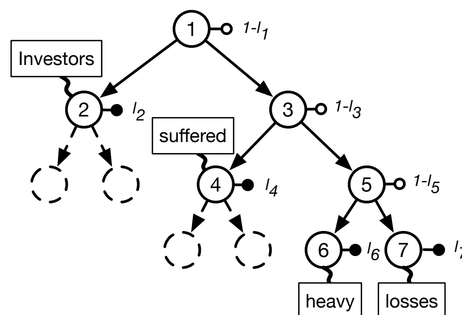

Figure 1: Our generative model is a recursive top-down neural network that recursively splits a root node embedding with some node-dependent probability. When the splitting stops, it emits a word with some probability. The joint probability of a sentence and its associated binary tree is the product of the probability of the tree $(1-l_1)(1-l_3)(1-l_5)l_2l_4l_6l_7$ and the probabilities of the word emitted at its leaves. We devise a novel marginalisation algorithm over binary trees to compute the likelihood of a sentence.

In this work, we propose a tree-based probabilistic decoder model for sequence-to-sequence tasks. Our model generates sentences from a latent tree structure that aims to reflect natural language syntax. The method assumes that each token in a sentence is emitted at the leaves of a full but latent binary tree (Fig. 1). The tree is obtained by recursively producing node embeddings from a root embedding with a recursive neural network. Word emission probabilities are function of the leaf embeddings. We describe a novel dynamic programming algorithm for exact marginalisation over the large number of latent binary trees.

[*]Equal contribution

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2291–2307
November 16 - 20, 2020. ©2020 Association for Computational Linguistics

Our generative model parametrizes a prior over binary trees with a stick-breaking process, similar to the "penetration probabilities" defined in Mochihashi and Sumita (2008). It is related to a long tradition of unsupervised grammar induction models that formulate a generative model of sentences (Klein and Manning, 2001; Bod, 2006; Klein and Manning, 2005).

Unlike more recent bottom-up approaches such as Kim et al. (2019a) which require the inside-outside algorithm (Baker, 1979) to marginalise over tree structures, our approach is top-down and comes with an efficient algorithm to perform marginalisation. Top-down models can be useful, as the decoder is encouraged by design to keep global context while generating sentences (Du and Black, 2019; Gū et al., 2018).

In the next section, we will describe the algorithm that marginalises over latent tree structures under some independence assumptions. We first introduce these assumptions and show that by introducing the notion of successive leaves, we can efficiently sum over different tree structures. We then introduce the details of the recursive architecture used. Finally, we present the experimental results of the model in Section 5.

2 Method

2.1 Generative Process

We assume that each sequence is generated by means of an underlying tree structure which takes the form of a *full binary tree*, which is a tree for which each node is either a leaf or has two children. A sequence of tokens is produced with the following generative process: first, sample a full binary tree T from a distribution $p(T)$. Denote the sets of leaves of T as $L(T)$. Then for each leaf v in $L(T)$, sample a token $x \in \mathcal{V}$, where $\mathcal{V}$ is the vocabulary, from a conditional distribution $p(x|v)$.

Under this model, the probability of a sequence $x_{1:N}$ can be obtained by marginalising over possible tree structures with N leaves:

$$\begin{aligned} p(x_{1:N}) &= \sum_T p(x_{1:N}, T) \\ &= \sum_T p(x_{1:N}|T)p(T) \end{aligned} \tag{1}$$

We assume that the probability of sequences with lengths different from the number of leaves in the tree is 0. Our generative process prescribes that, given the tree structure, the probability of each word is independent of the other words, i.e.:

$$p(x_{1:N}|T) = \prod_{n=1}^{N} p(x_n \mid L_n(T)), \tag{2}$$

where $L_n(T)$ represents the n-th leaf of T. In what follows, we describe an algorithm to efficiently marginalise over possible tree structures, such that the involved distributions can be parametrized by neural networks and can be trained end-to-end by maximizing log-likelihood of the observed sequences. We first describe how we model the prior $p(T)$ and then how to compute $p(x_{1:N})$ efficiently.

2.2 Probability of a full binary tree

We model the prior probability of a full binary tree $p(T)$ by using a branching process similar to the stick-breaking construction, which can be used to model a series of stochastic binary decisions until success (Sethuraman, 1994). In our model, we perform a series of binary decisions at each vertex, starting at the root and branching downwards. Each decision consists in whether to expand the current node by creating two children or not. This binary decision is therefore modeled with a Bernoulli random variable.

Let us define a complete binary tree T_C of depth D_C with vertices $\{v_1, \ldots, v_M\}, M = 2^{D_C+1} - 1$. Each vertex above is associated with a Bernoulli parameter l, $\theta = \{l_1, \ldots, l_{2^{D_C+1}-1}\}$, $l_i \in [0,1]$, modeling its split probability. The probabilities $(1 - l_i)$ are similar to the "penetration probabilities" mentioned in Mochihashi and Sumita (2008). A full binary tree depth $D \le D_C$ is contained in T_C, so we will refer to it as an *internal tree* from here on[1]. See Fig. 1 for an example of two internal trees with three leaves. Its probability can be expressed using parameters l_i as follows. The probability $p(T) = \pi(\mathsf{root})$, where π is defined recursively as:

$$\pi(v_i) = \begin{cases} l_i & \text{if } v_i \in L(T), \\ (1 - l_i) \cdot \pi(\mathsf{left}(v_i)) \cdot \pi(\mathsf{right}(v_i)) & \text{else} \end{cases} \tag{3}$$

where $\mathsf{left}(v_i)$ and $\mathsf{right}(v_i)$ are the left child and right child respectively.

[1] This is not to be confused with the notion of subtrees.

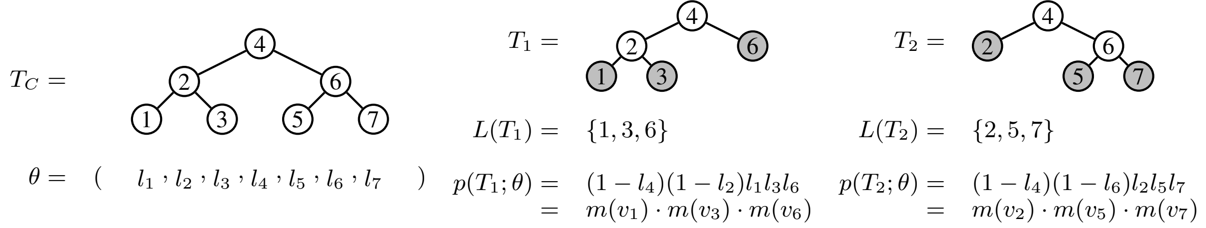

Figure 2: In this figure, root$(T_C) = v_4$. Given $N = 3$, there are two possible trees, T_1 and T_2. The probabilities of the trees can be expressed as the recurrent process described in Equation 3, or as a product of $m(\cdot)$ at the leaf vertices of the internal tree.

2.2.1 Memoizing the value at each vertex

We can compute Eq. (3) efficiently by storing a partial computation for each vertex and multiplying the values at the leaves to get the tree probability:

$$p(T; \theta) = \prod_{n=1}^{N} m(L_n(T)) \tag{4}$$

where $L_n(T)$ denotes the vertex corresponding to the n-th leaf of T. We define this value at the vertex v_i to be $m(v_i)$:

$$m(v_i) = l_i \prod_{v_j \in V_{i \to \text{root}}} (1 - l_j)^{\frac{1}{2^{|V_{i \to j}|}}} \tag{5}$$

where $V_{i \to j}$ denotes the set of vertices in the path from node v_i to node v_j inclusive. These values can be efficiently computed with this top-down recurrence relation:

$$m(v_i) = (\tilde{m}(\text{parent}(v_i)))^{\frac{1}{2}} \cdot l_i \tag{6}$$

$$\tilde{m}(v_i) = (\tilde{m}(\text{parent}(v_i)))^{\frac{1}{2}} \cdot (1 - l_i) \tag{7}$$

where the parent(v_i) is the parent of v_i, and $\tilde{m}(\text{parent}(\text{root})) = 1$. For example, in Fig. 2, $m(1) = (1 - l_4)^{1/4}(1 - l_2)^{1/2} l_1$ and we demonstrate the case for two internal trees with $D = 2$ and $N = 3$ leaves.

We can then use Eq. (2) and Eq. (4) to write the joint probability of a sequence and a tree:

$$p(x_{1:N}, T) = \prod_{n=1}^{N} p(x_n | L_n(T)) \cdot m(L_n(T)) \tag{8}$$

Note that the joint probability factorises as a product over the token probability and the value at the vertex. As we will see later, our method works by traversing the leaves of all possible internal trees, computing the product of the values at the leaves along the way. Therefore, expressing the probability of a full tree as a product of these values ensures that marginalisation stays tractable.

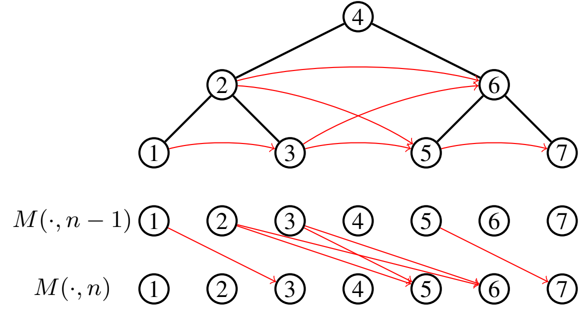

Figure 3: Successive leaf transitions for a tree of $D_C = 2$. The arrows show the possible transitions from each vertex. To enumerate $\mathcal{T}_3$ (trees with 3 leaves) we start at any of the vertices the left boundary (①, ②, or ④), and make 2 transitions (left-to-right arrows) over successive leaves to any vertex in the right boundary (④, ⑥ or ⑦), keeping track of the vertices visited along the way. There are two ways this can be done, which are the examples shown in Figure 2.

2.3 Marginalising over trees

Now that we can compute the probability of a given tree, we need to marginalise over all full binary trees with exactly N leaves. We will denote this formally by the set $\mathcal{T}_N = \{T : |L(T)| = N\}$. The crux of the problem surrounds marginalising over $\mathcal{T}_N$. We know $|\mathcal{T}_N| \leq C_{N-1}$, where C_n is the n-th Catalan number [2], with equality occuring when $N \leq D_C - 1$.

Successive leaves In order to efficiently enumerate all possible internal trees, we define a set of admissible transitions between the vertices of T_C. First, let us define the left and right boundaries of a T_C. Starting from the root node, traversing down the all left children recursively until the leftmost leaf, all vertices visited in this process belong to the left boundary B_l. This notion is similarly defined for all right children in the right boundary B_r. Given a vertex v, we define the *successive leaves* of v as any of the next possible leaves in a internal

[2] https://oeis.org/A000108

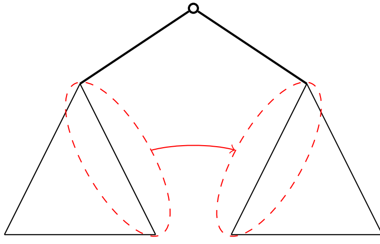

Figure 4: In a binary tree, the left boundary of any right subtree are all successive leaves of the right boundary of its corresponding left subtree.

binary tree in which v is a leaf. As an example, in Figure 3, vertices ⑤ and ⑥ are successive leaves of both vertices ② and ③. Therefore, if we start at a vertex in the left boundary and travel along these allowed transitions until we reach the right boundary, the vertices visited along this path describe the leaves of an internal tree. This notion is independent of the length of any sequence, and a traversal from the left boundary of T_C to the right boundary will induce the leaves of a valid internal T. As an example, in Figure 3, the admissible transitions ① → ③ → ⑥ form a valid internal tree, as well as ① → ③ → ⑤ → ⑦.

To list all pairs of allowed transitions v_i to v_j, we compute the Cartesian product of the vertices in the right boundary of the left subtree and the left boundary of the right subtree, and do this recursively for each vertex. See Figure 4 for an illustration of the concept. The pseudo-code for generating all such transitions in a tree is shown in Appendix B: SUCCESSIVELEAVES. The result of SUCCESSIVELEAVES(root) is the set $\mathcal{S}$, which contains pairs of vertices (v_i, v_j) such that v_j is a successive leaf of v_i. Taking $N-1$ transitions from the left boundary to the right boundary of T_C results in visiting the N leaves of an internal tree. Proof is in Appendix A.

Marginalisation We can use our transitions $\mathcal{S}$ to marginalise over internal trees with N leaves as follows: we fill a table $M(v, n)$ that contains the marginal probability of prefix $x_{1:n}$, where we sum over all partial trees for which vertex v has emitted token x_n:

$$M(v_i, n) = \sum_{T \,:\, L_n(T)=v_i} \prod_{n' \leq n} p(x_{n'} | L_{n'}(T)) \cdot m(L_{n'}(T)) \tag{9}$$

We first initialise the values at $M(v, 1)$ at the left boundary:

$$M(v_i, 1) = \begin{cases} p(x_1|v_i) \cdot m(v_i) & \text{if } v_i \in B_l \\ 0 & \text{else} \end{cases}$$

which should be the state of the table for all prefixes sequences of length 1. Then for $1 < n \leq N$,

$$M(v_i, n) = p(x_n|v_i) \cdot m(v_i) \sum_{v_j \,:\, (v_j, v_i) \in \mathcal{S}} M(v_j, n-1) \tag{10}$$

where we see that Eq. (9) can be recovered by pushing the product $p(x_n|v_i) \cdot m(v_i)$ inside the sum in Eq. (10). The sum describes the situation when vertices have more than one incoming arrow, as depicted in Fig. 3. It should be noted that a large number of these values will be zero, which signify that there are no incomplete trees that end on that vertex. In order to compute the marginalisation over $\mathcal{T}_N$, we have to finally sum over the values at the right boundary:

$$p(x_{1:N}) = \sum_{v_i \in B_r} M(v_i, N) \tag{11}$$

since valid full binary trees must also end on the right boundary of T_C[3]. Note that the values of any trajectory that do not form a full binary tree by $N-1$ iterations, i.e. those that do not reach the right boundary, do not get summed. Another interesting property is that full binary trees with fewer leaves than N would have their trajectories reach the right boundaries much earlier, and those values do not get propagated forward once they do.

2.4 Decoding from the model

During decoding, we can perform the following maximisation based on a modification of the marginalisation algorithm,

$$\arg\max_{x_{1:N}, T} p(x_{1:N}, T). \tag{12}$$

This technique borrows heavily from Viterbi (1967). We perform the same dynamic programming procedure as above, but replacing summations with maximizations, and maintaining a backpointer to the summand that was the highest:

$$M^*(v_i, n) = p(x_n|v_i) \cdot m(v_i) \cdot \max_{(v_j, v_i) \in \mathcal{S}} M^*(v_j, n-1) \tag{13}$$

[3] Since for any full binary tree, every node has either 0 or 2 children, this means that any full binary tree needs to have one leaf in B_r.

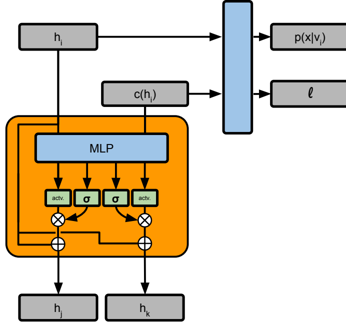

Figure 5: Schema of a single production function application. From the representation $\vec{h}_i$, (1) compute the context vector $c(\vec{h}_i)$ by attending on the encoder, (2) the distribution over word probabilities and the leaf probability parameter l, are computed, (3) apply the $\mathsf{Cell}(\cdot,\cdot)$ function to produce the child representations h_j and h_k. Repeat until the maximum depth is reached.

Since we do not know the length of the sequence being decoded, we need to decide on a stopping criteria. We know that any subsequent multiplication to values in $M(\cdot,\cdot)$ would decrease it, since $p(x_n|v_i) \cdot m(v_i) \leq 1$. Thus, we also know that if the current best full sequence has probability p^*, then if all probabilities at the frontier are $< p^*$, no sequence with a higher probability can be found. We can then stop the search, and return the current best solution. Algorithm 2 in the Appendix C contains the pseudo-code for decoding.

3 Architecture

3.1 Connectionist Tree (CTree) Decoder

We parameterize the emission probabilities $p(x|v_i)$ and the splitting probability at each vertex l_i with a recursive neural network. The neural network recursively splits a root embedding into internal hidden states of the binary tree structures via a *production function* f:

$$(\vec{h}_{\mathsf{left}(v)}, \vec{h}_{\mathsf{right}(v)}) = f(\vec{h}_v, \vec{c}_v) \tag{14}$$

where $\vec{h}_v$ is the embedding of the vertex v and $\vec{c}$ is a generic context embedding that can be optionally vertex dependent and carries external information, e.g. it can be used to pass information in an encoder-decoder setting.

We parameterise $f(\vec{h}_v, \vec{c})$ as a gated two layer neural network with a ReLU hidden layer:

$$\begin{aligned}
\vec{h} &= \mathsf{relu}(W_1\vec{h}_v + U_1\vec{c} + b_1) \\
[\vec{c}_{\mathsf{left}}; \vec{c}_{\mathsf{right}}] &= \mathsf{tanh}(\mathsf{layernorm}(W_2 \cdot \vec{h} + \vec{b}_2)) \\
[\vec{g}_{\mathsf{left}}; \vec{g}_{\mathsf{right}}] &= \mathsf{sigmoid}(W_3 \cdot \vec{h} + \vec{b}_3) \\
\vec{h}_{\mathsf{left}} &= \vec{g}_{\mathsf{left}} \odot \vec{c}_{\mathsf{left}} + (1 - \vec{g}_{\mathsf{left}}) \odot \vec{h}_v \\
\vec{h}_{\mathsf{right}} &= \vec{g}_{\mathsf{right}} \odot \vec{c}_{\mathsf{right}} + (1 - \vec{g}_{\mathsf{right}}) \odot \vec{h}_v
\end{aligned}$$

where $\mathsf{layernorm}$ is layer normalization (Ba et al., 2016). We fix the hidden size to be two times of the dimension of the input vertex embedding.

The splitting probability l_v and the emission probabilities $p(x|v)$ are defined as functions of the vertex embedding:

$$p(x|v) = g_x(\vec{h}_v); \;\; l_v = g_l(\vec{h}_v) \tag{15}$$

The leaf prediction g_l is a linear transform into a two-dimensional output space followed by a softmax. The specific form of the emission probability function g_x can vary with the task. Unless specified, g_x is an MLP.

3.2 Procedural Description

Starting with the root representation $\vec{h}_\rho$ and its eventual contextual information $\vec{c}_\rho$, we recursively apply f. This can be done efficiently in parallel breadth-wise, doubling the hidden representations at every level. We apply g_l at each level, and then Eq. (6) and Eq. (7) to get $m(v)$, which depend only on the parents. We then apply f recursively until a pre-defined depth D_C. We transform all the vertex embeddings using the emission function g_x in parallel, and multiply $p(x \mid v) \cdot m(v)$ for all vertices and words in the vocabulary. We have now computed the sufficient statistics in order to apply the algorithm described in the previous section to compute the marginal probability of the observed sentence.

D_C is a hyper-parameter that depends on memory and time constraints: if D_C is large, the number of representations grows exponentially with it, as does the time for computing the likelihood. If the depth of the latent trees used to generate the data has an upper bound, we can also restrict the class of trees being learned by setting D_C as well.

4 Related Work

Non-parametric Bayesian approaches to learning a hierarchy over the observed data has been proposed in the past (Ghahramani et al., 2010; Griffiths

et al., 2004). These works generally learn a prior on tree-structured data, and assumes a common superstructure that generated the corpus instead of assuming that each observed datapoint may have been produced by a different hierarchical structure. Our generative assumptions are generally stronger but they allow us for tractable marginalisation without costly iterative inference procedures, e.g. MCMC.

Our method shares similarities with the forward algorithm (Baum and Eagon, 1967; Baum and Sell, 1968) which computes likelihoods for Hidden Markov Models (HMM), and CTC (Graves et al., 2006). While the forward algorithm factors in the transition probabilities, both CTC and our algorithm have placed a conditional independence assumption in the factorisation of the likelihood of the output sequence. The inside-outside algorithm (Baker, 1979) is usually employed when it comes to learning parameters for PCFGs. Kim et al. (2019a) gives a modern treatment to PCFGs by introducing Compound PCFGs. In this work, the CFG production probabilities are conditioned on a continuous latent variable, and the entire model is trained using amortized variational inference (Kingma and Welling, 2013). This allows the production rules to be conditioned on a sentence-level random variable, allowing it to model correlations over rules that were not possible with a standard PCFG. However, all co-dependence between the rules can only be captured through the global latent variable. In CTC, Compound PCFGs, and our work, the fact that the dynamic programming algorithm is differentiable is exploited to train the model.

While typical language modelling is done with a left-to-right autoregressive structure, there has been recent work that change the conditional factorisation order (Cho et al., 2019; Yang et al., 2019), and even learn a good factorisation order (Stern et al., 2019; Gu et al., 2019). For hierarchical text generation, Chen et al. (2018) and Zhang et al. (2015b) have attempted to model this hierarchy using ground-truth parse trees from a parser. However, the parser was trained based on parses annotated using rules designed by linguists, which presents two challenges: (1) we may not always have these rules, particularly when it comes to low-resource languages, and (2) it may be possible that the structure required for different tasks are slightly different, enforcing the structure based on a universal parse structure may not be optimal. Jacob et al. (2018) attempts to learn a tree structure using discrete split and merge with REINFORCE (Williams, 1992). However, the method is known to have high variance (Tucker et al., 2017).

There has also been some work that use sequential models for learning a latent hierarchy. Chung et al. (2016) again uses discrete binary sampling units to learn a hierarchy. Shen et al. (2018) enforces an ordering to the hidden state of the LSTM (Hochreiter and Schmidhuber, 1997) that allows the hidden representations to be interpreted as a tree structure. In their follow up work, Shen et al. (2019) encodes sequences to a single vector representation, which we use in this work as the encoder.

5 Experiments

We evaluate our method on three different sequence-to-sequence tasks. Unless otherwise stated, we are using the Ordered Memory (OM) (Shen et al., 2019) as our encoder. Further details can be found in Appendix D.1.

5.1 SCAN

The SCAN dataset (Lake and Baroni, 2017) consists of a set of navigation commands as well as their corresponding action sequences. As an example, an input of `jump opposite left and walk thrice` shoud yield `LTURN LTURN JUMP WALK WALK WALK`. The dataset is designed as a test bed for examining the systematic generalization of neural models. We follow the experiment settings in Bastings et al. (2018), where the different splits test for different properties of generalisation. We apply our model to the 4 experimentation settings and compare our model with the baselines in the literature (See Table 1).

The SIMPLE split has the same data distribution for both the training set and test set. The TURN LEFT split partitions the data so that while `jump left`, and `turn right` would be examples present in the training set, `turn left` are not, but the model must be able to learn from these examples to produce `LTURN` when it sees `turn left` as input.

Lexical Attention Li et al. (2019) and Russin et al. (2019) propose a similar parameterization of the token output distribution based on key-value attention: the hidden states of the decoder (queries) attend on the hidden states of the encoder (keys), but only a-contextual word embeddings are used as

Model	Simple	+ Turn Left	+ Jump	Length
Bastings et al. (2018)	100 ± 0.0	59.1 ± 16.8	12.5 ± 6.6	18.1 ± 1.1
Bastings et al. (2018) - dep	100 ± 0.0	90.8 ± 3.6	0.7 ± 0.4	17.8 ± 1.7
Russin et al. (2019) (LA)	100 ± 0.0	99.9 ± 0.16	78.4 ± 27.4	15.2 ± 0.7
Li et al. (2019) (LA)	99.9 ± 0.0	99.7 ± 0.4	98.8 ± 1.4	20.3 ± 1.1
OM-Seq Cell + LA	99.8 ± 0.0	99.4 ± 1.4	3.5 ± 8.1	20.9 ± 3.1
BiRNN-CTree + LA	99.9 ± 0.0	85.5 ± 2.2	56.5 ± 15.8	19.8 ± 0.0
OM-CTree	99.9 ± 0.1	93.0 ± 7.5	0.1 ± 0.2	40.3 ± 22.5
OM-CTree + LA	100.0 ± 0.0	100.0 ± 0.0	80.1 ± 17.3	44.7 ± 33.5

Table 1: Results on the different splits on the SCAN dataset. The labels are written in the format Encoder-Decoder. CTree + LA is our decoder with lexical attention. Mean and standard deviation are over 10 runs.

walk opposite left after look left twice
lturn
lturn look lturn look
lturn walk

Figure 6: Example of a tree inferred by our model from SCAN.

values. This allows the model to make one-to-one mappings between input token embeddings and output token embeddings (e.g., `jump` in the input always maps to `JUMP` in the output), resulting in huge improvements in performance on the Jump split. We refer to this method as *lexical attention* (LA).

Results We report results in Table 1. Our model performs well on the SCAN splits. Figure 6 shows one tree induced from a model trained on Simple. The resulting parses hint at the model learning to "reuse" some lower-level concepts when `twice` appears in the input, for instance. The two most challenging tasks are Jump and Length splits. In Jump, the input token `jump` only appears alone during training and the model has to learn to use it in different contexts during testing. Surprisingly, this model fails to generalise in the Jump split, suggesting that the capability of our model to perform well on the Jump split may be dependent on the hierarchical decoding as well as the leaf attention.

The Length split partitions the data so that the distribution of output sequences seen in the training set is much shorter than those seen in the test set. Interestingly, our model converges to a solution that results in a 19.8% accuracy in 5 out of the 10 random seeds we use. In the other runs, the model achieves 25% or higher, with 2 runs achieving $> 99\%$ accuracy. The high variance of the model deserves more study, but we suspect in the failure cases, the model does not learn a meaningful concept of `thrice`. Overall, Length requires some generalisation at the structural level during decoding, and has thus far been the most challenging for current sequential models. Given the results, we believe our model has made some improvements on this front.

5.2 English Question Formation

McCoy et al. (2020) proposed linguistic synthetic tasks to test for hierarchical inductive biases in models. One such task is the formation of English questions: `the zebra does chuckle` $\rightarrow$ `does the zebra chuckle ?`. It gets challenging when further relative clauses are inserted into the sentence: `your zebras that don't dance do chuckle`. The heuristic that may work in the first case — moving the first verb to the front of the sentence — would fail, since the right output would be `do your zebras that don't dance chuckle ?`. The task involves having two modes of generation, depending on the final token of the input sentence. If it ends with `DECL`, the decoder simply has to copy the input. If it ends with `QUEST`, the decoder has to produce the question. The authors argue, and provide evidence, that the models that do this task well have syntactic structure. Like SCAN, a generalisation set is included to test for out-of-distribution examples and only the first-word accuracy is reported for the generalisation set.

Results Training our model on this task, we achieve comparable results to their models that are

MODEL	FULL (TEST)	FIRST-WORD (GEN.)
Structure information given		
TREE-TREE	0.96	0.99
SEQ-TREE	0.00	0.90
TREE-SEQ	0.96	0.13
No structure information		
SEQ-SEQ	0.88	0.03
SEQ-CTREE†*	1.00 ± 0.00	0.83 ± 0.19
OM-CTREE†*	1.00 ± 0.00	0.93 ± 0.07

Table 2: English Question Formation results. Our models are annotated with †, and we report mean and standard deviation over 5 runs. Models that use attention are noted with *.

given the syntactic structure of the sentence, after considering the results of the sequential models that they used. The results for this task are reported in Table 2.

5.3 Multi30k Translation

The Multi30k English-German translation task (Elliott et al., 2016), is a corpus of short English-German sentence pairs. The original dataset includes a picture for each pair, but we have excluded them to focus on the text translation task. Our baseline models include an LSTM sequence-to-sequence with attention, Transformer (Vaswani et al., 2017), and a non-autoregressive model LaNMT (Shu et al., 2020). For a fair comparison, we trained all models with negative log-likelihood loss or knowledge distillation (Kim and Rush, 2016) if applicable.

Results As shown in Table 3, our model achieved comparable performance to its autoregressive counterparts, and outperforms the non-autoregressive model. However, we did not observe significant performance improvements as a result of the generalisation capabilities shown in the previous experiments. This suggests further study is needed to overcome remaining issues before deep learning models can really utilise productivity in language.

On the other hand, examples in Figure 7 shows our model does acquire some grammatical knowledge. The model tends to generate all noun phrases (e.g. `an older man`, `a video game`) in separate subtrees. But it also tends to split the sentence before noun phrases. For example, the model splits the sub-clause `while in the air` into two different subtrees. Similarly, previous latent tree induction models (Shen et al., 2017, 2018) also shows a higher affinity for noun phrases compared to adjective and prepositional phrases.

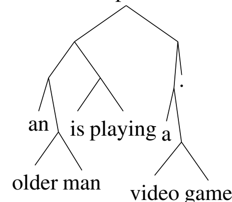

Figure 7: Example of a tree inferred by our model from Multi30K De-En.

	EN-DE		DE-EN	
	PARAM	BLEU	PARAM	BLEU
TRANSFORMER†	69M	33.6	65M	37.8
LSTM†	34M	35.2	30M	38.0
Non-autoregressive				
LANMT‡	96M	26.6	96M	27.9
+ DISTILL	96M	28.5	96M	32.0
OM-CTREE	20M	33.4	20M	34.4
+ DISTILL	20M	34.7	20M	36.6

Table 3: Multi30K results. † — Implemented by OpenNMT (Klein et al., 2017). ‡ — Trained and fine-tuned with the released code `https://github.com/zomux/lanmt`.

6 Conclusion

In this paper, we propose a new algorithm for learning a latent structure for sequences of tokens. Given the current interest in systematic generalisation and compositionality, we hope our work will lead to interesting avenues of research in this direction.

Firstly, the connectionist tree decoding framework allows for different architectural designs for the recurrent function used. Secondly, while the dynamic programming algorithm is an improvement over a naive enumeration over different trees, there is room for improvement. For one, exploiting the sparsity of the $M(\cdot, \cdot)$ table can perhaps result in some memory and time gains. Finally, the need to recursively expand to a complete tree results in exponential growth with respect to the input length.

These results, while preliminary, suggests that the method holds some potential. The experimental results reveal some interesting behaviours that require further study. Nevertheless, we demonstrate that it performs comparably to current algorithms, and surpasses current models in synthetic tasks that have been known to require structure in the models to perform well.

References

David Alvarez-Melis and Tommi S Jaakkola. 2016. Tree-structured decoding with doubly-recurrent neural networks.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

James K Baker. 1979. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132.

Joost Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela. 2018. Jump to better conclusions: Scan both left and right. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 47–55.

Leonard E Baum and John Alonzo Eagon. 1967. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360–363.

Leonard E Baum and George Sell. 1968. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227.

Rens Bod. 2006. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 865–872. Association for Computational Linguistics.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree neural networks for program translation. In *Advances in neural information processing systems*, pages 2547–2557.

Kyunghyun Cho, Hal Daumé III, Sean Welleck, et al. 2019. Non-monotonic sequential text generation. In *Proceedings of the 2019 Workshop on Widening NLP*, pages 57–59.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*.

Wenchao Du and Alan W Black. 2019. Top-down structurally-constrained neural response generation with lexicalized probabilistic context-free grammar. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3762–3771.

Wenyu Du, Zhouhan Lin, Yikang Shen, Timothy J. O'Donnell, Yoshua Bengio, and Yue Zhang. 2020. Exploiting syntactic structure for better language modeling: A syntactic distance approach. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Seattle, Washington.

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

Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. 2016. Multi30k: Multilingual English-German image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74. Association for Computational Linguistics.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 823–833.

Zoubin Ghahramani, Michael I Jordan, and Ryan P Adams. 2010. Tree-structured stick breaking for hierarchical data. In *Advances in neural information processing systems*, pages 19–27.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376.

Thomas L Griffiths, Michael I Jordan, Joshua B Tenenbaum, and David M Blei. 2004. Hierarchical topic models and the nested chinese restaurant process. In *Advances in neural information processing systems*, pages 17–24.

Jetic Gū, Hassan S. Shavarani, and Anoop Sarkar. 2018. Top-down tree structured decoding with syntactic connections for neural machine translation and parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 401–413, Brussels, Belgium. Association for Computational Linguistics.

Jiatao Gu, Qi Liu, and Kyunghyun Cho. 2019. Insertion-based decoding with automatically inferred generation order. *Transactions of the Association for Computational Linguistics*, 7:661–676.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Athul Paul Jacob, Zhouhan Lin, Alessandro Sordoni, and Yoshua Bengio. 2018. Learning hierarchical structures on-the-fly with a recurrent-recursive model for sequences. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 154–158.

Yoon Kim, Chris Dyer, and Alexander M Rush. 2019a. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019b. Unsupervised recurrent neural network grammars. *arXiv preprint arXiv:1904.03746*.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Dan Klein and Christopher D Manning. 2001. Natural language grammar induction using a constituent-context model. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, pages 35–42.

Dan Klein and Christopher D Manning. 2005. Natural language grammar induction with a generative constituent-context model. *Pattern recognition*, 38(9):1407–1419.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proc. ACL*.

Brenden M Lake and Marco Baroni. 2017. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv preprint arXiv:1711.00350*.

Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. 2019. Compositional generalization for primitive substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4284–4293.

R Thomas McCoy, Robert Frank, and Tal Linzen. 2020. Does syntax need to grow on trees? sources of hierarchical inductive bias in sequence-to-sequence networks. *arXiv preprint arXiv:2001.03632*.

Daichi Mochihashi and Eiichiro Sumita. 2008. The infinite markov model. In *Advances in neural information processing systems*, pages 1017–1024.

Jake Russin, Jason Jo, and Randall C O'Reilly. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*.

Jayaram Sethuraman. 1994. A constructive definition of dirichlet priors. *Statistica sinica*, pages 639–650.

Yikang Shen, Zhouhan Lin, Chin-Wei Huang, and Aaron Courville. 2017. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*.

Yikang Shen, Shawn Tan, Arian Hosseini, Zhouhan Lin, Alessandro Sordoni, and Aaron C Courville. 2019. Ordered memory. In *Advances in Neural Information Processing Systems*, pages 5038–5049.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2018. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*.

Raphael Shu, Jason Lee, Hideki Nakayama, and Kyunghyun Cho. 2020. Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. *AAAI*.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, volume 2010, pages 1–9.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. 2017. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2627–2636.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269.

Adina Williams, Andrew Drozdov*, and Samuel R Bowman. 2018. Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association of Computational Linguistics*, 6:253–267.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2015a. Top-down tree long short-term memory networks. *arXiv preprint arXiv:1511.00060*.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2015b. Tree recurrent neural networks with application to language modeling. *CoRR, abs/1511.00060*.

A Proofs

In this context, all trees are rooted.

Definition 1. *A **full binary tree** is a tree where each vertex has either 0 or 2 children.*

Definition 2. *A **complete binary tree** T_C is a tree where each vertex that is not a leaf has 2 children.*

Definition 3. *An **internal tree** T of a complete binary tree T_C is a full binary tree T such that* $\mathsf{root}(T) = \mathsf{root}(T_C)$ *and whose vertices and edges are a subset of T_C.*

Definition 4. *The set $\mathcal{T}(T_C)$ of all internal trees of T_C.*

Definition 5. *$L(T)$ is the ordered set of all leaf nodes in T, starting from the left-most leaf to the right-most leaf. Given a left and right subtree T' and T'' of the tree T,*

$$L(T) = [L(T'); L(T'')]$$

Definition 6. ***Left-most leaf** is $L_1(T)$ and the **right-most leaf** is $L_{|L(T)|}(T)$*

Definition 7. ***Successive leaf transitions** are pairs of vertices (v_i, v_j),*

$$\mathcal{S}(T_C) = \bigcup_{T \in \mathcal{T}(T_C)} \{(L_n(T), L_{n+1}(T)) \ : \ 1 \leq n < |L(T)|\}$$

where $L_n(T)$ is the n-th leaf of T

Definition 8. *A **left boundary** $B_l(T)$ of a tree is the set of vertices induced by recursively visiting the left vertex from the* root.

$$B_l(T) = \left\{v \ : \ v = \mathsf{left}^k(\mathsf{root}), k > 1\right\} \cup \{\mathsf{root}\}$$

*The notion is similarly defined for the **right boundary** B_r.*

Definition 9. *The probability $p(T) = \pi(\mathsf{root})$, where π is defined recursively as:*

$$\pi(v_i) = \begin{cases} l_i & \text{if } v_i \in L(T), \\ (1 - l_i) \cdot \pi(\mathsf{left}(v_i)) \cdot \pi(\mathsf{right}(v_i)) & \text{else} \end{cases}$$

where $\mathsf{left}(v_i)$ *and* $\mathsf{right}(v_i)$ *are the left child and right child respectively.*

Proposition 1. *If T' and T'' are the left and right subtrees of T respectively, and T'_C and T''_C are subtrees of T_C, then*

$$T \in \mathcal{T}(T_C) \rightarrow T' \in \mathcal{T}(T'_C), T'' \in \mathcal{T}(T''_C)$$

Proof.

$$\begin{aligned} \mathsf{root}(T_C) &= \mathsf{root}(T) \\ \mathsf{left}(\mathsf{root}(T)) &= \mathsf{root}(T') \\ &= \mathsf{left}(\mathsf{root}(T_C)) = \mathsf{root}(T'_C) \end{aligned}$$

Since the vertices of T' and T'' are subsets of vertices of T'_C and T''_C respectively, they are each internal trees of T'_C and T''_C. Therefore $T' \in \mathcal{T}(T'_C), T'' \in \mathcal{T}(T''_C)$ □

Proposition 2. *If for all $v_i \in L(T_C) \rightarrow l_i = 1$, then*

$$\sum_{T \in \mathcal{T}(T_C)} p(T) = 1$$

Proof. Base case: T_C is is of depth 0, then $\mathcal{T}(T_C) = \{T\}$, where $T = T_C = \text{root}$., and since root is a leaf $l = 1$.

Inductive case: Let the left and right subtrees of T_C be T'_C and T''_C respectively, and assume $\sum_{T \in \mathcal{T}(T'_C)} p(T) = 1$, and same for T''_C

$$\begin{aligned}
&\sum_{T \in \mathcal{T}(T_C)} p(T) \\
&\quad = l_{\text{root}} + \sum_{T \in (\mathcal{T}(T_C) \setminus \{\text{root}\}} p(T)
\end{aligned}$$

Second term has common factor, since root is not a leaf,

$$\begin{aligned}
&= l_{\text{root}} + (1 - l_{\text{root}}) \sum_{\substack{T' \in \mathcal{T}(T'_C) \\ T'' \in \mathcal{T}(T''_C)}} \pi(\text{root}(T')) \cdot \pi(\text{root}(T'')) \\
&= l_{\text{root}} + (1 - l_{\text{root}}) \sum_{\substack{T' \in \mathcal{T}(T'_C) \\ T'' \in \mathcal{T}(T''_C)}} p(T') \cdot p(T'') \\
&= l_{\text{root}} + (1 - l_{\text{root}}) \left(\sum_{T' \in \mathcal{T}(T'_C)} p(T') \right) \left(\sum_{T'' \in \mathcal{T}(T''_C)} p(T'') \right)
\end{aligned}$$

By the inductive assumption,

$$\begin{aligned}
&= l_{\text{root}} + (1 - l_{\text{root}}) \cdot 1 \cdot 1 \\
&= 1
\end{aligned}$$

□

Proposition 3. *Let*

$$\begin{aligned}
m(v_i) &= (\tilde{m}(\text{parent}(v_i)))^{\frac{1}{2}} \cdot l_i \\
\tilde{m}(v_i) &= (\tilde{m}(\text{parent}(v_i)))^{\frac{1}{2}} \cdot (1 - l_i)
\end{aligned}$$

then,

$$p(T) = \prod_{n=1}^{N} m(L_n(T))$$

Proof. We can write,

$$\prod_{n=1}^{N} m(L_n(T)) = \prod_{v \in V^N} (\tilde{m}(\text{parent}(v)))^{\frac{1}{2}} \cdot \pi(v) \tag{16}$$

where $V^N = L(T)$, and $|V^N| = N$.

If V^1, then $V^1 = \{\text{root}(T)\}$, then $m(\text{root}(T)) = l_{\text{root}(T)}$.

If $|V^N| > 1$, since T is a full binary tree, then there exists at least two vertices $v_i, v_j \in V$ such that

$\mathsf{parent}(v_i) = \mathsf{parent}(v_j) = v_k$. Let $V^{N-1} = (V \setminus \{v_i, v_j\}) \cup \{v_k\}$. Then,

$$\begin{aligned}
&\prod_{v \in V} (\tilde{m}(\mathsf{parent}(v)))^{\frac{1}{2}} \cdot \pi(v) \\
&\quad = (\tilde{m}(\mathsf{parent}(v_k)))^{\frac{1}{2}} \cdot (1 - l_k)\pi(v_i)\pi(v_j) \\
&\qquad \prod_{v \in (V \setminus \{v_i, v_j\})} (\tilde{m}(\mathsf{parent}(v)))^{\frac{1}{2}} \cdot \pi(v) \\
&\quad = (\tilde{m}(\mathsf{parent}(v_k)))^{\frac{1}{2}} \cdot \pi(v_k) \\
&\qquad \prod_{v \in (V \setminus \{v_i, v_j\})} (\tilde{m}(\mathsf{parent}(v)))^{\frac{1}{2}} \cdot \pi(v) \\
&\quad = \prod_{v \in V^{N-1}} (\tilde{m}(\mathsf{parent}(v)))^{\frac{1}{2}} \cdot \pi(v)
\end{aligned}$$

Then V^{N-1} forms another full binary tree T', where v_k is now a leaf, and we can assign $l_k := \pi(v_k)$ Applying this identity, we can repeatedly reduce the number of factors by 1, until we get V^1 ◻

Proposition 4. *If T is an internal tree of T_C,*

$$L_1(T) \in B_l(T_C), L_{|L(T)|}(T) \in B_r(T_C)$$

Proof. If $T = \mathsf{root}$, then the leftmost vertex is root, which is in B_l by definition.

Otherwise, from Definitions 3 & 1 we know that if $\mathsf{left}(v)$ for a given v is ϕ, then v is a leaf. We can then find the left-most leaf of T by recursively calling $v = \mathsf{left}(v)$, until $\mathsf{left}(v) = \phi$. Since all vertices of T are vertices of T_C, and both trees share root, the left-most leaf of T, $v \in B_l$ ◻

The argument for the rightmost vertex is symmetric.

Proposition 5. *Let T'_C and T''_C be left and right subtrees of T_C. Then,*

$$\mathcal{S}(T_C) = \mathcal{S}(T'_C) \cup \mathcal{S}(T''_C) \cup (B_l(T'_C) \times B_r(T''_C))$$

Proof. T_C is a complete tree so the left and right subtree T'_C and T''_C are both complete trees. For any $T \in \mathcal{T}(T_C)$, then by Definition 5, we can find T' and T'' which are internal trees of T'_C and T''_C respectively, such that $L(T) = [L(T'); L(T'')]$. Then,

For $1 \leq n < |L(T')|$,

$$\begin{aligned}
&(L_n(T), L_{n+1}(T)) \\
&\quad = (L_n(T'), L_{n+1}(T')) \in \mathcal{S}(T'_C)
\end{aligned}$$

For $|L(T')| + 1 \leq n < |L(T)|$,

$$\begin{aligned}
&(L_n(T), L_{n+1}(T)) \\
&\quad = (L_{n-|L(T')|}(T''), L_{n-|L(T')|+1}(T'')) \in \mathcal{S}(T''_C)
\end{aligned}$$

by Definition 7.

For $n = |L(T')|$, we know from Prop. 4,

$$\begin{aligned}
L_n(T) &= L_n(T') \in B_r(T'_C) \\
L_{n+1}(T) &= L_1(T'') \in B_l(T''_C)
\end{aligned}$$

Therefore,

$$(L_n(T), L_{n+1}(T)) \in B_l(T'_C) \times B_r(T''_C)$$

◻

B Successive Leaf Construction Algorithm

Algorithm 1 SUCCESSIVELEAVES

```
Input: vertex v_i
Output: successive leaf transitions S = {(v_j, v_k), ...}
Output: left boundary B_l = {i, ...}
Output: right boundary B_r = {i, ...}
if v_i is a leaf then
    S ← {}
    B_l, B_r ← {v_i}, {v_i}
else
    S', B'_l, B'_r ← SUCCESSIVELEAVES(left(v_i))
    S'', B''_l, B''_r ← SUCCESSIVELEAVES(right(v_i))
    S ← S' ∪ S'' ∪ (B'_r × B''_l)
    B_l ← B'_l ∪ {v_i}
    B_r ← B''_r ∪ {v_i}
end if
```

C Decoding Algorithm

Algorithm 2 DECODEJOINT

Input: $[p(\mathbf{x}|v_1), \ldots p(\mathbf{x}|v_{|V|})]$
Output: $x^*_{1:N^*}$
for all $v_i \in V$ **do**
  $m^*_{\text{arg}}(v_i) \leftarrow \arg\max_x p(\mathbf{x} = x|v_i)$ {Initialise}
  $m^*(i) \leftarrow \max_x p(\mathbf{x} = x|v_i)$
end for
$n \leftarrow 1$
for all $v_i \in B_l$ **do**
  $M^*(v_i, 1) \leftarrow m^*(v_i)$
end for
while $\max_{v \in V} M^*(v, n) \geq p^*$ **do**
  if $\max_{v_i \in B_r} M^*(v_i, n) > p^*$ **then**
    $N^* \leftarrow t$ {Compute current best}
    $v^* \leftarrow \arg\max_{v_i \in B_r} M^*(v_i, N^*)$
    $x^* \leftarrow [m^*_{\text{arg}}(v^*, N^*)]$
  end if
  $t \leftarrow t + 1$
  for all $v_i \in V$ **do**
    $M^*(v_i, n) \leftarrow m^*(v_i) \cdot \max_{v_j|(v_j,v_i) \in \mathcal{S}} M^*(v_j, n-1)$
    $M^*_{\text{arg}}(v_i, n) \leftarrow \arg\max_{v_j|(v_j,v_i) \in \mathcal{S}} M^*(v_j, n-1)$
  end for
end while
for $t \leftarrow N^*$ to 2 **do**
  $v^* \leftarrow M^*_{\text{arg}}(v^*, t)$ {Backtrace}
  $x^* \leftarrow [m^*_{\text{arg}}(v^*, t)].x^*$
end for

D Experiments

D.1 Encoder

Before the embeddings are fed into the OM, we first produce contextualised embeddings, by first feeding it into a one layer bidirectional Gated Recurrent Unit (GRU; Cho et al. 2014). We then expose the following representations from the encoder to the decoder:

Encode_ρ — Final representation computed by OM. Can be thought of as the root representation.

Encode_ι — Intermediate states ($\hat{M}_1 \ldots \hat{M}_S$) concatenated. Can be thought of as the representations of the internal nodes *and* the leaves.

Encode_ℓ — Input representations to the OM. Can be thought of as the representation of the leaves.

Encode_{ce} — Contextualized embeddings from the GRU.

Encode_e — Embeddings fed to the GRU.

We also use the $\mathsf{Cell}(\cdot, \cdot)$ function as defined in the paper.

D.2 SCAN sample trees

run around left thrice after turn left thrice

Figure 8: Erroneous tree example from the model trained on the LENGTH split.

D.3 Multi30k Translation Sample Trees

Ein älterer Mann spielt ein Videospiel.

Ein Mädchen an einer Küste mit einem Berg im Hintergrund.

Ein Junge greift sich ans Bein während er in die Luft springt.

Figure 9: Trees found by our model from Multi30K De-En.