Hierarchical Phrase-based Sequence-to-Sequence Learning

Bailin Wang
MIT
bailinw@mit.edu

Ivan Titov
University of Edinburgh
University of Amsterdam
ititov@inf.ed.ac.uk

Jacob Andreas
MIT
jda@mit.edu

Yoon Kim
MIT
yoonkim@mit.edu

Abstract

We describe a neural transducer that maintains the flexibility of standard sequence-to-sequence (seq2seq) models while incorporating hierarchical phrases as a source of inductive bias during training and as explicit constraints during inference. Our approach trains two models: a discriminative parser based on a bracketing transduction grammar whose derivation tree hierarchically aligns source and target phrases, and a neural seq2seq model that learns to translate the aligned phrases one-by-one. We use the same seq2seq model to translate at all phrase scales, which results in two inference modes: one mode in which the parser is discarded and only the seq2seq component is used at the sequence-level, and another in which the parser is combined with the seq2seq model. Decoding in the latter mode is done with the cube-pruned CKY algorithm, which is more involved but can make use of new translation rules during inference. We formalize our model as a source-conditioned synchronous grammar and develop an efficient variational inference algorithm for training. When applied on top of both randomly initialized and pretrained seq2seq models, we find that both inference modes perform well compared to baselines on small scale machine translation benchmarks.

1 Introduction

Despite the improvements in performance and data-efficiency enabled by recent advances in pre-training, standard neural sequence-to-sequence (seq2seq) models can still fail to model the hierarchical structure of sentences and be brittle with respect to novel syntactic structures (Lake and Baroni, 2018; Kim and Linzen, 2020; Weißenhorn et al., 2022). It has moreover not been clear how to incorporate explicit constraints such as translation rules (e.g., for translating idioms) into these black-box models without changing the underlying architecture. Classic grammar- and automaton-based approaches are well-suited for capturing hierarchical structure and can readily incorporate new rules, but have trouble with examples that do not conform exactly to the symbolic specifications. This paper describes a neural transducer that maintains the flexibility of neural seq2seq models but still models hierarchical phrase alignments between source and target sentences, which have been shown to be useful for transduction tasks (Chiang, 2005; Wong and Mooney, 2007, inter alia). This transducer bottoms out in an ordinary neural seq2seq model and can thus take advantage of pre-training schemes like BART (Lewis et al., 2020; Liu et al., 2020) and T5 (Raffel et al., 2020; Xue et al., 2021).

The transducer consists of two components learned end-to-end: a discriminative parser based on a bracketing transduction grammar (BTG; Wu, 1997) whose derivation tree hierarchically aligns
source and target phrases, and a neural seq2seq model that learns to translate the aligned phrases one-by-one. Importantly, a single seq2seq model is trained to translate phrases at all scales, including at the sentence-level, which results in two inference modes. In the first mode, we discard the parser and simply decode with the resulting seq2seq model at the sequence level, which maintains fast inference but still makes use of a seq2seq model that has been exposed to (and regularized by) latent hierarchical phrase alignments. This approach can be seen a form of learned data augmentation (Akyürek et al., 2021), wherein a model is guided away from wrong generalizations by additionally being trained on crafted data that coarsely express the desired inductive biases. In the second mode, we view the combined parser and seq2seq model as a source-conditioned neural synchronous grammar to derive a set of synchronous grammar rules along with their scores (a “neural phrase table”), and decode with a modified version of the classic cube-pruned CKY algorithm (Huang and Chiang, 2005). While more involved, this decoding scheme can incorporate explicit constraints and new translation rules during inference.

We formulate our approach as a latent variable model and use variational learning to efficiently maximize a lower bound on the log marginal likelihood. This results in an intuitive training scheme wherein the seq2seq component is trained on phrase tables derived from hierarchical alignments sampled from a variational posterior synchronous parser, as shown in Figure 1. When applied on top of both randomly initialized and pre-trained seq2seq models across various small-scale machine translation benchmarks, we find that both modes improve upon baseline seq2seq models.

2 Approach

Notation. We use $x$, $y$ to denote the source/target strings, $x_{i:k}$, $y_{l:r}$ to denote the source/target spans, and $x_i$ to denote the token with index $i$.

2.1 A Seq2Seq-Based Synchronous Grammar

The inversion transduction grammar (ITG), introduced by Wu (1997), defines a synchronous grammar that hierarchically generates source and target strings in tandem. The bracketing transduction grammar (BTG) is a simplified version of ITG with only one nonterminal, with the primary goal of modeling alignments between source and target words. We propose to use a variant of BTG which defines a stochastic monolingual grammar over target strings conditioned on source strings and can use a neural seq2seq model to generate phrases.

As a motivating example, consider the following English-Chinese example from Shao et al. (2018):\footnote{The phrase-level translations are: 他们 → They, 在 桥 上 → were on the bridge, 谈笑 风生 → laughing it up.}

They were laughing it up on the bridge

A translator for this sentence is faced with several problems: first, they must identify the appropriate unit of translation (number of phrases); two, they must reorder phrases if necessary; finally, they must be able translate phrase-by-phrase, taking into account phenomena such as idioms whose translations cannot be obtained by stitching together translations of subparts (e.g., 谈笑 风生 → laughing it up), while also making sure the final translation is fluent as a whole. We model this process with a source-conditioned synchronous grammar whose target derivation probabilistically refines the above process. Informally, this grammar first samples the number of segments to generate on the target ($n$), then segments and reorder source phrases ($d_{tree}$), and finally translates phrase-by-phrase while still conditioning on the full source sentence for fluent translations ($d_{leaf}$). This is shown in Figure 2.

More formally, this monolingual grammar over the target can be represented by a tuple

\[ G[x] = \left( \text{Root}, \{ T^n, S^n, I^n \}_{n=1}^{|x|}, C, \Sigma, R[x] \right), \]

where Root is the distinguished start symbol, $\{ T^n, S^n, I^n \}_{n=1}^{|x|}$ are nonterminals where $n$ denotes the number of segments associated with the nonterminal (to be explained below), $|x|$ is the source sentence length, $C$ is a special nonterminal that emits phrases, and $\Sigma$ is the target alphabet. The rule set $R[x]$ is given by a fixed set of context-free production rules. For the root node we have

\[ \forall n \in \{ 1, 2 \ldots |x| \}, \]

\[ \text{Root} \rightarrow T^n, \quad T^n \rightarrow S^n[x], \quad T^n \rightarrow I^n[x], \]

where $S, I$ represent the Straight and Inverted nonterminals. Binary nonterminal rules are given by,

\[ \forall i, j, k \in \{ 0, 1 \ldots |x| \} \quad \text{s.t.} \quad i < j < k \]

\[ S^n[x_{i:k}] \rightarrow I^l[x_{i:j}]S^r[x_{j:k}], \]

\[ S^n[x_{i:k}] \rightarrow I^l[x_{i:j}]I^r[x_{j:k}], \]

\[ I^n[x_{i:k}] \rightarrow S^l[x_{j:k}]I^r[x_{i:j}], \]

\[ I^n[x_{i:k}] \rightarrow S^l[x_{j:k}]S^r[x_{i:j}], \]

where each nonterminal is indexed by (i.e., aligned
Figure 2: Our seq2seq-based synchronous grammar defines a distribution over target derivations $d$ given source string $x$ by decomposing $p(d|x)$ into three components, $p(d|x) = p(n|x)p(d_{\text{tree}}|x,n)p(d_{\text{leaf}}|x,d_{\text{tree}})$. We first sample the number of segments in the target $n$ according to $p(n|x)$. Then we sample the tree topology $d_{\text{tree}} \sim p(d_{\text{tree}}|x,n)$, which gives a segmentation and reordering of the source sentence. Finally, the leaf derivations are given by translating the segmented and reordered source phrases one-by-one with a seq2seq model $p(d_{\text{leaf}}|x,d_{\text{tree}})$, where the phrase-by-phrase translations are contextualized against the entire source sentence. We show some example trees $d_{\text{tree}}$ and leaf derivations $d_{\text{leaf}}$ for various values of $n$. Note that each $n$ could have multiple segmentations and reorderings.

$p(n|x)$. For the distribution over the number of target segments we use a geometric distribution,

$$p(n|x) = \lambda(1-\lambda)^{n-1},$$

where $0 < \lambda < 1$, $1 \leq n < |x|$. This sets the probability of the upper bound $n = |x|$ to be $(1-\lambda)^{|x|-1}$. This distribution is implemented by the following configuration of the grammar:

$$\pi(\text{Root} \rightarrow T^n) = p(n|x).$$

The geometric distribution favors fewer target segments, which aligns with one of our goals which is to be able to use the resulting model as a regular seq2seq system (i.e., forcing Root $\rightarrow T^1$).

$p(d_{\text{tree}}|x,n)$. For the distribution over source segmentations and reorderings, we use a discriminative CRF parser whose scores are given by neural features over spans. The score for rule $R \in \mathcal{R}$, e.g., $S^n[x_{i:k}] \rightarrow I^1[x_{i:j}] A^r[x_{j:k}]$, is given by,

$$\pi(R) = \exp \left( e_A^\top \tilde{h}_{i:j} \tilde{H}_{i:j} \right),$$

where $A \in \{S,I\}$, $\tilde{h}_{i:j}$ and $\tilde{H}_{i:j}$ are the span features derived from the contextualized word representations of tokens at $i,j,k$ from a Transformer encoder as in (Kitaev and Klein, 2018), $f$ is an MLP that takes into two span features, and $e_A$ is a one-hot vector that extracts the corresponding scalar score. (Hence we use the same score all $n$).

These scores are globally normalized, i.e.,

$$p(d_{\text{tree}}|x,n) \propto \prod_{R \in \mathcal{R}_{\text{tree}}} \pi(R),$$

to obtain a distribution over $d_{\text{tree}}$, where $R$ are the rules in $d_{\text{tree}}$. Algorithms 1, 2 in appendix C give the dynamic programs for normalization/sampling.

$p(d_{\text{leaf}}|x,d_{\text{tree}})$. Finally, for the distribution over the leaf translations, the phrase transduc-
tion rule probabilities $p(C[x_{i:k}] \rightarrow v)$ are parameterized with a (potentially pretrained) neural seq2seq model. For the seq2seq component, instead of just conditioning on $x_{i:k}$ to parameterize $p_{\text{seq2seq}}(v|x_{i:k})$, in practice we use the contextualized representation of $x_{i:k}$ from a Transformer encoder that conditions on the full source sentence. Hence, our approach can be seen learning contextualized phrase-to-phrase translation rules. Specifically, for a length-$m$ target phrase we have,

$$p(d_{\text{leaf}}|x, d_{\text{tree}}) = \prod_{R \in d_{\text{leaf}}} \pi(R)$$

$$\pi(C[x_{i:k}] \rightarrow v) = \prod_{t=1}^{m} p_{\text{seq2seq}}(v_t | v_{<t}, h_{i:k}),$$

$\ h_{i:k} = \text{Encoder}(x_{i:k}),$

where $h_{i:k}$ refers to the contextualized representation of $x_{i:k}$ that is obtained after passing it through a Transformer encoder. The decoder attends over $h_{i:k}$ and $v_{<t}$ to produce a distribution over $v_t$. The two “exit” rules are assigned a score $\pi(A^1[x_{i:k}] \rightarrow C[x_{i:k}]) = 1$ where $A \in \{S, I\}$.

**Remark.** This grammar is a mixture of both locally and globally normalized models. In particular, $p(n|x)$ and $p(d_{\text{leaf}}|x, d_{\text{tree}})$ are locally normalized as the scores for these rules are already probabilities. In contrast, $p(d_{\text{tree}}|x, n)$ is obtained by globally normalizing the score of $d_{\text{tree}}$ with respect to a normalization constant $Z_{\text{tree}}(x, n)$. In the appendix we show how to convert the unnormalized scores $\pi$ in the globally normalized model into normalized probabilities $\tilde{\pi}$ (Algorithm 1), which will be used for CKY decoding in Section 2.4.

We also share the parameters between components whenever possible. For example, the encoder of the seq2seq component is the same encoder that is used to derive span features for $d_{\text{tree}}$. The decoder of the seq2seq component is used as part of the variational distribution when deriving span features for the variational parser. As we will primarily be working with small datasets, such parameter sharing will provide implicit regularization to each of the components of our grammar and mitigate against overfitting. We provide the full parameterization of each component in appendix A.

2.3 Learning

Our model defines a distribution over target trees $d = (n, d_{\text{tree}}, d_{\text{leaf}})$ (and by marginalization, target strings $y$) conditioned on a source string $x$,

$$p(y|x) = \sum_{d \in D(x,y)} p(d|x)$$

$$= \sum_{d \in D(x,y)} p(n|x)p(d_{\text{tree}}|x, n)p(d_{\text{leaf}}|x, d_{\text{tree}}),$$

where $D(x,y)$ is the set of trees such that yield$(D(x,y)) = (x, y)$. While this marginalization can theoretically be done in $O(L^7)$ time where $L = \min(|x|, |y|)$,\(^4\) we found it impractical in practice. We instead use variational approximations to the posteriors of $n$, $d_{\text{tree}}$, $d_{\text{leaf}}$ and optimize a lower bound on the log marginal likelihood whose gradient estimator can be obtained in $O(L^3)$ time. Each of the three variational distributions $q$ is analogous to the three distributions $p$ introduced previously, but additionally conditions on $y$.

$q(n|x, y)$. Similar to $p(n|x)$, we use a geometric distribution to model $q(n|x, y)$. The only difference is that we have $1 \leq n \leq \min(|x|, |y|)$.

$q(d_{\text{tree}}|x, y, n)$. This is parameterized with neural span scores over $x$ in the same way as in $p(d_{\text{tree}}|x, n)$, except that the span features also condition on the target $y$. That is, $h_{t:j}^y, h_{i:k}^y$ are now the difference features of the decoder’s contextualized representations over $x$ from a Transformer encoder-decoder, where $y$ is given to the encoder.\(^5\) The dynamic programs for normalization/sampling in this CRF is the same as in $p(d_{\text{tree}}|x, n)$.

$q(d_{\text{leaf}}|x, y, d_{\text{tree}})$. Observe that conditioned on $(x, y, d_{\text{tree}})$, the only source of uncertainty in $d_{\text{leaf}}$ is $C[x_{i:k}] \rightarrow y_{ax}$ for spans $y_{ax}$. That is, $q(d_{\text{leaf}}|x, y, d_{\text{tree}})$ is effectively a hierarchical segmentation model with a fixed topology given by $d_{\text{tree}}$. The segmentation model can be described by a probabilistic parser over the following grammar,

$$G[y, d_{\text{tree}}] = \langle \text{Root}, \{T^n\}_{n=1}^m, \mathcal{R}[y, d_{\text{tree}}] \rangle,$$

where rule set $\mathcal{R}[y, d_{\text{tree}}]$ includes (here $m = l+r$),

$$\text{Root} \rightarrow T^m[y], T^m[y_{ax}] \rightarrow T^l[y_{ab}]T^r[y_{bc}]$$

where the second rule exists only when the rule like $A^m \rightarrow B^lC^r$ is in $d_{\text{tree}}$. We use span features over

\(^3\)Locally normalized parameterization of $d_{\text{tree}}$ would require $\pi$ to take into account the number of segments $n$ (and $l, r$), which would lead to a more complex rule-scoring function. In our parameterization $\pi(R)$ does not depend on the number of segments, and $n, l, r$ only affect $p(d_{\text{tree}}|x, n)$ via the normalization $Z_{\text{tree}}(x, n)$.

\(^4\)For each $n \in [L_\text{max}]$, we must run the $O(n^6)$ bitext inside algorithm, so the total runtime is $\sum_{n=1}^{L_\text{max}} O(n^6) = O(L^7)$.

\(^5\)We use this parameterization (instead of, e.g., a bidirectional Transformer encoder over the concatenation of $x$ and $y$) in order to take advantage of pretrained multilingual encoder-decoder models such as mBART.
y for parameterization similarly to \( p(d_{\text{tree}}|x,n) \). Algorithms 3, 4 in appendix C give the normalization/sampling dynamic programs for this CRF.

**Optimization.** With the variational distributions in hand, we are now ready to give the following evidence lower bound (ignoring constant terms),

\[
\log p(y|x) \geq \mathbb{E}_{q(n|x,y)} \left[ \mathbb{E}_{q(d_{\text{tree}}|x,y,d_{\text{tree}})} \log p(d_{\text{tree}}|x,d_{\text{tree}}) \right] + \mathbb{H}[q(d_{\text{leaf}}|x,y,d_{\text{tree}})]
\]

\[
- \text{KL}[q(d_{\text{tree}}|x,y,n) \| p(d_{\text{tree}}|x,n)]
\]

(The KL between \( q(n|x,y) \) and \( p(n|x) \) is a constant and hence omitted.) See appendix B for the derivation. While the above objective seems involved, in practice it results in an intuitive training scheme. Let \( \theta \) be the grammar parameters and \( \phi \) be the variational parameters. Training proceeds as follows:

1. Sample number of target segments,
\[
n \sim q(n|x,y).
\]
2. Sample trees from variational parsers,
\[
d'_{\text{tree}} \sim q(d_{\text{tree}}|x,y,n),
d'_{\text{leaf}} \sim q(d_{\text{leaf}}|x,y,d'_{\text{tree}}),
\]
and obtain the set of hierarchical phrase alignments (i.e., phrase table) \( A \) implied by \( (d'_{\text{tree}},d'_{\text{leaf}}) \).
3. Train seq2seq model by backpropagating the leaf likelihood \( \log p(d'_{\text{leaf}}|x,d'_{\text{tree}}) \) to the seq2seq model,
\[
\sum_{(x,i,k,y:a,c) \in A} \nabla_{\theta} \log p_{\text{seq2seq}}(y\backslash a|x;i;k).
\]
4. Train variational parsers by backpropagating the score function and KL objectives with respect to \( q(d_{\text{tree}}|x,y,n) \),
\[
(\log p(d'_{\text{leaf}}|x,d'_{\text{tree}}) + \mathbb{H}[q(d_{\text{leaf}}|x,y,d'_{\text{tree}})])
\times \nabla_{\phi} \log q(d'_{\text{tree}}|x,y,n) - \nabla_{\phi} \text{KL}[\cdot],
\]
and also with respect to \( q(d_{\text{leaf}}|x,y,n) \),
\[
\log p(d'_{\text{leaf}}|x,d'_{\text{tree}}) \times \nabla_{\phi} \log q(d'_{\text{leaf}}|x,y,d'_{\text{tree}})
\]
\[
+ \nabla_{\phi} \mathbb{H}[q(d'_{\text{leaf}}|x,y,d'_{\text{tree}})].
\]
5. Train source parser by backpropagating \(-\nabla_{\phi} \text{KL}[\cdot]\) to \( p(d_{\text{tree}}|x,n) \).

For the score function gradient estimators we also employ a self-critical control-variate (Rennie et al., 2017) with argmax trees. The KL and entropy metrics are calculated exactly with inside algorithm with the appropriate semirings (Li and Eisner, 2009). This training scheme is shown in Figure 1. See Algorithms 5, 6 in appendix C for the entropy/KL dynamic programs.

**Complexity.** The above training scheme reduces time complexity from \( O(L^7) \) to \( O(L^3) \) since we do not marginalize over all possible number of segments and instead sample \( n \), and further decompose the derivation tree into \( (d_{\text{tree}},d_{\text{leaf}}) \) and sample from CRF parsers, which takes \( O(L^3) \). The KL/entropy calculations are also \( O(L^3) \).

### 2.4 Inference: Two Decoding Modes

During inference, we can use the seq2seq to either directly decode at the sentence-level (i.e., setting \( n = 1 \) and choosing \( \text{Root} \to T^1 \)), or generate phrase-level translations and compose them together. The first decoding strategy maintains the efficiency of seq2seq, and variational learning can be seen as a structured training algorithm that regularizes the overly flexible seq2seq model with latent phrase alignments. The second decoding strategy takes into account translations at all scales and can further incorporate constraints, such as new translation rules during decoding.

**Seq2Seq decoding.** Setting \( n = 1 \), decoding in \( G[x] \) this case is reduced to the standard sentence-level seq2seq decoding for the terminal rules \( C[x_0,|x|] \to v \) with beam search. We call this variant of the model BTG-1 Seq2Seq.

**CKY decoding.** Here we aim to find \( y \) with maximal marginal probability \( \text{argmax}_y \sum_{d \in T(x,y)} p(d|x) \). This requires exploring all possible derivations licensed by \( G[x] \). We employ CKY decoding algorithm based on the following recursion:

\[
C_{i:k}[v_1,v_2,S^n] = \sum_{l=1,r=n-l}^{n-1} \{ \\
C_{i,l}[v_1,I^l] \cdot C_{j:k}[v_2,S^r] \cdot \tilde{\pi}(S^n \to I^l S^r) + \\
C_{i:l}[v_1,I^l] \cdot C_{j:k}[v_2,\Gamma^r] \cdot \tilde{\pi}(S^n \to I^l \Gamma^r) \}
\]

\[
C_{l:k}[v_1,v_2,I^n] = \sum_{l=1,r=n-l}^{n-1} \{ \\
C_{j:k}[v_1,S^l] \cdot C_{j:k}[v_2,v_2,I^r] \cdot \tilde{\pi}(I^n \to S^l I^r) + \\
C_{j:k}[v_1,S^l] \cdot C_{j:k}[v_2,\Gamma^r] \cdot \tilde{\pi}(I^n \to S^l \Gamma^r) \}
\]

---

6We share the Transformer encoder/decoder between \( p_\theta(\cdot) \) and \( q_\phi(\cdot) \), and therefore the only parameters that are different between \( \theta \) and \( \phi \) are the MLPs’ parameters for scoring rules in the CRF parsers.
where \( C_{i,k}[v, A] \) stores the score of generating target phrase \( v \) based on source phrase \( x_{i:k} \) with non-terminal \( A \). In the above we abbreviate \( \tilde{\pi}(S^v \rightarrow I^v S^r) \) to mean \( \tilde{\pi}(S^{v_i} \rightarrow I^{v_j} S^{v_{i,j,k}}) \) since \( i, j, k \) are implied by the \( C \) terms that this quantity is multiplied with. Here \( v, v_1, v_2 \) denotes target phrases, and \( v_1 v_2 \) refers to the concatenation of two phrases. The chart is initialized by phrase pairs that are generated based on seq2seq.

\[
\begin{align*}
C_{i,k}[v, S^1] &= p_{\text{seq2seq}}(v | x_{i:k}) \cdot \pi(S^1 \rightarrow C), \\
C_{i,k}[v, I^1] &= p_{\text{seq2seq}}(v | x_{i:k}) \cdot \pi(I^1 \rightarrow C)
\end{align*}
\]

The final prediction is based on the chart item for the root node computed via:

\[
\begin{align*}
C[v, T^n] &= \mathbb{C}_{0,[x]}[v, S^n] \cdot \tilde{\pi}(T^n \rightarrow S^n) + \\
&\mathbb{C}_{0,[x]}[v, I^n] \cdot \tilde{\pi}(T^n \rightarrow I^n) \\
\mathbb{C}[v, \text{Root}] &= \sum_{n=1} C[v, T^n] \cdot \pi(\text{Root} \rightarrow T^n).
\end{align*}
\]

**Cube pruning.** Exact marginalization is still intractable since there are theoretically infinitely many phrases that can be generated by a seq2seq model (e.g., there is \( C_{i,k}[v, S^1] \) for each possible phrase \( v \in \Sigma^+ \)). We follow the cube pruning strategy from Huang and Chiang (2005) and only store the top-\( K \) target phrases within each chart item, and discard the rest. During the bottom-up construction of the chart, for \( A \in \{S, I\} \), \( C_{i,k}[v, A^1] \) only stores the \( K \) phrases generated by beam search with size \( K \), and \( C_{i,k}[v, A^n] \) constructs at most \( 2 \cdot (n - 1) \cdot K \cdot K \) target phrase translations and we only keep the top-\( K \) phrases. We call this decoding scheme **BTG-N Seq2Seq**. We generally found it to perform better than BTG-1 Seq2Seq but incur additional cost in decoding.\(^7\)

**External Translation Rules.** An important feature of CKY decoding is that we can constrain the prediction of BTG-Seq2Seq to incorporate new translation rules during inference. When we have access to new translation rules \( (x_{i:k} \rightarrow v) \) during inference (e.g., for idioms or proper nouns), we can enforce this by adding a rule \( C[x_{i:k}] \rightarrow v \) and setting the score \( \pi(C[x_{i:k}] \rightarrow v) = 1 \) (or some high value) during CKY decoding.

\(^7\)Merging two sorted lists requires \( O(K \log K) \) if implemented by priority queue (Huang and Chiang, 2005). Compared to the standard CKY algorithm, we also need to enumerate the number of segments \( n \). Hence, the best time complexity of CKY decoding is \( O((K \log K) n^4) \). In practice, we set a maximum number of segments \( N \) and treat it as a hyperparameter to tune.

## 3 Experimental Setup

We apply BTG-Seq2Seq to a spectrum sequence-to-sequence learning tasks, starting from diagnostic toy datasets and then gradually moving towards real machine translation tasks. Additional details (dataset statistics, model sizes, etc.) are given in appendix E.

**Baselines.** We use the standard Transformer-based seq2seq models (Vaswani et al., 2017) as the backbone module for all experiments. We consider three baselines: (1) a seq2seq model trained in a standard way, (2) a seq2seq model pretrained with phrase pairs extracted with an off-the-shelf tool, Fast-Align (Dyer et al., 2013),\(^8\) and (3) a trivial baseline in which the model is pretrained on phrase pairs generated by randomly splitting a source-target sentence pair into phrase pairs. The latter two approaches study the importance of using learned alignments.

**BTG-1 Seq2Seq and BTG-N Seq2Seq.** We focus on evaluating the two decoding modes of our model: CKY decoding where phrase-level translations are structurally combined to form a sentence-level prediction (BTG-N Seq2Seq), and standard “flat” Seq2Seq decoding where sentence-level translations are generated directly by forcing \( \text{Root} \rightarrow T^1 \) (BTG-1 Seq2Seq). In this way we test the effectiveness of hierarchical alignments both as a structured translation model and as a regularizer on standard Seq2Seq learning. BTG-N Seq2Seq is sometimes referred simply as BTG-Seq2Seq in cases where CKY decoding is activated by default (e.g., constrained inference).

**Backbone Transformers.** In each experiment, we use the same set of hyperparameters across models for fair comparison. One crucial hyperparameter for the backbone Transformers of BTG-Seq2Seq is to use relative positional encodings (as opposed to absolute ones) which are better at handling translation at phrase levels. Hence we use relative positional encodings whenever possible except when utilizing pretrained Transformers that have been pretrained with absolute positional embeddings.

## 4 Main Results

### 4.1 Toy SVO-SOV Translation

We start with a diagnostic translation task in which a model has to transform a synthetic sentence from
SVO to SOV structure. The dataset is generated with the following synchronous grammar,

\[
\text{Root } \rightarrow \langle \text{S VO}, \text{ S O V} \rangle, \\
\text{S } \rightarrow \text{NP}, \text{ O } \rightarrow \text{NP}, \text{ V } \rightarrow \text{VP}, \\
\text{NP } \rightarrow \langle \text{np}_s, \text{ np}_t \rangle, \text{ VP } \rightarrow \langle \text{vp}_s, \text{ vp}_t \rangle,
\]

where the noun phrase pairs \(\langle \text{np}_s, \text{ np}_t \rangle\) and the verb phrase pairs \(\langle \text{vp}_s, \text{ vp}_t \rangle\) are uniformly sampled from two nonoverlapping sets of synthetic phrase pairs with varying phrase lengths between 1 to 8. We create two generalization train-test splits.

**Novel-Position.** For this split, a subset of \(\text{np}_s, \text{ np}_t\) phrase pairs are only observed in one syntactic position during training (i.e., only as a subject or only as an object). At test time, they appear in a new syntactic role. Each remaining \(\text{np}_s, \text{ np}_t\) pairs appears in both positions in training.

**Few-Shot.** In this setting, each subject (or object) phrase is observed in three examples in three different examples. During the evaluation, each subject phrase appears with context words (i.e., a predicate and an object phrase) which are found in the training set but not with this subject phrase.

**Results.** In Table 1, we observe that BTG-N Seq2Seq perfectly solves these two tasks. We also observed that BTG-N Seq2Seq correctly recovers the latent trees when \(N = 3\). Interestingly, even without structured CKY decoding, BTG-1 Seq2Seq implicitly captures the underlying transduction rules, achieving near-perfect accuracy. In contrast, standard seq2seq models fail in both settings.

### 4.2 English-Chinese: Phrasal Few-shot MT

Next, we test our models on the English-Chinese translation dataset from Li et al. (2021), which was designed to test for compositional generalization in MT. We use a subset of the original dataset and create three few-shot splits similar to the few-shot setting in the previous task. For example in the NP split noun phrases are paired with new contexts during evaluation. Compared with the few-shot setting for the SVO-SOV task, this setup is more challenging due to a larger vocabulary and fewer training examples (approx. 800 data points).

**Results.** As shown in Table 1, both BTG-1 Seq2Seq and BTG-N Seq2Seq improve on the standard Seq2Seq by a significant margin. BTG-N Seq2Seq also outperforms the Fast-Align baseline\(^9\) on VP and PP splits and slightly lags behind NP, highlighting the benefits of jointly learning alignment along with the Seq2Seq model.

**Analysis.** In Figure 3 we visualize the induced phrasal alignments for an example translation by extracting the most probable derivation \(\text{argmax}_d p(d|x)\) for \(n = 3\), which encodes the phrase-level correspondence between \(x\) and \(y\). In this example “at a restaurant” is segmented and reordered before “busy night” according to the Chinese word order. We also give example translations in Table 2, which shows two failure modes of the standard seq2seq approach. The first failure mode is producing hallucinations. In the first example, the source phrase “hang out with the empty car he liked” is translated to a Chinese phrase with a completely different meaning. This Chinese translation exactly matches a Chinese training example, likely triggered by the shared word 空 (‘empty car’). The second failure mode is under-translation. In the second example, standard seqsSeq completely ignores the last phrase “it was so dark” in its Chinese translation, likely because this phrase never co-occurs with the current context. In contrast BTG-Seq2Seq is able to cover all source phrases, and

\(^9\)We pretrain on Fast-Align phrases and fine-tune on the original training set. This was found to perform better than jointly training on the combined dataset.
Figure 3: Induced hierarchical alignment by BTG-Seq2Seq on an English-Chinese example.

Table 2: Chinese translations from a baseline seq2seq model and our BTG-Seq2Seq approach. Source phrases that are translated incorrectly by standard seq2seq are highlighted in italics, and target translations are delineated with dashed underlines.

in this case, predicts the corresponding Chinese phrase for “it was so dark”.

4.3 German-English: Low-Data MT

Next, we move onto more a realistic dataset and consider a low data German-English translation task, following Sennrich and Zhang (2019). The original dataset contains TED talks from the IWSLT 2014 DE-EN translation task (Cettolo et al., 2014). Instead of training the seq2seq component from scratch, we use pre-trained mBART (Liu et al., 2020) to initialize the seq2seq component of our grammar (and the baseline seq2seq). This was found to perform much better than random initialization, and highlights an important feature of our approach which can work with arbitrary neural seq2seq models. (We explore this capability further in section 4.5.) The use of pre-trained mBART lets us consider extremely low-data settings, with 500 and 1000 sentence pairs, in addition to the 4978 setup from the original paper. We observe in Table 1 that even the standard seq2seq achieves surprisingly reasonable BLEU scores in this low-data setting—much higher that scores of seq2seq models trained from scratch (Sennrich and Zhang, 2019). BTG-N Seq2Seq and BTG-1 Seq2Seq again outperform the baseline seq2seq model.

Pretraining on Fast-Align harms performance, potentially due to it’s being unable to induce meaningful phrase pairs in such low data regimes. In contrast, all our aligners (i.e., parsers) can also leverage pretrained features from mBART (we use

Table 3: Experiments on injecting new translation rules during inference. Examples are with the mBART model finetuned on the IWSLT 2014 German-to-English dataset. We manually provide new translation rules to BTG-Seq2Seq, and decode with the constrained CKY algorithm. Injected phrases in the new predictions are highlighted with dashed underlines. We also provide results from Google Translate.

the same mBART model to give span scores for all our parsers), which can potentially improve structure induction and have a regularizing effect.

Injecting New Translation Rules. We investigate whether we can control the prediction of BTG-Seq2Seq by incorporating new translation rules during CKY inference. In particular, we give new translation rules to mBART-based BTG-Seq2Seq for DE-EN translation. Since the model has been trained only on 4978 pairs, the translations are far from perfect. But as shown in Table 3 we observe that BTG-Seq2Seq can translate proper nouns, time expression, and even idioms to an extent. The idiom case is particularly interesting as pure seq2seq systems are known to be too compositional when translating idioms (Dankers et al., 2022).

4.4 Cherokee-English: Low-Resource MT

Our next experiments consider a true low-resource translation task with Cherokee-English, using the dataset from Zhang et al. (2020). This dataset contains around 12k/1k/1k examples for train/dev/test, respectively. Unlike German-English, Cherokee was not part of the mBART pretraining set, and thus we train all our models from scratch. In Table 1 we see that BTG-N Seq2Seq and BTG-1 Seq2Seq again outperform the standard seq2seq.

4.5 Low Resource MT with Pretrained Models

The current trend in low resource MT is to first pretrain a single multilingual model on a large number

10Their best seq2seq model with 4978 sentences obtains 16.57 BLEU, while phrase-based SMT obtains 15.87.

11However standard SMT is a strong baseline for Cherokee-English: Zhang et al. (2020) obtain 14.5 (Chr→En) and 9.8 (En→Chr) with Moses (Koehn et al., 2007).
<table>
<thead>
<tr>
<th>Models</th>
<th>Mongolian (MN→EN)</th>
<th>Marathi (MR→EN)</th>
<th>Azerbaijani (AZ→EN)</th>
<th>Bengali (BN→EN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finetune mT5</td>
<td>8.6</td>
<td>8.5</td>
<td>10.2</td>
<td>8.4</td>
</tr>
<tr>
<td>BTG-1 mT5</td>
<td>10.7</td>
<td>9.6</td>
<td>11.5</td>
<td>9.6</td>
</tr>
<tr>
<td>BTG-N mT5</td>
<td>9.4</td>
<td>9.5</td>
<td>11.5</td>
<td>10.3</td>
</tr>
<tr>
<td>Finetune mBART50</td>
<td>11.2</td>
<td>11.6</td>
<td>15.5</td>
<td>13.6</td>
</tr>
<tr>
<td>BTG-1 mBART50</td>
<td>11.5</td>
<td>11.2</td>
<td>15.8</td>
<td>12.8</td>
</tr>
<tr>
<td>BTG-N mBART50</td>
<td>12.1</td>
<td>10.9</td>
<td>15.9</td>
<td>12.8</td>
</tr>
</tbody>
</table>

Table 4: BLEU scores on four low-resource machine translation tasks. The backbone seq2seq models of our BTG models are initialized with either mT5 (Xue et al., 2021) or mBART50 (Tang et al., 2020). Numbers in grey background are taken from Tang et al. (2020).

of language pairs (Tang et al., 2020; Fan et al., 2021; Costa-jussà et al., 2022) and then optionally finetune it on specific language pairs. In our final experiment, we explore whether we can improve upon models that have already pretrained on bitext data. We focus mBART50-large which has been pretrained on massive monolingual and bilingual corpora. We randomly pick four low-resource languages from the ML50 benchmark (Tang et al., 2020), each of which has fewer than 10K bitext pairs. We also experiment with mT5-base (Xue et al., 2021) to see if we can plug-and-play other pretrained seq2seq models.

From Table 4, we see that our BTG models outperform the standard finetuning baseline in four languages for mT5, and achieve slight improvement in two languages for mBART50. We conjecture that this is potentially due to (1) mBART50’s being already trained on a large amount of bitext data and (2) mT5’s use of relative positional encodings, which is more natural for translating at all phrase scales. Finally, although we focus on pretrained seq2seq models in this work, an interesting extension would be to consider prompt-based hierarchical phrase-based MT where \( p_{seq2seq}(v|\mathbf{x};k) \) is replaced with a large language model that has been appropriately prompted to translate phrases.\(^{12}\)

5 Related Work

Synchronous grammars Classic synchronous grammars and transducers have been widely explored in NLP for many applications (Shieber and Schabes, 1990; Wu, 1997; Eisner, 2003; Ding and Palmer, 2005; Nesson et al., 2006; Huang et al., 2006; Wong and Mooney, 2007; Wang et al., 2007; Graehl et al., 2008; Blunsom et al., 2008, 2009; Cohn and Lapata, 2009, *inter alia*). In this work, we focus on the formalism of bracketing transduction grammars, which have been used for modeling reorderings in SMT systems (Nakagawa, 2015; Neubig et al., 2012). Recent work has explored the coupling bracketing transduction grammars with with neural parameterization as a way to inject structural biases into neural sequence models (Wang et al., 2021). Our work is closely related to the neural QCFG (Kim, 2021), a neural parameterization of a quasi-synchronous grammars (Smith and Eisner, 2006), which also defines a monolingual grammar conditioned on the source. However they do not experiment with embedding a neural seq2seq model within their grammar. More generally our approach extends the recent line of work on neural parameterizations of classic grammars (Jiang et al., 2016; Han et al., 2017, 2019; Kim et al., 2019; Jin et al., 2019; Zhu et al., 2020; Yang et al., 2021b,a; Zhao and Titov, 2020, *inter alia*), although unlike in these works we focus on the transduction setting.

Data Augmentation Our work is also related to the line of work on utilizing grammatical or alignment structures to guide flexible neural seq2seq models via data augmentation (Jia and Liang, 2016; Fadaee et al., 2017; Andreas, 2020; Akyürek et al., 2021; Shi et al., 2021; Yang et al., 2022; Qiu et al., 2022) or auxiliary supervision (Cohn et al., 2016; Mi et al., 2016; Liu et al., 2016; Yin et al., 2021). In contrast to these works our data augmentation module has stronger inductive biases for hierarchical structure due to explicit use of latent tree-based alignments.

6 Conclusion

We presented a neural transducer that maintains the flexibility and seq2seq models but still incorporates hierarchical phrases both as a source of inductive bias during training and as explicit constraints during inference. We formalized our model as a synchronous grammar and developed an efficient variational inference algorithm for training. Our model performs well compared to baselines on small MT benchmarks both as a regularized seq2seq model and as a synchronous grammar.
7 Limitations

Our work has several limitations. While variational inference enables more efficient training than full marginalization ($O(L^3)$ vs. $O(L^7)$), it is still much more expensive compute- and memory-wise than regular seq2seq training. This currently precludes our approach as a viable alternative on large-scale machine translation benchmarks, though advances in GPU-optimized dynamic programming algorithms could improve efficiency (Rush, 2020). (However, as we show in our experiments it is possible to finetune a pretrained seq2seq MT system using our approach.) Cube-pruned CKY decoding is also much more expensive than regular seq2seq decoding. In general, grammar- and automaton-based models enable a greater degree of model introspection and interpretability. However our use of black-box seq2seq systems within the grammar still makes our approach not as interpretable as classic transducers.

Acknowledgements

This study was supported by MIT-IBM Watson AI Lab.

References


Phil Blunsom, Trevor Cohn, and Miles Osborne. 2009. Bayesian Synchronous Grammar Induction. In Proceedings of NeurIPS.


Wenjuan Han, Yong Jiang, and Kewei Tu. 2017. Dependency Grammar Induction with Neural Lexicalization and Big Training Data. In Proceedings of EMNLP.

Wenjuan Han, Yong Jiang, and Kewei Tu. 2019. Enhancing Unsupervised Generative Dependency Parser with Contextual Information. In Proceedings of ACL.


The basic strategy for our parameterization is to take as much advantage of the backbone Transformer seq2seq models as possible, which is the main source of rich neural representations (especially when working with pretrained models).

**Seq2Seq** $p(d_{seq}[x, d_{tree}])$. The leaf nodes of $d_{tree}$ are the production rules of $C[x_{i:k}] \rightarrow v$, which encode the correspondence between source segment $x_{i:k}$ and target segment $v$. The probability of such a rule is modeled by a conventional Transformer-based seq2seq model (Vaswani et al., 2017). In BTG-Seq2Seq, the underlying seq2seq is expected to handle segments at all levels, including sentence- and phrase-level translations. We found it beneficial to use two sets of special tokens to indicate the beginning/end of sentences, one for sentence-level translations and the other for phrase-level translations. Concretely, in addition to the typical ‘BOS’ and ‘EOS’ which are used to mark the beginning and the end of a sentence, we additionally use ‘Seg-BOS’ and ‘Seg-EOS’ to mark the beginning and the end of a segment.

**Source parser** $p(d_{tree}[x, n])$. To obtain a probabilistic parser, we need to assign a score to each rule involved in $d_{tree}$. For the following rules,

$$\forall n \in \{1, 2 \ldots |x|\},$$

$$T^n \rightarrow S^n[x], \quad T^n \rightarrow I^n[x],$$

we assign them a weight of 1 for simplicity, as we find that using trainable weights does not help during our initial experiments.

We use neural features to parameterize the score of the following rules:

$$\forall i,j,k \in \{0, 1 \ldots |x|\} \quad \text{s.t.} \quad i < j < k$$

$$S^n[x_{i:k}] \rightarrow I^l[x_{i:j}]S^r[x_{j:k}]$$

$$S^n[x_{i:k}] \rightarrow I^l[x_{i:j}]I^l[x_{j:k}]$$

$$I^n[x_{i:k}] \rightarrow S^l[x_{j:k}]I^l[x_{i:j}]$$

$$I^n[x_{i:k}] \rightarrow S^l[x_{j:k}]S^r[x_{i:j}].$$

For example the score

$$\pi(S^n[x_{i:k}] \rightarrow I^l[x_{i:j}]S^r[x_{j:k}]) = \exp(e^T_S f(\tilde{h}_{i:j}, \tilde{h}_{j:k}))$$

relies on the difference feature $\tilde{h}_{i:j}$ and $\tilde{h}_{j:k}$, following Kitaev and Klein (2018). In our parameterization, rules that have the same right-hand spans will have the same score regardless of their nonterminals. That is, the score of $I^n[x_{i:k}] \rightarrow S^l[x_{j:k}]I^l[x_{i:j}]$ is identical to
$I^n_{[x_{i:k}]} \rightarrow S^l_{[x_{j:k}]} S^r_{[x_{i:j}]}$. These difference features are calculated using

$$\tilde{h}_{i:j} = \text{Concat}[\tilde{h}_j - \tilde{h}_i; \tilde{h}_{j+1} - \tilde{h}_{i+1}]$$

where $\tilde{h}_i$ and $\tilde{h}_j$ are the “forward” and “backward” representations from Kitaev and Klein (2020).13

The score of a derivation tree is the sum of the scores of the production rules that form the tree, and we normalize the score to obtain the distribution over all possible derivations,

$$p(d_{\text{tree}}|x, n) = \left(\prod_{R \in d_{\text{tree}}} \pi(R)\right) / Z_{\text{tree}}(x, n),$$

where $Z_{d_{\text{tree}}}(x, n) = \sum_{d_{\text{tree}}} \left(\prod_{R \in d_{\text{tree}}} \pi(R)\right)$ is the partition function. The dynamic program for computing the partition function is given by Algorithm 1.

**Variational parser** $q(d_{\text{tree}}, d_{\text{leaf}}|x, y, n)$. We factorize the variational synchronous parser as

$$q(d_{\text{tree}}, d_{\text{leaf}}|x, y, n) = q(d_{\text{tree}}|x, y, n) \times q(d_{\text{leaf}}|x, y, d_{\text{tree}}),$$

which makes it possible to sample hierarchical alignments in $O(L^3)$ where $L = \min(|x|, |y|)$. The parameterization of $q(d_{\text{tree}}|x, y, n)$ is very similar to the parameterization of $p(d_{\text{tree}}|x, n)$, except that $h_i$ (which are used to compute the difference features) are the hidden states of a Transformer decoder, with the backbone Transformer seq2seq running from $y$ to $x$. This choice makes it possible to condition the contextualized representations of $x$ on the target $y$. Calculating the partition function uses the same dynamic program as in the prior parser (i.e., Algorithms 1). Algorithm 2 shows the dynamic program for sampling $d_{\text{tree}}$.

Given $x, y$ and $d_{\text{tree}}$, the only rules that remain unknown for the variational parser are the target segments $y_{a|c}$ in $C_{[x_{i:k}]} \rightarrow y_{a|c}$. Thus $q(d_{\text{leaf}}|x, y, d_{\text{tree}})$ is effectively a hierarchical segmentation model. We use a grammar with the following rules,

$$\text{Root} \rightarrow T^m[y], T^m[y_{a|c}] \rightarrow T^l[y_{a|b}] T^r[y_{b|c}],$$

to model the segmentations. Since the segmentation is conditioned on $d_{\text{tree}}$, we need to ensure that the tree structures of $d_{\text{leaf}}$ and $d_{\text{tree}}$ is identical. (Note that by conditioning on $d_{\text{tree}}$, we only utilize the tree topology of $d_{\text{tree}}$ for efficiency.)

We use the following function to score the rules:

$$\pi(T^m[y_{a|c}] \rightarrow T^m[y_{a|b}] T^r[y_{b|c}]) = \exp(f(\tilde{h}_{a|b}, \tilde{h}_{b|c})),$$

where $\tilde{h}_{a|b}, \tilde{h}_{b|c}$ are the difference features based on the contextualized representations of $y$, and $f$ is an MLP that emits a scalar score. We can then obtain a distribution over $d_{\text{leaf}}$ via,

$$q(d_{\text{leaf}}|x, y, n, d_{\text{tree}}) = \left(\prod_{R \in d_{\text{leaf}}} \pi(R)\right) / Z_{\text{leaf}}(d_{\text{tree}}, y)$$

where $Z_{d_{\text{leaf}}}(d_{\text{tree}}, y) = \sum_{d_{\text{leaf}}} \left(\prod_{R \in d_{\text{leaf}}} \pi(R)\right)$ is the partition function. Algorithm 3 shows the dynamic program for calculating the partition function above, and Algorithm 4 gives the dynamic program for sampling from $q(d_{\text{leaf}}|x, n, d_{\text{tree}})$. These dynamic programs modify the standard inside algorithm to only sum over chart entries that are valid under $d_{\text{tree}}$.

Since we share the encoder and decoder across all components (i.e., prior/variational parsers and the seq2seq model), the only additional parameters for BTG-Seq2Seq are the parameters of the MLP layers for the three CRF parsers.

**Number of segments** $p(n|x)$, $q(n|x, y)$. We use a truncated geometric distribution of the following form,

$$p(n) = \left\{ \begin{array}{ll}
\lambda(1 - \lambda)^{n-1}, & n \in \{1, 2, \ldots N - 1\} \\
(1 - \lambda)^{N-1}, & n = N
\end{array} \right.$$ 

where $N$ is the maximum number of segments. For the prior $p(n|x)$, we have $N = \min(|x|, |y|)$. In practice, we set an additional upper bound (chosen from $[3, 4, \ldots 8]$) for efficient training. $\lambda$ is chosen from $\{0.5, 0.6, 0.7, 0.8, 0.9\}$.

**B Evidence Lower Bound**

Figure 4 shows the lower bound derivation.

**C Dynamic Programs for the Parsers**

**Sampling Algorithms**

The algorithms for sampling from $p(d_{\text{tree}}|x, n)$ and $q(d_{\text{leaf}}|x, d_{\text{tree}})$ are provided in Algorithm 2 and 4. Sampling from $q(d_{\text{tree}}|x, y, n)$ is omitted, as it is the same algorithm as Algorithm 2.
We provide the dynamic program to compute $\mathbb{H}[q(d_{\text{leaf}}|x, y, d_{\text{tree}})]$ in Algorithm 5, and $\text{KL}[q(d_{\text{tree}}|x, y, n) \| p(d_{\text{tree}}|x, n)]$ in Algorithm 6. Note that $\text{KL}[q(n|x, y) \| p(n|x)]$ is a constant, and we can ignore it during optimization. In practice, the algorithms are implemented in log space for numerical stability.

**Gradient Estimation**

We use policy gradients (Williams, 1992) to optimize the lower bound. Two techniques are additionally employed to reduce variance. First, we use a self-critical control variable (Rennie et al., 2017) where the scores from argmax trees are used as a control variates. Second, we utilize a sum-and-sample strategy (Liu et al., 2019) where we always train on $n = 1$, i.e., sentence-level translations, and sample another $n > 1$ to explore segment-level translations.

**D Recipe for Using BTG-Seq2Seq**

Although it is expensive to train our BTG-Seq2Seq from scratch due to $O(L^3)$ time complexity, we find that a pretrain-then-finetune strategy is a practical way to use BTG-Seq2Seq. During the pretraining stage, we train a Transformer seq2seq in the standard way until convergence, or simply use public model checkpoints. Then during the fine-tuning stage, we use the pretrained model as the backbone seq2seq for our BTG-Seq2Seq. Usually we use a relatively smaller learning rate to finetune the backbone seq2seq during finetuning.

**E Dataset Statistics and Hyperparameters**

We show the statistics of the datasets and the hyperparameters of the model in Tables 5 and 6.
Algorithm 1 Inside algorithm for $p(d_{teq}|x, n)$. Rules are highlighted in blue

**Input:** $x$: source sentence, $n$: number of segments

**function** INSIDE-TREE($x, n$)

Initialize $β_{-}[S^1] = 0, β_{-}[I^1] = 0$

for $m = 2$ to $n$ do
  ▷ number of segments
  for $w = 1$ to $|x|$ do
    ▷ width of spans
    for $i = 0$ to $|x| - w$ do
      $k = i + w$
      ▷ start point
      for $j = i + 1$ to $k - 1$ do
        ▷ end point
        for $l = 1$ to $m - 1$ do
          ▷ number of segments in the left part
          ▷ number of segments in the right part
          $r = m - l$
          $β_{i,k}[S^m] = \pi(S^{m}[x_{i,k}] \rightarrow I^{l}[x_{i,j}]S^{r}[x_{j,k}]) \cdot β_{i,j}[I^l] \cdot β_{j,k}[S^r]$
          $β_{i,k}[I^m] = \pi(I^{m}[x_{i,k}] \rightarrow I^{l}[x_{i,j}]I^{r}[x_{j,k}]) \cdot β_{i,j}[I^l] \cdot β_{j,k}[I^r]$
          $β_{i,k}[R^m] = \pi(I^{m}[x_{i,k}] \rightarrow S^{l}[x_{i,j}]S^{r}[x_{j,k}]) \cdot β_{i,j}[S^l] \cdot β_{j,k}[S^r]$

  ▷ Compute locally-normalized scores $\tilde{\pi}$
  for $m = 2$ to $n$ do
    ▷ number of segments
    for $w = 1$ to $|x|$ do
      ▷ width of spans
      for $i = 0$ to $|x| - w$ do
        $k = i + w$
        ▷ start point
        for $j = i + 1$ to $k - 1$ do
          ▷ end point
          for $l = 1$ to $m - 1$ do
            ▷ number of segments in the left part
            ▷ number of segments in the right part
            $r = m - l$
            ▷ Denote $S^{m}[x_{i,k}] \rightarrow I^{l}[x_{i,j}]S^{r}[x_{j,k}]$ as $R_1$
            $\tilde{\pi}(R_1) = (\pi(R_1) \cdot β_{i,j}[I^l] \cdot β_{j,k}[S^r]) / β_{i,k}[S^m]$
            ▷ Denote $S^{m}[x_{i,k}] \rightarrow I^{l}[x_{i,j}]I^{r}[x_{j,k}]$ as $R_2$
            $\tilde{\pi}(R_2) = (\pi(R_2) \cdot β_{i,j}[I^l] \cdot β_{j,k}[I^r]) / β_{i,k}[S^m]$
            ▷ Denote $I^{m}[x_{i,k}] \rightarrow S^{l}[x_{i,j}]S^{r}[x_{j,k}]$ as $R_3$
            $\tilde{\pi}(R_3) = (\pi(R_3) \cdot β_{j,k}[S^l] \cdot β_{i,j}[S^r]) / β_{i,k}[I^m]$
            ▷ Denote $I^{m}[x_{i,k}] \rightarrow S^{l}[x_{i,j}]I^{r}[x_{j,k}]$ as $R_4$
            $\tilde{\pi}(R_4) = (\pi(R_4) \cdot β_{j,k}[S^l] \cdot β_{i,j}[I^r]) / β_{i,k}[I^m]$

  ▷ $T$ rules are assigned a trivial score: $\pi(T^{m} \rightarrow S^{n}[x]) = 1, \pi(T^{m} \rightarrow I^{n}[x]) = 1$
  $β[T^{m}] = β_{0:|x|}[S^n] + β_{0:|x|}[I^n], \ Z_{tree}(x, n) = β[T^{m}]$
  ▷ partition function
  $\tilde{\pi}(T^{m} \rightarrow S^{n}[x]) = β_{0:|x|}[S^n] / β[T^{m}], \ \tilde{\pi}(T^{m} \rightarrow I^{n}[x]) = β_{0:|x|}[I^n] / β[T^{m}]$

return $Z_{tree}(x, n), β, \tilde{\pi}$
Algorithm 2 Top-down sampling $d_{tree}$ from $p(d_{tree}|x, n)$. Rules are highlighted in blue

Input: $\tilde{\pi}$: normalized scores obtained from INSIDE-TREE($\cdot$), $n$: number of segments

function SAMPLE-TREE($\tilde{\pi}, n$)
    Initialize an empty tree $d_{tree}$
    Sample $A \in \{S, I\}$ w.r.t. $\tilde{\pi}(T^n\rightarrow A^n[x])$
    RECUR-SAMPLE-TREE($0, |x|, n, A, \tilde{\pi}, d_{tree}$)
    return $d_{tree}$

▷ Arguments: $i$: start point, $k$: end point, $n$: number of segments, $A$: nonterminal

function RECUR-SAMPLE-TREE($i, k, n, A, \tilde{\pi}, d_{tree}$)
    if $n = 1$ then
        Add rule $A^i[x_i:k] \rightarrow C[x_i:k]$ to $d_{tree}$
        return
    if $A = S$ then
        ▷ $A = I$
        $\triangleright 1 \leq l < n$, $A, B \in \{S, I\}, i < j < k$ are the random variables, $r = n - l$
        Sample rule $S^n[x_i:k] \rightarrow A^i[x_i:j]B^r[x_j:k]$ according to $\tilde{\pi}(\cdot)$
        Add $S^n[x_i:k] \rightarrow A^i[x_i:j]B^r[x_j:k]$ to $d_{tree}$
        RECUR-SAMPLE-TREE($i, j, l, A, \tilde{\pi}, d_{tree}$)
        RECUR-SAMPLE-TREE($j, k, r, B, \tilde{\pi}, d_{tree}$)
    else
        ▷ $1 \leq l < n$, $A, B \in \{S, I\}, i < j < k$ are the random variables, $r = n - l$
        Sample rule $I^n[x_i:k] \rightarrow A^i[x_i:j]B^r[x_j:i]$ according to $\tilde{\pi}(\cdot)$
        Add $I^n[x_i:k] \rightarrow A^i[x_i:j]B^r[x_j:i]$ to $d_{tree}$
        RECUR-SAMPLE-TREE($j, k, l, A, \tilde{\pi}, d_{tree}$)
        RECUR-SAMPLE-TREE($i, j, r, B, \tilde{\pi}, d_{tree}$)

Algorithm 3 Inside algorithm for $q(d_{leaf}|x, y, d_{tree})$.

Input: $y$: target sentence, $d_{tree}$: derivation tree

function INSIDE-LEAF($y, d_{tree}$)
    Infer number of segments $n$ from $d_{tree}$
    Initialize $\beta_{\cdot,\ldots} = 0$
    for $m = 2$ to $N$ do  ▷ number of segments
        for $w = 1$ to $|x|$ do  ▷ width of spans
            for $i = 0$ to $|x| - w$ do  ▷ start point
                $k = i + w$
                for $j = i + 1$ to $k - 1$ do  ▷ end point
                    for $l = 1$ to $m - 1$ do  ▷ number of segments in the left part
                        $r = m - l$
                        if there exist a rule like $A^m \rightarrow B^lC^r$ in $d_{tree}$ then
                            $\beta_{i,k}[T^m] += \pi(T^m[y_i:k] \rightarrow T^l[y_i,j]T^r[y_j:k]) \cdot \beta_{i,j}[T^l] \cdot \beta_{j,k}[T^r]$
                    end if
                end for
            end for
        end for
    end for
    return $\beta$
Algorithm 4: Top-down sampling \( d_{\text{leaf}} \) from \( q(d_{\text{leaf}} | x, y, d_{\text{tree}}) \).

**Input:** \( \beta \): inside scores returned from INSIDE-LEAF(), \( d_{\text{tree}} \): derivation tree, \( y \): target sentence

**function** SAMPLE-LEAF(\( \beta \), \( d_{\text{tree}} \), \( y \))

- Infer the number of segments \( n \) from \( d_{\text{tree}} \)
- Initialize an empty tree \( d_{\text{leaf}} \)
- RECUR-SAMPLE-LEAF(0, |\( y \)|, \( n \), \( \beta \), \( d_{\text{tree}} \), \( d_{\text{leaf}} \))
- return \( d_{\text{leaf}} \)

\[ \text{▷ Arguments: } i: \text{ start point, } k: \text{ end point, } n: \text{ number of segments} \]

**function** RECUR-SAMPLE-LEAF(\( i \), \( k \), \( n \), \( \beta \), \( d_{\text{leaf}} \), \( d_{\text{tree}} \))

- if \( n = 1 \) then
  - Add rule \( T^i[y_i:k] \rightarrow T^j[y_j:k] \) to \( d_{\text{leaf}} \)
  - return

- Infer \( l, r \) based on \( d_{\text{tree}} \) and \( n \)
- Compute \( Z = \sum_j (\beta_{i:j}[T^i] \cdot \beta_{j:k}[T^j]) \) \[ \text{▷ normalization term} \]
- for \( j = i + 1 \) to \( k - 1 \) do
  - \( \tilde{\pi}(T^i[y_i:k] \rightarrow T^j[y_j:k]) = (\beta_{i:j}[T^i] \cdot \beta_{j:k}[T^j])/Z \)

- Sample rule \( T^i[y_i:k] \rightarrow T^j[y_j:k] \) and add it to \( d_{\text{leaf}} \) \[ \text{▷ the rule only encodes split point } j \]
- RECUR-SAMPLE-LEAF(\( i \), \( j \), \( l \), \( r \), \( \beta \), \( d_{\text{tree}} \), \( d_{\text{leaf}} \))
- RECUR-SAMPLE-LEAF(\( j \), \( k \), \( r \), \( \beta \), \( d_{\text{tree}} \), \( d_{\text{leaf}} \))

Algorithm 5: Calculating the entropy of \( H[q(d_{\text{leaf}} | x, y, d_{\text{tree}})] \)

**Input:** \( \beta \): inside scores returned by INSIDE-LEAF(), \( d_{\text{tree}} \): derivation tree, \( y \): target sentence

**function** COMPUTE-LEAF-ENTROPY(\( \beta \), \( d_{\text{tree}} \), \( y \))

- Find the number of segments \( n \) from \( d_{\text{tree}} \)
- return RECUR-COMPUTE-LEAF-ENTROPY(0, |\( y \)|, \( n \), \( \beta \), \( d_{\text{tree}} \))

\[ \text{▷ Arguments: } i: \text{ start point, } k: \text{ end point, } n: \text{ number of segments} \]

**function** RECUR-COMPUTE-LEAF-ENTROPY(\( i \), \( k \), \( n \), \( \beta \), \( d_{\text{tree}} \))

- if \( n = 1 \) then
  - return 0

- Infer \( l, r \) based on \( d_{\text{tree}} \) and \( n \)
- Compute \( Z = \sum_j (\beta_{i:j}[T^i] \cdot \beta_{j:k}[T^j]) \) \[ \text{▷ normalization term} \]
- for \( j = i + 1 \) to \( k - 1 \) do
  - \( \tilde{\pi}(T^i[y_i:k] \rightarrow T^j[y_j:k]) = (\beta_{i:j}[T^i] \cdot \beta_{j:k}[T^j])/Z \)

- \( H = 0 \)
- for \( j = i + 1 \) to \( k - 1 \) do
  - \( H_l = \text{RECUR-COMPUTE-LEAF-ENTROPY}(i, j, l, \beta, d_{\text{tree}}) \)
  - \( H_r = \text{RECUR-COMPUTE-LEAF-ENTROPY}(j, k, r, \beta, d_{\text{tree}}) \)
  - Denote rule \( T^i[y_i:k] \rightarrow T^j[y_j:k] \) as \( R \)
  - \( H += (H_l + H_r - \log(\tilde{\pi}(R))) \cdot \tilde{\pi}(R) \)

- return \( H \) \[ \text{▷ return the entropy} \]
Algorithm 6 Computing the KL-divergence of KL[\( q(d_{\text{tree}}|x, y, n) \| p(d_{\text{tree}}|x, n) \)]

**Input:** \( \tilde{\pi}_q \): normalized rule scores returned by INSIDE-TREE(\( \cdot \)) for \( q(d_{\text{tree}}|x, y, n) \), \( \pi_p \): rule scores obtained similarly by calling INSIDE-TREE(\( \cdot \)) for \( p(d_{\text{tree}}|x, n) \), \( n \): number of segments, \( x \): source sentence

**function** COMPUTE-TREE-KL(\( \tilde{\pi}_q, \pi_p, n, x \))

1. Initialize KL[..] = 0
2. for \( m = 2 \) to \( n \) do ▷ number of segments
   1. for \( w = 1 \) to \( |x| \) do ▷ width of spans
      1. for \( i = 0 \) to \( |x| - w \) do ▷ start point
         1. \( k = i + w \) ▷ end point
         2. for \( j = i + 1 \) to \( k - 1 \) do ▷ number of segments in the left part
            1. \( r = m - l \) ▷ number of segments in the right part
               1. Denote \( S_{[x_{i+k}]\rightarrow l'}[x_{j+l}]\) as \( R_1 \)
               2. \( \text{KL}_{i,k}[S_{[x_{i+k}]}] = (\text{KL}_{i,j}[l'] + \text{KL}_{j:k}[S^l] + \log[\tilde{\pi}_q(R_1)] - \log[\tilde{\pi}_p(R_1)]) \cdot \tilde{\pi}_q(R_1) \)
               3. Denote \( S_{[x_{i+k}]\rightarrow l'}[x_{j+l}] \) as \( R_2 \)
               4. \( \text{KL}_{i,k}[S_{[x_{i+k}]}] = (\text{KL}_{i,j}[l'] + \text{KL}_{j:k}[l'] + \log[\tilde{\pi}_q(R_2)] - \log[\tilde{\pi}_p(R_2)]) \cdot \tilde{\pi}_q(R_2) \)
               5. Denote \( I_{[x_{i+k}]\rightarrow l'}[x_{j+l}] \) as \( R_3 \)
               6. \( \text{KL}_{i,k}[I_{[x_{i+k}]}] = (\text{KL}_{i,j}[l'] + \text{KL}_{j:k}[l'] + \log[\tilde{\pi}_q(R_3)] - \log[\tilde{\pi}_p(R_3)]) \cdot \tilde{\pi}_q(R_3) \)
               7. Denote \( I_{[x_{i+k}]\rightarrow l'}[x_{j+l}] \) as \( R_4 \)
               8. \( \text{KL}_{i,k}[I_{[x_{i+k}]}] = (\text{KL}_{i,j}[l'] + \text{KL}_{j:k}[l'] + \log[\tilde{\pi}_q(R_4)] - \log[\tilde{\pi}_p(R_4)]) \cdot \tilde{\pi}_q(R_4) \)
      3. Denote \( T_{[x_{i+k}]\rightarrow l'}[x_{j+l}] \) as \( R_5 \), \( R_6 \) respectively
3. \( \text{KL}[T_{[x_{i+k}]}] = (\text{KL}_{0:[x]}[S_{[x_{i+k}]}] + \log[\tilde{\pi}_q(R_5)] - \log[\tilde{\pi}_p(R_5)]) \cdot \tilde{\pi}_q(R_5) \)
3. \( \text{KL}[T_{[x_{i+k}]}] = (\text{KL}_{0:[x]}[I_{[x_{i+k}]}] + \log[\tilde{\pi}_q(R_6)] - \log[\tilde{\pi}_p(R_6)]) \cdot \tilde{\pi}_q(R_6) \)
4. return \( \text{KL}[T_{[x_{i+k}]}] \)