Neural QCFG is a grammar-based sequence-to-sequence (seq2seq) model with strong inductive biases on hierarchical structures. It excels in interpretability and generalization but suffers from expensive inference. In this paper, we study two low-rank variants of Neural QCFG for faster inference with different trade-offs between efficiency and expressiveness. Furthermore, utilizing the symbolic interface provided by the grammar, we introduce two soft constraints over tree hierarchy and source coverage. We experiment with various datasets and find that our models outperform vanilla Neural QCFG in most settings.

1 Introduction

Standard neural seq2seq models are versatile and broadly applicable due to its approach of factoring the output distribution into distributions over the next words based on previously generated words and the input (Sutskever et al., 2014; Gehring et al., 2017; Devlin et al., 2019). Despite showing promise in approximating complex output distributions, these models often fail when it comes to diagnostic tasks involving compositional generalization (Lake and Baroni, 2018; Bahdanau et al., 2019; Loula et al., 2018), possibly attributed to a lack of inductive biases for the hierarchical structures of sequences (e.g., syntactic structures), leading to models overfitting to surface clues.

In contrast to neural seq2seq models, traditional grammar-based models incorporate strong inductive biases to hierarchical structures but suffer from low coverage and the hardness of scaling up (Wong and Mooney, 2006; Bos, 2008). To benefit from both of these approaches, blending traditional methods and neural networks has been studied (Herzig and Berant, 2021; Shaw et al., 2021; Wang et al., 2021, 2022). In particular, Kim (2021) proposes Neural QCFG for seq2seq learning with a quasisynchronous context-free grammar (QCFG) (Smith and Eisner, 2006) that is parameterized by neural networks. The symbolic nature of Neural QCFG makes it interpretable and easy to impose constraints for stronger inductive bias, which leads to improvements in empirical experiments. However, all these advantages come at the cost of high time complexity and memory requirement, meaning that the model and data size is restricted, which leads to a decrease in text generation performance and limited application scenarios.

In this work, we first study low-rank variants of Neural QCFG for faster inference and lower memory footprint based on tensor rank decomposition (Rabanser et al., 2017), which is inspired by recent work on low-rank structured models (Cohen et al., 2013; Chiu et al., 2021; Yang et al., 2021, 2022). These variants allow us to use more symbols in Neural QCFG, which has been shown to be beneficial for structured latent variable models (Buhai et al., 2020; Chiu and Rush, 2020; Yang et al., 2021, 2022). Specifically, we study two low-rank variants with different trade-off between computation cost and ranges of allowed constraints: the efficient model (E model), following the decomposition method in TN-PCFG (Yang et al., 2021), and the expressive model (P model), newly introduced in this paper. Furthermore, we propose two new constraints for Neural QCFG, including a soft version of the tree hierarchy constraint used by vanilla Neural QCFG, and a coverage constraint which biases models in favour of translating all source tree nodes1. We conduct experiments on three datasets and our models outperform vanilla Neural QCFG in most settings. Our code is available at https://github.com/LouChao98/seq2seq_with_qcfg.

1Similar topics are discussed in the machine translation literature (Tu et al., 2016; Li et al., 2018, among others).
2 Preliminary: Neural QCFG

Let \( s_1, s_2 \) be the source and target sequences, and \( t_1, t_2 \) be the corresponding constituency parse trees (i.e., sets of labeled spans). Following previous work (Smith and Eisner, 2006; Kim, 2021), we consider QCFG in Chomsky normal form (CNF; Chomsky, 1959) with restricted alignments, which can be denoted as a tuple \( G[t_1] = (S, \mathcal{N}, \mathcal{P}, \Sigma, R[t_1], \theta) \), where \( S \) is the start symbol, \( \mathcal{N}/\mathcal{P}/\Sigma \) are the sets of nonterminals/preterminals/terminals respectively, \( R[t_1] \) is the set of grammar rules in three forms:

\[
\begin{align*}
S \rightarrow & A[\alpha_i] \quad \text{where} \ A \in \mathcal{N}, \ \alpha_i \in t_1, \\
A[\alpha_i] \rightarrow & B[\alpha_j]C[\alpha_k] \quad \text{where} \ A \in \mathcal{N}, \ B, C \in \mathcal{N} \cup \mathcal{P}, \ \alpha_i, \alpha_j, \alpha_k \in t_1, \\
D[\alpha_i] \rightarrow & w \quad \text{where} \ A \in \mathcal{P}, \ \alpha_i \in t_1, \ w \in \Sigma, \\
\end{align*}
\]

and \( \theta \) parameterizes rule probabilities \( p_\theta(r) \) for each \( r \in R[t_1] \).

Recently, Kim (2021) proposes Neural QCFG for seq2seq learning. He uses a source-side parser to model \( p(t_1 | s_1) \) and a QCFG to model \( p(t_2 | t_1) \). The log marginal likelihood of the target sequence \( s_2 \) is defined as follows:

\[
\log p_{\theta, \phi}(s_2 | s_1) = \log \sum_{t_1 \in T(s_1)} p_\theta(s_2 | t_1)p_{\phi}(t_1 | s_1)
\]

\[
= \log \sum_{t_1 \in T(s_1)} \sum_{t_2 \in T(s_2)} p_\theta(s_2 | t_1 | s_1),
\]

where \( T(\cdot) \) denotes the set of possible parse trees for a sequence and \( \theta, \phi \) are parameters. Due to the difficulty of marginalizing out \( t_1 \) and \( t_2 \) simultaneously, Kim (2021) resorts to maximizing the lower bound on the log marginal likelihood,

\[
\log p_{\theta, \phi}(s_2 | s_1) \geq \mathbb{E}_{t_1 \sim p_{\phi}(t_1 | s_1)} [\log p_{\theta}(s_2 | t_1)].
\]

3 Low-rank Models

Marginalizing \( t_2 \) in Neural QCFG has a high time complexity of \( O(|\mathcal{N}|(|\mathcal{N}| + |\mathcal{P}|)^2S^3T^3) \) where \( S/T \) are the source/target sequence lengths. In particular, the number of rules in QCFG contributes to a significant proportion, \( O(|\mathcal{N}|(|\mathcal{N}| + |\mathcal{P}|)^2S^3) \), of the complexity. Below, we try to reduce this complexity by rule decompositions in two ways.

3.1 Efficient Model (E Model)

Let \( R \) be a new set of symbols. The E model decomposes binary rules \( r_b \) into three parts: \( A[\alpha_i] \rightarrow R, R \rightarrow B[\alpha_j] \) and \( R \rightarrow C[\alpha_k] \) (Fig. 1a), where \( R \in R \) such that

\[
p(A[\alpha_i] \rightarrow B[\alpha_j]C[\alpha_k]) = \sum_R p(A[\alpha_i] \rightarrow R) \times p(R \rightarrow B[\alpha_j]) \times p(R \rightarrow C[\alpha_k]).
\]

In this way, \(|\mathcal{N}|(|\mathcal{N}| + |\mathcal{P}|)^2S^3 \) binary rules are reduced to only \( G_E := (3|\mathcal{N}| + 2|\mathcal{P}|)R|S \) decomposed rules, resulting in a time complexity of \( O(G_E T^3)^3 \) for marginalizing \( t_2 \). Further, the complexity can be improved to \( O(|R|T^3 + |R|^2T^2) \) using rank-space dynamic programming in Yang et al. (2022)^3.

However, constraints that simultaneously involve \( \alpha_i, \alpha_j, \alpha_k \) (such as the tree hierarchy constraint in vanilla Neural QCFG and those to be discussed in Sec. 4.1) can no longer be imposed because of two reasons. First, the three nodes are in separate rules and enforcing such constraints would break the separation and consequently undo the reduction of time complexity. Second, the rank-space dynamic programming algorithm prevents us from getting the posterior distribution \( p(\alpha_i, \alpha_j, \alpha_k | t_1, s_2) \), which is necessary for many methods of learning with constraints (e.g., Chang et al., 2008; Mann and McCallum, 2007; Ganchev et al., 2010) to work.

\[\text{3. Typically, we set } |R| = O(|\mathcal{N}| + |\mathcal{P}|).\]

\[\text{3. They describe the algorithm using TN-PCFG (Yang et al., 2021), which decomposes binary rules of PCFG, } A \rightarrow BC, \text{ into } A \rightarrow R, R \rightarrow B \text{ and } R \rightarrow C. \text{ For our case, one can define new symbol sets by coupling nonterminals with source tree nodes: } \mathcal{N}_t = \{(A, \alpha_i) | A \in \mathcal{N}, \alpha_i \in t_1\} \text{ and } \mathcal{P}_t = \{(A, \alpha_i) | A \in \mathcal{P}, \alpha_i \in t_1\}. \text{ Then our decomposition becomes identical to TN-PCFG and their algorithm can be applied directly.}\]
3.2 Expressive Model (P Model)

In the P model, we reserve the relation among \( \alpha_i, \alpha_j, \alpha_k \) and avoid their separation,

\[
p(A[\alpha_i] \rightarrow B[\alpha_j]C[\alpha_k]) = \sum_R p(A[\alpha_i] \rightarrow R) \times p(R, \alpha_i \rightarrow \alpha_j, \alpha_k) \times p(R, \alpha_j \rightarrow B) \times p(R, \alpha_k \rightarrow C),
\]

as illustrated in Fig. 1b. The P model is still faster than vanilla Neural QCFG because there are only \( G_P := |R|S^3 + (3|N| + 2|P|)|R|S \) decomposed rules, which is lower than vanilla Neural QCFG but higher than the E model. However, unlike the E model, the P model cannot benefit from rank-space
dynamic programming\(^4\) and has a complexity of \( O(|R|S^2T^3 + (|N| + |P|)|R|S + |R|S^3T^2) \) for marginalizing \( t_2 \).

Rule \( R, \alpha_i \rightarrow \alpha_j, \alpha_k \) is an interface for designing constraints involving \( \alpha_i, \alpha_j, \alpha_k \). For example, by setting \( p(R, \alpha_i \rightarrow \alpha_j, \alpha_k) = 0 \) for all \( R \in R \) and certain \( \alpha_i, \alpha_j, \alpha_k \), we can prohibit the generation \( A[\alpha_i] \rightarrow B[\alpha_j]C[\alpha_k] \) in the original QCFG.

With this interface, the P model can impose all constraints used by vanilla Neural QCFG as well as more advanced constraints introduced next section.

4 Constraints

4.1 Soft Tree Hierarchy Constraint

Denote the distance between two tree nodes\(^6\) as \( d(\alpha_i, \alpha_j) \) and define \( d(\alpha_i, \alpha_j) = \infty \) if \( \alpha_j \) is not a descendant of \( \alpha_i \). Then, the distance of a binary rule is defined as \( d(r) = \max(d(\alpha_i, \alpha_j), d(\alpha_i, \alpha_k)) \).

Neural QCFG is equipped with two hard hierarchy constraints. For \( A[\alpha_i] \rightarrow B[\alpha_j]C[\alpha_k] \), \( \alpha_j, \alpha_k \) are forced to be either descendants of \( \alpha_i \) (i.e., \( d(r) < \infty \)), or more strictly, distinct direct children of \( \alpha_i \) (i.e., \( d(r) = 1 \)). However, we believe the former constraint is too loose and the latter one is too tight. Instead, we propose a soft constraint based on distances: rules with smaller \( d(r) \) are considered more plausible. Specifically,

we encode the constraint into a reward function of rules, \( \zeta(d(r)) \), such that \( \zeta(1) > \zeta(2) > \ldots \) and \( \zeta(a)(\zeta(b) > \zeta(c)\zeta(d) \) for \( a + b = c + d \) and \( \max(a, b) < \max(c, d) \). A natural choice of the reward function is \( \zeta(d(r)) := d(r)e^{-d(r)} \). We optimize the expected rewards with a maximum entropy regularizer (Williams and Peng, 1991; Mnih et al., 2016), formulated as follows:

\[
\log \sum_{t_2 \in T(s_2)} p_{\theta}(t_2|t_1)\zeta(t_2) + \tau H[p_{\theta}(t_2|t_1, s_2)],
\]

where \( \zeta(t_2) = \prod_{r \in t_2} \zeta(d(r))^2 \), \( p_{\theta}(t_2|t_1, s_2) = p_{\theta}(t_2|t_1)/\sum_{t \in T(s_2)} p_{\theta}(t|t_1) \). \( H \) represents entropy, and \( \tau \) is a positive scalar.

4.2 Coverage Constraint

Our experiments on vanilla neural QCFG show that inferred alignments could be heavily imbalanced: some source tree nodes are aligned with multiple target tree nodes, while others are never aligned. This motivates us to limit the number of alignments per source tree node with an upper bound\(^8\), \( u \). Because the total number of alignments is fixed to \( |t_2| \), this would distribute alignments from popular source tree nodes to unpopular ones, leading to more balanced source coverage of alignments. We impose this constraint via optimizing the posterior regularization likelihood (Ganchev et al., 2010),

\[
E_{\theta_i} (\log p_{\theta}(s_2|t_1) + \gamma \min_{q \in Q} KL(q(t_2)||p_{\theta}(t_2|t_1, s_2))),
\]

where \( KL \) is the Kullback-Leibler divergence (KL), \( \gamma \) is a positive scalar and \( Q \) is the constraint set \( \{q(t_2)||E_{\phi}(q(t)) \leq \xi\}, \) i.e., expectation of feature vector \( \phi \) over any distribution in \( Q \) is bounded by constant vector \( \xi \). We define the target tree feature vector \( \phi(t_2) \in \mathbb{N}^{|t_1|} \) such that \( \phi(t_2) \) represents the count of source tree node \( \alpha_i \) being aligned by nodes in \( t_2 \) and \( \xi = u1 \). Ganchev et al. (2010) provide an efficient algorithm for finding the optimum \( q \), which we briefly review in Appx. C. After finding \( q \), the KL term of two tree distributions, \( q \) and \( p_{\theta} \), can be efficiently computed using the Torch-Struct library (Rush, 2020).

\(^{\text{4}}\)Below is an intuitive explanation. Assume there is only one nonterminal symbol. Then we can remove \( A, B, C \) because they are constants. The decomposition can be simplified to \( \alpha_i \rightarrow R, R\alpha_i \rightarrow \alpha_i\alpha_j \), which is equivalent to \( \alpha_i \rightarrow \alpha_i\alpha_j \), an undecomposed binary rule. The concept “rank-space” is undefined in an undecomposed PCFG.

\(^{\text{6}}\)It is better than \( O(G_P T^3) \) because we can cache some intermediate steps, as demonstrated in Cohen et al. (2013); Yang et al. (2021). Details can be found in Appx. A.

\(^{\text{7}}\)Below is an intuitive explanation. Assume there is only one nonterminal symbol. Then we can remove \( A, B, C \) because they are constants. The decomposition can be simplified to \( \alpha_i \rightarrow R, R\alpha_i \rightarrow \alpha_i\alpha_j \), which is equivalent to \( \alpha_i \rightarrow \alpha_i\alpha_j \), an undecomposed binary rule. The concept “rank-space” is undefined in an undecomposed PCFG.

\(^{\text{8}}\)Below is an intuitive explanation. Assume there is only one nonterminal symbol. Then we can remove \( A, B, C \) because they are constants. The decomposition can be simplified to \( \alpha_i \rightarrow R, R\alpha_i \rightarrow \alpha_i\alpha_j \), which is equivalent to \( \alpha_i \rightarrow \alpha_i\alpha_j \), an undecomposed binary rule. The concept “rank-space” is undefined in an undecomposed PCFG.

\(^{\text{9}}\)Below is an intuitive explanation. Assume there is only one nonterminal symbol. Then we can remove \( A, B, C \) because they are constants. The decomposition can be simplified to \( \alpha_i \rightarrow R, R\alpha_i \rightarrow \alpha_i\alpha_j \), which is equivalent to \( \alpha_i \rightarrow \alpha_i\alpha_j \), an undecomposed binary rule. The concept “rank-space” is undefined in an undecomposed PCFG.
5 Experiments

We conduct experiments on the three datasets used in Kim (2021). Details can be found in Appx. D.1.

5.1 SCAN

We first evaluate our models on four splits of the SCAN dataset (Lake and Baroni, 2018). We report accuracy in Tab. 1. The P model equipped with constraints can achieve almost perfect performance similar to vanilla Neural QCFG, while the E model fails due to a lack of constraints.

5.2 Style Transfer and En-Fr Translation

Next, we evaluate the models on the three hard transfer tasks from the StylePTB dataset (Lyu et al., 2021) and a small-scale En-Fr machine translation dataset (Lake and Baroni, 2018). Tab. 2 shows results of the models with different constraints.

5.3 Analysis

We study how the number of nonterminals affects performance. On our computer\textsuperscript{11}, we can use at most 18/64/128 nonterminals in vanilla Neural QCFG/the P model/the E model, showing that our low-rank models are more memory-friendly than vanilla Neural QCFG. We report results in Fig. 2. There is an overall trend of improved performance with more nonterminals (with some notable exceptions). When the numbers of nonterminals are

---

Following Kim (2021), we calculate the metrics for tasks from the StylePTB dataset using the nlg-eval library (Sharma et al. (2017); \url{https://github.com/Maluuba/nlg-eval}) and calculate BLEU for En-Fr MT using the multi-bleu script (Koehn et al. (2007); \url{https://github.com/moses-smt/mosesdecoder}).

\textsuperscript{9}We report speed and memory usage briefly in Sec 5.4 and in detail in Appx. D.3.

\textsuperscript{10}One NVIDIA TITIAN RTX with 24 GB memory.

---

<table>
<thead>
<tr>
<th>Approach</th>
<th>Simple</th>
<th>Jump</th>
<th>A. Right</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>vNQ\textsuperscript{1}</td>
<td>96.9</td>
<td>96.8</td>
<td>98.7</td>
<td>95.7</td>
</tr>
<tr>
<td>E\textsubscript{Model}</td>
<td>9.01</td>
<td>-</td>
<td>1.2</td>
<td>-</td>
</tr>
<tr>
<td>P\textsubscript{Model}</td>
<td>95.27</td>
<td>97.08</td>
<td>97.63</td>
<td>91.72</td>
</tr>
</tbody>
</table>

Table 1: Accuracy on the SCAN datasets. vNQ\textsuperscript{1} is vanilla Neural QCFG from Kim (2021). vNQ\textsuperscript{2} and P\textsubscript{Model} use the hard constraint $d(r) < \infty$.

<table>
<thead>
<tr>
<th>Approach</th>
<th>nil</th>
<th>+H\textsuperscript{1}</th>
<th>+H\textsuperscript{2}</th>
<th>+S</th>
<th>+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>vNQ\textsuperscript{1}</td>
<td>-</td>
<td>66.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>vNQ\textsuperscript{2}</td>
<td>71.42</td>
<td>71.56</td>
<td>71.62</td>
<td>73.86</td>
<td></td>
</tr>
<tr>
<td>E\textsubscript{Model}</td>
<td>73.48</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>74.25</td>
</tr>
<tr>
<td>P\textsubscript{Model}</td>
<td>75.06</td>
<td>69.88</td>
<td>73.11</td>
<td>75.44</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: BLEU-4 for tasks from the StylePTB dataset (the top three series) and BLEU for Fr-En machine translation against different models and constraints. vNQ\textsuperscript{2} is our reimplementation of Kim (2021). nil means that no constraint is placed. H\textsuperscript{1} and H\textsuperscript{2} is the hard constraint $d(r) < \infty$ and $d(r) = 1$, respectively. S is the soft tree hierarchy constraint. C is the coverage constraint. × means that the constraint is inapplicable and − means we do not run the experiment or Kim (2021) does not report the score.
Figure 3: The duration required to train one epoch on synthetic datasets with different length \((x = S = T)\). Thick and shallow lines are fitted curves based on time complexities of \(vNQ^2\), \(E_{\text{Model}}\) and \(P_{\text{Model}}\), i.e., \(O(x^6)\), \(O(x^3)\) and \(O(x^5)\).

Figure 4: Memory usage for training with batch size 1 on synthetic datasets with different length \((x = S = T)\).

5.4 Speed Comparison

We benchmark speed and memory usage using synthetic datasets with different sequence lengths. Fig. 3 and 4 illustrate the results. Compared to the standard Neural QCFG, the E model and P model are significantly faster and have a lower memory footprint. This enables them to model longer sequences effectively. For data construction and more results, please refer to Appx. D.3.

6 Conclusion

We have presented two low-rank variants of Neural QCFG based on decomposition for efficiency and two new constraints over tree hierarchy and source coverage. Experiments on three datasets validate the effectiveness and efficiency of our proposed models and constraints.

7 Limitations

First, unlike decoders in neural seq2seq models, which can attend to any previously generated tokens, QCFGs have a strong context-free independence assumption during generation. With this assumption, Neural QCFG cannot model some complex distributions. A potential solution is to use stronger grammars, such as RNNG (Dyer et al., 2016) and Transformer Grammars (TG; Sartran et al., 2022).

Second, we assume that both the grammars used by the source-side parser and QCFG are in CNF. Although it is convenient for discussion and implementation, CNF does not suit for modeling the structure of practical sequences. In semantic representations (e.g., Abstract Meaning Representation (Banarescu et al., 2013)), a predicate could have more than two arguments. Ideally, we should represent \(n\)-ary predicates with \(n\)-ary rules. However, for grammars in CNF, \(n - 1\) unnatural binary rules are required to represent \(n\)-ary predicates. In natural language, we will face semi-nally meaningless spans due to CNF, which is discussed in Sec 4.2.

Third, although using decomposition improves the speed and the memory requirement, our low-rank models still cost much more computation resources than neural seq2seq models for two main reasons. (1) A large amount of nonterminal symbols increase the memory cost significantly. (2) Because finding the most probable string \(t_2\) from \(p_\theta(t_2|t_1)\) is NP-hard (Sima’an, 1996; Lyngsø and Pedersen, 2002), we follow Kim (2021) to use a decoding strategy with heavy sampling. For real data, we may need to sample hundreds or thousands of sequences and then rank them, which can be much slower than the decoding of neural seq2seq models.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work was supported by the National Natural Science Foundation of China (61976139).

References


A Time Complexity of P Model

Let $\beta_{ij}, \beta_{jk} \in \mathbb{R}^{1+|N||R|}$ be two cells in the chart of the dynamic programming. $\beta_{ij}(x, y)$ denotes indexing into the matrix. Denote $A[\alpha_1] \rightarrow B[\alpha_2] C[\alpha_3]$, as $r_b$. The state transition equation is

$$\beta_{ik}(A, \alpha_1) = \sum_{j,B,C} p(r_b) \beta_{ij}(B, \alpha_2) \beta_{jk}(C, \alpha_3).$$

Let's define following terms:

$$\tilde{\beta}_{ij}(R, \alpha_2) = \sum_B p(R, \alpha_2 \rightarrow B) \beta_{ij}(B, \alpha_2)$$

$$\tilde{\beta}_{jk}(R, \alpha_3) = \sum_C p(R, \alpha_3 \rightarrow C) \beta_{jk}(C, \alpha_3)$$

$$\tilde{p} = p(A[\alpha_1] \rightarrow R)p(R, \alpha_1 \rightarrow \alpha_2, \alpha_3)$$

Then the state transition equation can be reformulated as:

$$\beta_{ik}(A, \alpha_1) = \sum_{R,\alpha_2,\alpha_3} \tilde{p} \tilde{\beta}_{ij}(R, \alpha_2) \tilde{\beta}_{jk}(R, \alpha_3),$$

where $\tilde{\beta}_{ij} \in \mathbb{R}^{1+|N||R|}$. We can compute $\tilde{\beta}_{ij}$ in $O((|N| + |P|)|R|)$ and cache it for composing $\tilde{\beta}_{ij}$. Then $\tilde{\beta}_{ik}$ can be computed in $O(|R|S^2T)$. Finally, we can compute $\hat{\beta}_{ik}$ in $O(|R||S^3|)$ by sum out $\alpha_2, \alpha_3$ first:

$$\hat{\beta}_{ik}(A, \alpha_1) = \sum_R p(A[\alpha_1] \rightarrow R) \sum_{\alpha_2,\alpha_3} p(R, \alpha_1 \rightarrow \alpha_2, \alpha_3) \tilde{\beta}_{ik}.$$

So, summing terms of all the above steps and counting the iteration over $i, k$, we will get $O(|R|S^2T^3 + (2|N| + |P|)|R|S + |R|S^3T^2)$.

B Neural Parameterization

We mainly follow (Kim, 2021) to parameterize the new decomposed rules. First, we add embeddings of terms on the same side together. For example, we do two additions $e_{lhs} = e_{R} + e_{\alpha}$ and $e_{rhs} = e_{\alpha} + e_{\lambda}$ for $R, \alpha_i \rightarrow \alpha_j, \alpha_k$, where $e_x$ denotes the embedding of $x$. Note that we use the same feedforward layer $f$ as (Kim, 2021) to obtain $e_x$ from some feature $h_x$, i.e., $e_x = f(h_x)$. Then, we compute the inner products of embeddings obtained in the previous step as unnormalized scores. For example, $p(R, \alpha_i \rightarrow \alpha_j, \alpha_k) \propto \exp(e_{lhs}^T e_{rhs})$.

C Posterior Regularization

The problem $\min_{q \in \mathcal{Q}} \mathbb{K} \mathcal{L}(q(t)) ||p(t_2|t_1, s_2))$ has the optimal solution

$$q^* = \frac{1}{Z(\lambda^*)} p(t_2|t_1, s_2) \exp\{-\lambda^* \phi(t_2)\},$$

where

$$Z(\lambda^*) = \sum_{t_2} p(t_2|s_1, t_1) \exp\{-\lambda^* \phi(t_2)\}$$

and $\lambda^*$ is the solution of the dual problem:

$$\max_{\lambda \geq 0} -b \cdot \lambda - \log Z(\lambda)$$

We can reuse the inside algorithm to compute $Z(\lambda^*)$ efficiently because our $\phi(t)$ can be factored as $p(t_2|t_1, s_2)$:

$$p(t_2|t_1, s_2) = \prod_{r \in t_2} p_{\theta}(r)$$

$$\phi(t) = \sum_{r \in t_2} \phi(r, t_1),$$

where $\phi(r, t_1) = 1$ if $t_1$ is in the left-hand side of $r$ and $\phi(r, t_1) = 0$ otherwise. Then, the solution $q^*$ can be written as

$$q^*(t_2) \propto \prod_{r \in t_2} p_{\theta}(r) \exp\{-\lambda \phi(r, t_1)\}.$$
D Experiments

D.1 Experimental Details

We implement vNQ, the E model, and the P model using our own codebase. We inherit almost all hyperparameters of Kim (2021) and a basic constraint: the target tree leaves/non-leaf nodes can only be aligned to source tree leaves/non-leaf nodes, and especially, the target tree root can only be aligned to the source tree root. One major difference is that, in our experiments, we do not use early-stopping and run fixed optimization steps, which are much more than the value set in Kim (2021) (i.e., 15). It is because in preliminary experiments, we found that the task metric (e.g., BLEU) almost always get improved consistently with the process of training, while the lowest perplexity occurs typically at an early stage (which is the criteria of early-stopping in Kim (2021)), and computing task metric is very expensive for Neural QCFGs. We report metrics on test sets averaged over three runs on all datasets except for SCAN. As mentioned in the code of Kim (2021), we need to run several times to achieve good performance on SCAN. Therefore, we report the maximum accuracy in twenty runs.

SCAN (Lake and Baroni, 2018) is a diagnostic dataset containing translations from English commands to machine actions. We conduct experiments on four splits: We evaluate our models on four splits of the SCAN (Lake and Baroni, 2018) dataset: simple, add primitive (jump), add template (around right) and length. The latter three splits are designed for evaluating compositional generalization. Following (Kim, 2021), we set $|\mathcal{N}| = 10$, $|\mathcal{P}| = 1$.

StylePTB (Lyu et al., 2021) is a text style transfer dataset built based on Penn Treebank (PTB; Marcus et al., 1993). Following Kim (2021), we conduct experiments on three hard transfer tasks: text to active to passive (2808 examples), adjective emphasis (696 examples) and verb emphasis (1201 examples). According to Tab. 2, we set $|\mathcal{N}| = |\mathcal{P}| = 32$, $|\mathcal{R}| = 100$ for the E model and set $|\mathcal{N}| = |\mathcal{P}| = 64$, $|\mathcal{R}| = 100$ for the P model.

En-Fr MT (Lake and Baroni, 2018) is a small-scale machine translation dataset. We use the same split as Kim (2021). The size of training/validate/test set is 6073/631/583. We set $|\mathcal{N}| = |\mathcal{P}| = 32$, $|\mathcal{R}| = 100$ for the E model and $|\mathcal{N}| = |\mathcal{P}| = 32$, $|\mathcal{R}| = 196$ for the P model.

D.2 Tune Hyperparameter

We tune hyperparameters according to metrics on validation sets, either manually or with the Bayesian Optimization and Hyperband (BOHB) search algorithm (Falkner et al., 2018) built in the wandb library. First, we tune $|\mathcal{N}|$, $|\mathcal{P}|$, $|\mathcal{R}|$ and the learning rate of parameters for parameterizing QCFG. We freeze hyperparameters related to the source-side parser, the contextual encoder (i.e., LSTM), and the TreeLSTM (Tai et al., 2015; Zhu et al., 2015). For the ATP task from StylePTB, we run the grid search to plot Fig. 2 and choose the best hyperparameters. For other tasks, we run about 20 trials according to BOHB for each manually set search range. Typically, the size of a search range is 256 (four choices for each tunable hyperparameter). Next, we tune the strength of the coverage constraint for all models by running with $\gamma = 0.5, 1, 2$.

D.3 Speed and Memory Usage Comparison

Tab. 3 shows the time and memory usage on synthetic datasets. Each synthetic dataset contains 1000 pairs of random sequences with the same length sampled from a vocabulary with size 5000, i.e., $\{(s_1, s_2)_1, \ldots (s_1, s_2)^{1000}\}$, $s_1, s_2 \in \Sigma$, $|\Sigma| = 5000$ where $v$ is the length. We set $|\mathcal{N}| = |\mathcal{P}| = 8$ for vanilla Neural QCFG and $|\mathcal{N}| = |\mathcal{N}| = 50$, $|\mathcal{R}| = 200$ for others. We train models on a computer with an NVIDIA GeForce RTX3090. Note that we disable the copy mechanism in Kim (2021) because of its complicated effects on memory usage, such that the results differ from Fig. 2 (in which models enable the copy mechanism).

12We run 100 epochs and evaluate task metrics on validation sets every 5 epochs.
<table>
<thead>
<tr>
<th>$v$</th>
<th>Approach</th>
<th>Constraint</th>
<th>Batch size</th>
<th>Time (s)</th>
<th>GPU Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vNQ$^2$</td>
<td>$nil$</td>
<td>8</td>
<td>25.6</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>$+H^1$</td>
<td>8</td>
<td>25.5</td>
<td>1.43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+H^2$</td>
<td>8</td>
<td>113.8</td>
<td>7.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+S$</td>
<td>8</td>
<td>60.5</td>
<td>2.46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>8</td>
<td>132.7</td>
<td>3.08</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>EModel</td>
<td>$nil$</td>
<td>8</td>
<td>20.1</td>
<td>1.59</td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>8</td>
<td>40.4</td>
<td>1.59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PModel</td>
<td>$nil$</td>
<td>8</td>
<td>30.7</td>
<td>3.78</td>
</tr>
<tr>
<td></td>
<td>$+H^1$</td>
<td>8</td>
<td>31.3</td>
<td>3.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+H^2$</td>
<td>8</td>
<td>64.0</td>
<td>6.41</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+S$</td>
<td>8</td>
<td>45.8</td>
<td>4.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>8</td>
<td>73.9</td>
<td>4.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vNQ$^2$</td>
<td>$nil$</td>
<td>8</td>
<td>341.2</td>
<td>14.49</td>
</tr>
<tr>
<td></td>
<td>$+H^1$</td>
<td>8</td>
<td>342.4</td>
<td>14.60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+H^2$</td>
<td>1</td>
<td>$\approx$16539.4</td>
<td>14.13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+S$</td>
<td>2</td>
<td>$\approx$1734.4</td>
<td>8.93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>2</td>
<td>$\approx$4657.1</td>
<td>12.24</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>EModel</td>
<td>$nil$</td>
<td>8</td>
<td>40.0</td>
<td>4.58</td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>4</td>
<td>173.4</td>
<td>14.48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PModel</td>
<td>$nil$</td>
<td>8</td>
<td>111.3</td>
<td>8.25</td>
</tr>
<tr>
<td></td>
<td>$+H^1$</td>
<td>8</td>
<td>110.8</td>
<td>8.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+H^2$</td>
<td>4</td>
<td>452.3</td>
<td>9.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+S$</td>
<td>8</td>
<td>269.8</td>
<td>18.76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>4</td>
<td>643.5</td>
<td>18.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vNQ$^2$</td>
<td>1</td>
<td>$\times$</td>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EModel</td>
<td>$nil$</td>
<td>8</td>
<td>82.5</td>
<td>14.95</td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>8</td>
<td>177.0</td>
<td>14.95</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>PModel</td>
<td>$nil$</td>
<td>4</td>
<td>$\approx$2102.7</td>
<td>16.78</td>
</tr>
<tr>
<td></td>
<td>$+H^1$</td>
<td>4</td>
<td>$\approx$2097.6</td>
<td>16.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+H^2$</td>
<td>1</td>
<td>$\times$</td>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+S$</td>
<td>2</td>
<td>$\approx$2729.3</td>
<td>10.63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$+C$</td>
<td>1</td>
<td>$\times$</td>
<td>$\times$</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Time and memory usage on synthetic datasets. We report statistics with as large as possible batch size (in 1, 2, 4, 8). $\times$ represents that we get an out-of-memory error even if we set batch size to 1. $\approx$ represents that the value is estimated using a small portion of the dataset.