Junction Tree Variational Autoencoder for Molecular Graph Generation

Wengong Jin [1] **Regina Barzilay** [1] **Tommi Jaakkola** [1]

Abstract

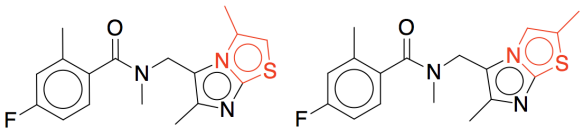
We seek to automate the design of molecules based on specific chemical properties. In computational terms, this task involves continuous embedding and generation of molecular graphs. Our primary contribution is the direct realization of molecular graphs, a task previously approached by generating linear SMILES strings instead of graphs. Our *junction tree variational autoencoder* generates molecular graphs in two phases, by first generating a tree-structured scaffold over chemical substructures, and then combining them into a molecule with a graph message passing network. This approach allows us to incrementally expand molecules while maintaining chemical validity at every step. We evaluate our model on multiple tasks ranging from molecular generation to optimization. Across these tasks, our model outperforms previous state-of-the-art baselines by a significant margin.

1. Introduction

The key challenge of drug discovery is to find target molecules with desired chemical properties. Currently, this task takes years of development and exploration by expert chemists and pharmacologists. Our ultimate goal is to automate this process. From a computational perspective, we decompose the challenge into two complementary subtasks: learning to represent molecules in a continuous manner that facilitates the prediction and optimization of their properties (encoding); and learning to map an optimized continuous representation back into a molecular graph with improved properties (decoding). While deep learning has been extensively investigated for molecular graph encoding (Duvenaud et al., 2015; Kearnes et al., 2016; Gilmer et al., 2017), the harder combinatorial task of molecular graph generation from latent representation remains under-explored.

[1]MIT Computer Science & Artificial Intelligence Lab. Correspondence to: Wengong Jin <wengong@csail.mit.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.



Cc1cn2c(CN(C)C(=O)c3ccc(F)cc3C)c(C)nc2s1
Cc1cc(F)ccc1C(=O)N(C)Cc1c(C)nc2scc(C)n12

Figure 1. Two almost identical molecules with markedly different canonical SMILES in RDKit. The edit distance between two strings is 22 (50.5% of the whole sequence).

Prior work on drug design formulated the graph generation task as a string generation problem (Gómez-Bombarelli et al., 2016; Kusner et al., 2017) in an attempt to side-step direct generation of graphs. Specifically, these models start by generating SMILES (Weininger, 1988), a linear string notation used in chemistry to describe molecular structures. SMILES strings can be translated into graphs via deterministic mappings (e.g., using RDKit (Landrum, 2006)). However, this design has two critical limitations. First, the SMILES representation is not designed to capture molecular similarity. For instance, two molecules with similar chemical structures may be encoded into markedly different SMILES strings (e.g., Figure 1). This prevents generative models like variational autoencoders from learning smooth molecular embeddings. Second, essential chemical properties such as molecule validity are easier to express on graphs rather than linear SMILES representations. We hypothesize that operating directly on graphs improves generative modeling of valid chemical structures.

Our primary contribution is a new generative model of molecular graphs. While one could imagine solving the problem in a standard manner – generating graphs node by node – the approach is not ideal for molecules. This is because creating molecules atom by atom would force the model to generate chemically invalid intermediaries (see, e.g., Figure 2), delaying validation until a complete graph is generated. Instead, we propose to generate molecular graphs in two phases by exploiting valid subgraphs as components. The overall generative approach, cast as a *junction tree variational autoencoder*, first generates a tree structured object (a junction tree) whose role is to represent the scaffold of subgraph components and their coarse relative arrangements. The components are valid chemical substructures automatically extracted from the training set using tree decomposition and are used as building blocks. In the sec-

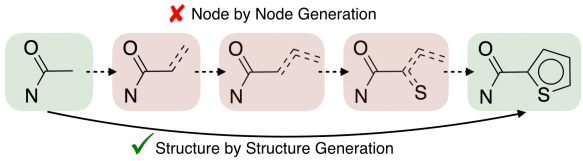


Figure 2. Comparison of two graph generation schemes: Structure by structure approach is preferred as it avoids invalid intermediate states (marked in red) encountered in node by node approach.

ond phase, the subgraphs (nodes in the tree) are assembled together into a coherent molecular graph.

We evaluate our model on multiple tasks ranging from molecular generation to optimization of a given molecule according to desired properties. As baselines, we utilize state-of-the-art SMILES-based generation approaches (Kusner et al., 2017; Dai et al., 2018). We demonstrate that our model produces 100% valid molecules when sampled from a prior distribution, outperforming the top performing baseline by a significant margin. In addition, we show that our model excels in discovering molecules with desired properties, yielding a 30% relative gain over the baselines.

2. Junction Tree Variational Autoencoder

Our approach extends the variational autoencoder (Kingma & Welling, 2013) to molecular graphs by introducing a suitable encoder and a matching decoder. Deviating from previous work (Gómez-Bombarelli et al., 2016; Kusner et al., 2017), we interpret each molecule as having been built from subgraphs chosen out of a vocabulary of valid components. These components are used as building blocks both when encoding a molecule into a vector representation as well as when decoding latent vectors back into valid molecular graphs. The key advantage of this view is that the decoder can realize a valid molecule piece by piece by utilizing the collection of valid components and how they interact, rather than trying to build the molecule atom by atom through chemically invalid intermediaries (Figure 2). An aromatic bond, for example, is chemically invalid on its own unless the entire aromatic ring is present. It would be therefore challenging to learn to build rings atom by atom rather than by introducing rings as part of the basic vocabulary.

Our vocabulary of components, such as rings, bonds and individual atoms, is chosen to be large enough so that a given molecule can be covered by overlapping components or *clusters* of atoms. The clusters serve the role analogous to cliques in graphical models, as they are expressive enough that a molecule can be covered by overlapping clusters without forming cluster cycles. In this sense, the clusters serve as cliques in a (non-optimal) triangulation of the molecular graph. We form a junction tree of such clusters and use it as the tree representation of the molecule. Since our choice of cliques is constrained a priori, we cannot guarantee that a junction tree exists with such clusters for an arbitrary

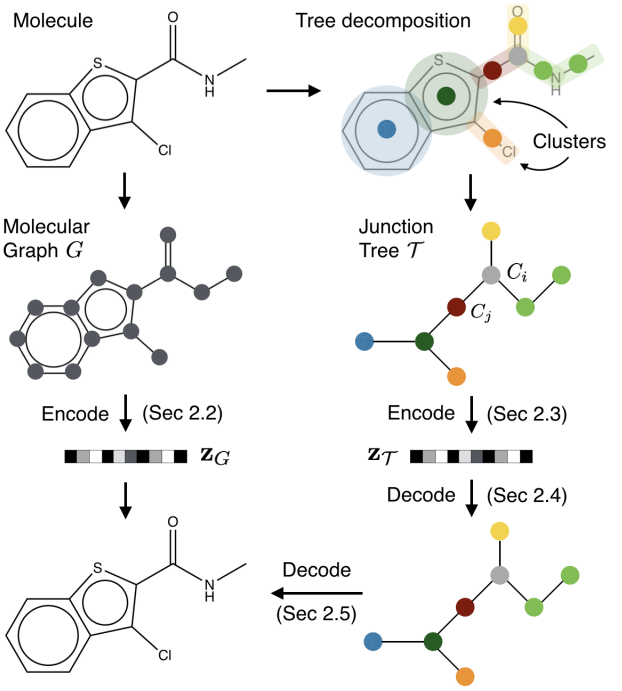


Figure 3. Overview of our method: A molecular graph G is first decomposed into its junction tree $\mathcal{T}_G$, where each colored node in the tree represents a substructure in the molecule. We then encode both the tree and graph into their latent embeddings $\mathbf{z}_\mathcal{T}$ and $\mathbf{z}_G$. To decode the molecule, we first reconstruct junction tree from $\mathbf{z}_\mathcal{T}$, and then assemble nodes in the tree back to the original molecule.

molecule. However, our clusters are built on the basis of the molecules in the training set to ensure that a corresponding junction tree can be found. Empirically, our clusters cover most of the molecules in the test set.

The original molecular graph and its associated junction tree offer two complementary representations of a molecule. We therefore encode the molecule into a two-part latent representation $\mathbf{z} = [\mathbf{z}_\mathcal{T}, \mathbf{z}_G]$ where $\mathbf{z}_\mathcal{T}$ encodes the tree structure and what the clusters are in the tree without fully capturing how exactly the clusters are mutually connected. $\mathbf{z}_G$ encodes the graph to capture the fine-grained connectivity. Both parts are created by tree and graph encoders $q(\mathbf{z}_\mathcal{T}|\mathcal{T})$ and $q(\mathbf{z}_G|G)$. The latent representation is then decoded back into a molecular graph in two stages. As illustrated in Figure 3, we first reproduce the junction tree using a tree decoder $p(\mathcal{T}|\mathbf{z}_\mathcal{T})$ based on the information in $\mathbf{z}_\mathcal{T}$. Second, we predict the fine grain connectivity between the clusters in the junction tree using a graph decoder $p(G|\mathcal{T}, \mathbf{z}_G)$ to realize the full molecular graph. The junction tree approach allows us to maintain chemical feasibility during generation.

Notation A molecular graph is defined as $G = (V, E)$ where V is the set of atoms (vertices) and E the set of bonds (edges). Let $N(x)$ be the neighbor of x. We denote sigmoid function as $\sigma(\cdot)$ and ReLU function as $\tau(\cdot)$. We use i, j, k for nodes in the tree and u, v, w for nodes in the graph.

2.1. Junction Tree

A tree decomposition maps a graph G into a *junction tree* by contracting certain vertices into a single node so that G becomes cycle-free. Formally, given a graph G, a junction tree $\mathcal{T}_G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ is a connected labeled tree whose node set is $\mathcal{V} = \{C_1, \cdots, C_n\}$ and edge set is $\mathcal{E}$. Each node or *cluster* $C_i = (V_i, E_i)$ is an induced subgraph of G, satisfying the following constraints:

1. The union of all clusters equals G. That is, $\bigcup_i V_i = V$ and $\bigcup_i E_i = E$.
2. Running intersection: For all clusters C_i, C_j and C_k, $V_i \cap V_j \subseteq V_k$ if C_k is on the path from C_i to C_j.

Viewing induced subgraphs as cluster labels, junction trees are labeled trees with label vocabulary $\mathcal{X}$. By our molecule tree decomposition, $\mathcal{X}$ contains only cycles (rings) and single edges. Thus the vocabulary size is limited ($|\mathcal{X}| = 780$ for a standard dataset with 250K molecules).

Tree Decomposition of Molecules Here we present our tree decomposition algorithm tailored for molecules, which finds its root in chemistry (Rarey & Dixon, 1998). Our cluster vocabulary $\mathcal{X}$ includes chemical structures such as bonds and rings (Figure 3). Given a graph G, we first find all its simple cycles, and its edges not belonging to any cycles. Two simple rings are merged together if they have more than two overlapping atoms, as they constitute a specific structure called bridged compounds (Clayden et al., 2001). Each of those cycles or edges is considered as a cluster. Next, a cluster graph is constructed by adding edges between all intersecting clusters. Finally, we select one of its spanning trees as the junction tree of G (Figure 3). As a result of ring merging, any two clusters in the junction tree have at most two atoms in common, facilitating efficient inference in the graph decoding phase. The detailed procedure is described in the supplementary.

2.2. Graph Encoder

We first encode the latent representation of G by a graph message passing network (Dai et al., 2016; Gilmer et al., 2017). Each vertex v has a feature vector $\mathbf{x}_v$ indicating the atom type, valence, and other properties. Similarly, each edge $(u, v) \in E$ has a feature vector $\mathbf{x}_{uv}$ indicating its bond type, and two hidden vectors $\boldsymbol{\nu}_{uv}$ and $\boldsymbol{\nu}_{vu}$ denoting the message from u to v and vice versa. Due to the loopy structure of the graph, messages are exchanged in a loopy belief propagation fashion:

$$\boldsymbol{\nu}_{uv}^{(t)} = \tau(\mathbf{W}_1^g \mathbf{x}_u + \mathbf{W}_2^g \mathbf{x}_{uv} + \mathbf{W}_3^g \sum_{w \in N(u) \setminus v} \boldsymbol{\nu}_{wu}^{(t-1)}) \quad (1)$$

where $\boldsymbol{\nu}_{uv}^{(t)}$ is the message computed in t-th iteration, initialized with $\boldsymbol{\nu}_{uv}^{(0)} = \mathbf{0}$. After T steps of iteration, we aggregate those messages as the latent vector of each vertex, which captures its local graphical structure:

$$\mathbf{h}_u = \tau(\mathbf{U}_1^g \mathbf{x}_u + \sum_{v \in N(u)} \mathbf{U}_2^g \boldsymbol{\nu}_{vu}^{(T)}) \quad (2)$$

The final graph representation is $\mathbf{h}_G = \sum_i \mathbf{h}_i / |V|$. The mean $\boldsymbol{\mu}_G$ and log variance $\log \boldsymbol{\sigma}_G$ of the variational posterior approximation are computed from $\mathbf{h}_G$ with two separate affine layers. $\mathbf{z}_G$ is sampled from a Gaussian $\mathcal{N}(\boldsymbol{\mu}_G, \boldsymbol{\sigma}_G)$.

2.3. Tree Encoder

We similarly encode $\mathcal{T}_G$ with a tree message passing network. Each cluster C_i is represented by a one-hot encoding $\mathbf{x}_i$ representing its label type. Each edge (C_i, C_j) is associated with two message vectors $\mathbf{m}_{ij}$ and $\mathbf{m}_{ji}$. We pick an arbitrary leaf node as the root and propagate messages in two phases. In the first bottom-up phase, messages are initiated from the leaf nodes and propagated iteratively towards root. In the top-down phase, messages are propagated from the root to all the leaf nodes. Message $\mathbf{m}_{ij}$ is updated as:

$$\mathbf{m}_{ij} = \text{GRU}(\mathbf{x}_i, \{\mathbf{m}_{ki}\}_{k \in N(i) \setminus j}) \quad (3)$$

where GRU is a Gated Recurrent Unit (Chung et al., 2014; Li et al., 2015) adapted for tree message passing:

$$\mathbf{s}_{ij} = \sum_{k \in N(i) \setminus j} \mathbf{m}_{ki} \quad (4)$$

$$\mathbf{z}_{ij} = \sigma(\mathbf{W}^z \mathbf{x}_i + \mathbf{U}^z \mathbf{s}_{ij} + \mathbf{b}^z) \quad (5)$$

$$\mathbf{r}_{ki} = \sigma(\mathbf{W}^r \mathbf{x}_i + \mathbf{U}^r \mathbf{m}_{ki} + \mathbf{b}^r) \quad (6)$$

$$\widetilde{\mathbf{m}}_{ij} = \tanh(\mathbf{W} \mathbf{x}_i + \mathbf{U} \sum_{k \in N(i) \setminus j} \mathbf{r}_{ki} \odot \mathbf{m}_{ki}) \quad (7)$$

$$\mathbf{m}_{ij} = (1 - \mathbf{z}_{ij}) \odot \mathbf{s}_{ij} + \mathbf{z}_{ij} \odot \widetilde{\mathbf{m}}_{ij} \quad (8)$$

The message passing follows the schedule where $\mathbf{m}_{ij}$ is computed only when all its precursors $\{\mathbf{m}_{ki} \mid k \in N(i) \setminus j\}$ have been computed. This architectural design is motivated by the belief propagation algorithm over trees and is thus different from the graph encoder.

After the message passing, we obtain the latent representation of each node $\mathbf{h}_i$ by aggregating its inward messages:

$$\mathbf{h}_i = \tau(\mathbf{W}^o \mathbf{x}_i + \sum_{k \in N(i)} \mathbf{U}^o \mathbf{m}_{ki}) \quad (9)$$

The final tree representation is $\mathbf{h}_{\mathcal{T}_G} = \mathbf{h}_{root}$, which encodes a rooted tree $(\mathcal{T}, root)$. Unlike the graph encoder, we do not apply node average pooling because it confuses the tree decoder which node to generate first. $\mathbf{z}_{\mathcal{T}_G}$ is sampled in a similar way as in the graph encoder. For simplicity, we abbreviate $\mathbf{z}_{\mathcal{T}_G}$ as $\mathbf{z}_{\mathcal{T}}$ from now on.

This tree encoder plays *two* roles in our framework. First, it is used to compute $\mathbf{z}_{\mathcal{T}}$, which only requires the bottom-up phase of the network. Second, after a tree $\widehat{\mathcal{T}}$ is decoded

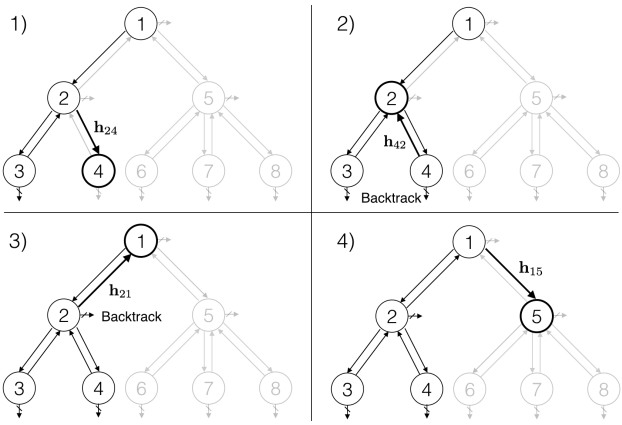


Figure 4. Illustration of the tree decoding process. Nodes are labeled in the order in which they are generated. 1) Node 2 expands child node 4 and predicts its label with message $\mathbf{h}_{24}$. 2) As node 4 is a leaf node, decoder backtracks and computes message $\mathbf{h}_{42}$. 3) Decoder continues to backtrack as node 2 has no more children. 4) Node 1 expands node 5 and predicts its label.

from $\mathbf{z}_\mathcal{T}$, it is used to compute messages $\widehat{\mathbf{m}}_{ij}$ over the entire $\widehat{\mathcal{T}}$, to provide essential contexts of every node during graph decoding. This requires both top-down and bottom-up phases. We will elaborate this in section 2.5.

2.4. Tree Decoder

We decode a junction tree $\mathcal{T}$ from its encoding $\mathbf{z}_\mathcal{T}$ with a tree structured decoder. The tree is constructed in a top-down fashion by generating one node at a time. As illustrated in Figure 4, our tree decoder traverses the entire tree from the root, and generates nodes in their depth-first order. For every visited node, the decoder first makes a *topological prediction*: whether this node has children to be generated. When a new child node is created, we predict its label and recurse this process. Recall that cluster labels represent subgraphs in a molecule. The decoder backtracks when a node has no more children to generate.

At each time step, a node receives information from other nodes in the current tree for making those predictions. The information is propagated through message vectors $\mathbf{h}_{ij}$ when trees are incrementally constructed. Formally, let $\tilde{\mathcal{E}} = \{(i_1, j_1), \cdots, (i_m, j_m)\}$ be the edges traversed in a depth first traversal over $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $m = 2|\mathcal{E}|$ as each edge is traversed in both directions. The model visits node i_t at time t. Let $\tilde{\mathcal{E}}_t$ be the first t edges in $\tilde{\mathcal{E}}$. The message $\mathbf{h}_{i_t,j_t}$ is updated through previous messages:

$$\mathbf{h}_{i_t,j_t} = \text{GRU}(\mathbf{x}_{i_t}, \{\mathbf{h}_{k,i_t}\}_{(k,i_t)\in\tilde{\mathcal{E}}_t, k\neq j_t}) \quad (10)$$

where GRU is the same recurrent unit as in the tree encoder.

Topological Prediction When the model visits node i_t, it makes a binary prediction on whether it still has children to be generated. We compute this probability by combining

Algorithm 1 Tree decoding at sampling time

Require: Latent representation $\mathbf{z}_\mathcal{T}$
Initialize: Tree $\widehat{\mathcal{T}} \leftarrow \emptyset$
function SampleTree(i, t)
  Set $\mathcal{X}_i \leftarrow$ all cluster labels that are chemically compatible with node i and its current neighbors.
  Set $d_t \leftarrow expand$ with probability p_t. ▷ Eq.(11)
  if $d_t = expand$ **and** $\mathcal{X}_i \neq \emptyset$ **then**
    Create a node j and add it to tree $\widehat{\mathcal{T}}$.
    Sample the label of node j from $\mathcal{X}_i$ ▷. Eq.(12)
    SampleTree($j, t+1$)
  end if
end function

$\mathbf{z}_\mathcal{T}$, node features $\mathbf{x}_{i_t}$ and inward messages $\mathbf{h}_{k,i_t}$ via a one hidden layer network followed by a sigmoid function:

$$p_t = \sigma(\mathbf{u}^d \cdot \tau(\mathbf{W}_1^d \mathbf{x}_{i_t} + \mathbf{W}_2^d \mathbf{z}_\mathcal{T} + \mathbf{W}_3^d \sum_{(k,i_t)\in\tilde{\mathcal{E}}_t} \mathbf{h}_{k,i_t}) \quad (11)$$

Label Prediction When a child node j is generated from its parent i, we predict its node label with

$$\mathbf{q}_j = \text{softmax}(\mathbf{U}^l \tau(\mathbf{W}_1^l \mathbf{z}_\mathcal{T} + \mathbf{W}_2^l \mathbf{h}_{ij})) \quad (12)$$

where $\mathbf{q}_j$ is a distribution over label vocabulary $\mathcal{X}$. When j is a root node, its parent i is a virtual node and $\mathbf{h}_{ij} = \mathbf{0}$.

Learning The tree decoder aims to maximize the likelihood $p(\mathcal{T}|\mathbf{z}_\mathcal{T})$. Let $\hat{p}_t \in \{0, 1\}$ and $\hat{\mathbf{q}}_j$ be the ground truth topological and label values, the decoder minimizes the following cross entropy loss:[1]

$$\mathcal{L}_c(\mathcal{T}) = \sum\nolimits_t \mathcal{L}^d(p_t, \hat{p}_t) + \sum\nolimits_j \mathcal{L}^l(\mathbf{q}_j, \hat{\mathbf{q}}_j) \quad (13)$$

Similar to sequence generation, during training we perform *teacher forcing*: after topological and label prediction at each step, we replace them with their ground truth so that the model makes predictions given correct histories.

Decoding & Feasibility Check Algorithm 1 shows how a tree is sampled from $\mathbf{z}_\mathcal{T}$. The tree is constructed recursively guided by topological predictions without any external guidance used in training. To ensure the sampled tree could be realized into a valid molecule, we define set $\mathcal{X}_i$ to be cluster labels that are chemically compatible with node i and its current neighbors. When a child node j is generated from node i, we sample its label from $\mathcal{X}_i$ with a renormalized distribution $\mathbf{q}_j$ over $\mathcal{X}_i$ by masking out invalid labels.

2.5. Graph Decoder

The final step of our model is to reproduce a molecular graph G that underlies the predicted junction tree $\widehat{\mathcal{T}} = (\widehat{\mathcal{V}}, \widehat{\mathcal{E}})$.

[1]The node ordering is not unique as the order within sibling nodes is ambiguous. In this paper we train our model with one ordering and leave this issue for future work.

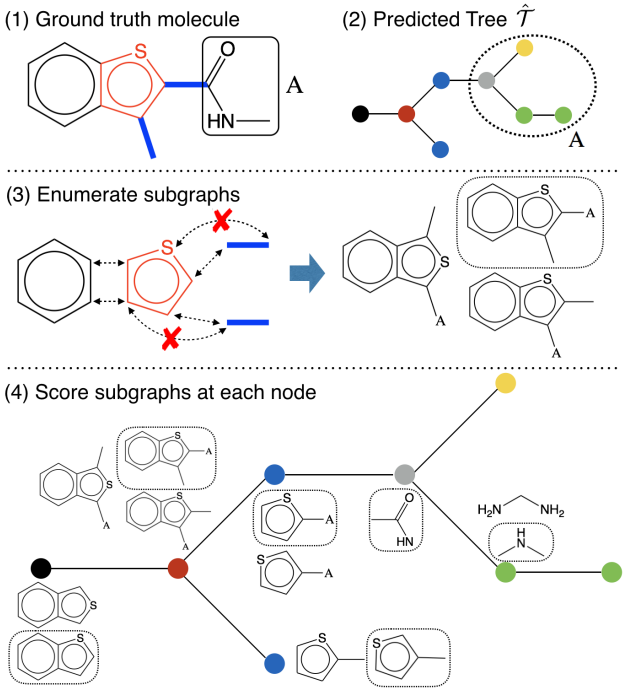


Figure 5. Decode a molecule from a junction tree. 1) Ground truth molecule G. 2) Predicted junction tree $\widehat{\mathcal{T}}$. 3) We enumerate different combinations between red cluster C and its neighbors. Crossed arrows indicate combinations that lead to chemically infeasible molecules. Note that if we discard tree structure during enumeration (i.e., ignoring subtree A), the last two candidates will collapse into the same molecule. 4) Rank subgraphs at each node. The final graph is decoded by putting together all the predicted subgraphs.

Note that this step is not deterministic since there are potentially many molecules that correspond to the same junction tree. The underlying degree of freedom pertains to how neighboring clusters C_i and C_j are attached to each other as subgraphs. Our goal here is to assemble the subgraphs (nodes in the tree) together into the correct molecular graph.

Let $\mathcal{G}(\mathcal{T})$ be the set of graphs whose junction tree is $\mathcal{T}$. Decoding graph $\hat{G}$ from $\widehat{\mathcal{T}} = (\widehat{\mathcal{V}}, \widehat{\mathcal{E}})$ is a structured prediction:

$$\hat{G} = \arg \max_{G' \in \mathcal{G}(\widehat{\mathcal{T}})} f^a(G') \tag{14}$$

where f^a is a scoring function over candidate graphs. We only consider scoring functions that decompose across the clusters and their neighbors. In other words, each term in the scoring function depends only on how a cluster C_i is attached to its neighboring clusters C_j, $j \in N_{\widehat{\mathcal{T}}}(i)$ in the tree $\widehat{\mathcal{T}}$. The problem of finding the highest scoring graph $\hat{G}$ – the assembly task – could be cast as a graphical model inference task in a model induced by the junction tree. However, for efficiency reasons, we will assemble the molecular graph one neighborhood at a time, following the order in which the tree itself was decoded. In other words, we start by sampling the assembly of the root and its neighbors according to their scores. Then we proceed to assemble the neighbors and their associated clusters (removing the degrees of freedom set by the root assembly), and so on.

It remains to be specified how each neighborhood realization is scored. Let G_i be the subgraph resulting from a particular merging of cluster C_i in the tree with its neighbors C_j, $j \in N_{\widehat{\mathcal{T}}}(i)$. We score G_i as a candidate subgraph by first deriving a vector representation $\mathbf{h}_{G_i}$ and then using $f_i^a(G_i) = \mathbf{h}_{G_i} \cdot \mathbf{z}_G$ as the subgraph score. To this end, let u, v specify atoms in the candidate subgraph G_i and let $\alpha_v = i$ if $v \in C_i$ and $\alpha_v = j$ if $v \in C_j \setminus C_i$. The indices α_v are used to mark the position of the atoms in the junction tree, and to retrieve messages $\widehat{\mathbf{m}}_{i,j}$ summarizing the subtree under i along the edge (i, j) obtained by running the tree encoding algorithm. The neural messages pertaining to the atoms and bonds in subgraph G_i are obtained and aggregated into $\mathbf{h}_{G_i}$, similarly to the encoding step, but with different (learned) parameters:

$$\begin{aligned} \boldsymbol{\mu}_{uv}^{(t)} &= \tau(\mathbf{W}_1^a \mathbf{x}_u + \mathbf{W}_2^a \mathbf{x}_{uv} + \mathbf{W}_3^a \widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)}) \\ \widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)} &= \begin{cases} \sum_{w \in N(u) \setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u = \alpha_v \\ \widehat{\mathbf{m}}_{\alpha_u, \alpha_v} + \sum_{w \in N(u) \setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u \neq \alpha_v \end{cases} \end{aligned} \tag{15}$$

The major difference from Eq. (1) is that we augment the model with tree messages $\widehat{\mathbf{m}}_{\alpha_u, \alpha_v}$ derived by running the tree encoder over the predicted tree $\widehat{\mathcal{T}}$. $\widehat{\mathbf{m}}_{\alpha_u, \alpha_v}$ provides a tree dependent positional context for bond (u, v) (illustrated as subtree A in Figure 5).

Learning The graph decoder parameters are learned to maximize the log-likelihood of predicting correct subgraphs G_i of the ground true graph G at each tree node:

$$\mathcal{L}_g(G) = \sum_i \left[f^a(G_i) - \log \sum_{G_i' \in \mathcal{G}_i} \exp(f^a(G_i')) \right] \tag{16}$$

where $\mathcal{G}_i$ is the set of possible candidate subgraphs at tree node i. During training, we again apply teacher forcing, i.e. we feed the graph decoder with ground truth trees as input.

Complexity By our tree decomposition, any two clusters share at most two atoms, so we only need to merge at most two atoms or one bond. By pruning chemically invalid subgraphs and merging isomorphic graphs, $|\mathcal{G}_i| \approx 4$ on average when tested on a standard ZINC drug dataset. The computational complexity of JT-VAE is therefore linear in the number of clusters, scaling nicely to large graphs.

3. Experiments

Our evaluation efforts measure various aspects of molecular generation. The first two evaluations follow previously proposed tasks (Kusner et al., 2017). We also introduce a third task — constrained molecule optimization.

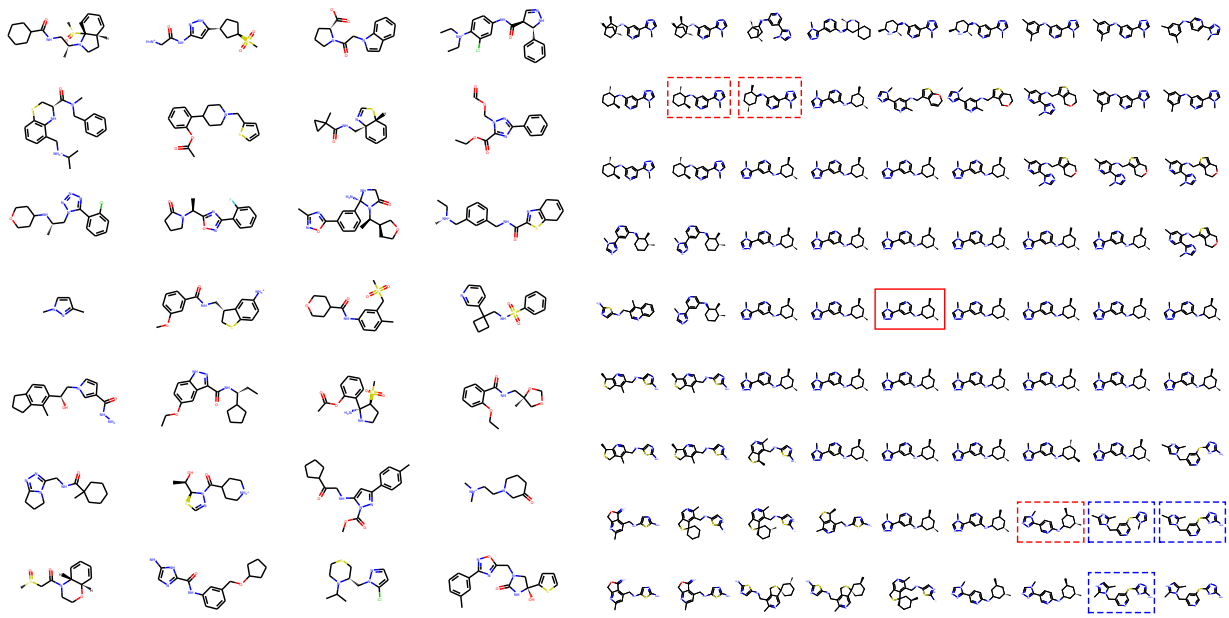


Figure 6. **Left**: Random molecules sampled from prior distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. **Right**: Visualization of the local neighborhood of a molecule in the center. Three molecules highlighted in red dashed box have the same tree structure as the center molecule, but with different graph structure as their clusters are combined differently. The same phenomenon emerges in another group of molecules (blue dashed box).

- **Molecule reconstruction and validity** We test the VAE models on the task of reconstructing input molecules from their latent representations, and decoding valid molecules when sampling from prior distribution. (Section 3.1)
- **Bayesian optimization** Moving beyond generating valid molecules, we test how the model can produce novel molecules with desired properties. To this end, we perform Bayesian optimization in the latent space to search molecules with specified properties. (Section 3.2)
- **Constrained molecule optimization** The task is to modify given molecules to improve specified properties, while constraining the degree of deviation from the original molecule. This is a more realistic scenario in drug discovery, where development of new drugs usually starts with known molecules such as existing drugs (Besnard et al., 2012). Since it is a new task, we cannot compare to any existing baselines. (Section 3.3)

Below we describe the data, baselines and model configuration that are shared across the tasks. Additional setup details are provided in the task-specific sections.

Data We use the ZINC molecule dataset from Kusner et al. (2017) for our experiments, with the same training/testing split. It contains about 250K drug molecules extracted from the ZINC database (Sterling & Irwin, 2015).

Baselines We compare our approach with SMILES-based baselines: 1) Character VAE (CVAE) (Gómez-Bombarelli et al., 2016) which generates SMILES strings character by character; 2) Grammar VAE (GVAE) (Kusner et al., 2017) that generates SMILES following syntactic constraints given by a context-free grammar; 3) Syntax-directed VAE (SD-VAE) (Dai et al., 2018) that incorporates both syntactic and semantic constraints of SMILES via attribute grammar. For molecule generation task, we also compare with GraphVAE (Simonovsky & Komodakis, 2018) that directly generates atom labels and adjacency matrices of graphs.

Model Configuration To be comparable with the above baselines, we set the latent space dimension as 56, i.e., the tree and graph representation $\mathbf{h}_T$ and $\mathbf{h}_G$ have 28 dimensions each. Full training details and model configurations are provided in the appendix.

3.1. Molecule Reconstruction and Validity

Setup The first task is to reconstruct and sample molecules from latent space. Since both encoding and decoding process are stochastic, we estimate reconstruction accuracy by Monte Carlo method used in (Kusner et al., 2017): Each molecule is encoded 10 times and each encoding is decoded 10 times. We report the portion of the 100 decoded molecules that are identical to the input molecule.

To compute validity, we sample 1000 latent vectors from the prior distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and decode each of these vectors 100 times. We report the percentage of decoded molecules that are chemically valid (checked by RDKit). For ablation study, we also report the validity of our model without validity check in decoding phase.

Results Table 1 shows that JT-VAE outperforms previous models in molecule reconstruction, and **always** pro-

Table 1. Reconstruction accuracy and prior validity results. Baseline results are copied from Kusner et al. (2017); Dai et al. (2018); Simonovsky & Komodakis (2018).

Method	Reconstruction	Validity
CVAE	44.6%	0.7%
GVAE	53.7%	7.2%
SD-VAE	76.2%	43.5%
GraphVAE	-	13.5%
JT-VAE (w/o check)	76.4%	93.5%
JT-VAE (full)	**76.7%**	**100.0%**

duces valid molecules when sampled from prior distribution. When validity check is removed, our model could still generates 93.5% valid molecules. This shows our method does not heavily rely on prior knowledge. As shown in Figure 6, the sampled molecules have non-trivial structures such as simple chains. We further sampled 5000 molecules from prior and found they are *all distinct* from the training set. Thus our model is not a simple memorization.

Analysis We qualitatively examine the latent space of JT-VAE by visualizing the neighborhood of molecules. Given a molecule, we follow the method in Kusner et al. (2017) to construct a grid visualization of its neighborhood. Figure 6 shows the local neighborhood of the same molecule visualized in Dai et al. (2018). In comparison, our neighborhood does not contain molecules with huge rings (with more than 7 atoms), which rarely occur in the dataset. We also highlight two groups of closely resembling molecules that have identical tree structures but vary only in how clusters are attached together. This demonstrates the smoothness of learned molecular embeddings.

3.2. Bayesian Optimization

Setup The second task is to produce novel molecules with desired properties. Following (Kusner et al., 2017), our target chemical property $y(\cdot)$ is octanol-water partition coefficients (logP) penalized by the synthetic accessibility (SA) score and number of long cycles.[2] To perform Bayesian optimization (BO), we first train a VAE and associate each molecule with a latent vector, given by the mean of the variational encoding distribution. After the VAE is learned, we train a sparse Gaussian process (SGP) to predict $y(m)$ given its latent representation. Then we perform five iterations of batched BO using the expected improvement heuristic.

For comparison, we report 1) the predictive performance of SGP trained on latent encodings learned by different VAEs, measured by log-likelihood (LL) and root mean square error (RMSE) with 10-fold cross validation. 2) The top-3 molecules found by BO under different models.

[2] $y(m) = logP(m) - SA(m) - cycle(m)$ where $cycle(m)$ counts the number of rings that have more than six atoms.

Table 2. Best molecule property scores found by each method. Baseline results are from Kusner et al. (2017); Dai et al. (2018).

Method	1st	2nd	3rd
CVAE	1.98	1.42	1.19
GVAE	2.94	2.89	2.80
SD-VAE	4.04	3.50	2.96
JT-VAE	**5.30**	**4.93**	**4.49**

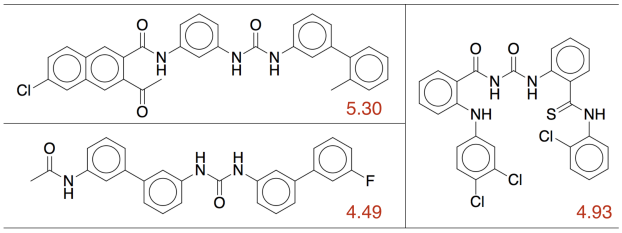


Figure 7. Best three molecules and their property scores found by JT-VAE using Bayesian optimization.

Results As shown in Table 2, JT-VAE finds molecules with significantly better scores than previous methods. Figure 7 lists the top-3 best molecules found by JT-VAE. In fact, JT-VAE finds over 50 molecules with scores over 3.50 (the second best molecule proposed by SD-VAE). Moreover, the SGP yields better predictive performance when trained on JT-VAE embeddings (Table 3).

3.3. Constrained Optimization

Setup The third task is to perform molecule optimization in a constrained scenario. Given a molecule m, the task is to find a different molecule m' that has the highest property value with the molecular similarity $sim(m, m') \geq \delta$ for some threshold δ. We use Tanimoto similarity with Morgan fingerprint (Rogers & Hahn, 2010) as the similarity metric, and penalized logP coefficient as our target chemical property. For this task, we jointly train a property predictor F (parameterized by a feed-forward network) with JT-VAE to predict $y(m)$ from the latent embedding of m. To optimize a molecule m, we start from its latent representation, and apply gradient ascent in the latent space to improve the predicted score $F(\cdot)$, similar to (Mueller et al., 2017). After applying $K = 80$ gradient steps, K molecules are decoded from resulting latent trajectories, and we report the molecule with the highest $F(\cdot)$ that satisfies the similarity constraint. A modification succeeds if one of the decoded molecules satisfies the constraint and is distinct from the original.

To provide the greatest challenge, we selected 800 molecules with the *lowest* property score $y(\cdot)$ from the test set. We report the success rate (how often a modification succeeds), and among success cases the average improvement $y(m') - y(m)$ and molecular similarity $sim(m, m')$ between the original and modified molecules m and m'.

Table 3. Predictive performance of sparse Gaussian Processes trained on different VAEs. Baseline results are copied from Kusner et al. (2017) and Dai et al. (2018).

Method	LL	RMSE
CVAE	-1.812 ± 0.004	1.504 ± 0.006
GVAE	-1.739 ± 0.004	1.404 ± 0.006
SD-VAE	-1.697 ± 0.015	1.366 ± 0.023
JT-VAE	$\mathbf{-1.658 \pm 0.023}$	$\mathbf{1.290 \pm 0.026}$

Table 4. Constrained optimization result of JT-VAE: mean and standard deviation of property improvement, molecular similarity and success rate under constraints $sim(m, m') \geq \delta$ with varied δ.

δ	Improvement	Similarity	Success
0.0	1.91 ± 2.04	0.28 ± 0.15	97.5%
0.2	1.68 ± 1.85	0.33 ± 0.13	97.1%
0.4	0.84 ± 1.45	0.51 ± 0.10	83.6%
0.6	0.21 ± 0.71	0.69 ± 0.06	46.4%

Results Our results are summarized in Table 4. The unconstrained scenario ($\delta = 0$) has the best average improvement, but often proposes dissimilar molecules. When we tighten the constraint to $\delta = 0.4$, about 80% of the time our model finds similar molecules, with an average improvement 0.84. This also demonstrates the smoothness of the learned latent space. Figure 8 illustrates an effective modification resulting in a similar molecule with great improvement.

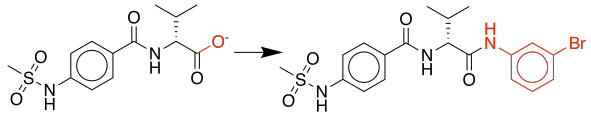


Figure 8. A molecule modification that yields an improvement of 4.0 with molecular similarity 0.617 (modified part is in red).

4. Related Work

Molecule Generation Previous work on molecule generation mostly operates on SMILES strings. Gómez-Bombarelli et al. (2016); Segler et al. (2017) built generative models of SMILES strings with recurrent decoders. Unfortunately, these models could generate invalid SMILES that do not result in any molecules. To remedy this issue, Kusner et al. (2017); Dai et al. (2018) complemented the decoder with syntactic and semantic constraints of SMILES by context free and attribute grammars, but these grammars do not fully capture chemical validity. Other techniques such as active learning (Janz et al., 2017) and reinforcement learning (Guimaraes et al., 2017) encourage the model to generate valid SMILES through additional training signal. Very recently, Simonovsky & Komodakis (2018) proposed to generate molecular graphs by predicting their adjacency matrices, and Li et al. (2018) generated molecules node by node. In comparison, our method enforces chemical validity and is more efficient due to the coarse-to-fine generation.

Graph-structured Encoders The neural network formulation on graphs was first proposed by Gori et al. (2005); Scarselli et al. (2009), and later enhanced by Li et al. (2015) with gated recurrent units. For recurrent architectures over graphs, Lei et al. (2017) designed Weisfeiler-Lehman kernel network inspired by graph kernels. Dai et al. (2016) considered a different architecture where graphs were viewed as latent variable graphical models, and derived their model from message passing algorithms. Our tree and graph encoder are closely related to this graphical model perspective, and to neural message passing networks (Gilmer et al., 2017). For convolutional architectures, Duvenaud et al. (2015) introduced a convolution-like propagation on molecular graphs, which was generalized to other domains by Niepert et al. (2016). Bruna et al. (2013); Henaff et al. (2015) developed graph convolution in spectral domain via graph Laplacian. For applications, graph neural networks are used in semi-supervised classification (Kipf & Welling, 2016), computer vision (Monti et al., 2016), and chemical domains (Kearnes et al., 2016; Schütt et al., 2017; Jin et al., 2017).

Tree-structured Models Our tree encoder is related to recursive neural networks and tree-LSTM (Socher et al., 2013; Tai et al., 2015; Zhu et al., 2015). These models encode tree structures where nodes in the tree are bottom-up transformed into vector representations. In contrast, our model propagates information both bottom-up and top-down.

On the decoding side, tree generation naturally arises in natural language parsing (Dyer et al., 2016; Kiperwasser & Goldberg, 2016). Different from our approach, natural language parsers have access to input words and only predict the topology of the tree. For general purpose tree generation, Vinyals et al. (2015); Aharoni & Goldberg (2017) applied recurrent networks to generate linearized version of trees, but their architectures were entirely sequence-based. Dong & Lapata (2016); Alvarez-Melis & Jaakkola (2016) proposed tree-based architectures that construct trees top-down from the root. Our model is most closely related to Alvarez-Melis & Jaakkola (2016) that disentangles topological prediction from label prediction, but we generate nodes in a depth-first order and have additional steps that propagate information bottom-up. This forward-backward propagation also appears in Parisotto et al. (2016), but their model is node based whereas ours is based on message passing.

5. Conclusion

In this paper we present a junction tree variational autoencoder for generating molecular graphs. Our method significantly outperforms previous work in molecule generation and optimization. For future work, we attempt to generalize our method for general low-treewidth graphs.

Acknowledgement

We thank Jonas Mueller, Chengtao Li, Tao Lei and MIT NLP Group for their helpful comments. This work was supported by the DARPA Make-It program under contract ARO W911NF-16-2-0023.

References

Aharoni, R. and Goldberg, Y. Towards string-to-tree neural machine translation. *arXiv preprint arXiv:1704.04743*, 2017.

Alvarez-Melis, D. and Jaakkola, T. S. Tree-structured decoding with doubly-recurrent neural networks. 2016.

Besnard, J., Ruda, G. F., Setola, V., Abecassis, K., Rodriguiz, R. M., Huang, X.-P., Norval, S., Sassano, M. F., Shin, A. I., Webster, L. A., et al. Automated design of ligands to polypharmacological profiles. *Nature*, 492(7428): 215–220, 2012.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Clayden, J., Greeves, N., Warren, S., and Wothers, P. *Organic Chemistry*. Oxford University Press, 2001.

Dai, H., Dai, B., and Song, L. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pp. 2702–2711, 2016.

Dai, H., Tian, Y., Dai, B., Skiena, S., and Song, L. Syntax-directed variational autoencoder for structured data. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyqShMZRb.

Dong, L. and Lapata, M. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*, 2016.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 2016. doi: 10.1021/acscentsci.7b00572.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pp. 729–734. IEEE, 2005.

Guimaraes, G. L., Sanchez-Lengeling, B., Farias, P. L. C., and Aspuru-Guzik, A. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Janz, D., van der Westhuizen, J., and Hernández-Lobato, J. M. Actively learning what makes a discrete sequence valid. *arXiv preprint arXiv:1708.04465*, 2017.

Jin, W., Coley, C., Barzilay, R., and Jaakkola, T. Predicting organic reaction outcomes with weisfeiler-lehman network. In *Advances in Neural Information Processing Systems*, pp. 2604–2613, 2017.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kiperwasser, E. and Goldberg, Y. Easy-first dependency parsing with hierarchical tree lstms. *arXiv preprint arXiv:1603.00375*, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017.

Landrum, G. Rdkit: Open-source cheminformatics. *Online). http://www. rdkit. org. Accessed*, 3(04):2012, 2006.

Lei, T., Jin, W., Barzilay, R., and Jaakkola, T. Deriving neural architectures from sequence and graph kernels. *arXiv preprint arXiv:1705.09037*, 2017.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. 2018. URL https://openreview.net/forum?id=Hy1d-ebAb.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. *arXiv preprint arXiv:1611.08402*, 2016.

Mueller, J., Gifford, D., and Jaakkola, T. Sequence to better sequence: continuous revision of combinatorial structures. In *International Conference on Machine Learning*, pp. 2536–2544, 2017.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pp. 2014–2023, 2016.

Parisotto, E., Mohamed, A.-r., Singh, R., Li, L., Zhou, D., and Kohli, P. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

Rarey, M. and Dixon, J. S. Feature trees: a new molecular similarity measure based on tree matching. *Journal of computer-aided molecular design*, 12(5):471–490, 1998.

Rogers, D. and Hahn, M. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50 (5):742–754, 2010.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Schütt, K., Kindermans, P.-J., Felix, H. E. S., Chmiela, S., Tkatchenko, A., and Müller, K.-R. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems*, pp. 992–1002, 2017.

Segler, M. H., Kogej, T., Tyrchan, C., and Waller, M. P. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *arXiv preprint arXiv:1701.01329*, 2017.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*, 2018.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Sterling, T. and Irwin, J. J. Zinc 15–ligand discovery for everyone. *J. Chem. Inf. Model*, 55(11):2324–2337, 2015.

Tai, K. S., Socher, R., and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pp. 2773–2781, 2015.

Weininger, D. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Zhu, X., Sobihani, P., and Guo, H. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pp. 1604–1612, 2015.