LEARNING MULTIMODAL GRAPH-TO-GRAPH TRANSLATION FOR MOLECULE OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We view molecule optimization as a graph-to-graph translation problem. The goal is to learn to map from one molecular graph to another with better properties based on an available corpus of paired molecules. Since molecules can be optimized in different ways, there are multiple viable translations for each input graph. A key challenge is therefore to model diverse translation outputs. Our primary contributions include a junction tree encoder-decoder for learning diverse graph translations along with a novel adversarial training method for aligning distributions of molecules. Diverse output distributions in our model are explicitly realized by low-dimensional latent vectors that modulate the translation process. We evaluate our model on multiple molecule optimization tasks and show that our model outperforms previous state-of-the-art baselines by a significant margin.

1 INTRODUCTION

The goal of drug discovery is to design molecules with desirable chemical properties. The task is challenging since the chemical space is vast and often difficult to navigate. One of the prevailing approaches, known as *matched molecular pair analysis* (MMPA) (Griffen et al., 2011; Dossetter et al., 2013), learns rules for generating "molecular paraphrases" that are likely to improve target chemical properties. The setup is analogous to machine translation: MMPA takes as input molecular pairs $\{(X, Y)\}$, where Y is a paraphrase of X with better chemical properties. However, current MMPA methods distill the matched pairs into graph transformation rules rather than treating it as a general translation problem over graphs based on parallel data.

In this paper, we formulate molecule optimization as *graph-to-graph translation*. Given a corpus of molecular pairs, our goal is to learn to translate input molecular graphs into better graphs. The proposed translation task involves many challenges. While several methods are available to encode graphs (Duvenaud et al., 2015; Li et al., 2015; Lei et al., 2017), generating graphs as output is more challenging without resorting to a domain-specific graph linearization. In addition, the target molecular paraphrases are diverse since multiple strategies can be applied to improve a molecule. Therefore, our goal is to learn multimodal output distributions over graphs.

To this end, we propose *junction tree encoder-decoder*, a refined graph-to-graph neural architecture that decodes molecular graphs with neural attention. To capture diverse outputs, we introduce stochastic latent codes into the decoding process and guide these codes to capture meaningful molecular variations. The basic learning problem can be cast as a variational autoencoder. We constrain the posterior inference over the latent codes to only depend on the target molecule Y rather than both on X and Y. The motivation is that the additional hints required for translation should depend only on the target type. Further, to avoid invalid translations, we propose a novel adversarial training method to align the distribution of graphs generated from the model using randomly selected latent codes with the observed distribution of valid targets. Specifically, we perform adversarial regularization on the level of the hidden states created as part of the graph generation.

We evaluate our model on three molecule optimization tasks, with target properties ranging from drug likeness to biological activity. As baselines, we utilize state-of-the-art graph generation methods (Jin et al., 2018; You et al., 2018a). We demonstrate that our model excels in discovering molecules with desired properties, yielding 35% to 94% relative gain over the best baseline. In addition, our model can translate a given molecule into a diverse set of compounds, significantly outperforming baselines in terms of diversity.

2 RELATED WORK

Molecule Generation/Optimization Prior work on molecule optimization approached the graph translation task through generative modeling (Gómez-Bombarelli et al., 2016; Segler et al., 2017; Kusner et al., 2017; Dai et al., 2018; Jin et al., 2018; Samanta et al., 2018; Li et al., 2018a) and reinforcement learning (Guimaraes et al., 2017; Olivecrona et al., 2017; Popova et al., 2018; You et al., 2018a). Earlier approaches represented molecules as SMILES strings (Weininger, 1988), while more recent methods represented them as graphs. Most of these methods coupled a molecule generator with a property predictor and solved the optimization problem by finding the molecule with the best property score. We argue that this type of approach is suboptimal as it depends on the reliability of the property predictor. While one can train a property predictor from a set of molecules with labeled properties, there is no guarantee that it generalizes well to the new chemical space proposed by the graph generator. We sidestep this problem by directly learning to translate a graph into a better graph, unifying graph generation and property estimation in one model.

Our approach is closely related to matched molecular pair analysis (MMPA) (Griffen et al., 2011; Dossetter et al., 2013) in drug de novo design, where the matched pairs are hard-coded into graph transformation rules. MMPA's main drawback is that large numbers of rules have to be realized (e.g. millions) to cover all the complex transformation patterns. In contrast, our approach uses neural networks to learn such transformations, which does not require the rules to be explicitly realized.

Graph Neural Networks Our work is related to graph encoders and decoders. Previous work on graph encoders includes convolutional (Scarselli et al., 2009; Bruna et al., 2013; Henaff et al., 2015; Duvenaud et al., 2015; Niepert et al., 2016; Defferrard et al., 2016; Kondor et al., 2018) and recurrent architectures (Li et al., 2015; Dai et al., 2016; Lei et al., 2017). Graph encoders have been widely applied to network analysis (Kipf & Welling, 2016; Hamilton et al., 2017; Xu et al., 2018) and chemistry (Kearnes et al., 2016; Gilmer et al., 2017; Schütt et al., 2017; Jin et al., 2017). Recently proposed graph decoders (Li et al., 2018b; Jin et al., 2018; You et al., 2018b; Liu et al., 2018) focus on learning generative models of graphs. While our model builds on Jin et al. (2018) to generate graphs, we contribute new techniques to learn multimodal graph-to-graph mappings.

Image/Text Style Translation Our work is closely related to image-to-image translation (Isola et al., 2017), which was later extended by Zhu et al. (2017) to learn multimodal mappings. Our adversarial training technique is inspired by recent text style transfer methods (Shen et al., 2017; Zhao et al., 2018) that adversarially regularize the continuous representation of discrete structures to enable end-to-end training. Our technical contribution is a novel adversarial regularization over graphs that constrains their scaffold structures in a continuous manner.

3 JUNCTION TREE ENCODER-DECODER

Our translation model extends the junction tree variational autoencoder (Jin et al., 2018) to an encoder-decoder architecture for learning graph-to-graph mappings. Following their work, we interpret each molecule as having been built from subgraphs (clusters of atoms) chosen from a vocabulary of valid chemical substructures. The clusters form a junction tree representing the *scaffold* structure of molecules (Figure 1), which is an important factor in drug design. Molecules are decoded hierarchically by first generating the junction trees and then combining the nodes of the tree into a molecule. This coarse-to-fine approach allows us to easily enforce the chemical validity of generated graphs, and provides an enriched representation that encodes molecules at different scales.

In terms of model architecture, the encoder is a graph message passing network that embeds both nodes in the tree and graph into continuous vectors. The decoder consists of a tree-structured decoder for predicting junction trees, and a graph decoder that learns to combine clusters in the predicted junction tree into a molecule. Our key departures from Jin et al. (2018) include a unified encoder architecture for trees and graphs, along with an attention mechanism in the tree decoding process.

3.1 TREE AND GRAPH ENCODER

Viewing trees as graphs, we encode both junction trees and graphs using graph message passing networks. Specifically, a graph is defined as $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the vertex set and $\mathcal{E}$ the edge set. Each node v has a feature vector $\boldsymbol{f}_v$. For atoms, it includes the atom type, valence, and other

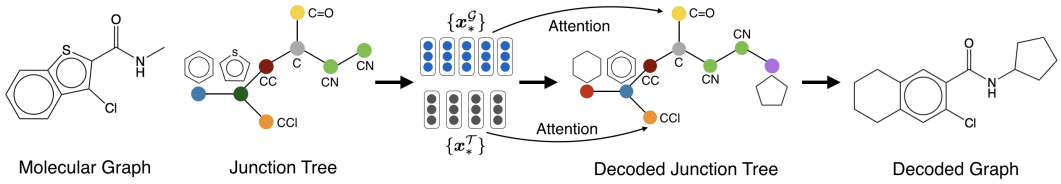

Figure 1: Illustration of our encoder-decoder model. Molecules are represented by their graph structures and junction trees encoding the *scaffold* of molecules. Nodes in the junction tree (which we call *clusters*) are valid chemical substructures such as rings and bonds. During decoding, the model first generates its junction tree and then combines clusters in the predicted tree into a molecule.

atomic properties. For clusters in the junction tree, $\boldsymbol{f}_v$ is a one-hot vector indicating its cluster label. Similarly, each edge $(u, v) \in \mathcal{E}$ has a feature vector $\boldsymbol{f}_{uv}$. Let $N(v)$ be the set of neighbor nodes of v. There are two hidden vectors $\boldsymbol{\nu}_{uv}$ and $\boldsymbol{\nu}_{vu}$ for each edge (u, v) representing the message from u to v and vice versa. These messages are updated iteratively via neural network $g_1(\cdot)$:

$$\boldsymbol{\nu}_{uv}^{(t)} = g_1\left(\boldsymbol{f}_u, \boldsymbol{f}_{uv}, \sum\nolimits_{w \in N(u) \setminus v} \boldsymbol{\nu}_{wu}^{(t-1)}\right) \tag{1}$$

where $\boldsymbol{\nu}_{uv}^{(t)}$ is the message computed in the t-th iteration, initialized with $\boldsymbol{\nu}_{uv}^{(0)} = \mathbf{0}$. In each iteration, all messages are updated asynchronously, as there is no natural order among the nodes. This is different from the tree encoding algorithm in Jin et al. (2018), where a root node was specified and an artificial order was imposed on the message updates. Removing this artifact is necessary as the learned embeddings will be biased by the artificial order.

After T steps of iteration, we aggregate messages via another neural network $g_2(\cdot)$ to derive the latent vector of each vertex, which captures its local graphical structure:

$$\boldsymbol{x}_u = g_2\left(\boldsymbol{f}_u, \sum\nolimits_{v \in N(u)} \boldsymbol{\nu}_{vu}^{(T)}\right) \tag{2}$$

Applying the above message passing network to junction tree $\mathcal{T}$ and to graph G yields two sets of vectors $\{\boldsymbol{x}_1^{\mathcal{T}}, \cdots, \boldsymbol{x}_n^{\mathcal{T}}\}$ and $\{\boldsymbol{x}_1^{\mathcal{G}}, \cdots, \boldsymbol{x}_n^{\mathcal{G}}\}$, which we call *source tree vectors* and *graph vectors*. $\boldsymbol{x}_i^{\mathcal{T}}$ is the embedding of tree node i, and $\boldsymbol{x}_j^{\mathcal{G}}$ is the embedding of graph node j.

3.2 Junction Tree Decoder

We generate a junction tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with a tree recurrent neural network with an attention mechanism. The tree is constructed in a top-down fashion by expanding the tree one node at a time. Formally, let $\tilde{\mathcal{E}} = \{(i_1, j_1), \cdots, (i_m, j_m)\}$ be the edges traversed in a depth first traversal over tree $\mathcal{T}$, where $m = 2|\mathcal{E}|$ as each edge is traversed in both directions. Let $\tilde{\mathcal{E}}_t$ be the first t edges in $\tilde{\mathcal{E}}$. At the t-th decoding step, the model visits node i_t and receives message vectors $\boldsymbol{h}_{ij}$ from its neighbors. The message $\boldsymbol{h}_{i_t,j_t}$ is updated through a tree Gated Recurrent Unit (Jin et al., 2018):

$$\boldsymbol{h}_{i_t,j_t} = \mathrm{GRU}(\boldsymbol{f}_{i_t}, \{\boldsymbol{h}_{k,i_t}\}_{(k,i_t) \in \tilde{\mathcal{E}}_t, k \neq j_t}) \tag{3}$$

Topological Prediction When the model visits node i_t, it first computes a predictive hidden state $\boldsymbol{h}_t$ by combining node features $\boldsymbol{f}_{i_t}$ and inward messages $\{\boldsymbol{h}_{k,i_t}\}$ via a one hidden layer network. The model then makes a binary prediction on whether to expand a new node or backtrack to the parent of i_t. This probability is computed by aggregating the source encodings $\{\boldsymbol{x}_*^{\mathcal{T}}\}$ and $\{\boldsymbol{x}_*^{\mathcal{G}}\}$ through an attention layer, followed by a feed-forward network ($\tau(\cdot)$ stands for ReLU and $\sigma(\cdot)$ for sigmoid):

$$\boldsymbol{h}_t = \tau(\boldsymbol{W}_1^d \boldsymbol{f}_{i_t} + \boldsymbol{W}_2^d \sum\nolimits_{(k,i_t) \in \tilde{\mathcal{E}}_t} \boldsymbol{h}_{k,i_t}) \tag{4}$$

$$\boldsymbol{c}_t^d = \mathrm{attention}\left(\boldsymbol{h}_t, \{\boldsymbol{x}_*^{\mathcal{T}}\}, \{\boldsymbol{x}_*^{\mathcal{G}}\}; \boldsymbol{U}_{att}^d\right) \tag{5}$$

$$\boldsymbol{p}_t = \sigma\left(\boldsymbol{u}^d \cdot \tau(\boldsymbol{W}_3^d \boldsymbol{h}_t + \boldsymbol{W}_4^d \boldsymbol{c}_t^d)\right) \tag{6}$$

Here we use $\mathrm{attention}(\cdot; \boldsymbol{U}_{att}^d)$ to mean the attention mechanism with parameters $\boldsymbol{U}_{att}^d$. It computes two set of attention scores $\{\boldsymbol{\alpha}_*^{\mathcal{T}}\}, \{\boldsymbol{\alpha}_*^{\mathcal{G}}\}$ (normalized by softmax) over source tree and graph vectors respectively. The output $\boldsymbol{c}_t^d$ is a concatenation of tree and graph attention vectors:

$$\boldsymbol{c}_t^d = \left[\sum_i \boldsymbol{\alpha}_{i,t}^{\mathcal{T}} \boldsymbol{x}_i^{\mathcal{T}}, \sum_i \boldsymbol{\alpha}_{i,t}^{\mathcal{G}} \boldsymbol{x}_i^{\mathcal{G}} \right] \tag{7}$$

Label Prediction If node j_t is a new child to be generated from parent i_t, we predict its label by

$$\boldsymbol{c}_t^l = \text{attention}(\boldsymbol{h}_{i_t,j_t}, \{\boldsymbol{x}_*^{\mathcal{T}}\}, \{\boldsymbol{x}_*^{\mathcal{G}}\}; \boldsymbol{U}_{att}^l) \tag{8}$$

$$\boldsymbol{q}_t = \text{softmax}(\boldsymbol{U}^l \tau(\boldsymbol{W}_1^l \boldsymbol{h}_{i_t,j_t} + \boldsymbol{W}_2^l \boldsymbol{c}_t^l)) \tag{9}$$

where $\boldsymbol{q}_t$ is a distribution over the label vocabulary and $\boldsymbol{U}_{att}^l$ is another set of attention parameters.

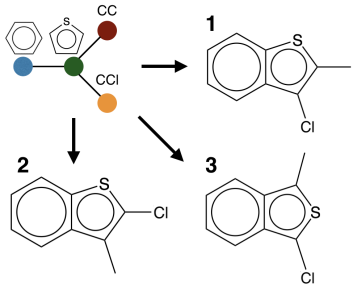

Figure 2: Multiple ways to assemble neighboring clusters in the junction tree.

3.3 Graph Decoder

The second step in the decoding process is to construct a molecular graph G from a predicted junction tree $\widehat{\mathcal{T}}$. This step is not deterministic since multiple molecules could correspond to the same junction tree. For instance, the junction tree in Figure 2 can be assembled into three different molecules. The underlying degree of freedom pertains to how neighboring clusters are attached to each other. Let $\mathcal{G}_i$ be the set of possible candidate attachments at tree node i. Each graph $G_i \in \mathcal{G}_i$ is a particular realization of how cluster C_i is attached to its neighboring clusters $\{C_j, j \in N_{\widehat{\mathcal{T}}}(i)\}$. The goal of the graph decoder is to predict the correct attachment between the clusters.

To this end, we design the following scoring function $f(\cdot)$ for ranking candidate attachments within the set $\mathcal{G}_i$. We first apply a graph message passing network over graph G_i to compute atom representations $\{\boldsymbol{\mu}_v^{G_i}\}$. Then we derive a vector representation of G_i through sum-pooling: $\boldsymbol{m}_{G_i} = \sum_v \boldsymbol{\mu}_v^{G_i}$. Finally, we score candidate G_i by computing dot products between $\boldsymbol{m}_{G_i}$ and the encoded source graph vectors: $f(G_i) = \sum_{u \in G} \boldsymbol{m}_{G_i} \cdot \boldsymbol{x}_u^{\mathcal{G}}$.

The graph decoder is trained to maximize the log-likelihood of ground truth subgraphs at all tree nodes (Eq. (10)). During training, we apply teacher forcing by feeding the graph decoder with ground truth junction tree as input. During testing, we assemble the graph one neighborhood at a time, following the order in which the junction tree was decoded.

$$\mathcal{L}_g(G) = \sum\nolimits_i \left[f(G_i) - \log \sum\nolimits_{G_i' \in \mathcal{G}_i} \exp(f(G_i')) \right] \tag{10}$$

4 Multimodal Graph-to-Graph Translation

Our goal is to learn a multimodal mapping between two molecule domains, such as molecules with low and high solubility, or molecules that are potent and impotent. During training, we are given a dataset of paired molecules $\{(X, Y)\} \subset \mathcal{X} \times \mathcal{Y}$ sampled from their joint distribution $P(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}, \mathcal{Y}$ are the source and target domains. It is important to note that this joint distribution is a many-to-many mapping. For instance, there exist many ways to modify molecule X to increase its solubility. Given a new molecule X, the model should be able to generate a diverse set of outputs.

To this end, we propose to augment the basic encoder-decoder model with low-dimensional latent vectors $\boldsymbol{z}$ to explicitly encode the multimodal aspect of the output distribution. The mapping to be learned now becomes $\mathcal{F} : (X, \boldsymbol{z}) \to Y$, with latent code $\boldsymbol{z}$ drawn from a prior distribution $P(\boldsymbol{z})$, which is a standard Gaussian $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. There are two challenges in learning this mapping. First, as shown in the image domain (Zhu et al., 2017), the latent codes are often ignored by the model unless we explicitly enforce the latent codes to encode meaningful variations. Second, the model should be properly regularized so that it does not produce invalid translations. That is, the translated molecule $\mathcal{F}(X, \boldsymbol{z})$ should always belong to the target domain $\mathcal{Y}$ given latent code $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. In this section, we propose two techniques to address these issues.

4.1 Variational Junction Tree Encoder-Decoder (VJTNN)

First, to encode meaningful variations, we derive latent code $\boldsymbol{z}$ from the embedding of ground truth molecule Y. The decoder is trained to reconstruct Y when taking as input both its vector encoding

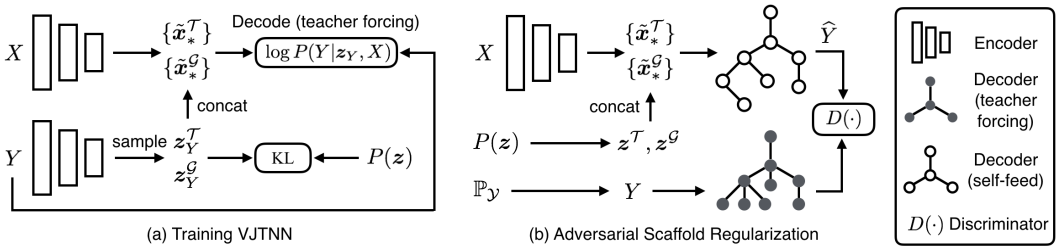

Figure 3: Multimodal graph-to-graph learning. Our model combines the strength of both variational JTNN and adversarial scaffold regularization.

$\boldsymbol{z}_Y$ and source molecule X. For efficient sampling, the latent code distribution is regularized to be close to the prior distribution, similar to a variational autoencoder. We also restrict $\boldsymbol{z}_Y$ to be a low dimensional vector to prevent the model from ignoring input X and degenerating to an autoencoder.

Specifically, we first embed molecules X and Y into their tree and graph vectors $\{\boldsymbol{x}_*^{\mathcal{T}}\}, \{\boldsymbol{x}_*^{\mathcal{G}}\}$; $\{\boldsymbol{y}_*^{\mathcal{T}}\}, \{\boldsymbol{y}_*^{\mathcal{G}}\}$, using the same encoder with shared parameters (Sec 3.1). Then we perform average pooling over the tree and graph vectors of Y: $\boldsymbol{y}_{\mathcal{T}} = \sum_i \boldsymbol{y}_i^{\mathcal{T}}/|\mathcal{T}_Y|$ and $\boldsymbol{y}_{\mathcal{G}} = \sum_i \boldsymbol{y}_i^{\mathcal{G}}/|\mathcal{G}_Y|$. We sample $\boldsymbol{z}_Y^{\mathcal{T}}$ and $\boldsymbol{z}_Y^{\mathcal{G}}$ from the posterior $Q(\cdot|Y)$ via the reparameterization trick (Kingma & Welling, 2013), where the mean and log variance are computed from $\boldsymbol{y}_{\mathcal{T}}$ and $\boldsymbol{y}_{\mathcal{G}}$ with two separate affine layers. Finally, we combine the latent code $\boldsymbol{z}_Y^{\mathcal{T}}$ and $\boldsymbol{z}_Y^{\mathcal{G}}$ with source tree and graph vectors:

$$\tilde{\boldsymbol{x}}_i^{\mathcal{T}} = \tau(\boldsymbol{W}_1^e \boldsymbol{x}_i^{\mathcal{T}} + \boldsymbol{W}_2^e \boldsymbol{z}_Y^{\mathcal{T}}); \qquad \tilde{\boldsymbol{x}}_i^{\mathcal{G}} = \tau(\boldsymbol{W}_3^e \boldsymbol{x}_i^{\mathcal{G}} + \boldsymbol{W}_4^e \boldsymbol{z}_Y^{\mathcal{G}}); \tag{11}$$

where $\tilde{\boldsymbol{x}}_*^{\mathcal{T}}$ and $\tilde{\boldsymbol{x}}_*^{\mathcal{G}}$ are "perturbed" tree and graph vectors of molecule X. The perturbed inputs are then fed into the decoder to synthesize the target molecule $\widehat{Y}$. The training objective follows a conditional variational autoencoder, including a reconstruction loss and a KL regularization term:

$$\mathcal{L}^{\text{VAE}}(X, Y) = -\mathbb{E}_{\boldsymbol{z}\sim Q}[\log P(Y|\boldsymbol{z}, X)] + \lambda_{\text{KL}} \mathcal{D}_{\text{KL}}[Q(\boldsymbol{z}|Y)||P(\boldsymbol{z})] \tag{12}$$

4.2 Adversarial Scaffold Regularization

Second, to avoid invalid translations, we force molecules decoded from latent codes $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ to follow the distribution of the target domain through adversarial training (Goodfellow et al., 2014). The adversarial game involves two components. The discriminator tries to distinguish real molecules in the target domain from fake molecules generated by the model. The generator (i.e. our encoder-decoder) tries to generate molecules indistinguishable from the molecules in the target domain.

The main challenge is how to integrate adversarial training into our decoder, as the discrete decisions in tree and graph decoding hinder gradient propagation. While it is possible to estimate the gradient using REINFORCE (Williams, 1992), training with these methods could be unstable due to the high variance of sampled gradients. Moreover, those methods require the model to assemble multiple junction tree samples into graphs, thus invoking candidate attachment enumeration multiple times and significantly slowing down the training process.

To overcome these issues, we instead apply *adversarial regularization* over continuous representations of decoded molecular structures, derived from the hidden states in the decoder (Shen et al., 2017; Zhao et al., 2018). That is, we replace the input of the discriminator with continuous embeddings of discrete outputs. For efficient training, we only enforce the adversarial regularization in the tree decoding step.[1] As a result, the adversary only matches the scaffold structure between translated molecules and true samples. While it is an approximation, we found this approach still yields a significant improvement when combined with VJTNN, as chemical properties are largely determined by their scaffold structures.

It remains to be specified how the continuous representation is computed, and how we can decode junction trees such that the gradient can be back-propagated through the entire tree decoding process.

[1]While it is desirable to apply the same idea to graph decoding, the subgraph enumeration requires the tree decoder to make "hard" decisions over node labels, blocking gradient propagation inevitably. We leave this issue for future work.

Algorithm 1 Training VJTNN with Adversarial Scaffold Regularization

1: **for** each training iteration **do**
2: *(1)* ***Train variational JTNN (Sec. 4.1)***
3: Sample batch $\{(X^{(i)}, Y^{(i)})\}_{i=1}^m \sim \mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$ and compute latent code $\boldsymbol{z}_Y^{(i)}$ for molecule $Y^{(i)}$.
4: Train the encoder/decoder by minimizing $\sum_i \mathcal{L}^{\text{VAE}}(X^{(i)}, Y^{(i)})$
5: *(2)* ***Train the discriminator*** $D(\cdot)$ ***for*** N ***iterations***
6: **for** $k \leftarrow 1$ to N **do**
7: Sample batch $\{X^{(i)}\}_{i=1}^m \sim \mathbb{P}_{\mathcal{X}}$ and $\{Y^{(i)}\}_{i=1}^m \sim \mathbb{P}_{\mathcal{Y}}$.
8: Let $\mathcal{T}^{(i)}$ be the junction tree of molecule $Y^{(i)}$. For each $\mathcal{T}^{(i)}$, compute its continuous representation $\boldsymbol{h}^{(i)}$ by unrolling the decoder with teacher forcing.
9: Encode each molecule $X^{(i)}$ with latent codes $\boldsymbol{z}^{(i)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
10: For each i, unroll the decoder by feeding the predicted labels and tree topologies to construct the translated junction tree $\widehat{\mathcal{T}}^{(i)}$, and compute its continuous representation $\widehat{\boldsymbol{h}}^{(i)}$.
11: Update $D(\cdot)$ by minimizing $\frac{1}{m}\sum_{i=1}^m -D(\boldsymbol{h}^{(i)}) + D(\widehat{\boldsymbol{h}}^{(i)})$ along with gradient penalty.
12: **end for**
13: *(3)* ***Train the encoder/decoder adversarially***
14: Sample batch $\{X^{(i)}\}_{i=1}^m \sim \mathbb{P}_{\mathcal{X}}$ and $\{Y^{(i)}\}_{i=1}^m \sim \mathbb{P}_{\mathcal{Y}}$.
15: Repeat lines 8-10.
16: Update encoder/decoder by minimizing $\frac{1}{m}\sum_{i=1}^m D(\boldsymbol{h}^{(i)}) - D(\widehat{\boldsymbol{h}}^{(i)})$.
17: **end for**

The decoder first predicts the label distribution $\boldsymbol{q}_{root}$ of the root of tree $\widehat{\mathcal{T}}$. Starting from the root, we incrementally expand the tree, guided by topological predictions, and compute the hidden messages $\{\boldsymbol{h}_{i_t,j_t}\}$ between nodes in the partial tree. At timestep t, the model decides to either expand a new node j_t or backtrack to the parent of node i_t. We denote this binary decision as $d(i_t, j_t) = \mathbf{1}_{\boldsymbol{p}_t > 0.5}$, which is determined by the topological score $\boldsymbol{p}_t$ in Eq.(6). For the true samples Y, the hidden messages are computed by Eq.(3) with teacher-forcing, namely replacing the label and topological predictions with their ground truth values. For the translated samples $\widehat{Y}$ from source molecules X, we replace the one-hot encoding $\boldsymbol{f}_{i_t}$ with its softmax distribution $\boldsymbol{q}_{i_t}$ over cluster labels in Eq.(3) and (4). Moreover, we multiply message $\boldsymbol{h}_{i_t,j_t}$ with the binary gate $d(i_t, j_t)$, to account for the fact that the messages should depend on the topological layout of the tree:

$$\boldsymbol{h}_{i_t,j_t} = \begin{cases} d(i_t,j_t) \cdot \text{GRU}(\boldsymbol{q}_{i_t}, \{\boldsymbol{h}_{k,i_t}\}_{(k,i_t)\in\tilde{\mathcal{E}}_t, k\neq j_t}) & \text{if } j_t \text{ is a child of node } i_t \\ (1-d(i_t,j_t)) \cdot \text{GRU}(\boldsymbol{q}_{i_t}, \{\boldsymbol{h}_{k,i_t}\}_{(k,i_t)\in\tilde{\mathcal{E}}_t, k\neq j_t}) & \text{vice versa} \end{cases} \tag{13}$$

As $d(i_t, j_t)$ is computed by a non-differentiable threshold function, we approximate its gradient with a straight-through estimator (Bengio et al., 2013; Courbariaux et al., 2016). Specifically, we replace the threshold function with a differentiable hard sigmoid function during back-propagation, while using the threshold function in the forward pass. This technique has been successfully applied to training neural networks with dynamic computational graphs (Chung et al., 2016).

Finally, after the tree $\mathcal{T}$ is completely decoded, we derive its continuous representation $\boldsymbol{h}_{\mathcal{T}}$ by concatenating the root label distribution $\boldsymbol{q}_{root}$ and the sum of its inward messages:

$$\boldsymbol{s}_{root} = \sum\nolimits_{k\in N(root)} \boldsymbol{h}_{k,root} \qquad \boldsymbol{h}_{\mathcal{T}} = [\boldsymbol{q}_{root}, \boldsymbol{s}_{root}] \tag{14}$$

We implement the discriminator $D(\cdot)$ as a multi-layer feedforward network, and train the adversary using Wasserstein GAN with gradient penalty (Arjovsky et al., 2017; Gulrajani et al., 2017). The whole algorithm is described in Algorithm 1.

5 EXPERIMENTS

Data Our graph-to-graph translation models are evaluated on three molecule optimization tasks. Following standard practice in MMPA, we construct training sets by sampling molecular pairs (X, Y) with significant property improvement and molecular similarity $sim(X, Y) \geq \delta$. The similarity constraint is also enforced at evaluation time to exclude arbitrary mappings that completely

Table 1: Translation performance on penalized logP task. GCPN results are from You et al. (2018a).

Method	$\delta = 0.6$		$\delta = 0.4$	
	Improvement	Diversity	Improvement	Diversity
MMPA	1.41 ± 0.49	**0.366**	2.00 ± 0.50	0.432
JT-VAE	0.28 ± 0.79	-	1.03 ± 1.39	-
GCPN	0.79 ± 0.63	-	2.49 ± 1.30	-
VSeq2Seq	0.80 ± 1.35	0.122	2.29 ± 1.99	0.294
VJTNN	1.48 ± 1.29	0.310	3.30 ± 1.80	0.497
VJTNN+GAN	$\mathbf{1.53 \pm 1.25}$	0.311	$\mathbf{3.37 \pm 1.81}$	**0.522**

ignore the input X. We measure the molecular similarity by computing Tanimoto similarity over Morgan fingerprints (Rogers & Hahn, 2010). Next we describe how these tasks are constructed.

- **Penalized logP** We first evaluate our methods on the constrained optimization task proposed by Jin et al. (2018). The goal is to improve the penalized logP score of molecules under the similarity constraint. Following their setup, we experiment with two similarity constraints ($\delta = 0.4$ and 0.6), and we extracted 120K and 53K translation pairs respectively from the ZINC dataset (Sterling & Irwin, 2015; Jin et al., 2018) for training. We use their validation and test sets for evaluation.
- **Drug likeness (QED)** Our second task is to improve drug likeness of compounds. Specifically, the model needs to translate molecules with QED scores (Bickerton et al., 2012) within the range $[0.7, 0.8]$ into the higher range $[0.9, 1.0]$. This task is challenging as the target range contains only the top 6.6% of molecules in the ZINC dataset. We extracted a training set of 88K molecule pairs with similarity constraint $\delta = 0.4$. The validation and test set contain 1000 molecules each.
- **Dopamine Receptor (DRD2)** The third task is to improve a molecule's biological activity against a biological target named the dopamine type 2 receptor (DRD2). We use a trained model from Olivecrona et al. (2017) to assess the probability that a compound is active. We ask the model to translate molecules with predicted probability $p < 0.05$ into active compounds with $p > 0.5$. The active compounds represent only 1.9% of the dataset. With similarity constraint $\delta = 0.4$, we derived a training set of 34K molecular pairs from ZINC and the dataset collected by Olivecrona et al. (2017). The validation and test set contain 1000 molecule each.

Baselines We compare our approaches (VJTNN and VJTNN+GAN) with the following baselines:

- **MMPA**: We utilized (Dalke et al., 2018)'s implementation to perform MMPA. Molecular transformation rules are extracted from the ZINC and Olivecrona et al. (2017)'s dataset for corresponding tasks. During testing, we translate a molecule multiple times using different matching transformation rules that have the highest average property improvements in the database (Appendix B).
- **Junction Tree VAE**: Jin et al. (2018) is a state-of-the-art generative model over molecules that applies gradient ascent over the learned latent space to generate molecules with improved properties. Our encoder-decoder architecture is closely related to their autoencoder model.
- **VSeq2Seq**: Our second baseline is a variational sequence-to-sequence translation model that uses SMILES strings to represent molecules and has been successfully applied to other molecule generation tasks (Gómez-Bombarelli et al., 2016). Specifically, we augment the architecture of Bahdanau et al. (2014) with stochastic latent codes learned in a similar way to our VAE model.
- **GCPN**: GCPN (You et al., 2018a) is a reinforcement learning based model that modifies a molecule by iteratively adding or deleting atoms and bonds. They also adopt adversarial training to enforce naturalness of the generated molecules.

Model Configuration For our models, we set the latent code dimension $|\boldsymbol{z}| = 8$, and KL regularization weight $\lambda_{\text{KL}} = 1.0$. We found that with a larger latent code dimension ($|\boldsymbol{z}| = 56$), the model degenerates to an autoencoder that reconstructs target molecule Y from the latent code $\boldsymbol{z}$ and completely ignores the source molecule X. In addition, our model performance is greatly degraded when $\lambda_{\text{KL}} < 1.0$. Due to limited space, we defer other hyper-parameter settings to the appendix.

5.1 Results

We quantitatively analyze the translation accuracy, diversity, and novelty of different methods.

Table 2: Translation performance on QED and DRD2 task. JT-VAE and GCPN results are computed by running Jin et al. (2018) and You et al. (2018a)'s open-source implementations.

Method	QED			DRD2		
	Success	Diversity	Novelty	Success	Diversity	Novelty
MMPA	20.8%	0.225	100%	35.6%	0.210	99.8%
JT-VAE	8.8%	-	-	3.4%	-	-
GCPN	9.4%	0.216	100%	4.4%	0.152	100%
VSeq2Seq	38.0%	0.199	78.8%	52.1%	0.027	20.1%
VJTNN	54.2%	0.387	99.3%	78.5%	0.206	79.0%
VJTNN+GAN	**56.9%**	0.377	99.7%	**81.0%**	0.208	79.1%

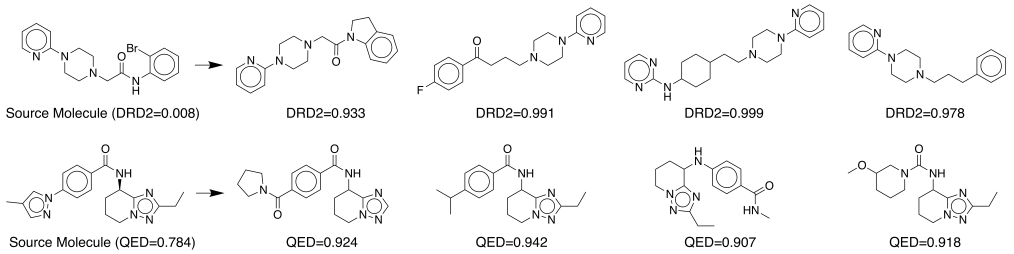

Figure 4: Examples of diverse translations learned by VJTNN+GAN on QED and DRD2 dataset.

Translation Accuracy We measure the translation accuracy as follows. On the penalized logP task, we follow the same evaluation protocol as GCPN. That is, for each source molecule, we decode K times with different latent codes $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and report the molecule having the highest property improvement under the similarity constraint. We set $K = 50$ so that it is comparable with baselines. On the QED and DRD2 datasets, we report the success rate of the learned translations. We define a translation as successful if one of the 50 translation candidates satisfies the similarity constraint and its property score falls in the target range (QED $\in [0.9, 1.0]$ and DRD2 > 0.5).

Tables 1 and 2 show the performance of all models across the three datasets. Our best model significantly outperforms all baselines, yielding 94% and 35% relative gain over GCPN on the logP dataset with $\delta = 0.6$ and 0.4, and more than 47% relative gain over VSeq2Seq across all the tasks. We also found that VJTNN+GAN has the most improvement on the QED and DRD2 tasks because their target domains are explicitly constrained by property ranges. Thus it is beneficial to constrain the model output distribution by adversarial training. We present more ablation studies in the appendix.

Diversity We define the diversity of a set of molecules as the average pairwise Tanimoto distance between them, where Tanimoto distance $dist(X, Y) = 1 - sim(X, Y)$. For each source molecule, we translate it 50 times (each with different latent codes), and compute the diversity over the set of validly translated molecules.[2] As we require valid translated molecules to be similar to a given compound, the diversity score is upper-bounded by the maximum allowed distance (e.g. for the QED and DRD2 datasets, the maximum diversity score is likely to be around 0.6). As shown in Tables 1 and 2, both of our methods have significantly higher diversity scores than VSeq2Seq across all datasets. On the logP dataset, our models achieve roughly 70% of the maximum allowed diversity. Figure 4 shows some examples of diverse translation over the QED and DRD2 tasks.

Novelty Lastly, we report how often our model discovers new molecules in the target domain that are unseen during training. This is an important metric as the ultimate goal of drug discovery is to design new molecules. Let $\mathcal{M}$ be the set of molecules generated by the model and $\mathcal{S}$ be the molecules given during training. We define novelty as $|\mathcal{M} - \mathcal{S}|/|\mathcal{M}|$. On the QED and DRD2 datasets, our models discover new compounds much more frequently than VSeq2Seq, again confirming the advantage of graph based methods. Both of our models perform similarly on this measure.

[2]To isolate the translation accuracy from the diversity measure, we exclude the failure cases from diversity calculation, namely excluding molecules that have no valid translation. Otherwise models with lower success rates will always have lower diversity.

6 CONCLUSION

In conclusion, we have evaluated various graph-to-graph translation models for molecule optimization. By combining the variational junction tree encoder-decoder with adversarial training, we can generate better and more diverse molecules than the baselines.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90, 2012.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

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

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

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

Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. *arXiv preprint arXiv:1802.08786*, 2018.

Andrew Dalke, Jerome Hert, and Christian Kramer. mmpdb: An open-source matched molecular pair platform for large multiproperty data sets. *Journal of chemical information and modeling*, 2018.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

Alexander G Dossetter, Edward J Griffen, and Andrew G Leach. Matched molecular pair analysis in drug discovery. *Drug Discovery Today*, 18(15-16):724–731, 2013.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 2016. doi: 10.1021/acscentsci.7b00572.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Ed Griffen, Andrew G Leach, Graeme R Robb, and Daniel J Warner. Matched molecular pairs as a medicinal chemistry tool: miniperspective. *Journal of medicinal chemistry*, 54(22):7739–7750, 2011.

Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pp. 5767–5777, 2017.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

Wengong Jin, Connor Coley, Regina Barzilay, and Tommi Jaakkola. Predicting organic reaction outcomes with weisfeiler-lehman network. In *Advances in Neural Information Processing Systems*, pp. 2604–2613, 2017.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8): 595–608, 2016.

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

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

Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017.

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

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. *arXiv preprint arXiv:1705.09037*, 2017.

Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *arXiv preprint arXiv:1801.07299*, 2018a.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018b.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L Gaunt. Constrained graph variational autoencoders for molecule design. *arXiv preprint arXiv:1805.09076*, 2018.

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

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48, 2017.

Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885, 2018.

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

Bidisha Samanta, Abir De, Niloy Ganguly, and Manuel Gomez-Rodriguez. Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*, 2018.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems*, pp. 992–1002, 2017.

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *arXiv preprint arXiv:1701.01329*, 2017.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems*, pp. 6830–6841, 2017.

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

David Weininger. 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.

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

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018a.

Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: A deep generative model for graphs. *arXiv preprint arXiv:1802.08773*, 2018b.

Junbo Jake Zhao, Yoon Kim, Kelly Zhang, Alexander M Rush, and Yann LeCun. Adversarially regularized autoencoders. *arXiv preprint arXiv:1706.04223*, 2018.

Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*, pp. 465–476, 2017.

A MODEL ARCHITECTURE

Tree and Graph Encoder For the graph encoder, functions $g_1(\cdot)$ and $g_2(\cdot)$ are parameterized as a one-layer neural network ($\tau(\cdot)$ represents the ReLU function):

$$\boldsymbol{\nu}_{uv}^{(t)} = \tau\left(\boldsymbol{W}_1^g \boldsymbol{f}_u + \boldsymbol{W}_2^g \boldsymbol{f}_{uv} + \sum\nolimits_{w\in N(u)\setminus v} \boldsymbol{W}_3^g \boldsymbol{\nu}_{wu}^{(t-1)}\right) \tag{15}$$

$$\boldsymbol{x}_u = \tau\left(\boldsymbol{U}_1^g \boldsymbol{f}_u + \sum\nolimits_{v\in N(u)} \boldsymbol{U}_2^g \boldsymbol{\nu}_{vu}^{(T)}\right) \tag{16}$$

For the tree encoder, since it updates the messages with more iterations, we parameterize function $g_1(\cdot)$ as a tree GRU function for learning stability (edge features $\boldsymbol{f}_{uv}$ are omitted because they are always zero). We keep the same parameterization for $g_2(\cdot)$, with a different set of parameters.

$$\boldsymbol{\nu}_{uv}^{(t)} = \mathrm{GRU}\left(\boldsymbol{f}_u, \{\boldsymbol{\nu}_{wu}^{(t-1)}\}_{w\in N(u)\setminus v}\right) \tag{17}$$

Tree Gated Recurrent Unit The tree GRU function $\mathrm{GRU}(\cdot)$ for computing message $\boldsymbol{h}_{ij}$ in Eq.(3) is defined as follows (Jin et al., 2018):

$$\boldsymbol{s}_{ij} = \sum\nolimits_{k\in N(i)\setminus j} \boldsymbol{h}_{ki} \tag{18}$$

$$\boldsymbol{z}_{ij} = \sigma(\boldsymbol{W}^z \boldsymbol{f}_i + \boldsymbol{U}^z \boldsymbol{s}_{ij} + \boldsymbol{b}^z) \tag{19}$$

$$\boldsymbol{r}_{ki} = \sigma(\boldsymbol{W}^r \boldsymbol{f}_i + \boldsymbol{U}^r \boldsymbol{h}_{ki} + \boldsymbol{b}^r) \tag{20}$$

$$\tilde{\boldsymbol{h}}_{ij} = \tanh\left(\boldsymbol{W}\boldsymbol{f}_i + \boldsymbol{U}\sum_{k\in N(i)\setminus j} \boldsymbol{r}_{ki} \odot \boldsymbol{h}_{ki} + \boldsymbol{b}\right) \tag{21}$$

$$\boldsymbol{h}_{ij} = (1 - \boldsymbol{z}_{ij}) \odot \boldsymbol{s}_{ij} + \boldsymbol{z}_{ij} \odot \tilde{\boldsymbol{h}}_{ij} \tag{22}$$

Tree Decoder Attention The attention mechanism is implemented as a bilinear function between decoder state $\boldsymbol{h}_t$ and source tree and graph vectors normalized by the softmax function:

$$\boldsymbol{\alpha}_{i,t}^{\mathcal{T}} = \frac{\exp(\boldsymbol{h}_t \boldsymbol{A}_{\mathcal{T}} \boldsymbol{x}_i^{\mathcal{T}})}{\sum_k \exp(\boldsymbol{h}_t \boldsymbol{A}_{\mathcal{T}} \boldsymbol{x}_k^{\mathcal{T}})} \qquad \boldsymbol{\alpha}_{i,t}^{\mathcal{G}} = \frac{\exp(\boldsymbol{h}_t \boldsymbol{A}_{\mathcal{T}} \boldsymbol{x}_i^{\mathcal{G}})}{\sum_k \exp(\boldsymbol{h}_t \boldsymbol{A}_{\mathcal{T}} \boldsymbol{x}_k^{\mathcal{G}})} \tag{23}$$

Graph Decoder We use the same graph neural architecture (Jin et al., 2018) for scoring candidate attachments. Let G_i be the graph resulting from a particular merging of cluster C_i in the tree with its neighbors C_j, $j \in N_{\widehat{\mathcal{T}}}(i)$, and let u, v denote atoms in the graph G_i. The main challenge of attachment scoring is *local isomorphism*: Suppose there are two neighbors C_j and C_k with the same cluster labels. Since they share the same cluster label, exchanging the position of C_j and C_k will lead to isomorphic graphs. However, these two cliques are actually not exchangeable if the subtree under j and k are different (Illustrations can be found in Jin et al. (2018)). Therefore, we need to incorporate information about those subtrees when scoring the attachments.

To this end, we define index $\alpha_v = i$ if $v \in C_i$ and $\alpha_v = j$ if $v \in C_j \setminus C_i$. The index α_v is used to mark the position of the atoms in the junction tree, and to retrieve messages $\boldsymbol{h}_{i,j}$ summarizing the subtree under i along the edge (i, j) obtained by running the tree encoding algorithm. The tree messages are augmented into the graph message passing network to avoid local isomorphism:

$$\boldsymbol{\mu}_{uv}^{(t)} = \tau(\boldsymbol{W}_1^a \boldsymbol{f}_u + \boldsymbol{W}_2^a \boldsymbol{f}_{uv} + \boldsymbol{W}_3^a \widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)}) \tag{24}$$

$$\widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)} = \begin{cases} \sum_{w\in N(u)\setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u = \alpha_v \\ \boldsymbol{h}_{\alpha_u,\alpha_v} + \sum_{w\in N(u)\setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u \neq \alpha_v \end{cases} \tag{25}$$

The final representation of graph G_i is $\boldsymbol{m}_{G_i} = \sum_v \boldsymbol{\mu}_v^{G_i}$, where

$$\boldsymbol{\mu}_u^{G_i} = \tau\left(\boldsymbol{U}_1^a \boldsymbol{f}_u + \sum\nolimits_{v\in N(u)} \boldsymbol{U}_2^a \boldsymbol{\mu}_{vu}^{(T)}\right) \tag{26}$$

Adversarial Scaffold Regularization Algorithm 2 describes the tree decoding algorithm for adversarial training. It replaces the ground truth input $\boldsymbol{f}_*$ with predicted label distributions $\boldsymbol{q}_*$, enabling gradient propagation from the discriminator.

Algorithm 2 Soft Tree Decoding for Adversarial Regularization

Require: Source tree and graph vectors $\{\boldsymbol{x}_*^{\mathcal{T}}\}, \{\boldsymbol{x}_*^{\mathcal{G}}\}$
Initialize: Tree $\widehat{\mathcal{T}} \leftarrow \emptyset$; Global counter $t \leftarrow 0$
function DecodeTree(i)
  repeat
    $t \leftarrow t + 1$
    Predict topology score $\boldsymbol{p}_t$ with Eq.(6), replacing $\boldsymbol{f}_i$ with predicted label distribution $\boldsymbol{q}_i$.
    if $\boldsymbol{p}_t \geq 0.5$ **then**
      Create a child j and add it to tree $\widehat{\mathcal{T}}$.
      Predict the node label distribution $\boldsymbol{q}_j$ with Eq.(9)
      Compute message $\boldsymbol{h}_{i,j}$ with Eq.(13)
      DecodeTree(j)
    end if
  until $\boldsymbol{p}_t < 0.5$
  Let j be the parent node of i. Compute message $\boldsymbol{h}_{i,j}$ with Eq.(13)
end function

B Experimental Details

Training Details We elaborate on the hyper-parameters used in our experiments. For our models, the hidden state dimension is 300 and latent code dimension $|\boldsymbol{z}| = 8$. The tree encoder runs message passing for 6 iterations, and graph encoder runs for 3 iterations. The entire model has 3.9M parameters. For VSeq2Seq, the encoder is a one-layer bidirectional LSTM and the decoder is a one-layer uni-directional LSTM. The attention scores are computed the same as Bahdanau et al. (2014). We set the hidden state dimension of the recurrent encoder and decoder to be 300 so that the VSeq2Seq model also has 3.8M parameters. For VSeq2Seq, we found that the model performance stays similar under different KL regularization weights ($\lambda_{\text{KL}} = 0.02$ and 1.0).

All models are trained with the Adam optimizer for 20 epochs with learning rate 0.001. We anneal the learning rate by 0.9 for every epoch. For adversarial training, our discriminator is a three-layer feed-forward network with hidden layer dimension 300. The activation function is LeakyReLU. The discriminator is trained for $N = 5$ iterations with gradient penalty weight $\beta = 10$.

Property Calculation The penalized logP is calculated using You et al. (2018a)'s implementation, which utilizes RDKit (Landrum, 2006) to compute clogP and synthetic accessibility scores. The QED scores are also computed using RDKit's built-in functionality. The DRD2 activity prediction model is downloaded from `https://github.com/MarcusOlivecrona/REINVENT/blob/master/data/clf.pkl`.

MMPA Procedure We utilized the open source toolkit `mmpdb` (Dalke et al., 2018) to perform matching molecular pair (MMP) analysis (`https://github.com/rdkit/mmpdb`). On the logP and QED tasks, we constructed a database of transformation rules extracted from the ZINC dataset (with test set molecules excluded). On the DRD2 task, the database is constructed from both ZINC and the dataset from Olivecrona et al. (2017). During testing, each molecule is translated 50 times with different matching rules. When there are more than 50 matching rules, we choose those with the highest average property improvement. This statistic is calculated during database construction.

Training Set Curation The training set of the penalized logP task is curated from the ZINC dataset of 250K molecules (Jin et al., 2018). A molecular pair (X, Y) is selected into the training set if the Tanimoto similarity $sim(X, Y) \geq \delta$ and the property improvement is greater than 0.5 (if $\delta = 0.6$) and 2.5 (if $\delta = 0.4$). On the QED and DRD2 tasks, we select training molecular pairs (X, Y) if $sim(X, Y) \geq 0.4$ and both X and Y fall into the source and target property range. Figure 5 shows the histogram of QED and DRD2 score distributions. On the QED dataset, the percentile of the source domain is 37.9-65.5%, and the target domain is 0-6.6%. On the DRD2 dataset, the percentile of target domain is 0-1.9%.

Ablation Study We investigate the translation accuracy of our models against the number of translation candidates K. In a real-world drug discovery setting, there is usually a budget on how many

drug candidates can be tested in the laboratory, as biological experiments are time-consuming in general. As shown in Figure 6, the model performance greatly increases when more translation candidates are included. We also notice that on the QED and DRD2 datasets, the performance difference between VJTNN and VJTNN+GAN is much larger when $K = 10$, compared to larger values of K. This shows that the model with adversarial training generates valid translations much more frequently. This is beneficial as it requires fewer experiments in the real scenario.

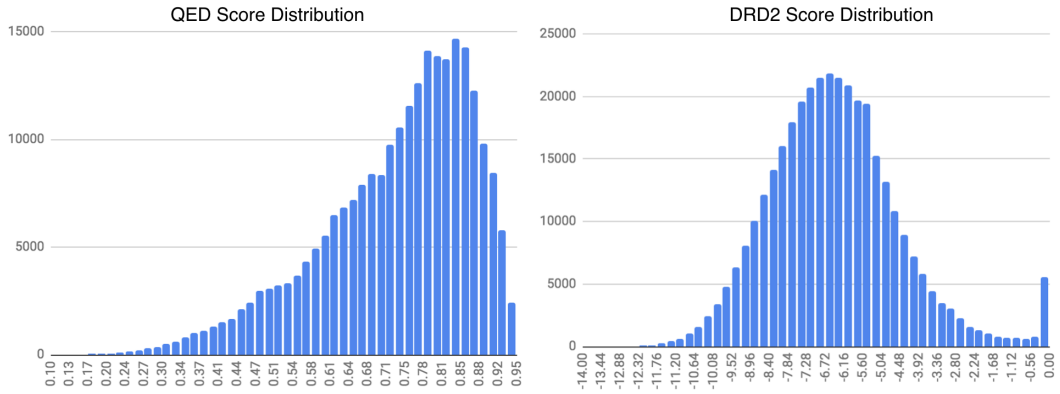

Figure 5: QED and DRD2 score histogram. The x-axis of DRD2 plot is shown in logarithmic scale.

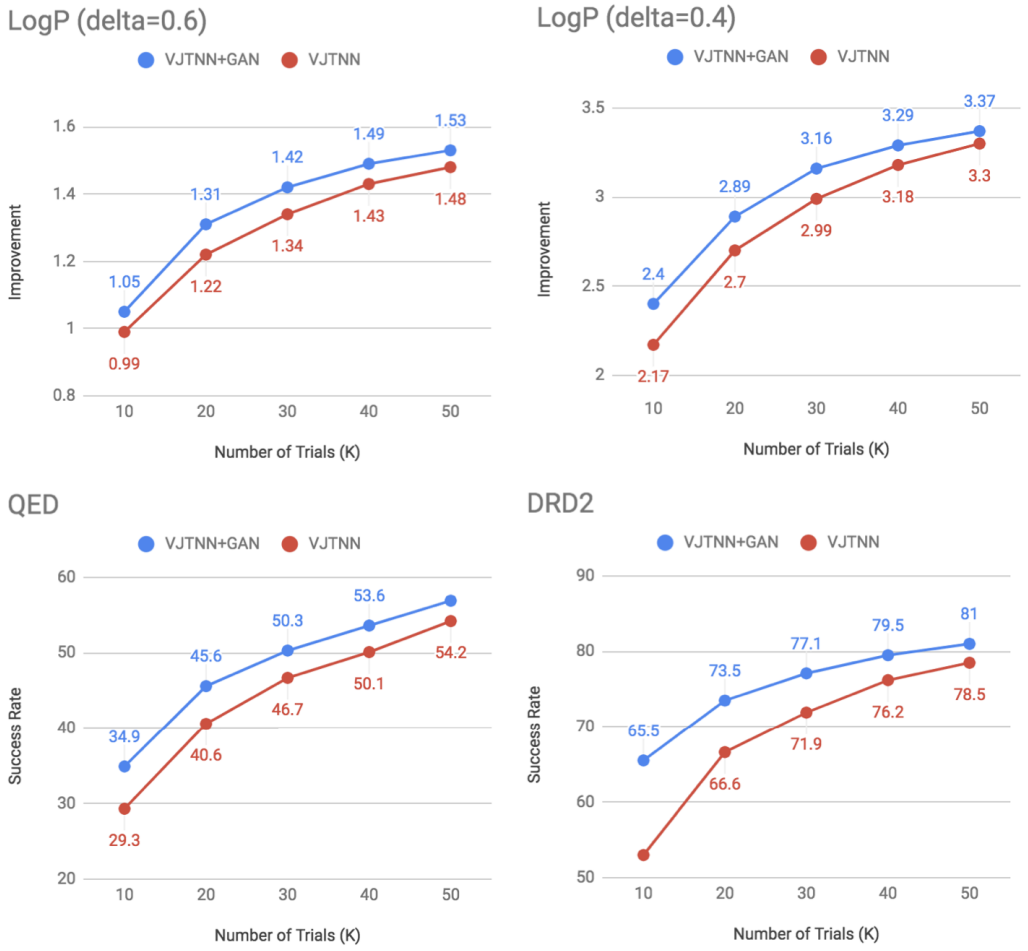

Figure 6: Ablation study on the translation accuracy against the number of translation attempts (K).