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 it only covers the most simple and common transformation patterns. In contrast, our approach uses neural networks to learn such translations, allowing us to learn far more complex transformations than hard-coded rules.

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 $G = (V, E)$ where $V$ is the vertex set and $E$ the edge set. Each node $v$ has a feature vector $f_v$. For atoms, it includes the atom type, valence, and other
atomic properties. For clusters in the junction tree, \( f_u \) is a one-hot vector indicating its cluster label. Similarly, each edge \( (u, v) \in \mathcal{E} \) has a feature vector \( f_{uv} \). Let \( N(v) \) be the set of neighbor nodes of \( v \). There are two hidden vectors \( \nu_{uv} \) and \( \nu_{vu} \) for each edge \( (u, v) \) representing the message from \( u \) to \( v \) and vice versa. These messages are updated iteratively via neural network \( g_1(\cdot) \):

\[
\nu_{uv}^{(t)} = g_1 \left( f_u, f_{uv}, \sum_{w \in N(u) \setminus v} \nu_{wv}^{(t-1)} \right)
\]

where \( \nu_{uv}^{(t)} \) is the message computed in the \( t \)-th iteration, initialized with \( \nu_{uv}^{(0)} = 0 \). 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 all messages are updated asynchronously, as there is no natural order among the nodes. This is 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:

\[
x_u = g_2 \left( f_u, \sum_{v \in N(u)} \nu_{vu}^{(T)} \right)
\]

Applying the above message passing network to junction tree \( T \) and to graph \( G \) yields two sets of vectors \( \{ x_1^T, \ldots, x_n^T \} \) and \( \{ x_1^G, \ldots, x_n^G \} \), which we call source tree vectors and graph vectors. \( x_i^T \) is the embedding of tree node \( i \), and \( x_j^G \) is the embedding of graph node \( j \).

### 3.2 Junction Tree Decoder

We generate a junction tree \( T = (\mathcal{V}, \mathcal{E}) \) 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 \( \mathcal{E} = \{(i_1, j_1), \ldots, (i_m, j_m)\} \) be the edges traversed in a depth first traversal over tree \( T \), where \( m = 2 |\mathcal{E}| \) as each edge is traversed in both directions. Let \( \mathcal{E}_t \) be the first \( t \) edges in \( \mathcal{E} \). At the \( t \)-th decoding step, the model visits node \( i_t \) and receives message vectors \( h_{ij} \) from its neighbors. The message \( h_{i_t,j_t} \) is updated through a tree Gated Recurrent Unit (Jin et al., 2018):

\[
h_{i_t,j_t} = \text{GRU}(f_{i_t}, \{ h_{k,i_t} \}_{k \neq i_t})
\]

**Topological Prediction** When the model visits node \( i_t \), it first computes a predictive hidden state \( h_t \) by combining node features \( f_{i_t} \) and inward messages \( \{ h_{k,i_t} \} \) via a 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 \( \{ x_i^T \} \) and \( \{ x_i^G \} \) through attention layer, followed by a feed-forward network \((\tau(\cdot)) \) stands for ReLU and \( \sigma(\cdot) \) for sigmoid):

\[
\begin{align*}
 h_t &= \tau(W^d f_{i_t} + W^d_2 \sum_{(k,i_t) \in \mathcal{E}_t} h_{k,i_t}) \\
 c^d_t &= \text{attention}(h_t, \{ x_i^T \}, \{ x_i^G \}; U_{att}^d) \\
 p_t &= \sigma(u^d \cdot \tau(W^d_3 h_t + W^d_4 c^d_t))
\end{align*}
\]

Here we use \( \text{attention}(\cdot; U_{att}^d) \) to mean the attention mechanism with parameters \( U_{att}^d \). It computes two set of attention scores \( \{ \alpha^T_d \}, \{ \alpha^G_d \} \) (normalized by softmax) over source tree and graph vectors.
respectively. The output $c^i_t$ is a concatenation of tree and graph attention vectors:

$$c^i_t = \sum_i \alpha^T_i x^T_i, \sum_i \alpha^G_i x^G_i$$  \hspace{1cm} (7)

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

$$c^i_t = \text{attention}(h_{i_t,j_t}, \{x^T_i\}, \{x^G_i\}, U^l_{att})$$  \hspace{1cm} (8)

$$q_t = \text{softmax}(U^T \tau(W^i_{h_{i_t,j_t}} + W^j_{c^i_t}))$$  \hspace{1cm} (9)

where $q_t$ is a distribution over the label vocabulary and $U^l_{att}$ is another set of attention parameters.

3.3 GRAPH DECODER

The second step in the decoding process is to construct a molecular graph $G$ from a predicted junction tree $\hat{T}$. This step is not deterministic since multiple molecules could correspond to the same junction tree. The underlying degree of freedom pertains to how neighboring clusters are attached to each other. Let $G_i$ be the set of possible candidate attachments at tree node $i$. Each graph $G_i \in G_i$ is a particular realization of how cluster $C_i$ is attached to its neighboring clusters $\{C_{j, j} \in N_\uparrow(i)\}$. 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 $G_i$. We first apply a graph message passing network over graph $G_i$ to compute atom representations $\{\mu^{G_i}\}$. Then we derive a vector representation of $G_i$ through sum-pooling: $m_{G_i} = \sum_u \mu^{G_i}$. Finally, we score candidate $G_i$ by computing dot products between $m_{G_i}$ and the encoded source graph vectors: $f(G_i) = \sum_u m_{G_i} \cdot x_u$.

During training, the graph decoder parameters are learned to maximize the log-likelihood of ground truth subgraphs at all tree nodes, as shown in Eq. (10). At testing time, we assemble the graph one neighborhood at a time, following the order in which the tree $\hat{T}$ was decoded.

$$\mathcal{L}_q(G) = \sum_i \left[ f(G_i) - \log \sum_{G_i' \in \mathcal{G}_i} \exp(f(G_i')) \right]$$  \hspace{1cm} (10)

4 MULTIMODAL GRAPH-TO-GRAHM 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 capable of generating a diverse set of outputs.

To this end, we propose to augment the basic encoder-decoder model with low-dimensional latent vectors $z$ to explicitly encode the multimodal aspect of the output distribution. The mapping to be learned now becomes $F : (X, z) \rightarrow Y$, with latent code $z$ drawn from a prior distribution $P(z)$, which is a standard Gaussian $\mathcal{N}(0, I)$. 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 $F(X, z)$ should always belong to the target domain $\mathcal{Y}$ given latent code $z \sim \mathcal{N}(0, I)$. In this section, we propose two techniques to address these issues.

4.1 VARIATIONAL JUNCTION TREE ENCODER-DECODER (VJTN}

First, to encode meaningful variations, we derive latent code $z$ from the embedding of ground truth molecule $Y$. The decoder is trained to reconstruct $Y$ when taking as input both its vector encoding $z_Y$ and source molecule $X$. For efficient sampling, the latent code distribution is regularized to be
Figure 2: Multimodal graph-to-graph learning. Our model combines the strength of both variational JTNN and adversarial scaffold regularization.

Specifically, we first embed molecules $X$ and $Y$ into their tree and graph vectors $\{x_T^X\}, \{x_G^X\}; \{y_T^Y\}, \{y_G^Y\}$, using the same encoder with shared parameters (Sec 3.1). Then we perform average pooling over the tree and graph vectors of $Y$: $y_T^Y = \sum_i y_T^i / |T_Y|$ and $y_G^Y = \sum_i y_G^i / |G_Y|$. We sample $z_T^Y$ and $z_G^Y$ from the posterior $Q(\cdot|Y)$ via the reparameterization trick (Kingma & Welling, 2013), where the mean and log variance are computed from $y_T$ and $y_G$ with two separate affine layers. Finally, we combine the latent code $z_T^Y$ and $z_G^Y$ with source tree and graph vectors:

$$\tilde{x}_i^T = \tau(W_e^1 x_i^T + W_e^2 z_T^Y); \quad \tilde{x}_i^G = \tau(W_e^3 x_i^G + W_e^4 z_G^Y);$$

(11)

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

$$L^{VAE}(X,Y) = -E_{z \sim Q}[\log P(Y|x, X)] + \lambda_{KL} D_{KL}[Q(z|Y)||P(z)]$$

(12)

### 4.2 Adversarial Scaffold Regularization

Second, to avoid invalid translations, we force molecules decoded from latent codes $z \sim \mathcal{N}(0, I)$ to follow the distribution of the target domain through adversarial training (Goodfellow et al., 2014). The adversarial objective involves two components. The discriminator tries to discriminate real molecules in the target domain from fake molecules generated by the model. The generator, namely 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. 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 VJTN, 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.

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

The decoder first predicts the label distribution \( q_{root} \) of the root of tree \( \hat{T} \). Starting from the root, we incrementally expand the tree, guided by topological predictions, and compute the hidden messages \( \{h_{i,t,J_t}\} \) 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 \( d(i_t, j_t) = 1_{p_t > 0.5} \), which is determined by the topological score \( p_t \) 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 \( \hat{Y} \) from source molecules \( X \), we replace the one-hot encoding \( f_{i_t} \) with its softmax distribution \( q_{i_t} \) over cluster labels in Eq. \( 3 \) and \( 4 \). Moreover, we multiply message \( h_{i_t, j_t} \) 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:

\[
    h_{i_t, j_t} = \begin{cases} 
    d(i_t, j_t) \cdot \text{GRU}(q_{i_t}, \{h_{k, i_t}\}_{(k, i_t) \in \mathcal{E}_i, k \neq j_t}) & \text{if } j_t \text{ is a child of node } i_t \\
    (1 - d(i_t, j_t)) \cdot \text{GRU}(q_{i_t}, \{h_{k, i_t}\}_{(k, i_t) \in \mathcal{E}_i, k \neq j_t}) & \text{vice versa}
    \end{cases}
\]

As \( d(i_t, j_t) \) is computed by a non-differentiable threshold function, we approximate its gradient with a straight-through estimator \( \text{Bengio et al., 2013} \). 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 \( \text{Chung et al., 2016} \).

Finally, after the tree \( T \) is completely decoded, we derive its continuous representation \( h_T \) by concatenating the root label distribution \( q_{root} \) and the sum of its inward messages:

\[
    s_{root} = \sum_{k \in N(root)} h_{k, root} \quad h_T = [q_{root}, s_{root}]
\]

We implement the discriminator \( D(\cdot) \) as a multi-layer feedforward network, and train the adversary using Wasserstein GAN with gradient penalty \( \text{Arjovsky et al., 2017} \). 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
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 (VJTN and VJTN+GAN) with the following baselines:

- **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 (Gomez-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**: On the logP task, we additionally compare with GCPN (You et al., 2018a), a reinforcement learning based model that modifies a molecule by iteratively adding or deleting atoms and bonds. They pretrain their model with expert trajectories and fine-tune the model with logP rewards. They also adopt adversarial training to enforce naturalness of the generated molecules.

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

**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 $z \sim \mathcal{N}(0, I)$, and report the molecule having the highest property

<table>
<thead>
<tr>
<th>Method</th>
<th>$\delta = 0.6$</th>
<th>Diversity</th>
<th>Improvement</th>
<th>Diversity</th>
</tr>
</thead>
<tbody>
<tr>
<td>JT-VAE</td>
<td>0.28 ± 0.79</td>
<td>-</td>
<td>1.03 ± 1.39</td>
<td>-</td>
</tr>
<tr>
<td>GCPN</td>
<td>0.79 ± 0.63</td>
<td>-</td>
<td>2.49 ± 1.30</td>
<td>-</td>
</tr>
<tr>
<td>VSeq2Seq</td>
<td>0.80 ± 1.35</td>
<td>0.122</td>
<td>2.29 ± 1.99</td>
<td>0.294</td>
</tr>
<tr>
<td>VJTN</td>
<td>1.48 ± 1.29</td>
<td>0.310</td>
<td>3.30 ± 1.80</td>
<td>0.497</td>
</tr>
<tr>
<td>VJTN+GAN</td>
<td>1.53 ± 1.25</td>
<td>0.311</td>
<td>3.37 ± 1.81</td>
<td>0.522</td>
</tr>
</tbody>
</table>

Table 1: Molecule translation results on penalized logP task.
Table 2: Molecule translation results on QED and DRD2 task.

<table>
<thead>
<tr>
<th>Method</th>
<th>QED</th>
<th></th>
<th></th>
<th>DRD2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Success</td>
<td>Diversity</td>
<td>Novelty</td>
<td>Success</td>
<td>Diversity</td>
<td>Novelty</td>
</tr>
<tr>
<td>JT-VAE</td>
<td>8.8%</td>
<td>-</td>
<td>-</td>
<td>3.4%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VSeq2Seq</td>
<td>38.0%</td>
<td>0.199</td>
<td>78.8%</td>
<td>52.1%</td>
<td>0.027</td>
<td>20.1%</td>
</tr>
<tr>
<td>VJTN</td>
<td>54.2%</td>
<td>0.387</td>
<td>99.3%</td>
<td>78.5%</td>
<td>0.206</td>
<td>79.0%</td>
</tr>
<tr>
<td>VJTN+GAN</td>
<td>56.9%</td>
<td>0.377</td>
<td>99.7%</td>
<td>81.0%</td>
<td>0.208</td>
<td>79.1%</td>
</tr>
</tbody>
</table>

Figure 3: Examples of diverse translations learned by VJTN+GAN on QED and DRD2 dataset.

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 GCNP 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 VJTN+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 $\text{dist}(X, Y) = 1 - \text{sim}(X, Y)$. 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. 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 3 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.

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.

---

2To 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.
REFERENCES


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):

$$\nu^{(t)}_{wu} = \tau \left( W^2_u f_u + W^2_{uv} f_{uv} + \sum_{w \in N(u) \setminus v} W^2_w \nu^{(t-1)}_{wu} \right)$$

$$x_u = \tau \left( U^1_u f_u + \sum_{v \in N(u)} U^2_{uv} \nu^{(T)}_{vu} \right)$$

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 $f_{uv}$ are omitted because they are always zero). We keep the same parameterization for $g_2(\cdot)$, with a different set of parameters.

$$\nu^{(t)}_{wu} = \text{GRU} \left( f_u, \{ \nu^{(t-1)}_{wu} \}_{w \in N(u) \setminus v} \right)$$

Tree Gated Recurrent Unit The tree GRU function $\text{GRU}(\cdot)$ for computing message $h_{ij}$ in Eq. (3) is defined as follows [Jin et al. 2018]:

$$s_{ij} = \sum_{k \in N(i) \setminus j} h_{ki}$$

$$z_{ij} = \sigma(W^2_{ij} f_i + U^1_s s_{ij} + b^s)$$

$$r_{ki} = \sigma(W^2_{ki} f_i + U^1_r h_{ki} + b^r)$$

$$\tilde{h}_{ij} = \tanh \left( W f_i + U \sum_{k \in N(i) \setminus j} r_{ki} \odot h_{ki} + b \right)$$

$$h_{ij} = (1 - z_{ij}) \odot s_{ij} + z_{ij} \odot \tilde{h}_{ij}$$

Tree Decoder Attention The attention mechanism is implemented as a bilinear function between decoder state $h_t$ and source tree and graph vectors normalized by the softmax function:

$$\alpha^T_{i,t} = \frac{\exp(h_t A^T x^T_i)}{\sum_k \exp(h_t A^T x^T_k)}$$

$$\alpha^G_{i,t} = \frac{\exp(h_t A^T x^G_i)}{\sum_k \exp(h_t A^T x^G_k)}$$

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 $C_j$, $j \in N_T(i)$, 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 $\alpha_v$ is used to mark the position of the atoms in the junction tree, and to retrieve messages $h_{i,j}$ 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:

$$\mu^{(t)}_{uv} = \tau \left( W^2_u f_u + W^2_{uv} f_{uv} + W^2_{v} \tilde{h}^{(t-1)}_{uv} \right)$$

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

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

$$\mu^{G_i}_v = \tau \left( U^1_u f_u + \sum_{v \in N(u)} U^2_{uv} \mu^{(T)}_{vu} \right)$$

Adversarial Scaffold Regularization Algorithm 2 describes the tree decoding algorithm for adversarial training. It replaces the ground truth input $f_i$ with predicted label distributions $q_v$, enabling gradient propagation from the discriminator.
Algorithm 2 Soft Tree Decoding for Adversarial Regularization

Require: Source tree and graph vectors \( \{x^T_*\}, \{x^G_*\} \)

1: Initialize: Tree \( \hat{T} \leftarrow \emptyset \); Global counter \( t \leftarrow 0 \)
2: function DecodeTree(i)
3: repeat
4: \( t \leftarrow t + 1 \)
5: Predict topology score \( p_t \) with Eq.(6), replacing \( f_i \) with predicted label distribution \( q_i \).
6: if \( p_t \geq 0.5 \) then
7: Create a child \( j \) and add it to tree \( \hat{T} \).
8: Predict the node label distribution \( q_j \) with Eq.(9)
9: Compute message \( h_{i,j} \) with Eq.(13)
10: DecodeTree(j)
11: end if
12: until \( p_t < 0.5 \)
13: Let \( j \) be the parent node of \( i \). Compute message \( h_{i,j} \) with Eq.(13)
14: end function

B  EXPERIMENTAL DETAILS

Training Details  We elaborate on the hyper-parameter settings used for our experiments. For our models, the hidden state dimension is 300 and latent code dimension \( |z| = 8 \). 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 unidirectional 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 (\( \lambda_{KL} = 0.02 \) and \( 1.0 \)).

All models are trained with the Adam optimizer for 20 epochs with default learning rate. 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 \( \beta = 10 \).

Dataset Statistics  The training set of the penalized logP task is curated from the ZINC250K dataset [Jin et al., 2018; You et al., 2018a]. When constructing molecular pairs, we only extract pairs with logP score improvement greater than 0.5 and 2.5 for \( \delta = 0.6 \) and 0.4 respectively.

The training set of the QED and DRD2 tasks are extracted by finding matched molecular pairs between source and target domains. Figure 1 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%.

Property Calculation  For penalized logP calculation, we directly used You et al. (2018a)'s implementation. The QED scores are computed using RDKit [Landrum, 2006]. The DRD2 activity prediction model is downloaded from [https://github.com/MarcusOlivecrona/REINVENT/blob/master/data/clf.pkl](https://github.com/MarcusOlivecrona/REINVENT/blob/master/data/clf.pkl).

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 2, 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 4: QED and DRD2 score histogram. The x-axis of DRD2 plot is shown in logarithmic scale.

Figure 5: Ablation study on the translation accuracy against the number of translation candidates $K = 10, 20, \cdots, 50$. 