Modular Flows: Differential Molecular Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Generating new molecules is fundamental to advancing critical applications such as drug discovery and material synthesis. Flows can generate molecules effectively by inverting the encoding process, however, existing flow models either require artifactual dequantization or specific node/edge orderings, lack desiderata such as permutation invariance, or induce discrepancy between encoding and decoding steps that necessitates *post hoc* validity correction. We circumvent these issues with novel continuous normalizing E(3)-equivariant flows, based on a system of node ODEs coupled as a graph PDE, that repeatedly reconcile locally toward globally aligned densities. Our models can be cast as message passing temporal networks, and result in superlative performance on the tasks of density estimation and molecular generation. In particular, our generated samples achieve state of the art on both the standard QM9 and ZINC250K benchmarks.

1 Introduction

Generative models have rapidly become ubiquitous in machine learning with advances from image synthesis (Ramesh et al., 2022) to protein design (Ingraham et al., 2019). Molecular generation (Stokes et al., 2020) has also received significant attention in discovering new drugs and materials. However, searching for valid molecules in large discrete spaces is challenging: drug-like structures range between 10^{23} and 10^{60} but only a tiny fraction ($\sim 10^8$) - has been synthesized (Polishchuk et al., 2013; Merz et al., 2020). Thus, learning representations that exploit appropriate molecular inductive biases (e.g., spatial correlations) becomes crucial.

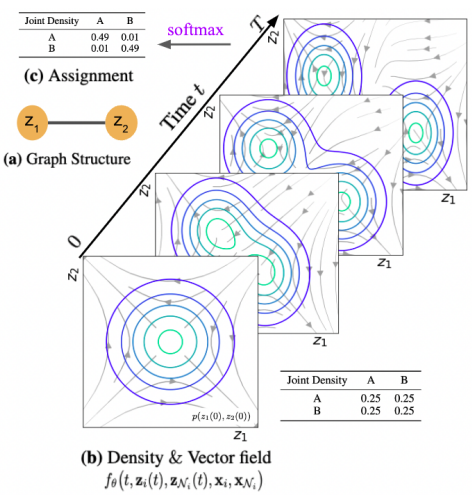


Figure 1: A toy illustration of `ModFlow` in action with a two-node graph. The two local flows - $\mathbf{z}_1$ and $\mathbf{z}_2$ - co-evolve toward a more complex joint density, both driven by the same differential f.

Earlier models focused on generating sequences based on the SMILES notation (Weininger, 1988), which were supplanted by generative models that capture valuable spatial information via embedding molecular with some graph neural network (GNNs) (Scarselli et al., 2009; Garg et al., 2020). This include variants of GANs (Goodfellow et al., 2014; Maziarka et al., 2020) which suffer the problem of mode collapse, VAEs (Kingma and Welling, 2013; Lim et al., 2018; Jin et al., 2018) which are susceptible to a distributional shift between the training data and the generated samples, and Normalizing Flows (Dinh et al., 2014, 2016). Flows are appealing since they enable estimating (and sampling from) complex data distributions using a sequence of invertible transformations from a tractable continuous distribution. Molecules

Submitted to the DLDE Workshop in the 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

are discrete, so many flow models (Madhawa et al., 2019; Honda et al., 2019; Shi et al., 2020) add noise during encoding leading to *dequantization* procedure which begets distortion and issues related to convergence (Luo et al., 2021). Moreover, many methods employ *post hoc* correction to ensure validity (Zang and Wang, 2020), effecting a discrepancy between the encoding and the decoded distributions.

We propose a coupled continuous normalizing E(3)-equivariant flows tailored to molecule generation, that bestow generative capabilities from neural partial differential equation (PDE) models on graphs (Chamberlain et al., 2021; Poli et al., 2019; Iakovlev et al., 2020). We seek to bring their efficacy and elegance as tools to generate complex objects, such as molecules. Specifically, a flow is associated with each node, which are conjoined as a joint ODE system conditioned on neighboring nodes. While these flows originate independently, they adjust progressively toward more complex joint distributions via interacting with the neighboring flows. We call the proposed method Modular Flows (`ModFlow`) to underscore that each node can be regarded as a module that coordinates with other modules. Experimental results show that `ModFlow`outperforms previous state-of-the-art methods over various molecule generation tasks.

2 Modular Flows

We focus on unsupervised learning of an underlying graph density $p(G)$ using a dataset $\mathcal{D} = \{G_n\}_{n=1}^N$ of observed molecular graphs G_n. We learn a generative flow model $p_\theta(G)$ specified by flow parameters θ and use it to sample novel high-probability molecules.

2.1 Molecular Representation

Graph representation. We represent the molecular graph $G = (V, E)$ as a tuple of vertices $V = (v_1, \ldots, v_M)$ and edges $E \subset V \times V$. Each vertex takes value as: $v \in \mathcal{A} = \{\mathtt{C}, \mathtt{H}, \mathtt{N}, \ldots\}$; while the edges $e \in \mathcal{B} = \{1, 2, 3\}$ represent abstract the type of bond. We assume that conditioned on the edges, the graph likelihood factorizes as a Categorical distribution over vertices given their latent representations:

$$p(G) := p(V|E, \{z\}) = \prod_{i=1}^{M} \mathrm{Cat}(v_i|\sigma(\mathbf{z}_i)), \tag{1}$$

where $\mathbf{z}_i = (z_{i\mathtt{C}}, z_{i\mathtt{H}}, \ldots) \in \mathbb{R}^{|\mathcal{A}|}$ is a set of atom score parameters of node i, and σ is the softmax function. We can obtain an alternative tree representation of molecules, similar to Jin et al. (2018) where we restrict clusters to be ring sub-structures. We obtain an extended alphabet $\mathcal{A}_{\mathrm{tree}} = \{\mathtt{C}, \mathtt{H}, \mathtt{N}, \ldots, \mathtt{C}_1, \mathtt{C}_2, \ldots\}$, where each cluster label $\mathtt{C}_r$ corresponds to the some ring-substructure in the label vocabulary χ.

2.2 Differential modular flows

We propose to model the atom scores $\mathbf{z}_i(t)$ as a Continuous-time Normalizing Flow (CNF) (Grathwohl et al., 2018) over time $t \in \mathbb{R}_+$. We assume the initial scores at time $t = 0$ as $\mathbf{z}_i(0) \sim \mathcal{N}(0, I)$ for each node i. Node scores evolve in parallel over time by a differential equation, where $\mathcal{N}_i$ is the set of neighbors, $\mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}$ is the positional information, and θ are the parameters of the flow function. By collecting all node differentials we obtain a *modular* joint, coupled ODE, which is our key contribution:,

$$\dot{\mathbf{z}}(t) = \begin{pmatrix} \dot{\mathbf{z}}_1(t) \\ \vdots \\ \dot{\mathbf{z}}_M(t) \end{pmatrix} = \begin{pmatrix} f_\theta\big(t, \mathbf{z}_1(t), \mathbf{z}_{\mathcal{N}_1}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big) \\ \vdots \\ f_\theta\big(t, \mathbf{z}_M(t), \mathbf{z}_{\mathcal{N}_M}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big) \end{pmatrix} \tag{2}$$

$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T \dot{\mathbf{z}}(t)dt. \tag{3}$$

The above system is usually solved via an ODE solver where gradients are computed via adjoint sensitivity method (Kolmogorov et al., 1962). This approach incurs a low memory cost, and explicitly controls numerical error. Notably, moving towards modular flows translates sparsity also to the adjoints.

Preposition 1: *Modular adjoints are sparser than regular adjoints. They can be computed by*

$$\frac{d\boldsymbol{\lambda}_i}{dt} = -\sum_{j\in\mathcal{N}_i\cup\{i\}} \boldsymbol{\lambda}_j(t)^\top \frac{\partial f\big(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big)}{\partial \mathbf{z}_j}, \tag{4}$$

where the partial derivatives $\frac{\partial f}{\partial \mathbf{z}} = [\frac{\partial f_i}{\partial \mathbf{z}_j}]_{ij}$ *are sparse* (See Appendix A.5 for derivation).

Equivariant local differential Our goal is to have a differential function f which satisfies the natural equivariances and invariances of molecules like translation, rotational (and reflection) equivariant and permutation equivariances. Therefore, we chose to use E(3)-Equivariant GNN (EGNN) (Satorras et al., 2021), which satisfies all the above criteria (See Appendix A.2 for details). The input to the EGNN are the node embeddings and the geometric information (polar coordinates (2D) and spherical polar coordinates (3D)). Interestingly, `ModFlow` can be viewed as a message passing temporal graph network (Rossi et al., 2020) as shown next.

Preposition 2: *Modular Flows can be casted as Temporal Graph Networks (TGN). The operations are listed in Table 3* (See Appendix A.3 for more details).

2.3 Training objective

Normalizing flows are predominantly trained to minimize KL divergence $\mathrm{KL}[p_{\mathrm{data}}||p_\theta]$ which is equivalent to maximizing their cross-entropy $\mathbb{E}_{p_{\mathrm{data}}}[\log p_\theta]$ (Papamakarios et al., 2021). However, this requires a mapping from discrete graphs G to continuous atom scores $\mathbf{z}(t)$. We reduce the learning problem to maximizing the score cross-entropy $\mathbb{E}_{\hat{p}_{\mathrm{data}}(\mathbf{z}(T))}[\log p_\theta(\mathbf{z}(T))]$, where we turn the observed set of graphs $\{G_n\}$ into a set of scores $\{\mathbf{z}_n\}$ by using one-hot encoding

$$\mathbf{z}_n(G_n;\epsilon) = (1-\epsilon)\,\mathrm{onehot}(G_n) + \frac{\epsilon}{|\mathcal{A}_f|}\mathbf{1}_{M(n)}\mathbf{1}_{|\mathcal{A}_f|}^\top,$$

where $\mathrm{onehot}(G_n)$ is a matrix of size $M(n) \times |\mathcal{A}_f|$ such that $G_n(i,k) = 1$ if $v_i = a_k \in \mathcal{A}_f$, $\mathbf{1}_q$ is a vector with q entries each set to 1; $\mathcal{A}_f \in \{\mathcal{A}, \mathcal{A}_{\mathrm{tree}}\}$; and $\epsilon \in [0,1]$ is added to model the noise in estimating the posterior $p(\mathbf{z}(T)|G)$ due to short-circuiting the inference process as shown in Fig.2. We exploit the (non-reversible) composition of the argmax and softmax operations to short-circuit the process, which keeps the forward and backward flows aligned. We thus maximize an objective over N training graphs,

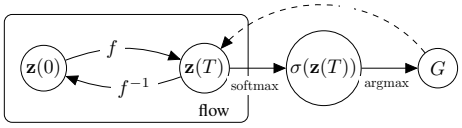


Figure 2: Plate diagram showing both the inference and generative components of `ModFlow`.

$$\arg\max_\theta \quad \mathcal{L} = \mathbb{E}_{\hat{p}_{\mathrm{data}}(\mathbf{z})} \log p_\theta(\mathbf{z}) \tag{5}$$

$$\approx \frac{1}{N}\sum_{n=1}^{N} \log p_T\big(\mathbf{z}(T) = \mathbf{z}_n\big) \tag{6}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{i=1}^{M(n)}\left[\log p_0(\mathbf{z}_i(0)) - \int_0^T \mathrm{tr}\frac{\partial f_\theta(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i})}{\partial \mathbf{z}_i(t)} dt\right]\right), \tag{7}$$

which factorizes over the size $M(n)$ of the n'th training molecule. In practice, we solve ODE integrals using numerical solvers, such as Runge-Kutta. We delegate this task to a general solver of the form $\texttt{ODESolve}(\mathbf{z}, f_\theta, T)$, where map f_θ is applied for T steps starting on $\mathbf{z}$. An optimizer `optim` is also required for updating θ.

2.4 Molecular generation

We generate novel molecules by sampling an initial state $\mathbf{z}(0) \sim \mathcal{N}(0, I)$ based on structure, and running the modular flow forward in time until $\mathbf{z}(T)$. This procedure maps a tractable base distribution p_0 to some more complex distribution p_T. We follow argmax to pick the most probable label assignment for each node (Zang and Wang, 2020).

3 Experiments

We evaluated `ModFlow` models trained, variously, on 2D coordinates, 3D coordinates, and their tree representation respectively on the task of molecular generation and compared it with many existing SOTA methods. Notably, `ModFlow` achieves state-of-the-art results without validity checks or post hoc correction.

We train and evaluate the models on ZINC250k (Irwin et al., 2012) and QM9 (Ramakrishnan et al., 2014) datasets. The molecules are in kekulized form with hydrogens removed by the RDkit software (Landrum et al., 2013). For evaluation, we used **Validity**, **Uniqueness**, **Novelty** and **Reconstruction**. These metrics measure the molecules obeying valency rules, non-duplicate generations, not present in the dataset, and the fraction of molecules that can be reconstructed from their encoding. We report the mean and the standard deviation from 5 different seed initialization, and generate 50,000 molecules for evaluation. All the implementation is done in PyTorch (Paszke et al., 2019). The input concatenates time and scalar vocabulary scores, per node. For more details, see Appendix A.4.

Tables 1 and 2 show the results on QM9 and ZINC250K. `ModFlow` achieves state-of-the-art results across all metrics. Notably, its reconstruction rate is 100% similar to other flow models; novelty and uniqueness scores are also very high. Moreover, `ModFlow` surpassed all early methods on validity (95%-99%). Additional results on property optimization and density estimation are shown in Appendix A.6 and A.7.

Table 1: Random generation performance on QM9 dataset. Results with * are taken from Luo et al. (2021). Higher values are better for all columns.

Method	Validity %	Uniqueness %	Novelty %	Reconstruction %
GVAE	60.2	9.3	80.9	96.0
GraphNVP*	83.1	99.2	58.2	100
GRF*	84.5	66	58.6	100
GraphAF*	67	94.2	88.8	100
GraphDF*	82.7	97.6	98.1	100
MoFlow*	89.0	98.5	96.4	100
`ModFlow` (2D-EGNN)	**96.2** $\pm$ 1.7	**99.5**	**100**	100
`ModFlow` (3D-EGNN)	**98.3** $\pm$ 0.7	99.1	**100**	100
`ModFlow` (JT-2D-EGNN)	**97.9** $\pm$ 1.2	99.2	**100**	100
`ModFlow` (JT-3D-EGNN)	**99.1** $\pm$ 0.8	99.3	**100**	100

Table 2: Random generation performance on ZINC250K dataset. Results with * are taken from Luo et al. (2021). Higher values are better for all columns.

Method	Validity %	Uniqueness %	Novelty %	Reconstruction %
MRNN	65	99.89	100	n/a
GraphNVP*	42.6	94.8	100	100
GRF*	73.4	53.7	100	100
GraphAF*	68	99.1	100	100
GraphDF*	89	99.2	100	100
MoFlow*	50.3	**99.9**	100	100
`ModFlow` (2D-EGNN)	**94.8** $\pm$ 1.0	99.4	100	100
`ModFlow` (3D-EGNN)	**95.4** $\pm$ 1.2	99.7	100	100
`ModFlow` (JT-2D-EGNN)	**97.4** $\pm$ 1.4	99.1	100	100
`ModFlow` (JT-3D-EGNN)	**98.1** $\pm$ 0.9	99.3	100	100

4 Conclusion

We proposed `ModFlow`, a new generative flow model where multiple flows interact locally according to a coupled ODE, resulting in accurate modeling of graph densities and high-quality molecular generation without any validity checks or correction. Interesting avenues open up, including the design of (a) more nuanced mappings between discrete and continuous spaces, and (b) extensions of modular flows to (semi-)supervised settings.

References

Andrew M. Bradley. PDE-constrained optimization and the adjoint method. Technical report, 2019.

Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pages 1407–1418. PMLR, 2021.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3419–3430. PMLR, 2020. URL `http://proceedings.mlr.press/v119/garg20c.html`.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL `https://arxiv.org/abs/1406.2661`.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Shion Honda, Hirotaka Akita, Katsuhiko Ishiguro, Toshiki Nakanishi, and Kenta Oono. Graph residual flow for molecular graph generation. *arXiv preprint arXiv:1909.13521*, 2019.

Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. *arXiv preprint arXiv:2006.08956*, 2020.

John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative models for graph-based protein design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/f3a4ff4839c56a5f460c88cce3666a2b-Paper.pdf`.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL `https://arxiv.org/abs/1312.6114`.

Andrei Nikolaevich Kolmogorov, Ye F Mishchenko, and Lev Semenovich Pontryagin. A probability problem of optimal control. Technical report, JOINT PUBLICATIONS RESEARCH SERVICE ARLINGTON VA, 1962.

Greg Landrum et al. Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling, 2013.

Jaechang Lim, Seongok Ryu, Jin Woo Kim, and Woo Youn Kim. Molecular generative model based on conditional variational autoencoder for de novo molecular design. *Journal of cheminformatics*, 10(1):1–9, 2018.

Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation, 2021. URL `https://arxiv.org/abs/2102.01189`.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

Lukasz Maziarka, Agnieszka Pocha, Jan Kaczmarczyk, Krzysztof Rataj, Tomasz Danel, and Michał Warchoł. Mol-cyclegan: a generative model for molecular optimization. *Journal of Cheminformatics*, 12(1):1–18, 2020.

Kenneth M. Merz, Gianni De Fabritiis, and Guo-Wei Wei. Generative models for molecular design. *Journal of Chemical Information and Modeling*, 60(12):5635–5636, 2020. doi: 10.1021/acs.jcim.0c01388. URL https://doi.org/10.1021/acs.jcim.0c01388. PMID: 33378853.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57): 1–64, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

Pavel G Polishchuk, Timur I Madzhidov, and Alexandre Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of computer-aided molecular design*, 27(8):675–679, 2013.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. URL https://arxiv.org/abs/2204.06125.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020. URL https://arxiv.org/abs/2006.10637.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks, 2021. URL https://arxiv.org/abs/2102.09844.

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. doi: 10.1109/TNN.2008.2005605.

Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M. Donghia, Craig R. MacNair, Shawn French, Lindsey A. Carfrae, Zohar Bloom-Ackermann, Victoria M. Tran, Anush Chiappino-Pepe, Ahmed H. Badran, Ian W. Andrews, Emma J. Chory, George M. Church, Eric D. Brown, Tommi S. Jaakkola, Regina Barzilay, and James J. Collins. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.e13, 2020. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2020.01.021. URL https://www.sciencedirect.com/science/article/pii/S0092867420301021.

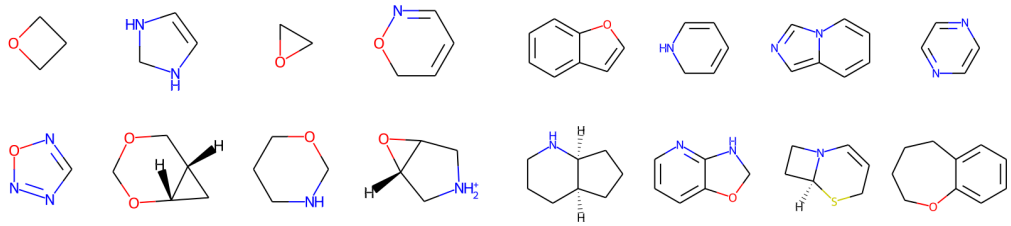
David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Chengxi Zang and Fei Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 617–626, 2020.

A Appendix

A.1 Tree Decomposition

The tree decomposition of the molecules follow closely to Jin et al. (2018) described algorithm. The rings in the molecule are extracted by RDKit's `GetRingInfo, GetSymmSSSR` function, where it provides the nodes corresponding to the ring sub-structure including the merged rings also. We constrain our clusters (vocabulary) to be unique ring-substructures present in the molecules. It is observed vocabulary of clusters, follows a skewed distribution over frequency of appearance within dataset. It means that only a subset (~ 30) of ring-substructures (labels) have high frequency of appearance within any molecule as compared to other substructures within the vocabulary. We simplify the vocabulary by only representing the 30 commonly occurring substructures of $\mathcal{A}_{tree}$. In Figure 3 , we have shown examples of cluster vocabulary of ring-substructure for different datasets.



(a) QM9 Dataset (b) ZINC250K Dataset

Figure 3: Examples of cluster vocabulary

A.2 Equivariant Graph Neural Networks (EGNN)

Equivariant Graph Neural Networks (Satorras et al., 2021) are a class of GNN which are translation, rotation and reflection equivariant (E(n)), and permutation equivariant with respect to an input set of points. These symmetries corresponds to E(3) group which can be extended to E(n) group. For input the set of node embeddings $\mathbf{h}^l = \{\mathbf{h}_0^l, \mathbf{h}_1^l, ..., \mathbf{h}_{M-1}^l\}$ and coordinate embeddings $\mathbf{x}^l = \{\mathbf{x}_0^l, \mathbf{x}_1^l, ..., \mathbf{x}_{M-1}^l\}$ and edge information E = (a_{ij}) and outputs a transformation on $\mathbf{h}^{l+1}$ and $\mathbf{x}^{l+1}$ as:

$$\mathbf{m}_{ij} = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, a_{ij}\right) \tag{8}$$

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \tag{9}$$

$$\mathbf{m}_i = \sum_{j\neq i}\mathbf{m}_{ij} \tag{10}$$

$$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right) \tag{11}$$

Here, the edge operation ϕ_e also take input the relative squared distance between two coordinates $\left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2$, with node embeddings $\mathbf{h}_i^l, \mathbf{h}_j^l$ and edge attribute a_{ij}. Subsequently, position of each node $\mathbf{x}_i^l$ is updated via a vector field in a radial direction. This means that, the position is updated by the weighted sum of all relative differences $(\mathbf{x}_i - \mathbf{x}_j)_{\forall j}$ by function ϕ_x. C is chosen to be $1/(M-1)$, which divides the sum by its number of elements. The Eq. 18 is the aggregation step, which can be limited to the neighbourhood $j \in \mathcal{N}(i)$ and Eq. 19 performs node operation ϕ_h. In case, when dealing with static coordinates $\mathbf{x}_i$ this becomes E(n) invariant.

A.3 Connection to Temporal Graph Networks

Temporal Graph Networks (Rossi et al., 2020) belong to class of methods applied on dynamic graphs. Specifically, it consist of combination of memory modules and graph-based operators acting as a neural model for dynamic graphs. First, for an event a message $\mathbf{m}_i'(t)$ is computed to update i's memory using previous time-step memory state ($\mathbf{s}_i\left(t^-\right)$). The msg is a learnable function like MLP, which aggregates the messages $\mathbf{m}_i'(t_1), \mathbf{m}_i'(t_2), ..., \mathbf{m}_i'(t_b)$ for $t_1, t_2.....t_b \leq t$. The aggregated message $\overline{\mathbf{m}}_i'(t)$ is then used to update the memory state of the node. Various choices of network architecture like RNNs, MLP etc can be incorporated as agg or mem. Finally, the node embeddings are updated as a summation over the neighbouring nodes $n_i^k([0,t])$, where h is a learnable function. The operations are listed in Table 3 respectively.

EGNN inherently does not have any time-dynamics, but when combined with modular flows it can be treated as a temporal EGNN, which can be casted as a variant of TGN. In Table 3, we have summarized the dynamics of both the architectures, and devised a mapping from one to another. From the above, we can see similarity in term of operations between two networks.

Table 3: Comparison and mapping of TGN to our method. We use the shorthand $\mathbf{r}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)$, a_{ij} are the edge attributes, $\mathbf{s}_i$ is memory state of node (TGN) and $\mathbf{v}_i, \mathbf{e}_{ij}$ are node and edge features.

Method	TGN layer	`ModFlow`
Edge	$\mathbf{m}_i'(t) = \mathrm{msg}\left(\mathbf{s}_i\left(t^-\right), t, \mathbf{v}_i(t)\right)$ $\overline{\mathbf{m}}_i'(t) = \mathrm{agg}\left(\mathbf{m}_i'\left(t_1\right), \ldots, \mathbf{m}_i'\left(t_b\right)\right)$	$\mathbf{m}_{ij}(t) = \phi_e\left(\mathbf{z}_i(t), \mathbf{z}_j(t), \left\|\mathbf{r}_{ij}(t)\right\|^2, a_{ij}\right)$ $\hat{\mathbf{m}}_{ij}(t) = \mathbf{r}_{ij}(t) \cdot \phi_x\left(\mathbf{m}_{ij}(t)\right)$ $\mathbf{m}_i(t) = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_i(t) = C \cdot \sum_{j \in \mathcal{N}(i)} \hat{\mathbf{m}}_{ij}(t)$
Memory state	$\mathbf{s}_i(t) = \mathrm{mem}\left(\overline{\mathbf{m}}_i'(t), \mathbf{s}_i\left(t^-\right)\right)$	$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \hat{\mathbf{m}}_i(t)$
Node	$\mathbf{z}_i'(t) = \sum_{j \in n_i^k([0,t])} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)\right)$	$\mathbf{z}_i(t+1) = \phi_h\left(\mathbf{z}_i(t), \mathbf{m}_i(t)\right)$

The role of the memory state $\mathbf{s}_i$ in TGN can be played by coordinate embeddings $\mathbf{x}_i$ in our method and messages can be formulated with respect to both in the networks. It is worthy to note that TGN also incorporates neighbouring node attributes, memory states to update the node embedding $\mathbf{z}_i'$ which is the similar to ours where the aggregated message over neighbouring nodes is used to update the node embedding $\mathbf{z}_i$, where (ϕ_h) is a learnable function. From the above similarities, we can see that the our method can be casted as a variant of TGN, which is also equivariant as compared to TGN.

A.4 Implementation Details

The model is implemented in PyTorch (Paszke et al., 2019). The EGNN implementation consist of 1 layer, and the embedding dimension is 32 and the neighbourhood is restricted to 1-hop neighbours of each node. The input are the node embeddings and the coordinate positions in polar coordinates (2D) and spherical polar coordinates (3D). The input dimension is the size of the vocabulary being considered with time concatenated to the input embedding. We solve the ODE system with Dormand–Prince adaptive step size scheme `dopri5` adaptive ODE solver, and the number of function evaluations lie roughly between $70 \sim 100$. Adam (Kingma and Ba, 2014) optimizer is used to train the model. Model is trained for 50 - 100 epochs. The training time of our model slightly above some one-shot discrete flows model where a single flow is characterizing whole system as compared to ours where each node has its own ODE. However, the training time is less than as compared to the Auto-regressive methods

Computational time is a crucial aspect for generative modeling involved in huge search spaces like molecules. We have compared the computational time (excluding preprocessing time) for generation of a molecule for ZINC250k and QM9 dataset reported in Table 4. The time is initially computed via generating 10000 molecules, then averaged across 5 runs. Notably, `ModFlow` is a one-hot generation model which is faster in order of magnitude from the auto-regressive models like GraphAF and GraphDF. However, the computational time depends heavily on the choices of frameworks and skills of implementations. `ModFlow` poses faster sampling time similar to any one-shot sampling technique and does not involve any additional validity checks and does not require multiple decoding like in JT-VAE.

Table 4: Generation time of various models performed on QM9 and ZINC250K dataset. Time is in seconds denoted as $\mathcal{O}$(second/molecule)

Method	ZINC250K	QM9
GraphEBM	1.12 ± 0.34	0.53 ± 0.16
GVAE	0.86 ± 0.12	0.46 ± 0.07
GraphAF	0.93 ± 0.14	0.56 ± 0.12
GraphDF	3.12 ± 0.56	1.92 ± 0.42
MoFlow	0.71 ± 0.14	0.31 ± 0.04
`ModFlow` (2D-EGNN)	0.46 ± 0.09	0.16 ± 0.04
`ModFlow` (3D-EGNN)	0.55 ± 0.13	0.24 ± 0.06
`ModFlow` (JT-2D-EGNN)	0.53 ± 0.07	0.21 ± 0.07
`ModFlow` (JT-3D-EGNN)	0.62 ± 0.11	0.28 ± 0.09

A.5 Derivation of modular adjoint

We present a standard adjoint gradient derivation (Bradley, 2019), and show that the adjoint of a graph neighborhood differential is sparse.

For completeness, we define an ODE system

$$\dot{\mathbf{z}}(t) = f(\mathbf{z}, t, \boldsymbol{\theta}) \tag{12}$$

$$\mathbf{z}(t) = \mathbf{z}_0 + \int_0^t f(\mathbf{z}, t, \boldsymbol{\theta}) d\tau, \tag{13}$$

where $\mathbf{z} \in \mathbb{R}^D$ is a state vector, $\dot{\mathbf{z}} \in \mathbb{R}^D$ is the state time differential defined by the vector field function f and parameterised by $\boldsymbol{\theta}$. The starting state is $\mathbf{z}_0$, and $t, \tau \in \mathbb{R}_+$ are time variables. Our goal is to solve a constrained problem

$$\min_{\boldsymbol{\theta}} \quad G(\boldsymbol{\theta}) = \int_0^T g(\mathbf{z}, t, \boldsymbol{\theta}) dt \tag{14}$$

$$s.t. \quad \dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta}) = 0, \qquad \forall t \in [0, T] \tag{15}$$

$$\mathbf{z}(0) - \mathbf{z}_0 = 0, \tag{16}$$

where G is the total loss that consists of instant loss functionals g. We desire to compute the gradients of the system $\nabla_{\boldsymbol{\theta}} G$.

We optimise the constrained problem by solving the Lagrangian

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = G(\boldsymbol{\theta}) + \int_0^T \boldsymbol{\lambda}(t)^\top (\dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta})) dt + \boldsymbol{\mu}^\top (\mathbf{z}(0) - \mathbf{z}_0) \tag{17}$$

$$= \int_0^T \Big[g(\mathbf{z}, t, \boldsymbol{\theta}) + \boldsymbol{\lambda}(t)^\top (\dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta})) \Big] dt + \boldsymbol{\mu}^\top (\mathbf{z}(0) - \mathbf{z}_0) \ . \tag{18}$$

The constraints are satisfied by the ODE definition. Hence, $\nabla_{\boldsymbol{\theta}} \mathcal{L} = \nabla_{\boldsymbol{\theta}} G$, and we can set values $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ freely. We use a shorthand notation $\frac{\partial a}{\partial b} = a_b$, and omit parameters from the functions for notational simplicity. Applying the chain rule, we note that the gradient becomes

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \nabla_{\boldsymbol{\theta}} G = \int_0^T \Big[g_{\mathbf{z}} \mathbf{z}_{\boldsymbol{\theta}} + g_{\boldsymbol{\theta}} + \boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} \mathbf{z}_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}} \Big] dt \ , \tag{19}$$

where the $\boldsymbol{\mu}$ term drops out since it does not depend on parameters $\boldsymbol{\theta}$. We apply integration by parts to swap the differentials in term $\boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}}$, resulting in

$$\int_0^T \boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}} dt = \boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=T} - \boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=0} - \int_0^T \dot{\boldsymbol{\lambda}}^\top \mathbf{z}_{\boldsymbol{\theta}} dt \ . \tag{20}$$

Substituting this into previous equation and rearranging the terms results in

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \int_0^T \underbrace{(g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} - \dot{\boldsymbol{\lambda}}^\top) \mathbf{z}_{\boldsymbol{\theta}}}_{0, \text{ if } \dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}}} dt + \int_0^T (g_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}}) dt + \underbrace{\boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=T}}_{0, \text{ if } \boldsymbol{\lambda}(T) = \mathbf{0}} - \underbrace{\boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=0}}_{0} . \tag{21}$$

The last term is removed since $\mathbf{z}(0)$ not depend on $\boldsymbol{\theta}$ as a constant, and thus $\mathbf{z}_{\boldsymbol{\theta}}(0) = 0$. The difficult term in the equation is $\mathbf{z}_{\boldsymbol{\theta}}$. We remove it by choosing

$$\dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}}. \tag{22}$$

Finally, we choose $\boldsymbol{\lambda}(T) = 0$ which also removes the second-to-last term. The choices lead to a final term

$$\nabla_{\boldsymbol{\theta}} G = \nabla_{\boldsymbol{\theta}} \mathcal{L} = \int_0^T (g_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}}) dt \tag{23}$$

$$s.t. \quad \dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} \tag{24}$$

$$\boldsymbol{\lambda}(T) = 0. \tag{25}$$

In the derivation the adjoint $\boldsymbol{\lambda}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{z}(t)} \in \mathbb{R}^D$ represents the change of loss with respect to instant states, and is another ODE system that runs backwards from $\boldsymbol{\lambda}(T) = 0$ until $\boldsymbol{\lambda}(0)$. The final gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ counts all adjoints within $[0, T]$ multiplied by the 'immediate' partial derivatives $f_{\boldsymbol{\theta}}$. The final gradient also takes into account the instant loss parameter derivatives. For simple MSE curve fitting, the instant loss has no parameters. The adjoint depends on the instant loss state derivatives $g_{\mathbf{z}}$. These are often only available for observations $\mathbf{y}_j$ at observed timepoints t_j. This can be represented by having a convenient loss

$$g(\mathbf{z}, t, \boldsymbol{\theta}) = \delta(t = t_j)\tilde{g}(\mathbf{z}, \mathbf{y}_j, t, \boldsymbol{\theta}), \tag{26}$$

and now the term $g_{\mathbf{z}}$ induces discontinuous jumps at observations. This does not pose problems in practice, since we can integrate the ODE in continuous segments between the observation instants.

The sparsity of the adjoint evolution is evident from the equation 24, where the $\dot{\boldsymbol{\lambda}}_i$ is an inner product between $\boldsymbol{\lambda}$ and one column of $\frac{\partial f}{\partial \mathbf{z}}$, which is invariant to non-neighbors. This gives a result

$$\frac{d\boldsymbol{\lambda}_i}{dt} = -\boldsymbol{\lambda}(t)^\top \frac{\partial f\big(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big)}{\partial \mathbf{z}} = -\sum_{j \in \mathcal{N}_i \cup \{i\}} \boldsymbol{\lambda}_j(t)^\top \frac{\partial f\big(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big)}{\partial \mathbf{z}_j}. \tag{27}$$

A.6 Density Estimation

We now demonstrate the high expressive power of Modular Flows (`ModFlow`) on learning highly discontinuous patterns on 2D grid graphs.

We considered two variants of chessboard pattern as (i) 4×4 grid where every node has opposite value to its neighbors (ii) 16×16 grid where blocks of 4×4 nodes have uniform values, but opposite across blocks, and (iii) a 20×20 grid describing alternating stripes. Figure 4 shows that we can learn a neural differential f_θ function that reproduces the patterns almost perfectly, indicating sufficient capacity to model complex patterns. Our model is able to transform the initial Gaussian into multi-modal and discontinuous distributions.

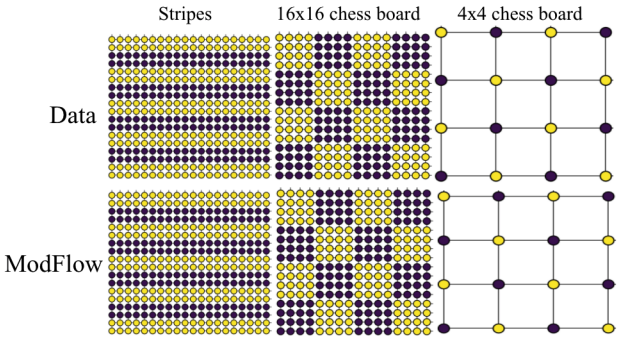
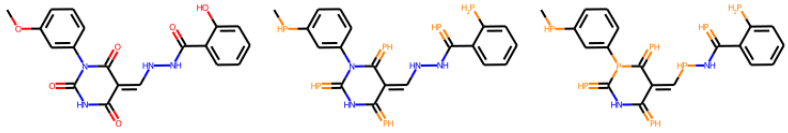


Figure 4: `ModFlow` can accurately learn to reproduce complex, discontinuous graph patterns.

A.7 Property-targeted Molecular Optimization

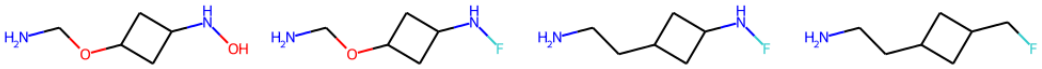
The task of molecular optimization is to search for the molecules, having a better chemical property. Specifically, we choose quantitative estimate of drug-likeness (QED) as our target chemical property. QED measures the potential of a molecule to be characterized as a drug. We used a pre-trained ModFlow model f, to encode a molecule $\mathcal{M}$ and get the embedding $Z = f(\mathcal{M})$, and further use linear regression to regress these embeddings to the QED scores of each molecule as our target chemical property. To discover new molecules, we interpolate in latent space of a molecule along the direction of increasing QED, via gradient ascend method, $Z' = Z + \lambda * \frac{dy}{dZ}$ where y is the QED score and λ is the length of the search step. The above method is conducting for K steps, and the new embedding Z' is decoded back to molecule space via reverse mapping $\mathcal{M}' = f^{-1}(\mathcal{Z}')$.

Figure 5 and 6 represents the molecules decoded from the learned latent space with linear regression for successful molecular optimization. Here, we select molecules with low QED score and interpolate its latent space neighbourhood along the direction of increasing QED and then decode back to molecule. However, the number of valid molecules decoded back varies on the query molecule. We also report the discovered novel molecules sorted by their QED scores in Table 5 . We can see that `ModFlow` is able to find novel molecules with high QED score.



QED: 0.332 QED: 0.428 QED: 0.557

Figure 5: Chemical property optimization for ZINC250K dataset. Given the left-most molecule, we interpolate in latent space along the direction which maximizes its QED property.



QED: 0.355 QED: 0.419 QED: 0.552 QED: 0.614

Figure 6: Chemical property optimization for QM9 dataset. Given the left-most molecule, we interpolate in latent space along the direction which maximizes its QED property.

Table 5: Property optimization performance evaluated by best QED scores. Baselines are taken from Luo et al. (2021).

Method	1st	2nd	3rd
ZINC (dataset)	0.948	0.948	0.948
JTVAE	0.925	0.911	0.910
GCPN	0.948	0.947	0.945
MRNN	0.844	0.799	0.736
GraphAF	0.948	0.948	0.947
GraphDF	0.948	0.948	0.948
MoFlow	0.948	0.948	0.948
`ModFlow` (2D-EGNN)	0.948	0.941	0.937
`ModFlow` (3D-EGNN)	0.948	0.937	0.931
`ModFlow` (JT-2D-EGNN)	0.947	0.941	0.939
`ModFlow` (JT-3D-EGNN)	0.948	0.948	0.945