Group Meeting - January 8, 2021

Paper review & Research progress

Truong Son Hy *

*Department of Computer Science
The University of Chicago

Ryerson Physical Lab

Papers

1. **Graph Normalizing Flows** (NeurIPS 2019), https://arxiv.org/abs/1905.13177
2. **Variational Inference with Normalizing Flows** (ICML 2015), https://arxiv.org/abs/1505.05770
3. **MolGAN: An implicit generative model for small molecular graphs**, https://arxiv.org/abs/1805.11973
4. **A Generative Model for Molecular Distance Geometry** (ICML 2020), https://arxiv.org/abs/1909.11459
5. **Constrained Graph Variational Autoencoders for Molecule Design** (NeurIPS 2018), https://proceedings.neurips.cc/paper/2018/hash/b8a03c5c15fcfa8dae0b03351eb1742f-Abstract.html
6. **Deep imitation learning for molecular inverse problems** (NeurIPS 2019), https://papers.nips.cc/paper/2019/hash/b0bef4c9a6e50d43880191492d4fc827-Abstract.html

Paper 1

Graph Normalizing Flows (NeurIPS 2019)
Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, Kevin Swersky
`https://arxiv.org/abs/1905.13177`

Normalizing Flows (1)

Normalizing flows (NFs) are a class of generative models that uses **invertible** mappings to transform an observed vector $\boldsymbol{x} \in \mathbb{R}^d$ to a latent vector $\boldsymbol{z} \in \mathbb{R}^d$ using a mapping function:

$$\boldsymbol{z} = f(\boldsymbol{x}), \qquad \boldsymbol{x} = f^{-1}(f(\boldsymbol{x}))$$

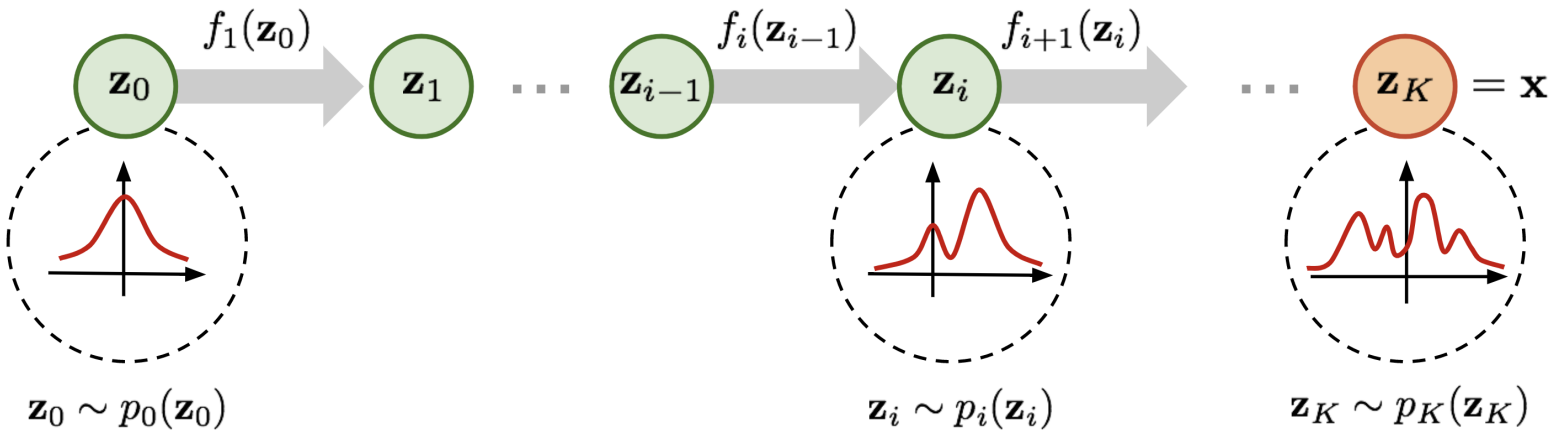
The change of variables:

$$P(\boldsymbol{z}) = P(\boldsymbol{x}) \left| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|^{-1}$$

For example, if we strech-out $\boldsymbol{x}$ 2 times: $\boldsymbol{z} = f(\boldsymbol{x}) = 2\boldsymbol{x}$. Then: $\partial f(\boldsymbol{x})/\partial \boldsymbol{x} = 2$, so the density is 2 times less dense: $P(\boldsymbol{z}) = P(\boldsymbol{x})/2$.

Normalizing Flows (2)

- With sufficiently expressive mapping, NFs can learn to map a complicated distribution into one that is well modeled as a Gaussian.
- The key is to find a mapping that is expressive, but with an efficiently computable determinant.



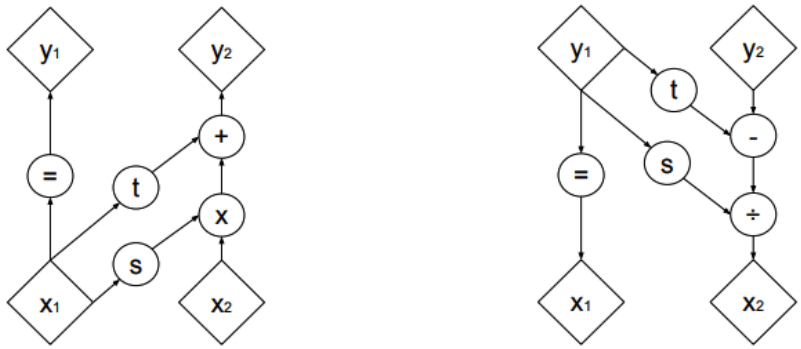
Normalizing Flows (3)

RealNVP partitions the dimensions of $\boldsymbol{x}$ into two sets of variables, $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$, and maps them into variables $\boldsymbol{z}^{(0)}$ and $\boldsymbol{z}^{(1)}$:

$$\boldsymbol{z}^{(0)} = \boldsymbol{x}^{(0)}$$

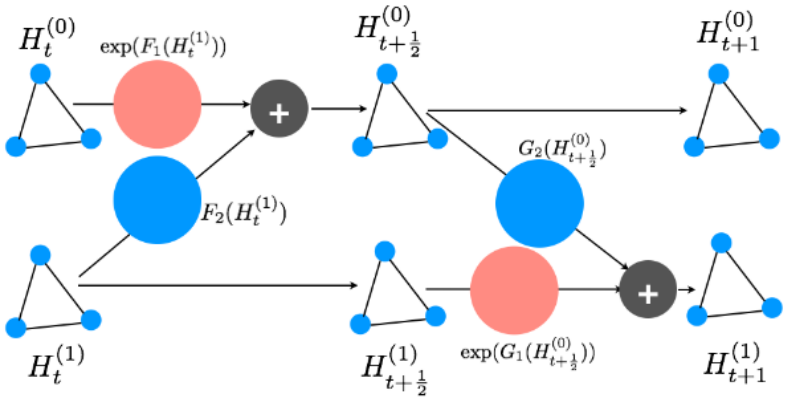
$$\boldsymbol{z}^{(1)} = \boldsymbol{x}^{(1)} \odot \exp(s(\boldsymbol{x}^{(0)})) + t(\boldsymbol{x}^{(0)})$$

where $\odot$ is the element-wise (Hadamard) product, s and t are **any** nonlinear functions.

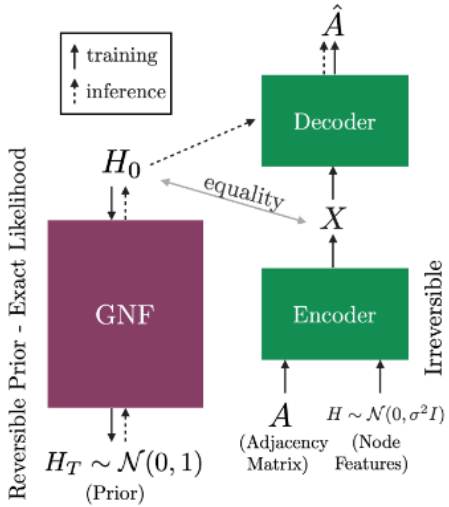


(a) Forward propagation (b) Inverse propagation

Overview



(a) Architecture of 1 step of message passing in a GRevNet: $H_t^{(0)}$, $H_t^{(1)}$ denote the two parts of the node-features of a particular node, $F_1(\cdot), F_2(\cdot)$ and $G_1(\cdot), G_2(\cdot)$ are 1-step MP transforms consisting of applying M_t and U_t once each. The scaling functions (F_2, G_2) are shown in red, whereas the translation functions (F_1, G_1) are shown in blue.



(b) A summary of our Graph generation pipeline using GNFs. The GNF model generates node features which are then fed into a decoder to generate the adjacency.

Figure 1: (a) GRevnet Message Passing, and (b) GNF generation pipeline

Note:

- When we partition each vertex features into half, we basically operate on 2 isomorphic graphs with different vertex features.
- F and G are any graph nets (that was mentioned in the talk and different from the paper).

Reversible Graph Neural Networks (GRevNets)

Forward:

$$
\begin{aligned}
H_{t+\frac{1}{2}}^{(0)} &= H_t^{(0)} \odot \exp\left(F_1\left(H_t^{(1)}\right)\right) + F_2(H_t^{(1)}) & H_{t+1}^{(0)} &= H_{t+\frac{1}{2}}^{(0)} \\
H_{t+\frac{1}{2}}^{(1)} &= H_t^{(1)} & H_{t+1}^{(1)} &= H_{t+\frac{1}{2}}^{(1)} \odot \exp\left(G_1\left(H_{t+\frac{1}{2}}^{(0)}\right)\right) + G_2\left(H_{t+\frac{1}{2}}^{(0)}\right)
\end{aligned} \tag{3}
$$

Inverse:

$$
\begin{aligned}
H_{t+\frac{1}{2}}^{(0)} &= H_{t+1}^{(0)} & H_t^{(1)} &= H_{t+\frac{1}{2}}^{(1)} \\
H_{t+\frac{1}{2}}^{(1)} &= \frac{\left(H_{t+1}^{(1)} - G_2\left(H_{t+\frac{1}{2}}^{(0)}\right)\right)}{\exp\left(G_1\left(H_{t+\frac{1}{2}}^{(0)}\right)\right)} & H_t^{(0)} &= \frac{\left(H_{t+\frac{1}{2}}^{(0)} - F_2(H_t^{(1)})\right)}{\exp\left(F_1\left(H_t^{(1)}\right)\right)}
\end{aligned} \tag{4}
$$

Note: I think the reason they have the middle step $t + 1/2$ is just to ensure all features are used. In the NFs, half the features are left unchanged.

GNFs for Structured Density Estimation

From the change of variables:

$$P(H_{t-1}) = P(H_t)\left|\frac{\partial H_{t-1}}{\partial H_t}\right|$$

From the chain rule:

$$P(\mathcal{G}) = P(H_T)\prod_{t=1}^{T}\left|\frac{\partial H_t}{\partial H_{t-1}}\right|$$

The Jacobians are given by lower triangular matrices $\rightarrow$ tractable. GNFs can model expressive distributions in continuous spaces over **sets** of vectors. We choose the prior:

$$P(H_T) = \prod_{i=1}^{N}\mathcal{N}(\boldsymbol{h}_i|0,1)$$

Experiments

Dataset/Task		GNN	GRevNet	Neumann RBP
Cora	Semi-Supervised	71.9	**74.5**	56.5
Cora	(1% Train)	55.5	**55.8**	54.6
Pubmed	Semi-Supervised	**76.3**	76.0	62.4
Pubmed	(1% Train)	76.6	**77.0**	58.5
PPI	Inductive	**0.78**	0.76	0.70

Model	mu	alpha	HOMO	LUMO	gap	R2
GNN	0.474	0.421	**0.097**	0.124	0.170	27.150
GrevNet	**0.462**	**0.414**	0.098	0.124	**0.169**	**26.380**

Model	ZPVE	U0	U	H	G	Cv
GNN	0.035	0.410	**0.396**	**0.381**	0.373	0.198
GrevNet	0.036	**0.390**	0.407	0.418	**0.359**	**0.195**

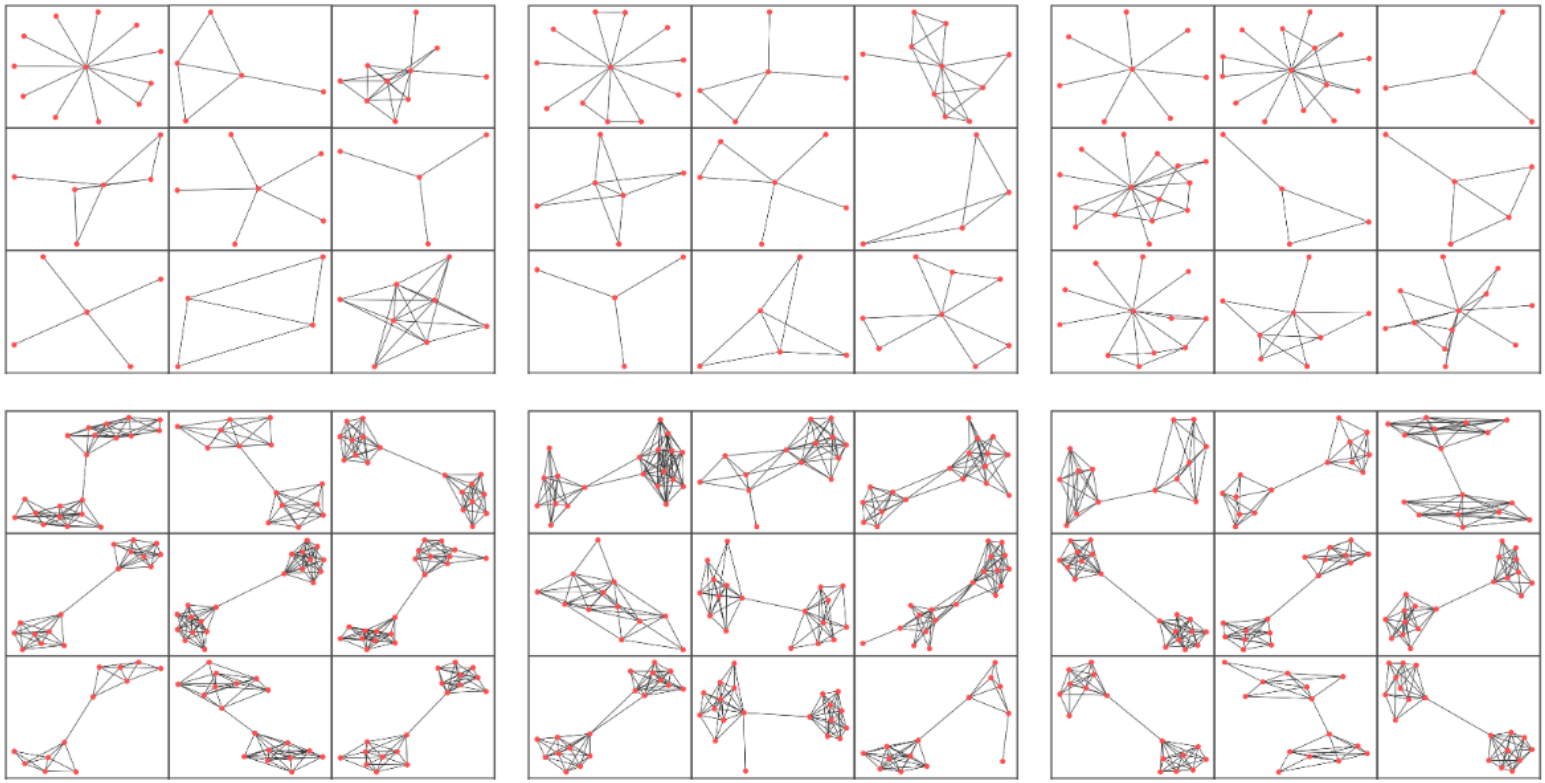
Table 1: Top: performance in terms of accuracy (Cora, Pubmed) and Micro F1 scores (PPI). For GNN and GrevNet, number of MP steps is fixed to 4. For Neumann RBP, we use 100 steps of MP. These values are averaged out over 3-5 runs with different seeds. Bottom: performance in terms of Mean Absolute Error (lower is better) for independent regression tasks on QM9 dataset. Number of MP steps is fixed to 4. The model was trained for 350k steps, as in [6].

Experiments

	COMMUNITY-SMALL			EGO-SMALL		
MODEL	DEGREE	CLUSTER	ORBIT	DEGREE	CLUSTER	ORBIT
GRAPHVAE	0.35	0.98	0.54	0.13	0.17	0.05
DEEPGMG	0.22	0.95	0.4	0.04	0.10	0.02
GRAPHRNN	0.08	0.12	0.04	0.09	0.22	0.003
GNF	0.20	0.20	0.11	0.03	0.10	0.001
GRAPHRNN(1024)	**0.03**	**0.01**	**0.01**	0.04	0.05	0.06
GNF(1024)	0.12	0.15	0.02	**0.01**	**0.03**	**0.0008**

Table 4: Graph generation results depicting MMD for various graph statistics between the test set and generated graphs. GRAPHVAE and DEEPGMG are reported directly from [30]. The second set of results (GRAPHRNN, GNF) are from evaluating the GraphRNN evaluation scheme with node distribution matching turned on. We trained 5 separate models of each type and performed 3 trials per model, then averaged the result over 15 runs. The third set of results (GRAPHRNN (1024), GNF (1024)) are obtained when evaluating on the test set over all 1024 generated graphs (no sub-sampling of the generated graphs based on node similarity). In this case, we trained and evaluated the result over 5 separate runs per model.

Experiments



(a) Training data (b) GNF samples (c) GRAPHRNN samples

Figure 3: Dataset examples and samples, drawn randomly, from the generative models. Top row: EGO-SMALL, bottom row: COMMUNITY-SMALL.

Paper 2

MolGAN: An implicit generative model for small molecular graphs

Nicola De Cao, Thomas Kipf

`https://arxiv.org/abs/1805.11973`

Overview (1)

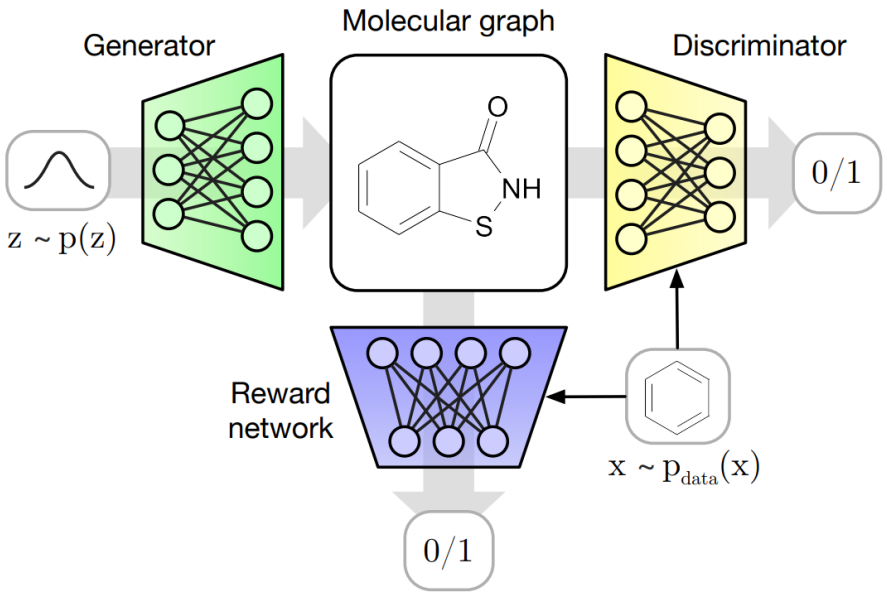


Figure 1. Schema of MolGAN. A vector $\boldsymbol{z}$ is sampled from a prior and passed to the generator which outputs the graph representation of a molecule. The discriminator classifies whether the molecular graph comes from the generator or the dataset. The reward network tries to estimate the reward for the chemical properties of a particular molecule provided by an external software.

Note:

- Improved WGAN: Wesserstein-1 distance (Kantorovich-Rubinstein duality).
- Gradient clipping.

Overview (2)

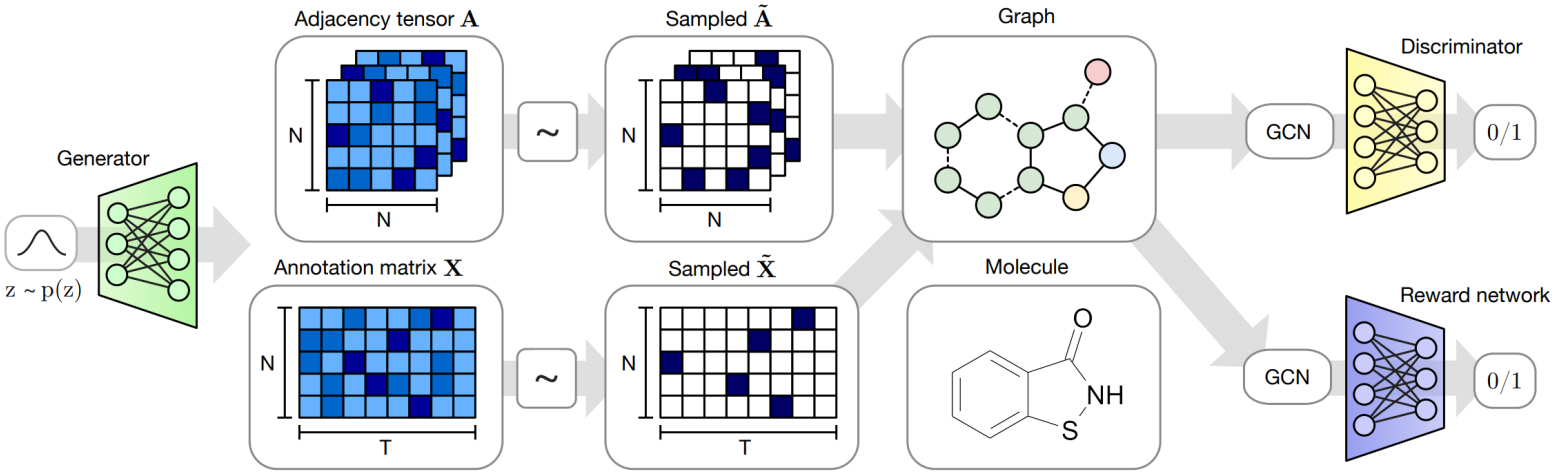


Figure 2. Outline of MolGAN. From *left*: the generator takes a sample from a prior distribution and generates a dense adjacency tensor $\boldsymbol{A}$ and an annotation matrix $\boldsymbol{X}$. Subsequently, sparse and discrete $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{X}}$ are obtained from $\boldsymbol{A}$ and $\boldsymbol{X}$ respectively via categorical sampling. The combination of $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{X}}$ represents an annotated molecular graph which corresponds to a specific chemical compound. Finally, the graph is processed by both the discriminator and reward networks that are invariant to node order permutations and based on Relational-GCN (Schlichtkrull et al., 2017) layers.

Discretization

This is the reason why I couldn't make the vanilla GAN converge. As this discretization process is non - differentiable, three options for gradient - based training:

1. Continuous $\boldsymbol{A}$ and $\boldsymbol{X}$.
2. Add Gumbel noise to them before passing to the Discriminator (still continuous).
3. Use a straight through gradient based on categorical re-parameterization with the Gumbel - Softmax.

Note: From the code `https://github.com/nicola-decao/MolGAN/blob/master/models/gan.py`, I think they actually used option 2.

Experiments (1)

Objective	Algorithm	Valid (%)	Unique (%)	Time (h)	Diversity	Druglikeliness	Synthesizability	Solubility
Druglikeliness	ORGAN	88.2	69.4*	9.63*	0.55	0.52	0.32	0.35
	OR(W)GAN	85.0	8.2*	10.06*	0.95	0.60	0.54	0.47
	Naive RL	97.1	54.0*	9.39*	0.80	0.57	0.53	0.50
	MolGAN	**99.9**	2.0	1.66	0.95	**0.61**	0.68	0.52
	MolGAN (QM9)	**100.0**	2.2	4.12	**0.97**	**0.62**	0.59	0.53
Synthesizability	ORGAN	96.5	45.9*	8.66*	0.92	0.51	0.83	0.45
	OR(W)GAN	97.6	30.7*	9.60*	**1.00**	0.20	0.75	0.84
	Naive RL	97.7	13.6*	10.60*	0.96	0.52	0.83	0.46
	MolGAN	**99.4**	2.1	1.04	0.75	0.52	**0.90**	0.67
	MolGAN (QM9)	**100.0**	2.1	2.49	0.95	0.53	**0.95**	0.68
Solubility	ORGAN	94.7	54.3*	8.65*	0.76	0.50	0.63	0.55
	OR(W)GAN	94.1	20.8*	9.21*	0.90	0.42	0.66	0.54
	Naive RL	92.7	100.0*	10.51*	0.75	0.49	0.70	0.78
	MolGAN	**99.8**	2.3	0.58	**0.97**	0.45	0.42	**0.86**
	MolGAN (QM9)	**99.8**	2.0	1.62	**0.99**	0.44	0.22	**0.89**
All/Alternated	ORGAN	96.1	97.2*	10.2*	0.92	**0.52**	0.71	0.53
All/Simultaneously	*MolGAN*	**97.4**	2.4	2.12	0.91	0.47	**0.84**	**0.65**
All/Simultaneously	*MolGAN (QM9)*	**98.0**	2.3	5.83	**0.93**	0.51	**0.82**	**0.69**

Table 2. Gray cells indicate directly optimized objectives. Baseline results are taken from Guimaraes et al. (2017) (Table 1) and * indicates results reproduced by us using the code provided by the authors.

Experiments (2)

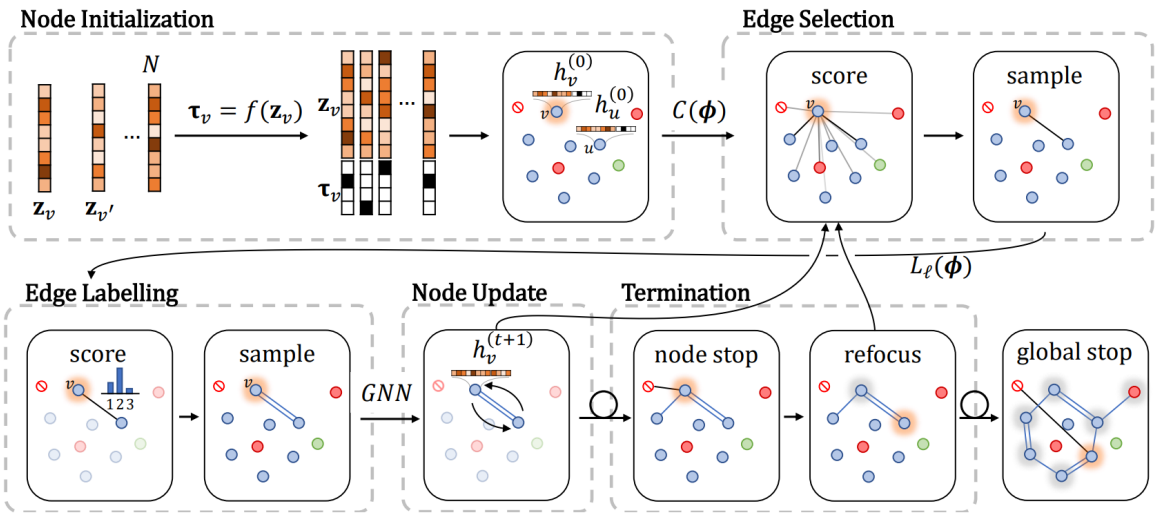
Algorithm	Valid	Unique	Novel
CharacterVAE	10.3	67.5	90.0
GrammarVAE	60.2	9.3	80.9
GraphVAE	55.7	**76.0**	61.6
GraphVAE/imp	56.2	42.0	75.8
GraphVAE NoGM	81.0	24.1	61.0
MolGAN	**98.1**	10.4	**94.2**

Table 3. Comparison with different algorithms on QM9. Values are reported in percentages. Baseline results are taken from Simonovsky & Komodakis (2018).

Note: I think this model suffers the **mode collapse** phenomenon of GAN, high validity and high novel but actually few molecules are generated, many are just redundant.

Research update (1)

Move to **autoregressive** approach (not RL) – Constrained Graph Variational Autoencoders for Molecule Design (NeurIPS 2018):



At first, we sample the each vertex latent z independently (so this is the first order), then iteratively we add new edge to the existing graph (given randomly selected node as the start). We apply second-order message passing (with gated recurrent architecture) to produce the probability of (u, v) where one vertex is in the existing graph and the another one is outside; and also the probability of its label.

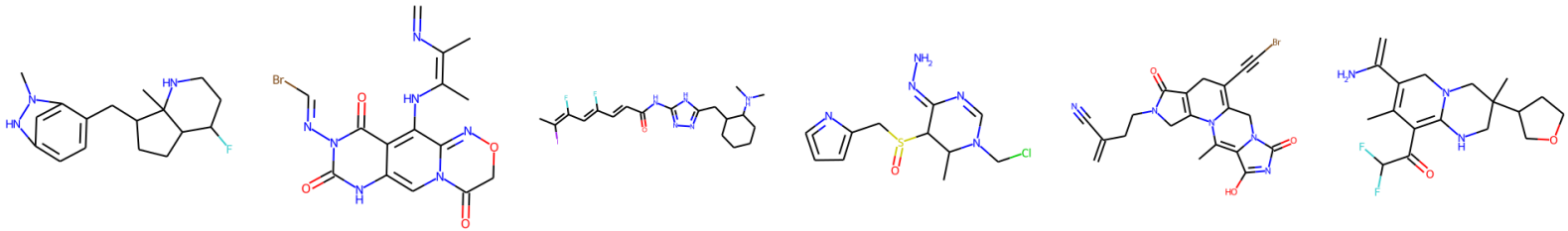
Research update (2)

The difference between this architecture and RL approach is:

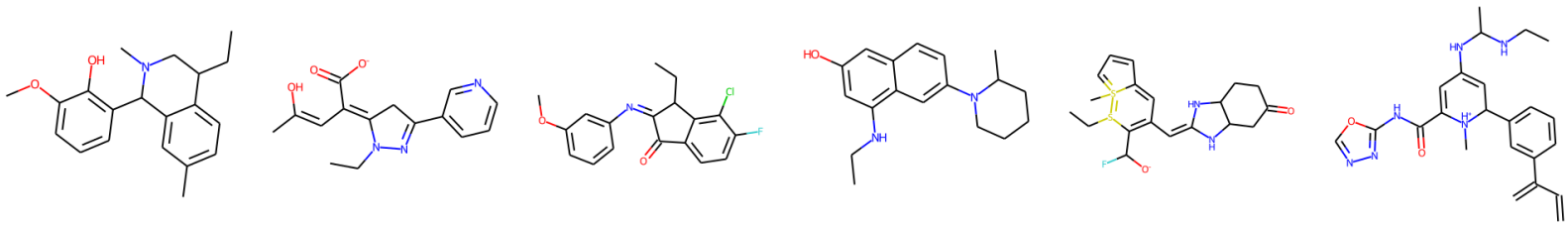
1. It uses VAE.
2. In the generation process, the RL selects a new atom type during the construction but this one basically has the atom types fixed at the beginning, we only select a new edge/bond.
3. RL is **unstable** and hard to train; only after we construct the whole molecular graph, we might get some rewards. And in RL, atom labels are not known, the model must decide to add an atom into the canvas or terminate.
4. It is more like **intimidation learning** in Eric Jonas's paper: actually we break down the generation process into multiple classifications, each classification is given an input as a partial graph and we have to predict the next edge (only the next edge, because the vertex/atom labels are known already).

Research update (3)

I think it is easier for a graph nets in general to do the classification tasks rather than **all-at-once** generation.



Examples generated with (global) Sn/Maron



Examples generated with (global) Sn/Maron + (local) CCN 1D
It seems to like Benzen rings (in almost every generated molecule).

I am going to wrap PCCN into TensorFlow for CCN 2D, and more.

Research update (4) – Need for higher order

A Generative Model for Molecular Distance Geometry (ICML 2020):

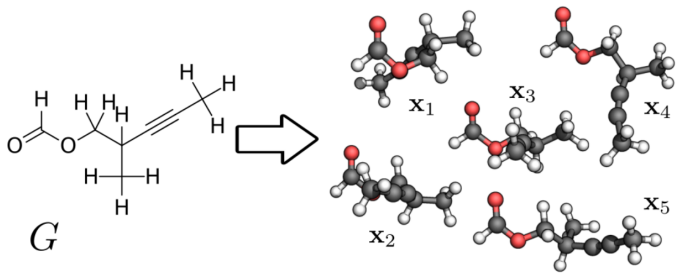


Figure 1. Standard graph representation G of a molecule (left) with a set of possible conformations $\{\mathbf{x}_i\}$ (right). It is the goal of this work to generate such conformations from the graph representation of a molecule. Conformations feature the same atom types and bonds but the atoms are arranged differently in space. These differences arise from rotations around and stretching of bonds in the molecule. Hydrogen (H), carbon (C), and oxygen (O) atoms are colored white, gray, and red, respectively.

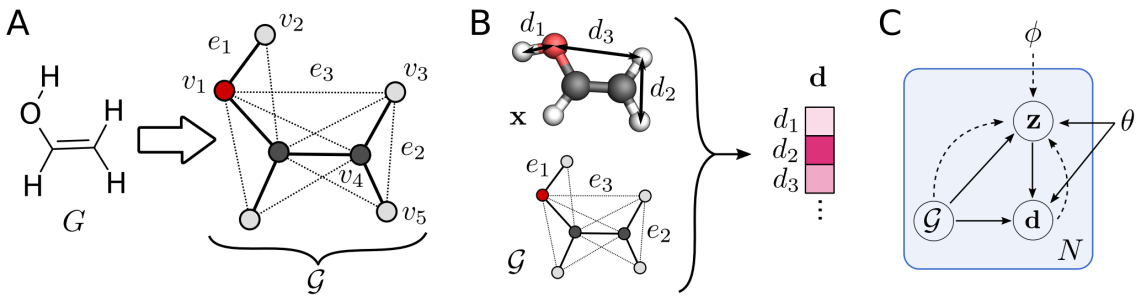


Figure 2. A) The structural formula of a molecule G is converted to an extended molecular graph $\mathcal{G}$ consisting of nodes representing atoms (circles, e.g., v_1) and edges representing molecular bonds (solid lines, e.g., $e_1 \in E_{\text{bond}}$) and auxiliary edges (dotted lines, e.g., $e_2 \in E_{\text{angle}}$ and $e_3 \in E_{\text{dihedral}}$). B) The distances $\mathbf{d}$ are extracted from a conformation $\mathbf{x}$ based on the edges E. C) Graphical model of the variational autoencoder: generative model $p_\theta(\mathbf{d}|\mathbf{z},\mathcal{G})p_\theta(\mathbf{z}|\mathcal{G})$ (solid lines) and variational approximation $q_\phi(\mathbf{z}|\mathbf{d},\mathcal{G})$ (dashed lines).