GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation

By: Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, Jian Tang

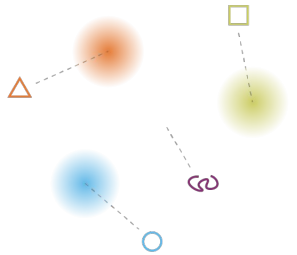
Presented by: Daniel Ahn and Xuan Lin

Introduction

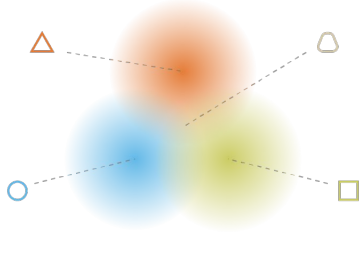
- Molecular design with desired properties
  - Applications to drug discovery and materials science
  - Used to help find drugs for Covid-19 vaccine although recommendations were ultimately not used
- Large number of potential molecules due to number of atoms, types of bonds, orientations, etc.
  - Search space estimated to be as large as 10^{33}
- Amenable to machine learning approaches due to generative models and representation of chemical structure as graphs

Related Works: Variational Autoencoder

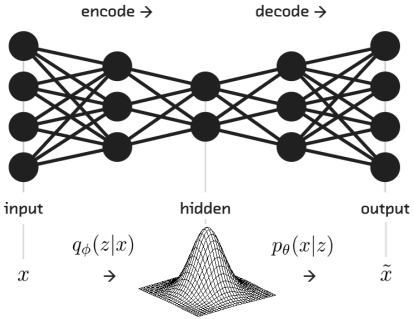
- Autoencoders with a continuous latent space
- Force the latent space to be continuous by projecting input into properties of distributions
- Generates outputs by sampling from learned distributions
- Minimizes ELBO loss



what can happen without regularisation

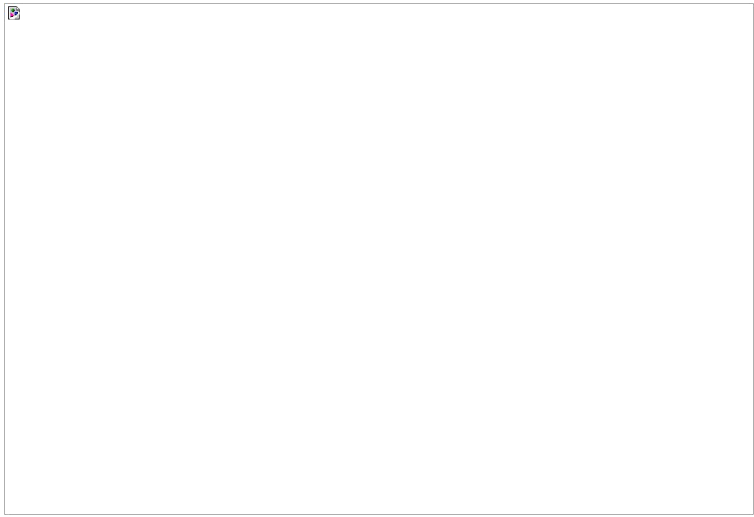


what we want to obtain with regularisation



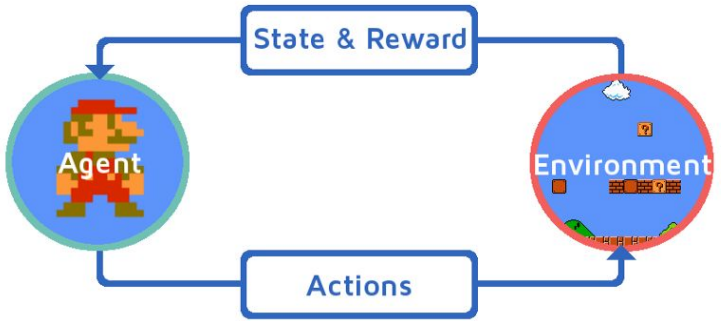
Related Works: Generative Adversarial Network

- Generator and discriminator compete in a zero sum game
  - Generator must generate outputs that fool the discriminator
  - Discriminator must correctly distinguish between generated and real outputs
- No log likelihood maximization
  - Uses discriminator as a proxy for distance between learned and target distributions



Related Works: Graph Convolutional Policy Network

- Problem reformulated in terms of states and actions with associated rewards
  - State: current graph structure
  - Action: new additions to graph
  - Reward: property of resulting graph
- Learn a correlation with graph state and next recommended addition by accruing reward for high performing final graphs



Related Works: Autoregressive Models

- Predicted value on time step t is a function of values on timesteps **T** < t

$$x_t = c + \sum_{i}^{t-1} \phi_i x_i + \epsilon$$

- Useful for modelling sequential data and processes

Background: Normalizing Flow

- An invertible mapping

$$f : \mathcal{E} \rightarrow \mathcal{Z}$$

- where $\mathcal{E}$ is the latent distribution and $\mathcal{Z}$ is the data distribution
- Data distribution given by

$$p_Z(z) = p_\mathcal{E}\left(f_\theta^{-1}(z)\right)\left|\det \frac{\partial f_\theta^{-1}(z)}{\partial z}\right|.$$

Background: Autoregressive Flow

- Latent distribution modeled by a gaussian over the previous time steps

$$p(z_d|z_{1:d-1}) = \mathcal{N}(z_d|\mu_d, (\alpha_d)^2), \text{ where } \mu_d = g_\mu(z_{1:d-1};\theta), \alpha_d = g_\alpha(z_{1:d-1};\theta).$$

- g_μ and g_α are neural networks parameterized by Θ
- Data distribution calculated with the affine transformation

$$f_\theta(\epsilon_d) = z_d = \mu_d + \alpha_d \cdot \epsilon_d;\ f_\theta^{-1}(z_d) = \epsilon_d = \frac{z_d - \mu_d}{\alpha_d}.$$

Quiz

T/F: For autoregressive conditional probabilities parameterized by Gaussians, the mean and standard deviation of the latent distribution are only a function of the current sampled output

- True
- **False**

Background: Graph Representation Learning

- Graph with n nodes, d types of nodes, and b types of edges is represented by G = (A, X)
  - A is the adjacency matrix $A \in \{0,1\}^{n \times n \times b}$
    - A_{ijk} = 1 if there is an edge of type k between nodes i and j
  - X is the node feature matrix $X \in \{0,1\}^{n \times d}$
    - X_i = 1 if the node has attribute i
- A relational graph convolutional network (R-GCN) is used to learn node embeddings
  - Embedding at layer l is given by

$$H^l = \mathrm{Agg}\left(\mathrm{ReLU}\left(\{\tilde{D}_i^{-\frac{1}{2}}\tilde{E}_i\tilde{D}_i^{-\frac{1}{2}}H^{l-1}W_i^l\}\middle| i \in (1,\ldots,b)\right)\right)$$

  - where $\tilde{E}$= A[:, :,i] with a self loop and $\tilde{D}$ = the degree matrix of $\tilde{E}$

Methods: GraphAF

- Graph generation is modeled by a sequential process
  - Given graph G_i, a new node is added to the subgraph, $p(X_i, G_i)$
  - After the nodes is added, an edge is added to connect the node to the rest of the graph, $p(A_{ij}, G_i, X_i, A_{i,1:j-1})$
    - To allow for no edges to be connected, add a new edge type to indicate no connection
  - Repeat until either
    - Graph reaches maximum number of nodes
    - Newly added node cannot be connected to rest of the graph
- The autoregressive flow training process is parallelizable
  - Training is twice as fast on GraphAF compared to GCPN

Methods: Dequantization

- Normalizing flow requires continuous inputs
  - A and X are discrete values so they must be transformed into continuous values
  - Use dequantization to transform G = (A,X) to z = (z^A, z^X)

$$z_i^X = X_i + u,\ u \sim U[0,1)^d;\ z_{ij}^A = A_{ij} + u,\ u \sim U[0,1)^{b+1}$$

Methods: Model Latent Distribution

- Apply autoregressive flow to the dequantized graph representation
  - Latent distribution modeled by a gaussian

$$
\begin{aligned}
p(z_i^X|G_i) =& \mathcal{N}(\mu_i^X, (\alpha_i^X)^2), \\
& \text{where } \mu_i^X = g_{\mu^X}(G_i), \alpha_i^X = g_{\alpha^X}(G_i), \\
p(z_{ij}^A|G_i, X_i, A_{i,1:j-1}) =& \mathcal{N}(\mu_{ij}^A, (\alpha_{ij}^A)^2),\ j \in \{1, 2, \ldots, i-1\}, \\
& \text{where } \mu_{ij}^A = g_{\mu^A}(G_i, X_i, A_{i,1:j-1}), \alpha_{ij}^A = g_{\alpha^A}(G_i, X_i, A_{i,1:j-1}).
\end{aligned}
$$

- In order to calculate the parameters μ and α, we must learn the neural networks g_μ and g_α
  - Use a R-GCN to get node embeddings from G_i and feed embeddings to MLPs

$$
\begin{aligned}
&\text{R-GCN: } H_i^L = \text{R-GCN}(G_i),\ \tilde{h}_i = \text{sum}(H_i^L); \\
&\text{Node-MLPs: } g_{\mu^X} = m_{\mu^X}(\tilde{h}_i),\ g_{\alpha^X} = m_{\alpha^X}(\tilde{h}_i); \\
&\text{Edge-MLPs: } g_{\mu^A} = m_{\mu^A}(\tilde{h}_i, H_{i,i}^L, H_{i,j}^L),\ g_{\alpha^A} = m_{\alpha^A}(\tilde{h}_i, H_{i,i}^L, H_{i,j}^L)
\end{aligned}
$$

Methods: Sample and Predict

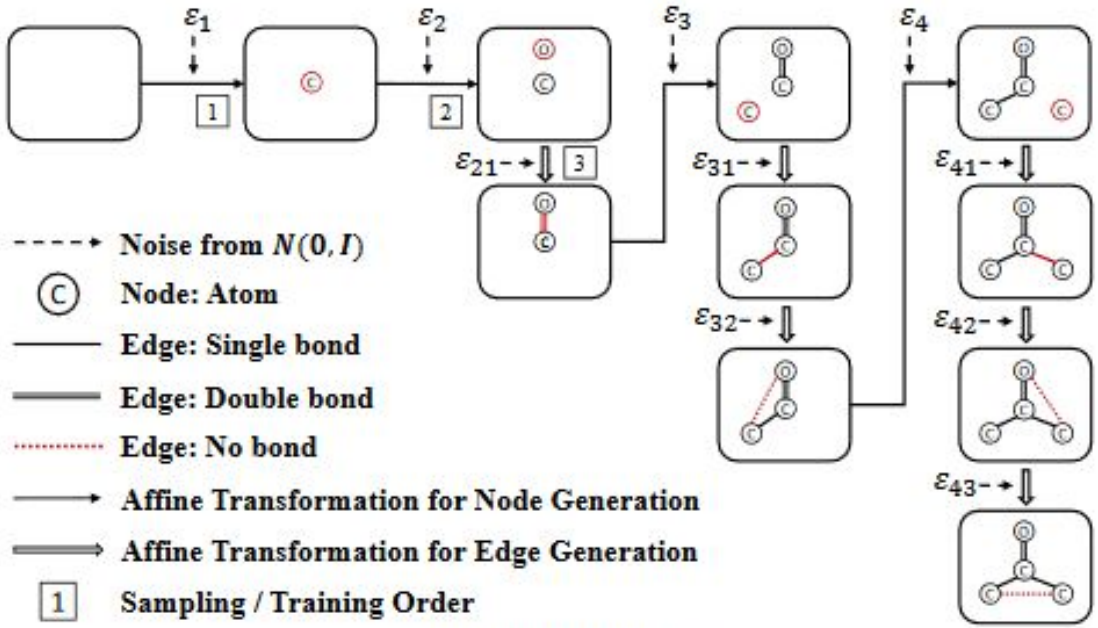
- Sample ϵ_i and ϵ_{ij} from the base Gaussian distribution
  - Apply affine transformation to recover the graph representation of the sample

$$z_i^X = \epsilon_i \odot \alpha_i^X + \mu_i^X,\ \epsilon_i \in \mathbb{R}^d;$$
$$z_{ij}^A = \epsilon_{ij} \odot \alpha_{ij}^A + \mu_{ij}^A,\ j \in \{1, 2, \ldots, i-1\},\ \epsilon_{ij} \in \mathbb{R}^{b+1}$$

- Discretize the real value graph $z = (z^A, z^X)$ to $G = (A, X)$ by using argmax

$$X_i = v^d_{\text{argmax}(z_i^X)}$$
$$A_{ij} = v^{b+1}_{\text{argmax}(z_{ij}^A)}$$

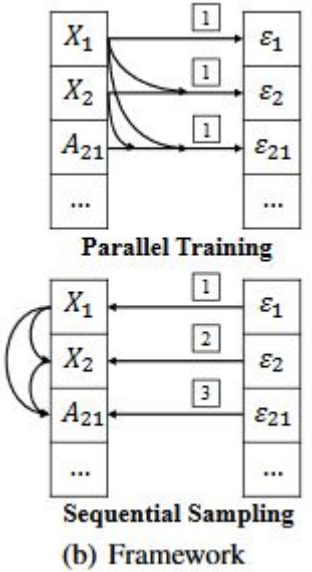
Methods: GraphAF



(a) Sampling Phases

Methods: Efficient Training

- Jacobian of the inverse mapping of normalizing flow is lower triangular
  - Determinant of lower triangular jacobian can be efficiently computed
- Masking nodes and edges in a graph can be used to train R-GCN in parallel
  - Satisfy the autoregressive property by forcing every masked graph to be a subgraph of previous graph
    - Use G_i to predict ϵ_i and G_{i-1} to predict ϵ_{i-1} and $G_{i-1} \subset G_i$



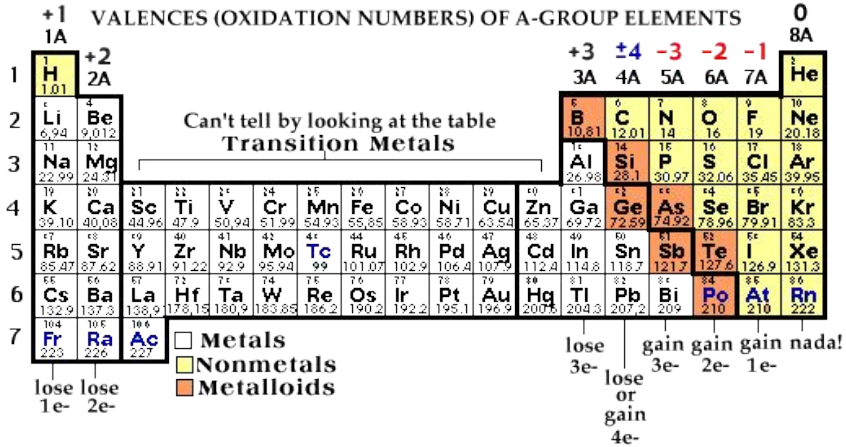
Quiz

SC: Why GraphAF training efficient?

A. The Jacobian of the inverse of normalizing flow is sparse
B. GraphAF utilizes transformers for graph generation which is more parallelizable than RNNs
C. Iterations of the graph generation can be trained independently because masking satisfies the autoregressive property

Methods: Chemistry Recap

- Atoms need to have a specific number of electrons in their outermost orbital to be stable
  - Number of electrons it can give up or accept is known as the valence number
  - For nonmetals and some metals, valence number can be determined by the group number
  - For most metals, valence number can change based on the type of bond formed



Methods: Validity Constraints

- Valence number can be used as a constraint to determine if a molecule is valid

$$\sum_j |A_{ij}| \leq \text{Valency}(X_i) \text{ and } \sum_i |A_{ij}| \leq \text{Valency}(X_j)$$

- If newly added edge violates the valency constraint, reject the edge

Quiz

SC: How does GraphAF handle valency constraints in the generation process?

A. Use the constraint as a term in the loss function and optimize using gradient descent
B. At every step in sequential generation, check if the newly generated molecule obeys the valency constraint
C. After the molecule has been generated, use a pruning algorithm to remove excess edges

Methods: Property Optimization

- Use reinforcement learning to choose nodes and edges that meet a given criteria
- State
  - The current subgraph
  - Only allow state transitions that satisfy the valency constraint
- Policy
  - The autoregressive model defined earlier
  - Given the current state, the subgraph, choose the best action, which node and edges to add

Methods: Property Optimization

- Reward Function
  - Utilizes intermediate rewards, calculated when an action is taken, and final rewards, calculated when the generation procedure is complete
    - Intermediate reward: negative reward if immediate action violates valency rule
    - Final reward: Desired property score, logP, and drug likeness, QED
      - Distributed to intermediate steps with a discount factor

$$L(\theta) = -E_{G\sim p_\theta}\Big\{E_i\Big[\min\left(r_i(\theta)V(G_i,X_i),\text{clip}\left(r_i(\theta),1-\epsilon,1+\epsilon\right)V(G_i,X_i)\right) + E_j\left[\min\left(r_{ij}(\theta)V(G_{ij},A_{ij}),\text{clip}\left(r_{ij}(\theta),1-\epsilon,1+\epsilon\right)V(G_{ij},A_{ij})\right)\right]\Big]\Big\}$$

- where V(state, action) is the advantage function, $r_i(\theta) = \frac{p_\theta(X_i|G_i)}{p_{\theta_{old}}(X_i|G_i)}$, $r_{ij}(\theta) = \frac{p_\theta(A_{ij}|G_{ij})}{p_{\theta_{old}}(A_{ij}|G_{ij})}$ and $G_{ij} = G_i \cup X_i \cup A_{i,1:j-1}$

Experiments

Evaluation Tasks

1. Density Modeling and Generation: learn the data distribution and generate realistic and diverse molecules
2. Property Optimization: generating novel molecules with optimized chemical properties
3. Constrained Property Optimization: modifying the given molecule to improve desired properties while satisfying a similarity constraint

Data ZINC250k molecular dataset: 250, 000 drug-like molecules

Baselines JT-VAE (Jin et al., 2018), GCPN, MolecularRNN (Popova et al., 2019), GraphNVP (Madhawa et al., 2019)

Metrics

Validity is the percentage of valid molecules among all the generated graphs.

Uniqueness is the percentage of unique molecules among all the generated molecules.

Novelty is the percentage of generated molecules not appearing in training set.

Reconstruction is the percentage of the molecules that can be reconstructed from latent vectors.

The above metrics are calculated from 10,000 randomly generated molecules.

Density Modeling and Generation

Table 2: Comparison of different models on density modeling and generation. *Reconstruction* is only evaluated on latent variable models. *Validity w/o check* is only evaluated on models with valency constraints. Result with † is obtained by running GCPN's open-source code. Results with ‡ are taken from Popova et al. (2019).

Method	Validity	Validity w/o check	Uniqueness	Novelty	Reconstruction
JT-VAE	100%	—	100%‡	100%‡	76.7%
GCPN	100%	20%†	99.97%‡	100%‡	—
MRNN	100%	65%	99.89%	100%	—
GraphNVP	42.60%	—	94.80%	100%	100%
GraphAF	100%	68%	99.10%	100%	100%

- GraphAF (as flow based model) holds perfect reconstruction ability compared with VAE approaches.
- GraphAF achieves a 100% validity rate due to the valency check during sequential generation (GraphNVP only achieves 42.60% due to its one-shot sampling process)
- JT-VAE and GCPN take around 24 and 8 hours, respectively, while GraphAF only takes 4 hours

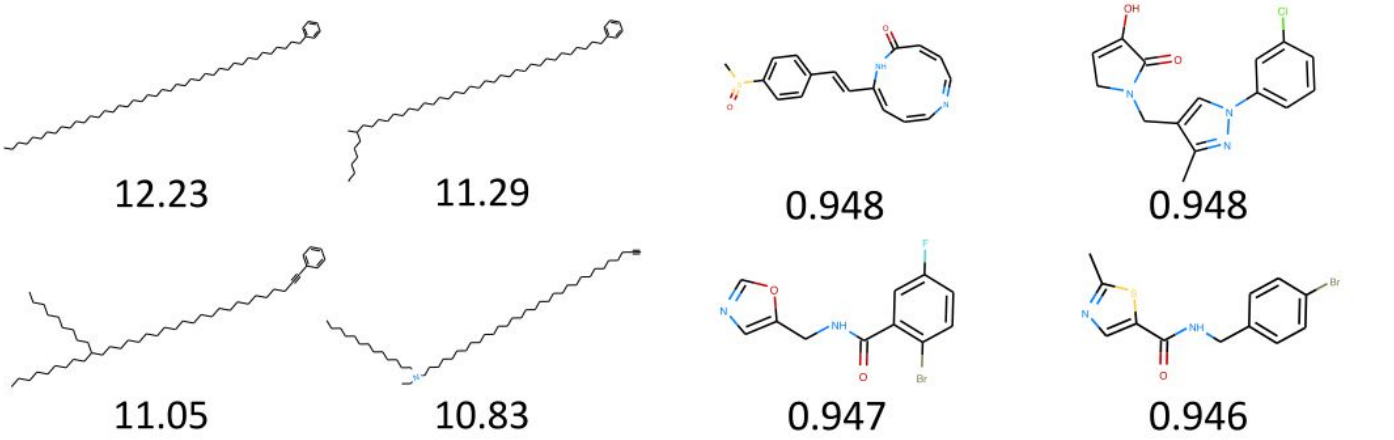
Property Optimization

Table 5: Comparison of the top 3 property scores of generated molecules.

Method	Penalized logP				QED			
	1st	2nd	3rd	Validity	1st	2nd	3rd	Validity
ZINC (Dataset)	4.52	4.30	4.23	100.0%	0.948	0.948	0.948	100.0%
JT-VAE (Jin et al., 2018)	5.30	4.93	4.49	100.0%	0.925	0.911	0.910	100.0%
GCPN (You et al., 2018a)	7.98	7.85	7.80	100.0%	**0.948**	0.947	0.946	100.0%
MRNN[1] (Popova et al., 2019)	8.63	6.08	4.73	100.0%	0.844	0.796	0.736	100.0%
GraphAF	**12.23**	**11.29**	**11.05**	100.0%	**0.948**	**0.948**	**0.947**	100.0%

- logP score penalized by ring size and synthetic accessibility.
- QED measures the druglikeness of the molecules.
- Pretrain the GraphAF network for 300 epochs for likelihood modeling, and apply the RL process to fine-tune the network

Property Optimization



(a) Penalized logP optimization (b) QED optimization

Property Optimization

GraphAF v.s. GCPN

- RL process is similar
- GCPN uses GAN model which is known to suffer from the mode collapse problem
- GraphAF uses flow which is flexible at modeling complex distribution and generating diverse data, encouraging exploration during RL

Constrained Property Optimization

Table 6: Comparison of results on constrained property optimization.

δ	JT-VAE			GCPN			GraphAF		
	Improvement	Similarity	Success	Improvement	Similarity	Success	Improvement	Similarity	Success
0.0	1.91 ± 2.04	0.28 ± 0.15	97.5%	4.20 ± 1.28	0.32 ± 0.12	100%	$\mathbf{13.13 \pm 6.89}$	0.29 ± 0.15	100%
0.2	1.68 ± 1.85	0.33 ± 0.13	97.1%	4.12 ± 1.19	0.34 ± 0.11	100%	$\mathbf{11.90 \pm 6.86}$	0.33 ± 0.12	100%
0.4	0.84 ± 1.45	0.51 ± 0.10	83.6%	2.49 ± 1.30	0.47 ± 0.08	100%	$\mathbf{8.21 \pm 6.51}$	0.49 ± 0.09	99.88%
0.6	0.21 ± 0.71	0.69 ± 0.06	46.4%	0.79 ± 0.63	0.68 ± 0.08	100%	$\mathbf{4.98 \pm 6.49}$	0.66 ± 0.05	96.88%

- Optimize penalized logP for 800 molecules in ZINC250k with the lowest scores
- Set the initial states as sub-graphs randomly sampled from 800 molecules to be optimized during generation

Conclusion

- GraphAF is the first flow-based autoregressive model for generating realistic and diverse molecular graphs.
- GraphAF generates novel and 100% valid molecules in empirical experiments, and its training process is fast.
- The generative process is fine-tuned with reinforcement learning to optimize the properties of generated molecules,