Constrained Graph Variational Autoencoders for Molecule Design

(NIPS 2018)

Qi Liu[1] , Miltiadis Allamanis[2] , Marc Brockschmidt[2] , and Alexander L. Gaunt[2]

https://qdata.github.io/deep2Read

Presenter: Arshdeep Sekhon

Fall 2018

Constrained Graph V

Motivation

- Generate graphs representing distribution in training data
- Generate graphs that obey hard constraints specific to the task
- generating and optimizing chemical molecules : important real-world application

Summary

- Constrained Graph Variational Autoencoder: Graph as input and Graph as output
- encoder-decoder are Gated Graph Neural Nets in a variational autoencoder (VAE)
- incorporate hard domain-specific constraints

Related Work

Key Challenge: sampling directly from a joint distribution over all configurations of labeled nodes and edges is intractable for reasonably sized graphs. Related Work:

- Uncorrelated generation
- Sequential generation

Related Work

Key Challenge: sampling directly from a joint distribution over all configurations of labeled nodes and edges is intractable for reasonably sized graphs. Related Work:

- Uncorrelated generation
- Sequential generation :
  - lose permutation symmetry
  - challenging to marginalize out the construction trace when computing the log-likelihood of a graph in the VAE objective.

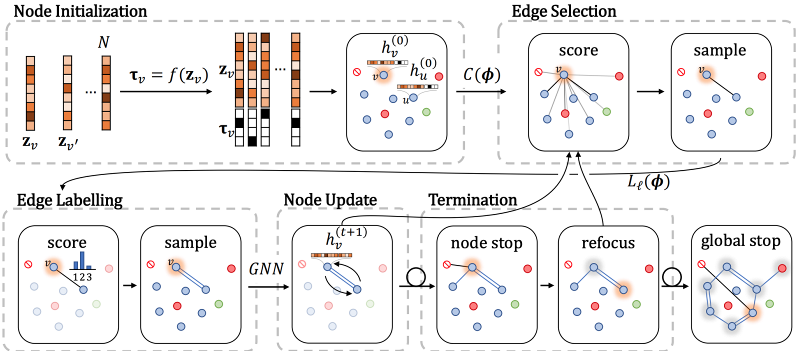
Related Work: Uncorrelated generation

- The Erdos-Rnyi G(n, p) random graph model
- each edge exists with independent probability p.
- GraphVAEs : where the decoder emits independent probabilities governing edge and node existence and labels.

Related Work: Sequential generation

- generating a graph from an auxiliary stream of information that imposes an order on construction steps.
- an autoregressive model for graphs without the auxiliary stream.
    - good results, but each decision is conditioned on a full history of the generation sequence
    - stability and scalability problems

Method



Node Initialization
N
$\mathbf{z}_v$
$\mathbf{z}_{v'}$
$\boldsymbol{\tau}_v = f(\mathbf{z}_v)$
$\mathbf{z}_v$
$\boldsymbol{\tau}_v$
$h_v^{(0)}$
$h_u^{(0)}$
v
u
$C(\boldsymbol{\phi})$
Edge Selection
score
sample
$L_\ell(\boldsymbol{\phi})$
Edge Labelling
score
123
sample
GNN
Node Update
$h_v^{(t+1)}$
Termination
node stop
refocus
global stop

Method: Generative Model/Decoder

- process is seeded with N[1] vectors z_v
- form a latent specification for the graph to be generated
- generation of edges between nodes:
  - focus : zoom into one node
  - focus: deterministic queue using breadth first search
  - expand: chooses edges to add from the focus node.

[1]N is an upper bound on the number of nodes in the final graph

Method: *expand* function

- ideally, make expand condition upon the full history of the generation.
- learned model is likely to learn to reproduce generation traces
- underlying data usually only contains fully formed graphs: trace is an artifact
- this would lead to extremely deep computation graphs, as even small graphs easily have many dozens of edges;
- makes training of the resulting models very hard

Method: *expand* function

- **Solution**: condition expand only upon the partial graph structure G^t generated so far
- intuitively, learning how to complete a partial graph without using any information about how the partial graph was generated.

Method: Node Initalization

- state $h_v^{(t=0)}$ with each node v
- z_v is drawn from the d-dimensional standard normal $N(0, I)$
- $h_v^{(t=0)}$ is the concatenation $[z_v, \tau_v]$
- τ_v is an interpretable one-hot vector : the node type.
- τ_v is derived from z_v by sampling from the softmax output of a learned mapping $\tau_v : f(z_v)$ [2]
- The interpretable component of $h_v^{(t=0)}$: τ_v gives us a means to enforce hard constraints during generation.
- node-level variables,to global representations $H(t)$ and H_{init},
- also initialize a special stop node to a learned representation h_ϕ for algorithm termination

[2] f is an NN

Method: Node Update

- Whenever we obtain a new graph $G^{(t+1)}$, discard $h_v^{(t)}$ and compute new representations $h_v^{(t+1)}$ for all nodes taking their (possibly changed) neighborhood into account.
- This is implemented using a standard gated graph neural network (GGNN) G_{dec} for S steps

$$\mathbf{m}_v^{(0)} = \mathbf{h}_v^{(0)} \qquad \mathbf{m}_v^{(s+1)} = \mathrm{GRU}\left[\mathbf{m}_v^{(s)}, \sum_{v \overset{\ell}{\leftrightarrow} u} E_\ell(\mathbf{m}_u^{(s)})\right] \qquad \mathbf{h}_v^{(t+1)} = \mathbf{m}_v^{(S)}$$

- the sum runs over all edges in the current graph and E is an edge-type specific neural network
- since $h_v^{(t+1)}$ is computed from $h_v^{(0)}$ rather than $h_v^{(t)}$, the representation $h_v^{(t+1)}$ is independent of the generation history of $G^{(t+1)}$

Method: Edge Selection and Labeling

- first pick a focus node v from our queue
- function expand then selects edges $v \overset{l}{\longleftrightarrow} u$ from v to u with label l as follows.
- For each non-focus node u, we construct a feature vector where $d_{v,u}$ is the graph distance between v and u.
- We use these representations to produce a distribution over candidate edges:

$$p(v \overset{l}{\longleftrightarrow} u|\phi_{v,u}^{(t)}) = p(l|\phi_{v,u}^{(t)}, v \overset{l}{\longleftrightarrow} u) \cdot p(\overset{l}{\longleftrightarrow} u)|\phi_{v,u}^{(t)}). \qquad (1)$$

- The factors are calculated as softmax outputs from neural networks C (determining the target node for an edge) and L' (determining the type of the edge):

$$p(v \leftrightarrow u \mid \boldsymbol{\phi}_{v,u}^{(t)}) = \frac{M_{v\leftrightarrow u}^{(t)} \exp[C(\boldsymbol{\phi}_{v,u}^{(t)})]}{\sum_w M_{v\leftrightarrow w}^{(t)} \exp[C(\boldsymbol{\phi}_{v,w}^{(t)})]}, \quad p(\ell \mid \boldsymbol{\phi}_{v,u}^{(t)}) = \frac{m_{v \overset{\ell}{\leftrightarrow} u}^{(t)} \exp[L_\ell(\boldsymbol{\phi}_{v,u}^{(t)})]}{\sum_k m_{v \overset{k}{\leftrightarrow} u}^{(t)} \exp[L_k(\boldsymbol{\phi}_{v,u}^{(t)})]}.$$

Masking

- $M^(t)v \overset{u}{\longleftrightarrow}$ and $m(t)v \overset{l}{\longleftrightarrow} u$ are binary masks that forbid edges that violate constraints.
- New edges are sampled from these distributions
- any nodes that are connected to the graph for the first time are added to the focus queue.
- only consider undirected edges in this paper, extension to directed graphs

Method: Generative: Termination

- keep adding edges to a node v using expand and G_{dec} until an edge to the stop node is selected.
- Node v then loses focus and becomes closed (mask M ensures that no further edges will ever be made to v).
- The next focus node is selected from the focus queue.
- a single connected component is grown in a breadth-first manner.
- Edge generation continues until the queue is empty

Method: training the generative model: Encoder

- The encoder of our VAE is a GGNN G_{enc} that embeds each node in an input graph G to a diagonal normal distribution in d-dimensional latent space parametrized by mean μ_v and standard deviation σ_v vectors.
- The latent vectors z_v are sampled from these distributions
- VAE regularizer term measuring the KL divergence between the encoder distribution and the standard Gaussian prior:

$$L_{latent} = \Sigma_{v \in G} KL(N(\mu, diag^2_{\sigma_v})||N(0, I)) \tag{2}$$

Training the Decoder

- decoder: generative procedure described previously
- condition generation on a latent sample from the encoder distribution during training

Node Initialization

- first sample a node specification z_v for each node v
- independently for each node generate the label τ_v using the learned function f.
- The probability of re-generating the labels $\tau*_v$ observed in the encoded graph is given by a sum over node permutations

$$p(\mathcal{G}^{(0)} \mid \mathbf{z}) = \sum_{\mathcal{P}} p(\boldsymbol{\tau} = \mathcal{P}(\boldsymbol{\tau}^*) \mid \mathbf{z}) > \prod_v p(\boldsymbol{\tau}_v = \boldsymbol{\tau}_v^* \mid \mathbf{z}_v).$$

- This inequality provides a lower bound given by the single contribution from the ordering used in the encoder

Edge Selection and Labelling

- During training, provide supervision on the sequence of edge additions based on breadth-first traversals of each graph in the dataset D.
- to learn a distribution over graphs (and not graph generation traces), we would need to train with an objective that computes the log-likelihood of each graph by marginalizing over all possible breadth-first traces.
- computationally intractable: only compute a Monte-Carlo estimate of the marginal on a small set of sampled traces.
- expand model is not conditioned on full traces, and instead only considers the partial graph generated so far.

Edge Selection and Labeling

- Jensens inequality to show that the log-likelihood of a graph G is loosely lower bounded by the expected log-likelihood of all the traces $\prod$ that generate it.

$$\log p(\mathcal{G} \mid \mathcal{G}^{(0)}) = \log \sum_{\pi \in \Pi} p(\pi \mid \mathcal{G}^{(0)}) \geq \log(|\Pi|) + \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \log p(\pi \mid \mathcal{G}^{(0)})$$

- decompose each full generation trace $\pi \in \prod$ into a sequence of steps of the form (t, v, ϵ), where v is the current focus node and $\epsilon = vl u$is the edge added at step t:

$$\log p(\pi \mid \mathcal{G}^{(0)}) = \sum_{(t,v,\epsilon)\in\pi} \left\{ \log p(v \mid \pi, t) + \log p(\epsilon \mid \mathcal{G}^{(t-1)}, v) \right\}$$

Edge Selection and Labeling

- first term corresponds to the choice of v as focus node at step t of trace π.
- focus function is fixed: this choice is uniform in the first focus node and then deterministically follows a breadth-first queuing system.
- A summation over this term thus evaluates to the constant $\log(1/N)$.

Edge Selection and Labeling

- the second term only conditioned on the current graph (and not the whole generation history $G^{(0)} \dots G^{(t1)}$).
- consider the set of generation states S of all valid state pairs $s = (G^{(t)}, v)$ of a partial graph $G^{(t)}$ and a focus node v.
- We use $|s|$ to denote the multiplicity of state s in $\prod$, i.e., the number of traces that contain graph $G^{(t)}$ and focus on node v.
- Let E_s denote all edges that could be generated at state s (the edges from the focus node v that are present in the graph G from the dataset, but are not yet present in $G^{(t)}$.)

Edge Selection and Labeling

- Then, each of these appears uniformly as the next edge to generate in a trace for all $|s|$ occurrences of s in a trace from $\prod$
- rearrange a sum over paths into a sum over steps

$$\frac{1}{|\Pi|}\sum_{\pi\in\Pi}\sum_{(t,v,\epsilon)\in\pi}\log p(\epsilon \mid s) = \frac{1}{|\Pi|}\sum_{s\in\mathcal{S}}\sum_{\epsilon\in\mathcal{E}_s}\frac{|s|}{|\mathcal{E}_s|}\log p(\epsilon \mid s)$$
$$= \mathbb{E}_{s\sim\Pi}\left[\frac{1}{|\mathcal{E}_s|}\sum_{\epsilon\in\mathcal{E}_s}\log p(\epsilon \mid s)\right]$$

- $|s|/|\prod|$: the probability of observing state s in a random draw from all states in $\prod$
- $L_{recon} = \sum_{G\in D} log[p(G|G^{(0)}) \cdot p(G^{(0)}|z)]$

Optimizing Graph Properties

- perform (local) optimization of these graphs with respect to some numerical property, Q.
- gradient ascent in the continuous latent space using a differentiable gated regression model $R(zv) = \Sigma_v \sigma(g_1(z_v))g_2(z_v)$, where g_1 and g_2 are neural networks
- During training, L2 distance loss L_Q between $R(z_v)$ and the labeled properties Q.
- at test time, we can sample an initial latent point z_v and then use gradient ascent to a locally optimal point z_v^* subject to an L2 penalty that keeps the z_v^* within the standard normal prior of the VAE.
- Decoding from the point z_v^* then produces graphs with an optimized property Q.

Objective

$$L = L_{recon.} + \lambda_1 L_{latent} + \lambda_2 L_Q, \tag{3}$$

Experiments

Datasets"

- QM9: 134k samples with 9 atoms, , the molecules in this dataset only reflect basic structural constraints.
- ZINC: 250k drug compounds 23 atoms, molecules are bigger (23 heavy atoms) and structurally more complex than the molecules in QM9.
- CEPDB: 250k organic molecules 28 heavy atoms, structurally very complex, containing six to seven rings each

In the encoder, molecular graphs are presented with nodes annotated with onehot vectors τ_v indicating their atom type and charge.

Experiments: Datasets

- edge types: single, double, triple covalent bonds
- Encoder Inputs: node one hot vector that indicate atom type and charge

Experiemnts: Valency Masking

- valency : the number of bonds that that atom can make in a stable molecule
- valency rules : some atoms can only make certain types of bonds
- valency of 2: 2 bonds
- Mask: the number of bonds b_v never exceeds the valency b_v* of the node
- if bonds less than the valency: add hydrogen atoms
- this generates valid molecules: parse the graph to SMILES string using RDKitParser
- also handle avoidance of edge duplication and self loops

$$M^{(t)}_{v\leftrightarrow u} = \mathbb{1}(b_v < b_v^*) \times \mathbb{1}(b_u < b_u^*) \times \mathbb{1}(\text{no } v \leftrightarrow u \text{ exists}) \times \mathbb{1}(v \neq u) \times \mathbb{1}(u \text{ is not closed})$$

Baselines

- DeepGAR: DeepAutoregressive model
- GrammarVAE
- ChemVAE
- GraphVAE
- random graph models

train on the three datasets and then sample 20k molecules from the trained models

Metrics

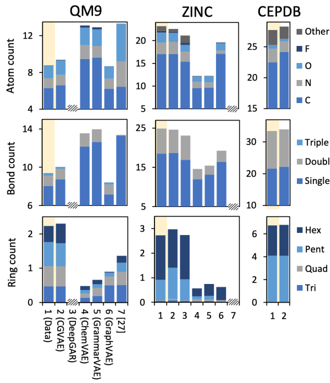
- Novelty: fraction of molecules not appearing in the training data
- syntactic validitty
- uniqueness: ratio of sample set size before and after deduplication of identical molecules
- average atom type
- average bond type
- the average number of 3-, 4-, 5-, and 6-membered cycles in each molecule.

Results

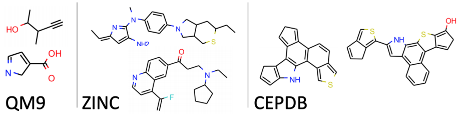
(a)

	Measure	2: CGVAE	3: [21]	4: [8]	5: [17]	6: [29]	7: [27]
QM9	% valid	100	-	10.00	30.00	61.00	98.00
	% novel	94.35	-	90.00	95.44	85.00	100
	% unique	98.57	-	67.50	9.30	40.90	99.86
ZINC	% valid	100	89.20	17.00	31.00	14.00	-
	% novel	100	89.10	98.00	100	100	-
	% unique	99.82	99.41	30.98	10.76	31.60	-
CEPDB	% valid	100	-	8.30	0.00	-	-
	% novel	100	-	90.05	-	-	-
	% unique	99.62	-	80.99	-	-	-

(b)

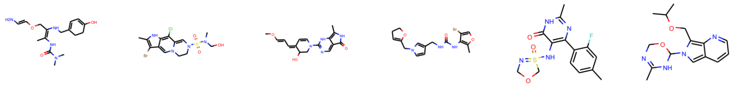


(c)



Results: Directed Molecule Generation

- predict the Quantitative Estimate of Drug-Likeness (QED) directly from latent space
- generate molecules with very high QED values by performing gradient ascent in the latent space using the trained QED-scoring network.



Pred. QED	0.5686	0.6685	0.7539	0.8376	0.9013	0.9271
Real QED	0.5345	0.6584	0.7423	0.8298	0.8936	0.9383