CATEGORICAL NORMALIZING FLOWS VIA CONTINUOUS TRANSFORMATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite their popularity, to date, the application of normalizing flows on categorical data stays limited. The current practice of using dequantization to map discrete data to a continuous space is inapplicable as categorical data has no intrinsic order. Instead, categorical data have complex and latent relations that must be inferred, like the synonymy between words. In this paper, we investigate *Categorical Normalizing Flows*, that is normalizing flows for categorical data. By casting the encoding of categorical data in continuous space as a variational inference problem, we jointly optimize the continuous representation and the model likelihood. Using a factorized decoder, we introduce an inductive bias to model any interactions in the normalizing flow. As a consequence, we do not only maintain a close-to-exact likelihood estimate compared to having a joint decoder, but also make it possible to scale up to a large number of categories that is currently impossible with discrete normalizing flows. Based on Categorical Normalizing Flows, we propose GraphCNF a permutation-invariant generative model on graphs. GraphCNF implements a three step approach modeling the nodes, edges and adjacency matrix stepwise to increase efficiency. On molecule generation, GraphCNF outperforms both one-shot and autoregressive flow-based state-of-the-art.

1 INTRODUCTION

Normalizing Flows have been popular for tasks with continuous data like image modeling (Dinh et al., 2017; Kingma and Dhariwal, 2018; Ho et al., 2019) and speech generation (Kim et al., 2019; Prenger et al., 2019) by providing efficient parallel sampling and exact density evaluation. The concept that normalizing flows rely on is the rule of change of variables, a continuous transformation designed for continuous data. However, there exist many data types typically encoded as discrete, categorical variables, like language and graphs, where normalizing flows are not straightforward to apply.

To address this, it has recently been proposed to discretize the transformations inside normalizing flows to act directly on discrete data. Unfortunately, these discrete transformations have shown to be limited in terms of the vocabulary size and layer depth due to gradient approximations (Hoogeboom et al., 2019; Tran et al., 2019). For the specific case of discrete but ordinal data, like images where integers represent quantized values, a popular strategy is to add a small amount of noise to each value (Dinh et al., 2017; Ho et al., 2019). It is unnatural, however, to apply such dequantization techniques for the general case of categorical data, where values represent categories with no intrinsic order. Treating these categories as integers for dequantization biases the data to a non-existing order, and makes the modeling task significantly harder. In addition, relations between categories are often multi-dimensional, for example word meanings, which cannot be represented with dequantization.

In this paper, we investigate normalizing flows for the general case of categorical data. To account for discontinuity, we propose continuous encodings in which different categories correspond to unique, non-overlapping and thus close-to-deterministic volumes in a continuous latent space. Instead of pre-specifying the non-overlapping volumes per category, we resort to variational inference to jointly learn those and model the likelihood by a normalizing flow at the same time. This work is not the first to propose variational inference with normalizing flows, mostly considered for improving the flexibility of the approximate posterior (Kingma et al., 2016; Rezende and Mohamed, 2015; Van Den Berg et al., 2018). Different from previous works, we use variational inference to learn a continuous representation $\boldsymbol{z}$ of the discrete categorical data $\boldsymbol{x}$ to a normalizing flow. A similar

idea has been investigated in (Ziegler and Rush, 2019), who use a variational autoencoder structure with the normalizing flow being the prior. As both their decoder and normalizing flow model (complex) dependencies between categorical variables, (Ziegler and Rush, 2019) rely on intricate yet sensitive learning schedules for balancing the likelihood terms. Instead, we propose to separate the representation and relation modeling by factorizing the decoder both over the categorical variable x and the conditioning latent z. This forces the encoder and decoder to focus only on the mapping from categorical data to continuous encodings, and not model any interactions. By inserting this inductive bias, we move all complexity into the flow and achieve a close-to exact likelihood estimate. We call this approach *Categorical Normalizing Flows* (CNF).

Categorical Normalizing Flows can be applied to any task involving categorical variables, but we primarily focus on modeling graphs. Current state-of-the-art approaches often rely on autoregressive models (Li et al., 2018; Shi et al., 2020; You et al., 2018) that view graphs as sequences, although there exists no intrinsic order of the node. In contrast, normalizing flows can perform generation in parallel making a definition of order unnecessary. By treating both nodes and edges as categorical variables, we employ our variational inference encoding and propose GraphCNF. GraphCNF is a novel permutation-invariant normalizing flow on graph generation which assigns equal likelihood to any ordering of nodes. Meanwhile, GraphCNF efficiently encodes the node attributes, edge attributes and graph structure in three consecutive steps. As shown in the experiments, the improved encoding and flow architecture allows GraphCNF to outperform significantly both the autoregressive and parallel flow-based state-of-the-art. Further, we show that Categorical Normalizing Flows can be used in problems with regular categorical variables like modeling natural language or sets.

Our contributions are summarized as follows. Firstly, we propose Categorical Normalizing Flows using variational inference with a factorized decoder to obtain a close-to exact likelihood estimate and scale up to large number of categories. Secondly, starting from the Categorical Normalizing Flows, we propose GraphCNF, a permutation-invariant normalizing flow on graph generation. On molecule generation, GraphCNF sets a new state-of-the-art for flow-based methods outperforming one-shot and autoregressive baselines. Finally, we show that simple mixture models for encoding distributions are accurate, efficient and generalize across a multitude of setups, including sets language and graphs.

2 Categorical Normalizing Flows

2.1 Normalizing Flows on continuous data

A normalizing flow (Rezende and Mohamed, 2015; Tabak and Vanden Eijnden, 2010) is a generative model that models a probability distribution $p(\boldsymbol{z}^{(0)})$ by applying a sequence of invertible, smooth mappings $f_1, ..., f_K : \mathbb{R}^d \to \mathbb{R}^d$. Using the rule of change of variables, the likelihood of the input $\boldsymbol{z}^{(0)}$ is determined as follows:

$$p(\boldsymbol{z}^{(0)}) = p(\boldsymbol{z}^{(K)}) \cdot \prod_{k=1}^{K} \left| \det \frac{\partial f_k(\boldsymbol{z}^{(k-1)})}{\partial \boldsymbol{z}^{(k-1)}} \right| \tag{1}$$

where $\boldsymbol{z}^{(k)} = f_k(\boldsymbol{z}^{(k-1)})$, and $p(\boldsymbol{z}^{(K)})$ represents a prior distribution. This calculation requires to compute the Jacobian for the mappings $f_1, ..., f_K$, which is expensive for arbitrary functions. Thus, the mappings are often designed to allow an efficient computation of its determinant. One of such is the coupling layer proposed by Dinh et al. (2017) which showed to work well with neural networks. For a detailed introduction to normalizing flows, we refer the reader to Kobyzev et al. (2019).

2.2 Normalizing Flows on Categorical Data

We define $\boldsymbol{x} = \{x_1, ..., x_S\}$ to be a multivariate, categorical random variable, where each element x_i is itself a categorical variable of K categories with no intrinsic order. For instance, $\boldsymbol{x}$ could be a sentence with x_i being the words. Our goal is to learn the joint probability mass function, $P_{\text{model}}(\boldsymbol{x})$, via a normalizing flow. Specifically, as normalizing flows constitute a class of continuous transformations, we aim to learn a continuous latent space in which each categorical choice of a variable x_i maps to one distribution of a continuous variable $\boldsymbol{z}_i \in \mathbb{R}^d$.

Compared to variational autoencoders (Kingma and Welling, 2014) and latent normalizing flows (Ziegler and Rush, 2019), we want to ensure that all modeling complexity is solely in the prior, and

keep a lossless reconstruction from latent space. To implement this, we simplify the decoder by factorizing the decoder over latent variables: $p(\boldsymbol{x}|\boldsymbol{z}) = \prod_i p(x_i|\boldsymbol{z}_i)$. Factorizing the conditional likelihood means that we enforce independence between the categorical variables x_i given their learned continuous encodings $\boldsymbol{z}_i$. Therefore, any interaction between the categorical variables $\boldsymbol{x}$ must be learned inside the normalizing flow, which incentivizes the decoder to be deterministic, namely to precisely model x and obtain high likelihoods. As a consequence, the encoder $q(\boldsymbol{z}|\boldsymbol{x})$ is being optimized to provide suitable representations of the categorical variables to the flow while separating the different categories in latent space to improve the decoding. As the decoder becomes almost deterministic, the KL divergence between the true and approximate posterior is encouraged to be close to zero. Overall, our objective becomes:

$$\mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}\left[\log P_{\text{model}}(\boldsymbol{x})\right] \geq \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}\mathbb{E}_{\boldsymbol{z}\sim q(\cdot|\boldsymbol{x})}\left[\log \frac{p_{\text{model}}(\boldsymbol{z})\prod_i p(x_i|\boldsymbol{z}_i)}{q(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{2}$$

We refer to this framework as *Categorical Normalizing Flows*. In contrast to dequantization, the continuous encoding $\boldsymbol{z}$ is not bounded by the domain of the encoding distribution. Instead, the partitioning is jointly learned with the model likelihood. Furthermore, we can freely choose the dimensionality of the continuous variables, $\boldsymbol{z}_i$, to fit the number of categories and their relations.

Modeling the encoder The encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and decoder $p(x_i|\boldsymbol{z}_i)$ can be implemented in several ways. The first and main setup we consider is to encode each category by a logistic distribution with a learned mean and scaling. Therefore, our overall encoding distribution is a mixture of K logistics, one per category. With g denoting the logistic, the encoder becomes $q(\boldsymbol{z}|\boldsymbol{x}) = \prod_{i=1}^{N} g(\boldsymbol{z}_i|\mu(x_i), \sigma(x_i))$. In this setup, the true decoder likelihood can actually be found by applying Bayes: $p(x_i|\boldsymbol{z}_i) = \frac{\tilde{p}(x_i)q(\boldsymbol{z}_i|x_i)}{\sum_{\hat{x}}\tilde{p}(\hat{x})q(\boldsymbol{z}_i|\hat{x})}$ with $\tilde{p}(x_i)$ being a prior over categories. Therefore, by using a mixture model, we do not have a unknown KL divergence between approximate and true posterior constituting an ELBO.

The mixture model is simple, efficient and removes the ELBO, but might be limited in the distributions it can express. To test whether a greater encoding flexibility is needed, we experiment with adding flows conditioned on the categories which transform each logistic into a more complex distribution. We refer to this approach as linear flows. Taking a step further, we can also represent the encoder $q(\boldsymbol{z}|\boldsymbol{x})$ with a flow across categorical variables, similar to variational dequantization (Ho et al., 2019). Experiments presented in Section 5 show however that a simple mixture of logistic usually suffices.

3 Graph generation with Categorical Normalizing Flows

A graph $\mathcal{G} = (V, E)$ is defined by a set of nodes V, and a set of edges E representing connections between nodes. When modeling a graph, we must take into account the node and edge attributes, often represented by categorical data, as well as the overall graph structure. Moreover, nodes and edges are better viewed as sets and not sequences since any permutation represents the same graph and assigned the same likelihood.

We propose GraphCNF, a normalizing flow for graph generation, which is invariant to the order of nodes by generating all nodes and edges at once. Given a graph $\mathcal{G}$, we model each node and edge as a separate categorical variable where the categories correspond to their discrete attributes. To represent the graph structure, i.e. between which pairs of nodes does or does not exist an edge, we add an extra category to the edges representing the missing or *virtual* edges. Hence, to model an arbitrary graph, we consider an edge variable for every possible tuple of nodes.

To apply normalizing flows on the node and edge categorical variables, we map them into continuous latent space using Categorical Normalizing Flows. Subsequent coupling layers map those representations to a continuous prior distribution. Thereby, GraphCNF uses two crucial design choices for graph modeling: (1) we perform the generation stepwise by first encoding the nodes, edges and then the adjacency matrix for improved efficiency, and (2) we introduce an inductive bias that the model assigns equal likelihood to any ordering of the nodes.

3.1 Three-step generation

Modeling all edges including the virtual ones requires a significant amount of latent variables and is computationally expensive. However, normalizing flows have been shown to benefit from splitting of

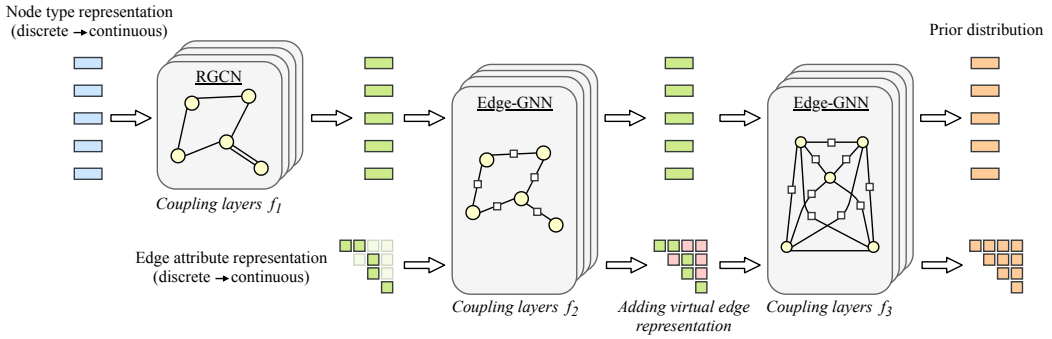

Figure 1: Visualization of GraphCNF for an example graph of five nodes. We add the node and edge attributes, as well as the virtual edges stepwise to the latent space while leveraging the graph structure in the coupling layers. The last step considers a fully connected graph with features per edge.

latent variables at earlier layers while increasing efficiency (Dinh et al., 2017; Kingma and Dhariwal, 2018). Thus, we propose to add the node types, edge attributes and graph structure stepwise to the latent space as visualized in Figure 1. In the first step, we encode the nodes into continuous latent space, $\boldsymbol{z}_0^{(V)}$, using Categorical Normalizing Flows. On those, we apply a group of coupling layers, f_1, which additionally use the adjacency matrix and the edge attributes, denoted by E_{attr}, as input. Thus, we can summarize the first step as:

$$z_1^{(V)} \quad = \quad f_1\big(z_0^{(V)}; E, E_{attr}\big) \tag{3}$$

The second step incorporates the edge attributes, E_{attr}, into latent space. Hence, all edges of the graph except the virtual edges are encoded into latent variables, $\boldsymbol{z}_0^{(E_{attr})}$, representing their attribute. The following coupling layers, denoted by f_2, transform both the node and edge attribute variables:

$$z_2^{(V)}, z_1^{(E_{attr})} \quad = \quad f_2\big(z_1^{(V)}, z_0^{(E_{attr})}; E\big) \tag{4}$$

Finally, we add the virtual edges to the latent variable model as $z_0^{(E^*)}$. Thereby, we need to slightly adjust our encoding from Categorical Normalizing Flows as we consider the virtual edges as an additional category of the edges. While the other categories are already encoded by $\boldsymbol{z}_1^{(E_{attr})}$, we add a separate encoding distribution for the virtual edges, for which we use a simple logistic. Meanwhile, the decoder needs to be applied on all edges, as we need to distinguish the continuous representation between virtual and non-virtual edges. Overall, the mapping can be summarized as:

$$z_3^{(V)}, z_1^{(E)} \quad = \quad f_3\big(z_2^{(V)}, z_0^{(E)}\big) \quad \text{where} \quad z_0^{(E)} = \big[z_1^{(E_{attr})}, z_0^{(E^*)}\big] \tag{5}$$

where the latent variables $z_3^{(V)}$ and $z_1^{(E)}$ are trained to follow a prior distribution. During sampling, we first inverse f_3 and determine the general graph structure. Next, we inverse f_2 and reconstruct the edge attributes. Finally, we apply the inverse of f_1 and determine the node types.

3.2 Permutation-invariant graph modeling

To achieve permutation invariance with respect to the likelihood estimate, the transformations of the coupling layers need to be independent of the node order. This includes both the split of variables that will be transformed, and the network model that predicts the transformation parameters. We ensure the first aspect by applying a channel masking strategy (Dinh et al., 2017), where the split is performed over the latent dimensions for each node and edge separately making it independent to the node order. For the second aspect, we leverage the graph structure in the coupling networks and apply graph neural networks. In the first step of GraphCNF, f_1, we use a Relation GCN (Schlichtkrull et al., 2018) which incorporates the categorical edge attributes into the layer. For the second and third step, we need a graph network that supports the modeling of both node and edge features. We implement this by alternating between updates of the edge and the node features. Specifically, given node features $\boldsymbol{v}^t$ and edge features $\boldsymbol{e}^t$ at layer t, we update those as follows:

$$\boldsymbol{v}^{t+1} = f_{node}(\boldsymbol{v}^t; \boldsymbol{e}^t), \qquad \boldsymbol{e}^{t+1} = f_{edge}(\boldsymbol{e}^t; \boldsymbol{v}^{t+1}) \tag{6}$$

We call this network Edge-GNN, and compare different implementations of f_{node} and f_{edge} in Appendix B. Using both design choices, GraphCNF models a permutation invariant distribution.

4 Related work

Dequantization Applying continuous normalizing flows on discrete data leads to undesired density models where arbitrarily high likelihoods are placed on particular values (Theis et al., 2016; Uria et al., 2013). A common solution to this problem is to *dequantize* the data $\boldsymbol{x}$ by adding noise $\boldsymbol{u} \in [0, 1)^D$. Theis et al. (2016) have shown that modeling $p_{\text{model}}(\boldsymbol{x} + \boldsymbol{u})$ lower-bounds the discrete distribution $P_{\text{model}}(\boldsymbol{x})$. The noise distribution $q(\boldsymbol{u}|\boldsymbol{x})$ is usually uniform or learned by a second normalizing flow. The latter is referred to as *variational dequantization* and has been proven to be crucial for state-of-the-art image modeling (Ho et al., 2019; Hoogeboom et al., 2020). Categories, however, are not quantized values, so that ordering them as integers introduces bias to the representation.

Discrete NF Recent works have investigated normalizing flows with discretized transformations. Hoogeboom et al. (2019) proposed to use additive coupling layers with rounding operators for ensuring discrete output. Tran et al. (2019) discretizes the output by a Gumbel-Softmax approximating an argmax operator. Thereby, the coupling layers resemble a reversible shift operator. While both approaches achieved competitive results, due to discrete operations the gradient approximations have been shown to introduce new challenges, such as limiting the number of layers or distribution size.

Latent NF Several works have investigated the application of normalizing flows in variational auto-encoders (VAEs) (Kingma and Welling, 2014) for increasing the flexibility of the approximate posterior (Kingma et al., 2016; Van Den Berg et al., 2018; Tomczak and Welling, 2017). However, VAEs model a lower bound of the true likelihood. To minimize this gap, Ziegler and Rush (2019) proposed Latent Normalizing Flows that move the main model complexity into the prior by using normalizing flows. In contrast to CNFs, Latent NF have a joint decoder over the latent space, $p(\boldsymbol{x}|\boldsymbol{z})$, which allows the modeling of interactions between variables in the decoder. Thus, instead of an inductive bias to push all complexity to the normalizing flow, Latent NF rely on a loss-scheduling weighting the decoder loss much higher. This pushes the decoder to be deterministic, but can lead to unstable optimization due to neglecting the flow's likelihood. Further, experiments on sequence tasks show Latent NF to be competitive but is still outperformed by an LSTM baseline as we observed in our experiments.

Graph modeling The first generation models on graphs have been autoregressive (Liao et al., 2019; You et al., 2018), generating nodes and edges in a sequential order. While being efficient in memory, they are slow in sampling and assume an order in the set of nodes. The first application of normalizing flows for graph generation was introduced by Liu et al. (2019), where a flow modeled the node representations of a pretrained autoencoder. Recent works of GraphNVP (Madhawa et al., 2019) and GraphAF (Shi et al., 2020) proposed normalizing flows for molecule generation. GraphNVP consists of two separate flows, one for modeling the adjacency matrix and a second for modeling the node types. Although allowing parallel generation, the model is sensitive to the node order due to its masking strategy and feature networks in the coupling layer. GraphAF is an autoregressive normalizing flow sampling nodes and edges sequentially but allowing parallel training. However, both flows use standard uniform dequantization to represent the node and edge categories. VAE have also been proposed for latent-based graph generation (Simonovsky and Komodakis, 2018; Ma et al., 2018; Liu et al., 2018; Jin et al., 2018). Although those models can be permutation-invariant, they model a lower bound and do not provide a lossless reconstruction from latent space.

5 Experimental results

We start our experiments with evaluating GraphCNF on two benchmarks for graph generation, namely molecule generation and graph coloring. Further, to test generality we evaluate CNFs on other categorical problems, specifically language and set modeling. For the normalizing flows, we use a sequence of logistic mixture coupling layers (Ho et al., 2019) mapping a mixture of logistic distributions back into a single mode. Before each coupling layer, we include an activation normalization layer and invertible 1x1 convolution (Kingma and Dhariwal, 2018). For reproducibility, we provide all hyperparameter details in Appendix D, and make our code publicly available.

Table 1: Performance on molecule generation trained on Zinc250k (Irwin et al., 2012), calculated on 10k samples and averaged over 4 runs. The standard deviation of those runs can be found in Appendix C. Scores of the baselines are taken from their respective papers.

Method	Validity	Uniqueness	Novelty	Reconstruction	Parallel	Manual Rules
JT-VAE [8]	100%	100%	100%	71%	✗	✓
GraphAF [34]	68%	99.10%	100%	100%	✗	✗
R-VAE [21]	34.9%	100%	–	54.7%	✓	✗
GraphNVP [22]	42.60%	94.80%	100%	100%	✓	✗
GraphCNF	83.41%	99.99%	100%	100%	✓	✗
+ Sub-graphs	96.35%	99.98%	99.98%	100%	✓	✗

5.1 Molecule generation

Modeling and generating graphs is crucial in biology and chemistry for applications such as drug discovery, where molecule generation has emerged as a common benchmark (Jin et al., 2018; Shi et al., 2020). In a molecule graph the nodes are atoms and the edges represent bonds between atoms, both represented by categorical features. Using a dataset of existing molecules, the goal is to learn a distribution of valid molecules as not all possible combinations of atoms and bonds are valid. We perform experiments on the Zinc250k (Irwin et al., 2012) dataset which consists of 250,000 drug-like molecules. The molecules contain up to 38 atoms of 9 different types, with three different bond types possible between the atoms. For comparability, we follow the preprocessing of Shi et al. (2020).

We compare to baselines that consider molecules as graph and not as text representation. As per VAE-based approaches, we consider R-VAE (Ma et al., 2018) and Junction-Tree VAE (JT-VAE) (Jin et al., 2018). R-VAE is a one-shot generation model using regularization to ensure semantic validity. JT-VAE represents a molecule as junction tree of sub-graphs which are obtained from the training dataset. We also compare our model to GraphNVP (Madhawa et al., 2019) and GraphAF (Shi et al., 2020). The models are evaluated by sampling 10,000 examples and measuring the proportion of valid molecules. We also report the proportion of unique molecules and novel samples that are not in the training dataset. These metrics prevent models from memorizing a small subset of graphs. Finally, the reconstruction rate describes whether graphs can be accurately decoded from latent space. Normalizing Flows naturally score 100% due to their invertible mapping, and we achieve the same with our encoding despite no guarantees.

Table 1 shows that GraphCNF generates almost twice as many valid molecules than other one-shot approaches. Yet, the validity and uniqueness stay at almost 100%. Even the autoregressive normalizing flow, GraphAF, is outperformed by GraphCNF by 15%. However, the rules for generating valid molecules can be enforced in autoregressive models by masking out the invalid outputs. This has been the case for JT-VAE as it has been trained with those manual rules, and thus achieves an validity of 100%. Nevertheless, we are mainly interested in the model's capability of learning the rules by itself and being not specific to any application. While GraphNVP and GraphAF sample with a lower standard deviation from the prior to increase validity, we explicitly sample from the original prior to underline that our model covers the whole latent space well. Surprisingly, we found out that most invalid graphs actually consist of two or more that in isolation are valid. This can happen as one-shot models have no guidance for generating a single connected graph. By taking the largest sub-graph of these predictions, we obtain a validity ratio of 96.35% making our model generate almost solely valid molecules *without any manually encoded rules*. We also evaluated our model on the Moses (Polykovskiy et al., 2018) dataset, and achieved similar scores as shown in Appendix C.

5.2 Graph coloring

Graph coloring is a well-known combinatorial problem (Bondy et al., 1976) where for a graph $\mathcal{G}$, each node is assigned to one of K colors. Yet, two adjacent nodes cannot have the same color. Modeling the distribution of valid color assignments to arbitrary graphs is NP-complete. To train models on such a distribution, we generate a dataset of valid graph colorings for randomly sampled graphs. To further investigate the effect of complexity, we create two dataset versions, one with graphs of size $10 \leq |V| \leq 20$ and another with $25 \leq |V| \leq 50$, as larger graphs are commonly harder to solve.

Table 2: Results on the graph coloring problem. Runtimes are measured on a NVIDIA TitanRTX GPU with a batch size of 128. The standard deviations of the results can be found in Appendix D.2.

	$10 \leq \lvert V \rvert \leq 20$			$25 \leq \lvert V \rvert \leq 50$		
Method	**Validity**	**Bits per node**	**Time**	**Validity**	**Bits per node**	**Time**
VAE	44.95%	0.84bpd	0.05s	7.75%	0.64bpd	0.10s
RNN+Smallest_first	76.86%	0.73bpd	0.69s	32.27%	0.50bpd	2.88s
RNN+Random	88.62%	0.70bpd	0.69s	49.28%	0.46bpd	2.88s
RNN+Largest_first	93.41%	0.68bpd	0.69s	**71.32%**	**0.43**bpd	2.88s
GraphCNF	**94.56%**	**0.67**bpd	0.28s	66.80%	0.45bpd	0.54s
− Affine coupling	93.90%	0.69bpd	0.12s	65.78%	0.47bpd	0.35s

Table 3: Results on language modeling. The reconstruction error is shown in brackets.

Model	**Penn Treebank**	**text8**	**Wikitext103**
LSTM baseline	1.28 ±0.01	1.44 ±0.01	4.81 ±0.05
Latent NF - 1 layer	1.30±0.01 (0.01)	1.61±0.02 (0.03)	6.39±0.19 (1.78)
Categorical NF - 1 layer	1.27 ±0.01 (0.00)	1.45 ±0.01 (0.00)	5.43 ±0.09 (0.32)

For graph coloring we rely on GraphCNF and compare to a variational autoencoder and an autoregressive model generating one node at a time. As no edges are being modeled here, we only use the first step of GraphCNF's three-step generation. For all models, we apply the same Graph Attention network (Veličković et al., 2018). As autoregressive models require a manually prescribed node order, we compare the following: a *random* ordering per graph, *largest_first* which is inspired by heuristics of automated theorem provers that start from the nodes with the most connections, and *smallest_first*, where we reverse the order of the previous heuristic. We evaluate the models by measuring the likelihood of color assignments to unseen test graphs in bits per nodes. Secondly, we sample one color assignment per model for each test graph, and report the proportion of valid colorings.

The results in Table 2 show that the node ordering has indeed a significant effect on the autoregressive model's performance. While the smallest_first ordering leads to only 32% valid solutions on the large dataset, reversing the order simplifies the task for the model such that it generates more than twice as many valid color assignments. In contrast, GraphCNF is invariant of the order of nodes. Despite generating all nodes in parallel, it outperforms all node orderings on the small dataset, while being close to the best ordering on the larger dataset. This invariance property is especially beneficial in tasks where an optimal order of nodes is not known, like molecule generation. Although having more parameters, the sampling with GraphCNF is also considerably faster than the autoregressive models. The sampling time can further be improved by replacing the logistic mixture coupling layers by affine ones. Due to the lower complexity, we see a slight decrease in validity and bits per dimension, but can verify that logistic mixture couplings are not crucial for CNFs.

5.3 Language Modeling

To compare CNF with Latent NF, we test both models on language modeling. We experiment with two popular character-level datasets, Penn Treebank (Marcus et al., 1994) and text8 (Mahoney, 2011) with a vocabulary size of $K = 51$ and $K = 27$ respectively. We also test a word-level dataset, Wikitext103 (Merity et al., 2017), with $K = 10,000$ categories, which Discrete NF cannot handle due to its gradient approximations (Tran et al., 2019). We follow the setup of Ziegler and Rush (2019) for the Penn Treebank and train on sequences of 256 tokens for the other two datasets. Both Latent NF and CNF apply a single mixture coupling layer being autoregressive across time and latent dimensions, and differ only in the encoding/decoding strategy. The applied LSTM in the coupling layers is shown as additional RNN baseline in Table 3. Categorical Normalizing Flow performs on par with their autoregressive baseline, only slightly underperforming on Wikitext103 due to using a single flow layer. Latent NF however performs considerably worse on text8 and Wikitext103 due to a non-deterministic decoding and higher fluctuation in loss that we experienced during training. This

Table 4: Results on set modeling. Metric used is bits per categorical variable (dimension).

Model	**Set shuffling**	**Set summation**
Discrete NF (Tran et al., 2019)	3.87 ±0.04	2.51 ±0.00
Variational Dequant. (Ho et al., 2019)	3.01 ±0.02	2.29 ±0.01
Latent NF (Ziegler and Rush, 2019)	**2.78** ±0.00	2.26 ±0.01
CNF + Mixture model	**2.78** ±0.00	**2.24** ±0.00
CNF + Linear flows	**2.78** ±0.00	2.25 ±0.00
CNF + Variational Encoding	2.79 ±0.01	2.25 ±0.01
Optimal	2.77	2.24

shows the importance of a factorized likelihood, and underlines the two benefits of CNFs. Firstly, CNFs are more stable and simpler to train as no loss scheduling is required, and the likelihood remains an exact estimate with the mixture model encoding. Secondly, the encoding of CNFs is much more efficient than Latent NFs. We conclude that CNFs are more widely applicable, and Latent NFs might be the better choice if the decoder can explicitly help, *i.e.*, prior knowledge.

5.4 Set modeling

Finally, we present experiments on sets of categorical variables, for which we create two toy datasets with known data likelihood: *set shuffling* and *set summation*. The goal is to assign high likelihood only to those possible sets that occur in the dataset, and shows how accurately our flows can model an arbitrary, discrete dataset. In set shuffling, we model a set of N categorical variables each having one out of N categories. Each category has to appear exactly once, which leads to $N!$ possible assignments that need to be modeled. In set summation, we again consider a set of size N with N categories, but those categories represent the actual integers $1, 2, ..., N$ and have to sum to an arbitrary number, L. In contrast to set shuffling, the data is ordinal, which we initially expected to help dequantization methods. For both experiments we set $N = 16$ and $L = 42$.

In Table 4, we compare CNFs to applying variational dequantization (Ho et al., 2019), Latent Normalizing Flows (Ziegler and Rush, 2019) and Discrete Normalizing Flows (Tran et al., 2019). The results show that CNFs achieve nearly optimal performance. Although we model a lower bound in continuous space, our flows can indeed model discrete distributions precisely. Interestingly, representing the categories by a simple mixture model is sufficient for achieving these results. We observe the same trend in domains with more complex relations between categories, such as on graphs and language modeling, presumably because both the coupling layers and the prior distribution rest upon logistic distributions as well. Variational dequantization performs worse on the shuffling dataset, while on set summation with ordinal data, the gap to the optimum is smaller. The same holds for Discrete NFs, although it is worth noting that unlike CNFs, optimizing Discrete NFs had issues due to their gradient approximations. Latent Normalizing Flows with the joint decoder achieve similar performance as CNF, which can be attributed to a close-to deterministic decoding. When looking at the encoding space (see Appendix A.4 for visualization), we see that Latent NF has indeed learned a mixture model as well. Hence, the added complexity is not needed on this simple dataset.

6 Conclusion

We present Categorical Normalizing Flows which learn a categorical, discrete distribution by jointly optimizing the representation of categorical data in continuous latent space, and the model likelihood of a normalizing flow. Thereby, we apply variational inference with a factorized posterior to maintain an almost unique decoding while allowing flexible encoding distributions. We find that a plain mixture model is sufficient for modeling discrete distributions accurately while providing an efficient way for encoding and decoding categorical data. Compared to a joint posterior, CNFs are more stable, efficient and do not model an ELBO. Furthermore, GraphCNF, a normalizing flow on graph modeling based on CNFs, outperforms autoregressive and one-shot approaches on molecule generation and graph coloring while being invariant to the node order. This emphasizes the potential of normalizing flows on categorical tasks, especially for such with non-sequential data.

REFERENCES

John Adrian Bondy, Uppaluri Siva Ramachandra Murty, and others. 1976. *Graph theory with applications*, volume 290. Macmillan London.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using Real NVP. In *5th International Conference on Learning Representations, ICLR 2017*, Toulon, France.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415v3*.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. 2019. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2722–2730, Long Beach, California, USA. PMLR.

Emiel Hoogeboom, Taco S. Cohen, and Jakub M. Tomczak. 2020. Learning Discrete Distributions by Dequantization. *arXiv preprint arXiv:2001.11235v1*.

Emiel Hoogeboom, Jorn W. T. Peters, Rianne van den Berg, and Max Welling. 2019. Integer Discrete Flows and Lossless Compression. In *Advances in Neural Information Processing Systems 32*, pages 12134–12144, Vancouver, BC, Canada.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. 2012. ZINC: A Free Tool to Discover Chemistry for Biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768.

Wengong Jin, Regina Barzilay, and Tbmmi Jaakkola. 2018. Junction tree variational autoencoder for molecular graph generation. *35th International Conference on Machine Learning, ICML 2018*, 5:3632–3648.

Sungwon Kim, Sang-Gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. 2019. FloWaveNet : A Generative Flow for Raw Audio. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3370–3378, Long Beach, California, USA. PMLR.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR) 2015*, San Diego, CA, USA.

Diederik P. Kingma and Prafulla Dhariwal. 2018. Glow: Generative Flow with Invertible 1x1 Convolutions. In *Advances in Neural Information Processing Systems*, volume 31, pages 10215–10224. Curran Associates, Inc.

Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. 2016. Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 29:4743–4751.

Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR*, Banff, AB, Canada.

Ivan Kobyzev, Simon Prince, and Marcus A. Brubaker. 2019. Normalizing Flows: Introduction and Ideas. *arXiv preprint arXiv:1908.09257v1*.

Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. 2019. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, volume 2019-Novem, pages 879–885. IEEE Computer Society.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. 2019. Efficient Graph Generation with Graph Recurrent Attention Networks. In H Wallach, H Larochelle, A Beygelzimer, F D'Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 4255–4265. Curran Associates, Inc.

Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. 2019. Graph Normalizing Flows. In *Advances in Neural Information Processing Systems*, pages 13556–13566.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2020. On the Variance of the Adaptive Learning Rate and Beyond. In *International Conference on Learning Representations*.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. 2018. Constrained Graph Variational Autoencoders for Molecule Design. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7795–7804. Curran Associates, Inc.

Tengfei Ma, Jie Chen, and Cao Xiao. 2018. Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7113–7124. Curran Associates, Inc.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. 2019. GraphNVP: An Invertible Flow Model for Generating Molecular Graphs. *arXiv preprint arXiv:1905.11600v1*.

Matt Mahoney. 2011. Large text compression benchmark. *Benchmark published at http://mattmahoney.net/dc/text.html*.

Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*, pages 114–119, Plainsboro, NJ. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer Sentinel Mixture Models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. *preprint (http://www.fit.vutbr.cz/imikolov/rnnlm/char. pdf)*, 8.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H Wallach, H Larochelle, A Beygelzimer, F d' Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, Artur Kadurin, Simon Johansson, Hongming Chen, Sergey Nikolenko, Alan Aspuru-Guzik, and Alex Zhavoronkov. 2018. Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. *arXiv preprint arXiv:1811.12823v3*, pages 1–17.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. 2019. Waveglow: A flow-based generative network for speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE.

Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. 2018. Semi-supervised User Geolocation via Graph Convolutional Networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2009–2019, Melbourne, Australia. Association for Computational Linguistics.

Danilo Jimenez Rezende and Shakir Mohamed. 2015. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, Lille, France.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web*, pages 593–607, Cham. Springer International Publishing.

Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. 2020. GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation. In *International Conference on Learning Representations*.

Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. In *International Conference on Artificial Neural Networks*, volume abs/1802.0, pages 412–422.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Esteban Tabak and Eric Vanden Eijnden. 2010. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233.

Lucas Theis, Aäron Van Den Oord, and Matthias Bethge. 2016. A note on the evaluation of generative models. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Jakub M. Tomczak and Max Welling. 2017. Improving Variational Auto-Encoders using Householder Flow. *arXiv preprint arXiv:1611.09630v4*.

Dustin Tran, Keyon Vafa, Kumar Krishna Agrawal, Laurent Dinh, and Ben Poole. 2019. Discrete Flows: Invertible Generative Models of Discrete Data. In *Advances in Neural Information Processing Systems*, pages 14692–14701.

Benigno Uria, Iain Murray, and Hugo Larochellehugo. 2013. RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, volume 26, pages 2175–2183.

Rianne Van Den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. 2018. Sylvester normalizing flows for variational inference. *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 1:393–402.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations*.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5708–5717, Stockholmsmässan, Stockholm Sweden. PMLR.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv preprint arXiv:1812.08434*.

Zachary M. Ziegler and Alexander M. Rush. 2019. Latent Normalizing Flows for Discrete Sequences. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 7673–7682, Long Beach, California, USA. PMLR.