Semi-Discrete Normalizing Flows through Differentiable Tessellation

Ricky T. Q. Chen
Meta AI
rtqichen@meta.com

Brandon Amos
Meta AI
bamos@meta.com

Maximilian Nickel
Meta AI
maxn@meta.com

Abstract

Mapping between discrete and continuous distributions is a difficult task and many have had to resort to heuristical approaches. We propose a tessellation-based approach that directly learns quantization boundaries in a continuous space, complete with exact likelihood evaluations. This is done through constructing normalizing flows on convex polytopes parameterized using a simple homeomorphism with an efficient log determinant Jacobian. We explore this approach in two application settings, mapping from discrete to continuous and vice versa. Firstly, a *Voronoi dequantization* allows automatically learning quantization boundaries in a multidimensional space. The location of boundaries and distances between regions can encode useful structural relations between the quantized discrete values. Secondly, a *Voronoi mixture model* has near-constant computation cost for likelihood evaluation regardless of the number of mixture components. Empirically, we show improvements over existing methods across a range of structured data modalities.

1 Introduction

Likelihood-based models have seen increasing usage across multiple data modalities. Across a variety of modeling approaches, the family of normalizing flows stands out as a large amount of structure can be incorporated into the model, aiding its usage in modeling a wide variety of domains such as images [7, 23], graphs [29], invariant distributions [2, 25] and molecular structures [42]. However, the majority of works focus on only continuous functions and continuous random variables. This restriction can make it difficult to apply such models to distributions with implicit discrete structures, *e.g.* distributions with discrete symmetries, multimodal distributions, distributions with holes.

In this work, we incorporate discrete structure into standard normalizing flows, while being entirely composable with any other normalizing flow. Specifically, we propose a homeomorphism between the unbounded domain and convex polytopes, which are defined through a learnable tessellation of the domain, *i.e.* a set of disjoint subsets that together fully cover the domain. This homeomorphism is cheap to compute, has a cheap inverse, and results in an efficient formulation of the resulting change in density. In other words, this transformation is highly scalable—at least computationally.

Our method has the potential to be useful for a variety of applications. Firstly, the learned tessellation naturally allows defining a dequantization method for discrete variables. This allows likelihood-based models that normally only act on continuous spaces, such as normalizing flows, to work directly on discrete data in a flexible choice of embedding space with the ability to capture relational structure. Secondly, if we take each convex polytope as the support of a single mixture component, then the full tessellation defines a mixture model with disjoint components. This allows us to scale to mixtures of normalizing flows while retaining the compute cost of a single model. Following semi-discrete optimal transportation [37], which defines couplings between discrete and continuous measures, we refer to our models as semi-discrete normalizing flows that learn transformations between discrete and continuous random variables.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

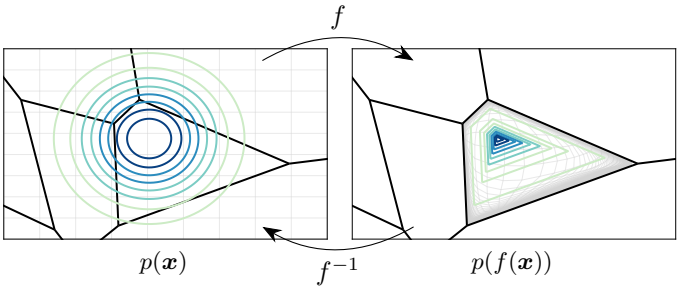


Figure 1: We propose an invertible mapping f between $\mathbb{R}^D$ and a convex polytope, which is parameterized based on a differentiable Voronoi tessellation of $\mathbb{R}^D$. This mapping adds discrete structure into normalizing flows, and its inverse f^{-1} and log determinant Jacobian can both be efficiently computed.

2 Preliminaries

Normalizing Flows This family of generative models typically includes any model that makes use of invertible transformations f to map samples between distributions. The relationship between the original and transformed density functions have a closed form expression,

$$p_z(f(\boldsymbol{x})) = p_x(\boldsymbol{x}) \left| \det \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|^{-1}. \tag{1}$$

For instance, p_z can be a simple base distribution while f is learned so that the resulting p_x is close to some target distribution. Many different choices of f have been discussed in the literature, leading to different trade-offs and use cases [24, 36].

Furthermore, if the domain and codomain of f are different, then p_x and p_z can have different supports, *i.e.* regions where probability is non-zero. Currently, existing designs of f that have this property act independently for each dimension, such as the logit transform [7]. To the best of our knowledge, the use of general support-modifying invertible transformations have not been discussed extensively in the literature.

Dequantization In order to model discrete data with density models, a standard approach is to combined with dequantization methods. These methods provide a correspondence between discrete values and convex subsets, where a discrete variable $y \in \mathcal{Y}$ is randomly placed within a disjoint subset of $\mathbb{R}^D$. There is also a correspondence between the likelihood values of the discrete random variable and the dequantized random variable [44].

Let A_y denote the subset corresponding to y and let $q(\boldsymbol{x}|y)$ be the dequantization model which has a bounded support in A_y. Then any density model $p(\boldsymbol{x})$ with support over $\mathbb{R}^D$ satisfies

$$\log p(y) \geq \mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x}|y)} \left[\log(\mathbb{1}_{[\boldsymbol{x}\in A_y]} p(\boldsymbol{x})) - \log q(\boldsymbol{x}|y) \right] = \mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x}|y)} \left[\log p(\boldsymbol{x}) - \log q(\boldsymbol{x}|y) \right] \tag{2}$$

Thus with an appropriate choice of dequantization, maximizing the likelihood under the density model $p(\boldsymbol{x})$ is equivalent to maximizing the likelihood under the discrete model $p(y)$.

In existing dequantization methods, the value of D and the choice of subsets A_y are entirely dependent on the type of discrete data (ordinal vs non-ordinal) and the number of discrete values $|\mathcal{Y}|$ in the non-ordinal case. Furthermore, the subsets A_y do not interact with one another and are fixed during training. In contrast, we conjecture that important relations between discrete values should be modeled as part of the parameterization of A_y, and that it'd be useful to be able to automatically learn the boundaries of A_y based on gradients.

Disjoint mixture models Building mixture models is one of the simplest methods for creating more flexible distributions from simpler one, and mixture models can typically be shown to be universal density estimators [12]. However, in practice this often requires a large number of mixture components, which quickly becomes computationally expensive. This is because evaluating the likelihood under a mixture model requires evaluating the likelihood of each component. To alleviate this issue, Dinh et al. [8] recently proposed to use components with disjoint support. In particular, let $\{A_k\}_{k=1}^K$ be disjoint subsets of $\mathbb{R}^D$, such that each mixture component is defined on one subset and

has support restricted to that particular subset. The likelihood of the mixture model then simplifies to

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} p(\boldsymbol{x}|k)p(k) = \sum_{k=1}^{K} \mathbb{1}_{[\boldsymbol{x} \in A_k]} p(\boldsymbol{x}|k)p(k) = p(\boldsymbol{x}|k = g(\boldsymbol{x}))p(k = g(\boldsymbol{x})) \tag{3}$$

where $g : \mathbb{R}^D \to \{1, \ldots, K\}$ is a set identification function that satisfies $\boldsymbol{x} \in A_{g(\boldsymbol{x})}$. This framework allows building large mixture models while no longer having a compute cost that scales with K. In contrast to variational approaches with discrete latent variables, the use of disjoint subsets provides an exact (log-)likelihood and not a lower bound.

3 Voronoi Tessellation for Normalizing Flows

We first discuss how we parameterize each subset as a convex polytope, and in the next part, discuss constructing distributions on each convex polytope using a differentiable homeomorphism.

Parameterizing disjoint subsets We separate the domain into subsets through a Voronoi tessellation [48]. This induces a correspondence between each subset and a corresponding *anchor point*, which provides a differentiable parameterization of the tessellation for gradient-based optimization.

Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K\}$ be a set of anchor points in $\mathbb{R}^D$. The *Voronoi cell* for each anchor point is

$$V_k \triangleq \{\boldsymbol{x} \in \mathbb{R}^D : \|\boldsymbol{x_k} - \boldsymbol{x}\| < \|\boldsymbol{x}_i - \boldsymbol{x}\|, i = 1, \ldots, K\}, \tag{4}$$

i.e. it defines a subset containing all points which have the anchor point as their nearest neighbor. Together, these subsets $\{V_k\}_{k=1}^K$ form a *Voronoi tessellation* of $\mathbb{R}^D$. Each subset can equivalently be expressed in the form of a convex polytope,

$$V_k = \{\boldsymbol{x} \in \mathbb{R}^D : \boldsymbol{a}_i^\mathsf{T} \boldsymbol{x} < b_i \ \forall i \neq k\}, \quad \text{where } \boldsymbol{a}_i = 2(\boldsymbol{x}_i - \boldsymbol{x}_k)^\mathsf{T}, b_i = \|\boldsymbol{x}_i\|^2 - \|\boldsymbol{x}_k\|^2. \tag{5}$$

For simplicity, we also include box constraints so that all Voronoi cells are bounded in all directions.

$$V_k = \{\boldsymbol{x} \in \mathbb{R}^D : \underbrace{\boldsymbol{a}_i^\mathsf{T} \boldsymbol{x} < b_i \ \forall i \neq k}_{\text{Voronoi cell}}, \ \underbrace{\mathbf{c}^{(l)} < \boldsymbol{x} < \mathbf{c}^{(r)}}_{\text{box constraints}}\} \tag{6}$$

Thus, the learnable parameters of this Voronoi tessellation are the anchor points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K\}$, and the box constraints $\mathbf{c}_l, \mathbf{c}_r \in \mathbb{R}^D$. These will be trained using gradients from an invertible transformation and resulting distribution defined in each Voronoi cell, which we discuss next.

Invertible mapping onto subset Within the normalizing flow framework, defining a probability distribution on a bounded support is equivalent to defining an invertible transformation that maps from the unbounded support onto the appropriate support. Figure 2 illustrates the transformation of mapping onto a shaded region.

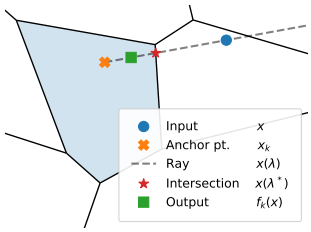


Figure 2: An illustration.

Given a cell V_k for some $k \in \{1, \ldots, K\}$, we construct an invertible mapping $f_k : \mathbb{R}^D \to V_k$ by following 2 steps. Let $\boldsymbol{x} \in \mathbb{R}^D$. First, if $\boldsymbol{x} = \boldsymbol{x_k}$, the anchor point of V_k, we simply set $f_k(\boldsymbol{x}) = \boldsymbol{x}$. Otherwise:

1. Determine where the ray starting from $\boldsymbol{x_k}$ in the direction of $\boldsymbol{x}$ intersects with the boundary of V_k.

First define the direction $\boldsymbol{\delta_k}(\boldsymbol{x}) \triangleq \frac{\boldsymbol{x} - \boldsymbol{x_k}}{\|\boldsymbol{x} - \boldsymbol{x_k}\|}$ and the ray $\boldsymbol{x}(\lambda) \triangleq \boldsymbol{x_k} + \lambda \boldsymbol{\delta_k}(\boldsymbol{x})$, with $\lambda > 0$. Since V_k is convex, we can frame this as a linear programming problem.

$$\max \ \lambda \quad \text{s.t.} \quad \boldsymbol{x}(\lambda) \in \overline{V_k}, \lambda \geq 0 \tag{7}$$

where $\overline{V_k}$ is the closure of V_k. We discuss solving this efficiently in Section 3.1.

Let λ^* be the solution. This solution exists if V_k is bounded, which is always true due to the use of box constraints. Then $\boldsymbol{x}(\lambda^*)$ will be the point of intersection.

This first step solves for the farthest point in the Voronoi cell in the direction of x. Using this knowledge, we can now map all points that lie on this ray onto the Voronoi cell.

2. Apply an invertible transformation such that any point on the ray $\{\boldsymbol{x}(\lambda) : \lambda > 0\}$ is mapped onto the line segment $\{\boldsymbol{x_k} + \alpha(\boldsymbol{x}(\lambda^*) - \boldsymbol{x_k}) : \alpha \in (0, 1)\}$.

There are many possible choices for designing this transformation. An intuitive choice is to use a monotonic transformation of the relative distance from $\boldsymbol{x}$ to the anchor point $\boldsymbol{x_k}$.

$$f_k(\boldsymbol{x}) \triangleq \boldsymbol{x_k} + \alpha_k \left(\frac{\|\boldsymbol{x} - \boldsymbol{x_k}\|}{\|\boldsymbol{x}(\lambda^*) - \boldsymbol{x_k}\|} \right) (\boldsymbol{x}(\lambda^*) - \boldsymbol{x_k}) \tag{8}$$

where α_k is an appropriate *invertible* squashing function from R^+ to $(0, 1)$. In our experiments, we use $\alpha_k(h) = \text{softsign}(\gamma_k h)$ where γ_k is a learned cell-dependent scale. Other choices should also work, such as a monotonic neural network; however, depending on the application, we may need to compute α_k^{-1} for computing the inverse mapping, so it's preferable to have an analytical inverse.

3.1 Remarks and Propositions

Box constraints There can be continuity problems if a Voronoi cell is unbounded, as the solution to Equation (8) does not exist if $x(\lambda^*)$ diverges. Furthermore, when solving Equation (7), it can be difficult to numerically distinguish between an unbounded cell and one whose boundaries are very far away. It is for these reasons that we introduce box constraints (Equation 6) in the formulation of Voronoi cells which allows us to sidestep these issues for now.

Solving for λ^* Equation (7) can be solved numerically, but this approach is prone to numerical errors and requires implicit differentiation through convex optimization solutions [1]. We instead note that the solution of Equation (7) can be expressed in closed form, since it is always going to be the intersection of the ray $\boldsymbol{x}(\lambda)$ with the nearest linear constraint.

Let $\boldsymbol{a}_i^\mathsf{T} \boldsymbol{x} = b_i$ be the plane that represents one of the linear constraints in Equation (6), which are expressed in Equation (4). Let λ_i be the intersection of this plane with the ray, *i.e.* it is the solution to $\boldsymbol{a}_i^\mathsf{T} \boldsymbol{x}(\lambda_i) = b_i$, then the solution is simply the smallest positive λ_i, which satisfies all the linear constraints:

$$\lambda^* = \min\{\lambda_i : \lambda_i > 0\}, \quad \text{where } \lambda_i = \frac{b_i - \boldsymbol{a}_i^\mathsf{T} \boldsymbol{x_k}}{\boldsymbol{a}_i^\mathsf{T} \boldsymbol{\delta_k}(\boldsymbol{x})}. \tag{9}$$

There are a total of $K + 2D - 1$ linear constraints, including the Voronoi cell boundaries and box constraints, which can be computed fully in parallel. This also allows end-to-end differentiation of the mapping f_k via automatic differentiation, providing the ability to learn the parameters of f_k, *i.e.* parameters of α_k and the Voronoi tessellation.

Homeomorphism with continuous density We can show that f_k is a bijection, and both f_k and f_k^{-1} are continuous. This allows us to use f_k within the normalizing flows framework, as a mapping between a distribution p_x defined on R^D and the transformed distribution p_z on V_k. Furthermore, the Jacobian is continuous almost everywhere. Proofs are in Appendix A.

Proposition 1 The mapping $f_k : \mathbb{R}^D \to V_k$ as defined in the 2-step procedure is a homeomorphism.

Proposition 2 If $p_x(x)$ is continuous, then the transformed density $p_z(f_k(x))$ is continuous a.e.

Efficient computation of the log det Jacobian As f_k is a mapping in D dimensions, computing the log determinant Jacobian for likelihood computation can be costly and will scale poorly with D if computed naïvely. Instead, we note that the Jacobian of f_k is highly structured. Intuitively, because f_k depends only on the direction $\boldsymbol{\delta_k}(\boldsymbol{x})$ and the distance away from $\boldsymbol{x_k}$, it only has two degrees of freedom regardless of D. In fact, the Jacobian of f_k can be represented as a rank-2 update on a scaled identity matrix. This allows us to use the matrix determinant lemma to reformulate the log determinant Jacobian in a compute- and memory-efficient form. We summarize this in a proposition.

Proposition 3 Let the transformation $f_k(x)$ and all intermediate quantities be as defined in Section 3 for some given input x. Then the Jacobian factorizes as

$$\frac{\partial f_k(x)}{\partial x} = cI + u_1 v_1^\mathsf{T} + u_2 v_2^\mathsf{T} \tag{10}$$

for some $c \in \mathbb{R}, u_i \in \mathbb{R}^D, v_i \in \mathbb{R}^D$, and its log determinant has the form

$$\log \left| \det \frac{\partial f_k(x)}{\partial x} \right| = \log |1 + w_{11}| + \log \left| 1 + w_{22} - \frac{w_{12} w_{21}}{1 + w_{11}} \right| + D \log c \tag{11}$$

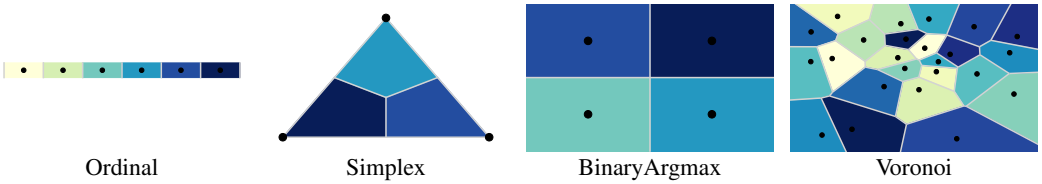


Figure 3: Existing dequantization methods can be seen as special cases of Voronoi dequantization with fixed anchor points. In additional to decoupling the dimension of the embedding space from the number of discrete values, the Voronoi dequantization can learn to model the similarities of discrete values through the positions and boundaries of their Voronoi cells.

where $w_{ij} = c^{-1} v_i^\mathsf{T} u_j$. This expression for the log determinant only requires inner products between vectors in R^D. To reduce notational clutter, exact formulas for c, u_1, u_2, v_1, v_2 are in Appendix A. All vectors used in computing Equation (11) are either readily available as intermediate quantities after computing $f_k(x)$ or are gradients of scalar functions and can be efficiently computed through reverse-mode automatic differentiation. The only operations on these vectors are dot products, and no large D-by-D matrices are ever constructed. Compared to explicitly constructing the Jacobian matrix, this is more efficient in both compute and memory, and can readily scale to large values of D.

4 Application settings

We discuss two applications of our method to likelihood-based modeling of discrete and continuous data. The first is a new approach to dequantization which allows training a model of discrete data using density models that normally only act on continuous variables such as normalizing flows. Compared to existing dequantization methods [7, 18, 38], the Voronoi dequantization is not restricted to fixed dimensions and can benefit from learning similarities between discrete values. In fact, existing approaches can be seen as special cases of a Voronoi dequantization.

The second is a formulation of disjoint mixture models, where each component lies strictly within each subset of a Voronoi tessellation. To the best of our knowledge, disjoint mixture models have not been explored significantly in the literature and have not been successfully applied in more than a couple dimensions. Compared to an existing method [8], the Voronoi mixture model is not restricted to acting individually for each dimension.

Voronoi dequantization Let y be a discrete random variable with finite support $\mathcal{Y}$. Then we can use tessellation to define subsets for dequantizing y as long as there are at least as many Voronoi cells—equivalently, anchors points—as the number of discrete values, *i.e.* $K \geq |\mathcal{Y}|$. By assigning each discrete value to a Voronoi cell, we can then define a dequantization model $q(\boldsymbol{x}|y)$ by first sampling from a base distribution $\boldsymbol{z} \sim q(\boldsymbol{z}|y)$ in $\mathbb{R}^D$ and then applying the mapping $\boldsymbol{x} = f_k(\boldsymbol{z})$ from Section 3 to construct a distribution over V_k. We can then obtain probabilities $q(\boldsymbol{x}|y)$ efficiently and train the dequantization alongside the density model $p(\boldsymbol{x})$ on $\mathbb{R}^D$.

Sampling from the model $p(y|\boldsymbol{x})$ is straightforward and deterministic after sampling $\boldsymbol{x}$. We can write $y = g(\boldsymbol{x})$ where g is the set identification function satisfying $\boldsymbol{x} \in V_{g(\boldsymbol{x})}$. From Equation (4), it is easy to see that $g(\boldsymbol{x})$ is the nearest neighbor operation, $g(\boldsymbol{x}) = \arg\min_k \|\boldsymbol{x} - \boldsymbol{x_k}\|$.

We are free to choose the number of dimensions D, where a smaller D assigns less space for each Voronoi cell induces dequantized distributions that are easier to fit, while a larger D allows the anchor points and Voronoi cells more room to change over the course of training. When the anchor points are fixed to specific positions, we can recover the disjoint subsets used by prior methods as special cases. We illustrate this in Figure 3.

Voronoi mixture models Let $\{V_k\}$ be a Voronoi tessellation of $\mathbb{R}^D$. Then a disjoint mixture model can be constructed by defining distributions on each Voronoi cell. Here we make use of the inverse mapping $f_k^{-1} : V_k \to \mathbb{R}^D$ (discussed in Appendix B) so that we only need to parameterize distributions over $\mathbb{R}^D$. Let $\boldsymbol{x}$ be a point in $\mathbb{R}^D$, our Voronoi mixture model assigns the density

$$\log p_{\text{mix}}(\boldsymbol{x}) = \log p_{\text{comp}}(f^{-1}_{k=g(\boldsymbol{x})}(\boldsymbol{x})|k = g(\boldsymbol{x})) + \log \left|\det \frac{\partial f_k^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right| + \log p(k = g(\boldsymbol{x})) \tag{12}$$

where p_{comp} can be any distribution over $\mathbb{R}^D$, including another disjoint mixture model. This can easily be composed with normalizing flow layers where, in addition to the change of variable due to f_k^{-1}, we also apply the change in density resulting from choosing one out of K components.

5 Related Work

Normalizing flows for discrete data Invertible mappings have been proposed for discrete data, where discrete values are effectively rearranged from a factorized distribution. In order to parameterize the transformation, Hoogeboom et al. [17], van den Berg et al. [47] use quantized ordinal transformations, while Tran et al. [45] takes a more general approach of using modulo addition on one-hot vectors. These approaches suffer from gradient bias due to the need to use discontinuous operations and do not have universality guarantees since it's unclear whether simple rearrangement is sufficient to transform any joint discrete distribution into a factorized distribution. In contrast, the dequantization approach provides a universality guarantee since the lower bound in Equation (2) is tight when $p(\boldsymbol{x})\mathbb{1}_{[\boldsymbol{x}\in A_y]} \propto q(\boldsymbol{x}|y)$, with a proportionality equal to $p(y)$.

Dequantization methods Within the normalizing flows literature, the act of adding noise was originally used for ordinal data as a way to combat numerical issues [7, 46]. Later on, appropriate dequantization approaches have been shown to lower bound the log-likelihood of a discrete model [16, 44]. For non-ordinal data, many works have proposed simplex-based approaches. Early works on relaxations [20, 32] proposed continuous distribution on the simplex that mimic the behaviors of discrete random variables; however, these were only designed for the use with a Gumbel base distribution. Potapczynski et al. [38] extend this to a Gaussian distribution—although it is not hard to see this can work with any base distribution—by designing invertible transformations between R^D and the probability simplex with K vertices, with $D = K - 1$, where K is the number of classes of a discrete random variable.

Intuitively, after fixing one of the logits, the softmax operation is an invertible transformation on the bounded domain $\Delta^{K-1} = \{\boldsymbol{x} \in \mathbb{R}^K : \sum_i^K \boldsymbol{x}_i = 1, \boldsymbol{x}_i > 0\}$. The $(K-1)$-simplex can then be broken into K subsets, each corresponding to a particular discrete value.

$$A_k^{\text{simplex}} = \{\boldsymbol{x} \in \Delta^{K-1} : \boldsymbol{x}_k > \boldsymbol{x}_i \ \forall i \neq k\}. \tag{13}$$

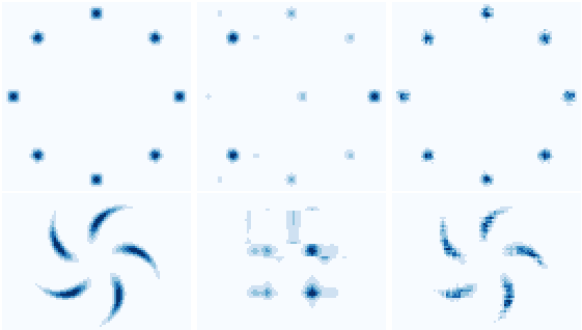
More recently, Hoogeboom et al. [18] proposed ignoring the simplex constraint and simply use

$$A_k^{\text{argmax}} = \{\boldsymbol{x} \in \mathbb{R}^K : \boldsymbol{x}_k > \boldsymbol{x}_i \ \forall i \neq k\}, \tag{14}$$

which effectively increases the number of dimensions by one compared to the simplex approach. However, both approaches force the dimension of the continuous space D to scale with K. In order to make their approach work when K is large, [18] propose binarizing the data as a preprocessing step. In contrast, Voronoi dequantization has full flexibility in choosing D regardless of K.

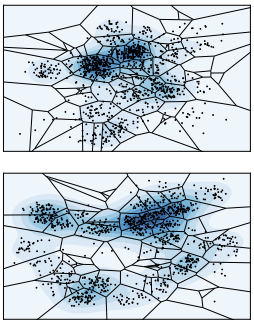
Related models for discrete data Among other related methods, a recent work proposed normalizing flows on truncated supports [43] but had to resort to rejection sampling for training. Furthermore, their subsets are not disjoint by construction. Prior works [28, 31] also proposed removing the constraint that subsets are disjoint, and instead work with general mixture models with unbounded support, relying on the conditional model $p(y|\boldsymbol{x})$ being sufficiently weak so that the task of modeling is forced onto a flow-based prior. They have achieved good performance on a number of tasks, similar to general approaches that combine normalizing flows with variational inference [19, 33, 50]. However, they lose the computational savings and the deterministic decoder $p(y|\boldsymbol{x})$ gained from using disjoint subsets. On the other hand, quantization based on nearest neighbor have been used for learning discrete latent variables [34, 40], but no likelihoods are constructed, the boundaries are not explicitly differentiated through, and the model relies on training with heuristical updates.

Disjoint mixture models The computational savings from using disjoint subsets was pointed out by Dinh et al. [8]. However, their method only works in each dimension individually. They transform samples using a linear spline, which is equivalent to creating subsets based on the knots of the spline and applying a linear transformation within each subset. Furthermore, certain parameterizations of the spline can lead to discontinuous density functions, whereas our disjoint mixture has a density function that is continuous almost everywhere (albeit exactly zero on the boundaries; see Section 7



Target PMF Discrete Flow Voronoi Flow

Figure 4: **Quantized 2D data.** Voronoi dequantization can model complex relations between discrete values. No knowledge of ordering is given to the models.



Dequantized samples

Figure 5: The model learns to cluster discrete values with similar probability values.

for more discussion). The use of monotonic splines have previously been combined with normalizing flows [10], but interestingly, since the splines in Dinh et al. [8] are not enforced to be monotonic, the full transformation is not bijective and acts like a disjoint mixture model. Ultimately, their experiments were restricted to two-dimensional syntheic data sets, and it remained an interesting research question whether disjoint mixture models can be successfully applied in high dimensions.

6 Experiments

We experimentally validate our semi-discrete approach of combining Voronoi tessellation with likelihood-based modeling on a variety of data domains: discrete-valued UCI data sets, itemset modeling, language modeling, and disjoint mixture modeling for continuous UCI data. The goal of these experiments is not to show state-of-the-art results in these domains but to showcase the relative merit of Voronoi tessellation compared to existing methods. For this reason, we just use simple coupling blocks with affine transformations [6, 7] as a base flow model in our experiments, unless stated otherwise. When comparing to Discrete Flows [45], we use their bipartite layers. Details regarding preprocessing for data sets can be found in Appendix C, and detailed experimental setup is in Appendix D. All tables, when shown, display standard deviations across three random seeds.

6.1 Quantized 2D data

We start by considering synthetic distributions over two discrete variables, to qualitatively compare against Discrete Flows [45]. Figure 4 shows the target probability mass functions, which are created by a 2D distribution where each dimension is quantized into 91 discrete values. Though the data is ordinal, no knowledge of ordering is given to the models. We see that Discrete Flows has trouble fitting these, because it is difficult to rearrange the target distribution into a factorized distribution. For our model, we dequantize each discrete variable with a Voronoi tessellation in $\mathbb{R}^2$. We then learn a flow model on the combined $\mathbb{R}^4$, parameterized by multilayer perceptrons (MLPs). In Figure 5 we visualize the learned Voronoi tessellation and samples from our model. The learned tessellation seems to group some of discrete values that occur frequently together, to reduce the number of modes.

6.2 Discrete-valued UCI data

We experiment with complex data sets where each discrete variable can have a varying number of classes. Furthermore, these discrete variables may have hidden semantics. To this end, we use unprocessed data sets from the UCI database [9]. The only pre-processing we perform is to remove variables that only have one discrete value. We then take 80% as train, 10% as validation, and 10% as the test set. Most of these data sets have a combination of both ordinal and non-ordinal variables, and we expect the non-ordinal variables to exhibit relations that are unknown (*e.g.* spatial correlations).

We see that Discrete Flows can be competitive with dequantization approaches, but can also fall short on more challenging data sets such as the `USCensus90`, the largest data set we consider with 2.7

Table 1: **Discrete UCI data sets.** Negative log-likelihood results on the test sets in nats.

Method	**Connect4**	**Forests**	**Mushroom**	**Nursery**	**PokerHands**	**USCensus90**
Voronoi Deq.	12.92$\pm$0.07	14.20$\pm$0.05	9.06$\pm$0.05	9.27$\pm$0.04	19.86$\pm$0.04	24.19$\pm$0.12
Simplex Deq.	13.46$\pm$0.01	16.58$\pm$0.01	9.26$\pm$0.01	9.50$\pm$0.00	19.90$\pm$0.00	28.09$\pm$0.08
BinaryArgmax Deq.	13.71$\pm$0.04	16.73$\pm$0.17	9.53$\pm$0.01	9.49$\pm$0.00	19.90$\pm$0.01	27.23$\pm$0.02
Discrete Flow	19.80$\pm$0.01	21.91$\pm$0.01	22.06$\pm$0.01	9.53$\pm$0.01	19.82$\pm$0.03	55.62$\pm$0.35

Table 2: **Permutation-invariant discrete itemset modeling.**

Model (Dequantization)	**Retail** (nats)	**Accidents** (nats)
CNF (Voronoi)	9.44$\pm$2.34	7.81$\pm$2.84
CNF (Simplex)	24.16$\pm$0.21	19.19$\pm$0.01
CNF (BinaryArgmax)	10.47$\pm$0.42	6.72$\pm$0.23
Determinantal Point Process	20.35$\pm$0.05	15.78$\pm$0.04

Table 3: **Language modeling.**

Dequantization	**text8** (bpc)	**enwik8** (bpc)
Voronoi (D=2)	1.39$\pm$0.01	1.46$\pm$0.01
Voronoi (D=4)	1.37$\pm$0.00	1.41$\pm$0.00
Voronoi (D=6)	1.37$\pm$0.00	1.40$\pm$0.00
Voronoi (D=8)	1.36$\pm$0.00	1.39$\pm$0.01
BinaryArgmax [18]	1.38	1.42
Ordinal [18]	1.43	1.44

million examples and 68 different discrete variables of varying types. For dequantization methods, simplex [38] and binary argmax [18] approaches are mostly on par. We do see a non-negligible gap in performance between these baselines and Voronoi dequantization for most of the data sets, likely due to the ability to learn semantically useful relations between the values of each discrete variable. For instance, the `Connect4` data set contains positions of a two-player board game with the same name, which exhibit complex spatial dependencies between the board pieces, and the `USCensus90` data set contains highly correlated variables and is often used to test clustering algorithms.

6.3 Permutation-invariant itemset modeling

An appeal of using normalizing flows in continuous space is the ability to incorporate invariance to specific group symmetries into the density model. For instance, this can be done by ensuring the ordinary differential equation is equivariant [2, 25, 42] in a continuous normalizing flow [4, 13] framework. Here we focus on invariance with respect to permutations, *i.e.* sets of discrete variables. This invariance cannot be explicitly modeled by Discrete Flows as they require an ordering or a bipartition of discrete variables. We preprocessed a data set of retail market basket data from an anonymous Belgian retail store [3] and a data set of anonymized traffic accident data [11], which contain sets of discrete variables each with 765 and 269 values, respectively. Without the binarization trick, simplex dequantization must use a large embedding dimensions and performs worse than a determinental point process baseline [26]. Meanwhile, our Voronoi dequantization has no problems staying competitive as its embedding space is freely decoupled from the number of discrete values.

6.4 Language modeling

Language modeling a widely used benchmark for discrete models [18, 28, 45]. Here we used the open source code provided by Hoogeboom et al. [18] with the exact same autoregressive flow model and optimizer setups. The only difference is replacing their binary argmax with our Voronoi dequantization. Results are shown in Table 3, where we tried out multiple embedding dimensions D. Generally, we find $D = 2$ to be too low and can stagnate training since the Voronoi cells are more constrained, while larger values of D improve upon argmax dequantization.

6.5 Voronoi mixture models

We test disjoint mixture modeling with Voronoi tessellation on continuous data sets. Figure 6 shows results of training a simple normalizing flow on 2D data, which has trouble completely separating all the modes. Adding in a Voronoi mixture allows us to better fit these multimodal densities.

To test Voronoi mixture models on higher dimensions, we apply our approach to data sets preprocessed by Papamakarios et al. [35], which range from 6 to 63 dimensions. We add a disjoint mixture component after a few layers of normalizing flows, with each component modeled by a normalizing

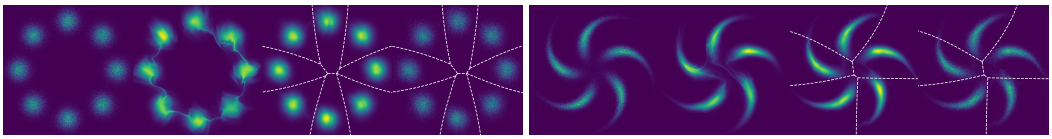


Figure 6: Tessellation is done in a transformed space; nonlinear boundaries are shown.

Table 4: **Disjoint mixture modeling.** NLL on the test sets in nats. *Baseline results from [13, 35].

Method	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Real NVP*	-0.17±0.01	-8.33±0.07	18.71±0.01	13.55±0.26	-153.28±0.89
MAF*	-0.24±0.01	-10.08±0.01	17.73±0.01	12.24±0.22	-154.93±0.14
FFJORD*	-0.46±0.01	-8.59±0.12	14.92±0.08	10.43±0.22	-157.40±0.19
Base Coupling Flow	-0.44±0.01	-11.75±0.02	16.78±0.08	10.87±0.06	-155.14±0.04
Voronoi Disjoint Mixture	-0.52±0.01	-12.63±0.05	16.16±0.01	10.24±0.14	-156.59±0.14

flow conditioned on a embedding of the component index. The number of mixture components ranges from 16 to 64, with complete sweeps and hyperparameters detailed in Appendix D. For comparison, we also show results from the baseline coupling flow that uses the same number of total layers as the disjoint mixture, as well as a strong but computationally costly density model FFJORD [13]. From Table 4, we see that the disjoint mixture model approach allows us to increase complexity quite a bit, with almost no additional cost compared to the baseline flow model.

7 Conclusion and Discussion

We combine Voronoi tessellation with normalizing flows to construct a new invertible transformation that has learnable discrete structure. This acts as a learnable mapping between discrete and continuous distributions. We propose two applications of this method: a Voronoi dequantization that maps discrete values into a learnable convex polytope, and a Voronoi mixture modeling approach that has around the same compute cost as a single component. We showcased the relative merit of our approach across a range of data modalities with varying hidden structure. Below, we discuss some limitations and possible future directions.

Diminishing density on boundaries. The distributions within each Voronoi cell necessarily go to zero on the boundary between cells due to the use of a homeomorphism from an unbounded domain. This is a different and undesirable property as opposed to the partitioning method used by Dinh et al. [8]. However, alternatives could require more compute cost in the form of solving normalization constants. Balancing expressiveness and compute cost is a delicate problem that sits at the core of probabilistic machine learning.

Design of homeomorphisms and tessellations. Our proposed homeomorphism is simple and computationally scalable, but this comes at the cost of smoothness and expressiveness. As depicted in Figure 1, the transformed density has a non-smooth probability density function. This non-smoothness exists when the ray intersects with multiple boundaries. This may make optimization more difficult as the gradients of the log-likelihood objective can be discontinuous. Additionally, our use of Euclidean distance can become problematic in high dimensions, as this metric can cause almost all points to be nearly equidistant, resulting in a behavior where all points lie seemingly very close to the boundary. Improvements on the design of homeomorphisms to bounded domains could help alleviate these problems. Additionally, more flexible tessellations—such as the Laguerre tessellation—and additional concepts from semi-discrete optimal transport [14, 27, 37] may be adapted to improve semi-discrete normalizing flows.

References

[1] Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, Z. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 2019.

[2] Biloš, M. and Günnemann, S. Scalable normalizing flows for permutation invariant densities. In *International Conference on Machine Learning*, pp. 957–967. PMLR, 2021.

[3] Brijs, T., Swinnen, G., Vanhoof, K., and Wets, G. Using association rules for product assortment decisions: A case study. In *Knowledge Discovery and Data Mining*, pp. 254–260, 1999.

[4] Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

[5] Chen, R. T. Q. torchdiffeq, 2018. URL https://github.com/rtqichen/torchdiffeq.

[6] Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[7] Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. *International Conference on Learning Representations*, 2017.

[8] Dinh, L., Sohl-Dickstein, J., Larochelle, H., and Pascanu, R. A rad approach to deep mixture models. *arXiv preprint arXiv:1903.07714*, 2019.

[9] Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[10] Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. *Advances in Neural Information Processing Systems*, 32:7511–7522, 2019.

[11] Geurts, K., Wets, G., Brijs, T., and Vanhoof, K. Profiling high frequency accident locations using association rules. In *Proceedings of the 82nd Annual Transportation Research Board, Washington DC. (USA), January 12-16*, pp. 18pp, 2003.

[12] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

[13] Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. FFJORD: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[14] Gruber, P. M. Optimum quantization and its applications. *Advances in Mathematics*, 186(2): 456–497, 2004.

[15] Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[16] Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pp. 2722–2730. PMLR, 2019.

[17] Hoogeboom, E., Peters, J. W., Berg, R. v. d., and Welling, M. Integer discrete flows and lossless compression. *Advances in Neural Information Processing Systems*, 2019.

[18] Hoogeboom, E., Nielsen, D., Jaini, P., Forré, P., and Welling, M. Argmax flows and multinomial diffusion: Towards non-autoregressive language models. *Advances in Neural Information Processing Systems*, 2021.

[19] Huang, C.-W., Dinh, L., and Courville, A. Augmented normalizing flows: Bridging the gap between generative flows and latent variable models. *arXiv preprint arXiv:2002.07101*, 2020.

[20] Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations*, 2017.

[21] Kim, H., Papamakarios, G., and Mnih, A. The lipschitz constant of self-attention. In *International Conference on Machine Learning*, pp. 5562–5571. PMLR, 2021.

[22] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[23] Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. *Advances in Neural Information Processing Systems*, 2018.

[24] Kobyzev, I., Prince, S., and Brubaker, M. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[25] Köhler, J., Klein, L., and Noé, F. Equivariant flows: exact likelihood generative learning for symmetric densities. In *International Conference on Machine Learning*, pp. 5361–5370. PMLR, 2020.

[26] Kulesza, A. and Taskar, B. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012. ISSN 1935-8237. doi: 10.1561/2200000044.

[27] Lévy, B. A numerical algorithm for l2 semi-discrete optimal transport in 3d. *ESAIM: Mathematical Modelling and Numerical Analysis*, 49(6):1693–1715, 2015.

[28] Lippe, P. and Gavves, E. Categorical normalizing flows via continuous transformations. *International Conference on Learning Representations*, 2021.

[29] Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. *Advances in Neural Information Processing Systems*, 2019.

[30] Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *International Conference on Learning Representations*, 2019.

[31] Ma, X., Zhou, C., Li, X., Neubig, G., and Hovy, E. Flowseq: Non-autoregressive conditional sequence generation with generative flow. *arXiv preprint arXiv:1909.02480*, 2019.

[32] Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations*, 2017.

[33] Nielsen, D., Jaini, P., Hoogeboom, E., Winther, O., and Welling, M. Survae flows: Surjections to bridge the gap between vaes and flows. *Advances in Neural Information Processing Systems*, 33, 2020.

[34] Oord, A. v. d., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[35] Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

[36] Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[37] Peyré, G., Cuturi, M., et al. Computational optimal transport. *Center for Research in Economics and Statistics Working Papers*, (2017-86), 2017.

[38] Potapczynski, A., Loaiza-Ganem, G., and Cunningham, J. P. Invertible gaussian reparameterization: Revisiting the gumbel-softmax. *arXiv preprint arXiv:1912.09588*, 2019.

[39] Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[40] Razavi, A., van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pp. 14866–14876, 2019.

[41] Rudin, W. et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1964.

[42] Satorras, V. G., Hoogeboom, E., Fuchs, F. B., Posner, I., and Welling, M. E(n) equivariant normalizing flows. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021.

[43] Tan, S., Huang, C.-W., Sordoni, A., and Courville, A. Learning to dequantise with truncated flows. In *International Conference on Learning Representations*, 2022.

[44] Theis, L., Oord, A. v. d., and Bethge, M. A note on the evaluation of generative models. *International Conference on Learning Representations*, 2016.

[45] Tran, D., Vafa, K., Agrawal, K., Dinh, L., and Poole, B. Discrete flows: Invertible generative models of discrete data. *Advances in Neural Information Processing Systems*, 32:14719–14728, 2019.

[46] Uria, B., Murray, I., and Larochelle, H. Rnade: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems*, 26, 2013.

[47] van den Berg, R., Gritsenko, A. A., Dehghani, M., Sønderby, C. K., and Salimans, T. Idf++: Analyzing and improving integer discrete flows for lossless compression. In *International Conference on Learning Representations*, 2020.

[48] Voronoi, G. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908(133):97–102, 1908.

[49] Zhang, G., Wang, C., Xu, B., and Grosse, R. Three mechanisms of weight decay regularization. *International Conference on Learning Representations*, 2019.

[50] Ziegler, Z. and Rush, A. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, pp. 7673–7682. PMLR, 2019.

A Proofs

Proposition 1 The mapping $f_k : \mathbb{R}^D \to V_k$ as defined in the 2-step procedure is a homeomorphism.

Proof. Let $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{x_k}$ be a given anchor point corresponding to a Voronoi cell V_k. If $\boldsymbol{x} \neq \boldsymbol{x_k}$, then $\boldsymbol{x}$ is uniquely represented by the tuple (Δ, δ) where $\Delta = \|\boldsymbol{x} - \boldsymbol{x_k}\|$ and $\delta = \frac{\boldsymbol{x}-\boldsymbol{x_k}}{\|\boldsymbol{x}-\boldsymbol{x_k}\|}$, since $\boldsymbol{x} = \boldsymbol{x_k} + \Delta\delta$. Since δ uniques defines the ray $\{\boldsymbol{x_k} + \lambda\delta; \lambda > 0\}$ and $\boldsymbol{x}(\lambda^*)$, then because α_k in Equation (8) is a bijection in Δ, f_k is a bijection. For $\boldsymbol{x} \neq \boldsymbol{x_k}$, the continuity of f_k follows from Rudin et al. [41, Theorem 4.7] since Δ and $\boldsymbol{x}(\lambda^*)$ are continuous in $\boldsymbol{x}$ and α_k is continuous in Δ. Then since $f_k(\boldsymbol{x}) \to \boldsymbol{x_k}$ as $\boldsymbol{x} \to \boldsymbol{x_k}$ from all directions, this justifies the choice of setting $f_k(\boldsymbol{x}) = \boldsymbol{x_k}$ when $\boldsymbol{x} = \boldsymbol{x_k}$. Finally, by the invariance of domain theorem, since V_k is an open set in $\mathbb{R}^D$, f_k is an open map and the inverse f_k^{-1} is continuous, and we can conclude f is a homeomorphism between $\mathbb{R}^D$ and V_k. □

Proposition 2 If $p_x(x)$ is continuous, then the transformed density $p_z(f_k(x))$ is continuous a.e.

Proof. See proof of Proposition 3 below for the form of the Jacobian. For the case where $\boldsymbol{x} \neq \boldsymbol{x_k}$, all quantities in Equations (15) to (22) are continuous with respect to $\boldsymbol{x}$. Hence the Jacobian $\frac{\partial f_k(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is continuous, and since it is always full-rank, then the composition $\left|\det \frac{\partial f_k(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|$ is continuous [41, Theorem 4.7] and so is the product $p_x(x)\left|\det \frac{\partial f_k(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|$ [41, Theorem 4.9]. □

Proposition 3 Let the transformation $f_k(x)$ and all intermediate quantities be as defined in Section 3 for some given input x. Then the Jacobian factorizes as

$$\frac{\partial f_k(x)}{\partial x} = cI + u_1 v_1^\mathsf{T} + u_2 v_2^\mathsf{T} \tag{10}$$

for some $c \in \mathbb{R}, u_i \in \mathbb{R}^D, v_i \in \mathbb{R}^D$, and its log determinant has the form

$$\log\left|\det \frac{\partial f_k(x)}{\partial x}\right| = \log|1 + w_{11}| + \log\left|1 + w_{22} - \frac{w_{12}w_{21}}{1+w_{11}}\right| + D\log c \tag{11}$$

where

$$\Delta \triangleq \|x - x_k\| \tag{15}$$

$$c \triangleq \alpha_k(\tilde{\Delta})\lambda^*\Delta^{-1} \tag{16}$$

$$u_1 \triangleq \left[\alpha_k(\tilde{\Delta})\Delta^{-1} - \frac{\partial \alpha_k(\tilde{\Delta})}{\partial \tilde{\Delta}}\right]\boldsymbol{\delta_k}(\boldsymbol{x}) \tag{17}$$

$$v_1 \triangleq \frac{\partial \lambda^*}{\partial \boldsymbol{\delta_k}(\boldsymbol{x})} \tag{18}$$

$$u_2 \triangleq \left[\left(\frac{\partial \alpha_k(\tilde{\Delta})}{\partial \tilde{\Delta}}\right) - \alpha_k(\tilde{\Delta})\lambda^*\Delta^{-1} - \alpha_k(\tilde{\Delta})\Delta^{-1}\left(\left(\frac{\partial \lambda^*}{\partial \boldsymbol{\delta_k}(\boldsymbol{x})}\right)^\mathsf{T}\boldsymbol{\delta_k}(\boldsymbol{x})\right)\right. \tag{19}$$

$$\left. + \left(\frac{\partial \alpha_k(\tilde{\Delta})}{\partial \tilde{\Delta}}\right)\left(\left(\frac{\partial \lambda^*}{\partial \boldsymbol{\delta_k}(\boldsymbol{x})}\right)^\mathsf{T}\boldsymbol{\delta_k}(\boldsymbol{x})\right)\right]\boldsymbol{\delta_k}(\boldsymbol{x}) \tag{20}$$

$$v_2 \triangleq \boldsymbol{\delta_k}(\boldsymbol{x}) \tag{21}$$

$$w_{ij} \triangleq c^{-1} v_i^\mathsf{T} u_j, \text{ for } i, j \in \{1, 2\} \tag{22}$$

Proof. We first write the Jacobian of f_k in the form of $cI + u_1 v_1^\mathsf{T} + u_2 v_2^\mathsf{T}$ where c is a scalar, and u_1, u_2, v_1, v_2 are vectors of size D. Define the shorthand $\Delta \triangleq \|x - x_k\|$, $\Delta^* \triangleq \|x(\lambda^*) - x_k\|$, and $\tilde{\Delta} \triangleq \Delta/\Delta^*$. To simplify notation, we use the short-hands $\alpha_k = \alpha_k(\tilde{\Delta})$, $\delta_k = \delta_k(x)$. Then the

Jacobian follows

$$\frac{\partial f_k(x)}{\partial x} \tag{23}$$

$$=\frac{\partial}{\partial x}\left(x_k+\alpha_k(x(\lambda^*)-x_k)\right) \tag{24}$$

$$=\left(x(\lambda^*)-x_k\right)\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\frac{\partial\tilde{\Delta}}{\partial x}\right)^{\mathsf{T}}+\alpha_k\left(\frac{\partial x(\lambda^*)}{\partial x}\right) \tag{25}$$

$$=\left(x(\lambda^*)-x_k\right)\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\frac{1}{\Delta^*}\delta_k^{\mathsf{T}}-\frac{\Delta}{(\Delta^*)^2}(x(\lambda^*)-x_k)^{\mathsf{T}}\frac{\partial x(\lambda^*)}{\partial x}\right)+\alpha_k\left(\frac{\partial x(\lambda^*)}{\partial x}\right) \tag{26}$$

$$=\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\delta_k\delta_k^{\mathsf{T}}-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\Delta\delta_k\delta_k^{\mathsf{T}}\frac{\partial x(\lambda^*)}{\partial x}+\alpha_k\left(\frac{\partial x(\lambda^*)}{\partial x}\right) \tag{27}$$

$$=\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\delta_k\delta_k^{\mathsf{T}}+\left(\alpha_k-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\Delta\delta_k\delta_k^{\mathsf{T}}\right)\left(\frac{\partial x(\lambda^*)}{\partial x}\right) \tag{28}$$

$$=\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\delta_k\delta_k^{\mathsf{T}}+\left(\alpha_k-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\Delta\delta_k\delta_k^{\mathsf{T}}\right)\left(\lambda^*\Delta^{-1}I-\lambda^*\Delta^{-1}\delta_k\delta_k^{\mathsf{T}}+\delta_k\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\left(\frac{\partial\delta_k}{\partial x}\right)\right) \tag{29}$$

$$=\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\delta_k\delta_k^{\mathsf{T}}+\left(\alpha_k-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\Delta\delta_k\delta_k^{\mathsf{T}}\right)\left(\lambda^*\Delta^{-1}I-\lambda^*\Delta^{-1}\delta_k\delta_k^{\mathsf{T}}+\Delta^{-1}\delta_k\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\left(I-\delta_k\delta_k^{\mathsf{T}}\right)\right) \tag{30}$$

$$=\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\delta_k\delta_k^{\mathsf{T}}+\left(\alpha_k-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\Delta\delta_k\delta_k^{\mathsf{T}}\right)\left(\lambda^*\Delta^{-1}I-\lambda^*\Delta^{-1}\delta_k\delta_k^{\mathsf{T}}+\Delta^{-1}\delta_k\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}-\Delta^{-1}\left(\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\delta_k\right)\delta_k\delta_k^{\mathsf{T}}\right) \tag{31}$$

$$=\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\delta_k\delta_k^{\mathsf{T}}+\alpha_k\lambda^*\Delta^{-1}I-\alpha_k\lambda^*\Delta^{-1}\delta_k\delta_k^{\mathsf{T}}+\alpha_k\Delta^{-1}\delta_k\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}-\alpha_k\Delta^{-1}\left(\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\delta_k\right)\delta_k\delta_k^{\mathsf{T}} \tag{32}$$

$$-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\lambda^*\delta_k\delta_k^{\mathsf{T}}+\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\lambda^*\left(\delta_k^{\mathsf{T}}\delta_k\right)\delta_k\delta_k^{\mathsf{T}}-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\delta_k^{\mathsf{T}}\delta_k\right)\delta_k\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}} \tag{33}$$

$$+\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\delta_k^{\mathsf{T}}\delta_k\right)\left(\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\delta_k\right)\delta_k\delta_k^{\mathsf{T}} \tag{34}$$

$$=\alpha_k\lambda^*\Delta^{-1}I+\left[\alpha_k\Delta^{-1}-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\delta_k^{\mathsf{T}}\delta_k\right)\right]\delta_k\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}} \tag{35}$$

$$+\left[\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)-\alpha_k\lambda^*\Delta^{-1}-\alpha_k\Delta^{-1}\left(\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\delta_k\right)-\lambda^*\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\delta_k^{\mathsf{T}}\delta_k-1\right)+\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\delta_k^{\mathsf{T}}\delta_k\right)\left(\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\delta_k\right)\right]\delta_k\delta_k^{\mathsf{T}} \tag{36}$$

$$=\alpha_k\lambda^*\Delta^{-1}I+\left[\alpha_k\Delta^{-1}-\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\right]\delta_k\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}} \tag{37}$$

$$+\left[\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)-\alpha_k\lambda^*\Delta^{-1}-\alpha_k\Delta^{-1}\left(\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\delta_k\right)+\left(\frac{\partial\alpha_k}{\partial\tilde{\Delta}}\right)\left(\left(\frac{\partial\lambda^*}{\partial\delta_k}\right)^{\mathsf{T}}\delta_k\right)\right]\delta_k\delta_k^{\mathsf{T}} \tag{38}$$

Now that $\frac{\partial f_k(x)}{\partial x}$ is in the form of $cI+u_1v_1^{\mathsf{T}}+u_2v_2^{\mathsf{T}}$, where

$$c\triangleq\alpha_k(\tilde{\Delta})\lambda^*\Delta^{-1} \tag{39}$$

$$u_1\triangleq\left[\alpha_k(\tilde{\Delta})\Delta^{-1}-\frac{\partial\alpha_k(\tilde{\Delta})}{\partial\tilde{\Delta}}\right]\boldsymbol{\delta_k}(\boldsymbol{x}) \tag{40}$$

$$v_1\triangleq\frac{\partial\lambda^*}{\partial\boldsymbol{\delta_k}(\boldsymbol{x})} \tag{41}$$

$$u_2\triangleq\left[\left(\frac{\partial\alpha_k(\tilde{\Delta})}{\partial\tilde{\Delta}}\right)-\alpha_k(\tilde{\Delta})\lambda^*\Delta^{-1}-\alpha_k(\tilde{\Delta})\Delta^{-1}\left(\left(\frac{\partial\lambda^*}{\partial\boldsymbol{\delta_k}(\boldsymbol{x})}\right)^{\mathsf{T}}\boldsymbol{\delta_k}(\boldsymbol{x})\right)+\left(\frac{\partial\alpha_k(\tilde{\Delta})}{\partial\tilde{\Delta}}\right)\left(\left(\frac{\partial\lambda^*}{\partial\boldsymbol{\delta_k}(\boldsymbol{x})}\right)^{\mathsf{T}}\boldsymbol{\delta_k}(\boldsymbol{x})\right)\right]\boldsymbol{\delta_k}(\boldsymbol{x}) \tag{42}$$

$$v_2\triangleq\boldsymbol{\delta_k}(\boldsymbol{x}) \tag{43}$$

Applying the matrix determinant lemma twice, we can show that

$$\det(cI+u_1v_1^{\mathsf{T}}+u_2v_2^{\mathsf{T}})=(1+v_2^{\mathsf{T}}(cI+u_1v_1^{\mathsf{T}})^{-1}u_2)\det(cI+u_1v_1^{\mathsf{T}}) \tag{44}$$

$$=\left[\left(1+c^{-1}v_2^{\mathsf{T}}u_2-\frac{c^{-2}}{1+c^{-1}v_1^{\mathsf{T}}u_1}\right)v_2^{\mathsf{T}}u_1v_1^{\mathsf{T}}u_2\right](1+c^{-1}v_1^{\mathsf{T}}u_1)c^D \tag{45}$$

We simplify this by defining the scaled dot products,

$$w_{ij}\triangleq c^{-1}v_i^{\mathsf{T}}u_j,\ \text{for } i,j\in\{1,2\}. \tag{46}$$

Then

$$\log\left|\det\frac{\partial f_k(x)}{\partial x}\right| = \log|1+w_{11}| + \log\left|1+w_{22}-\frac{w_{12}w_{21}}{1+w_{11}}\right| + D\log c \tag{47}$$

□

Intermediate steps above used the following gradient identities.

$$\begin{aligned}\frac{\partial \boldsymbol{\delta_k}(\boldsymbol{x})}{\partial x} =& \frac{\partial}{\partial x}(x-x_k)\Delta^{-1}\\ =& \Delta^{-1}I + (x-x_k)\left(\frac{\partial}{\partial x}\left(\Delta^2\right)^{-\frac{1}{2}}\right)^{\mathsf{T}}\\ =& \Delta^{-1}I + (x-x_k)\left(\frac{-1}{2\Delta^3}\right)\left(\frac{\partial\Delta^2}{\partial x}\right)^{\mathsf{T}}\\ =& \Delta^{-1}I + (x-x_k)\left(\frac{-1}{2\Delta^3}\right)2\left(x-x_k\right)^{\mathsf{T}}\\ =& \Delta^{-1}\left(I-\boldsymbol{\delta_k}(\boldsymbol{x})\boldsymbol{\delta_k}(\boldsymbol{x})^{\mathsf{T}}\right)\end{aligned} \tag{48}$$

$$\frac{\partial\Delta}{\partial x} = \frac{\partial\Delta}{\partial x} = \frac{\partial}{\partial x}\left(\Delta^2\right)^{\frac{1}{2}} = \frac{1}{\Delta}(x-x_k) = \boldsymbol{\delta_k}(\boldsymbol{x}) \tag{49}$$

B The inverse mapping

We use the inverse mapping $f_k^{-1}: V_k \to \mathbb{R}^D$ for training disjoint mixture models, so we next describe how to compute this. Let $\boldsymbol{z} = f_k(\boldsymbol{x})$.

Conveniently, since both $\boldsymbol{x}$ and $\boldsymbol{z}$ lie on the ray $\{\boldsymbol{x}(\lambda) : \lambda > 0\}$, we know $\boldsymbol{\delta_k}(\boldsymbol{x}) = \boldsymbol{\delta_k}(\boldsymbol{z})$. So the first step is the same as the forward procedure: we solve for λ^* and $\boldsymbol{x}(\lambda^*)$. Following this, we then recover $\boldsymbol{x}$ by inverting Step 2 of the forward procedure.

This inverse transformation is given by

$$\tilde{\alpha} = \frac{\boldsymbol{z}-\boldsymbol{x_k}}{\boldsymbol{x}(\lambda^*)-\boldsymbol{x_k}} \tag{50}$$

$$\tilde{\Delta} = \alpha_k^{-1}(\tilde{\alpha}_1) \tag{51}$$

$$\Delta = \tilde{\Delta}\,\|\boldsymbol{x}(\lambda^*)-\boldsymbol{x_k}\| \tag{52}$$

$$x = \Delta\boldsymbol{\delta_k}(\boldsymbol{z}) + \boldsymbol{x_k} \tag{53}$$

Equation (50) is an element-wise division. Since $\tilde{\alpha}$ will the same in all dimensions, we can simply pick a dimension in Equation (51). In our experiments, the inverse α_k^{-1} can be computed analytically, though since it is just a scalar function, simple methods like bisection can also work when the inverse is not known analytically. Lastly, Equation (53) follows from the observation that $\boldsymbol{\delta_k}(\boldsymbol{x}) = \boldsymbol{\delta_k}(\boldsymbol{z})$.

Log determinant of the inverse. We can also use Proposition 3 to compute the log determinant of the inverse transform without needing to recompute $f_k(x)$. The only difference is a sign: $\log\left|\det\frac{\partial f_k^{-1}(z)}{\partial z}\right| = -\log\left|\det\frac{\partial f_k(x)}{\partial x}\right|$. The required quantities, Δ, $\boldsymbol{x}(\lambda^*)$, and $\boldsymbol{\delta_k}(\boldsymbol{x})$, are readily available after computing $x = f_k^{-1}(z)$. The gradients with respect to quantities of x can be expressed using gradients with respect to quantities of z,

$$\frac{\partial\alpha_k(\Delta)}{\partial\Delta} = \left(\frac{\partial\alpha_k^{-1}(\tilde{\alpha})}{\partial\tilde{\alpha}}\right)^{-1} \qquad \text{and} \qquad \frac{\partial\lambda^*}{\partial\boldsymbol{\delta_k}(\boldsymbol{x})} = \frac{\partial\lambda^*}{\partial\boldsymbol{\delta_k}(\boldsymbol{z})}, \tag{54}$$

which are accessible through automatic differentiation.

C Data sets

C.1 UCI Data sets

The main preprocessing we did was to (i) remove the "label" attribute from each data set, and (ii) remove attributes that only ever take on one value. Apart from this, the `USCensus90` data set contains a unique identifier for each row, which was removed. Descriptions for all data set are below.

Connect4 [webpage] This data set contains all legal 8-ply positions in the game of Connect Four in which neither player has won yet, and in which the next move is not forced. The original task was to predict which player would win, which has been removed during preprocessing. There are a total **42** discrete variables (one for each location on the board), each with **3** possible discrete values (taken by player 1, taken by player 2, blank). This data set was randomly split into 54045 training examples, 6755 validation examples, and 6757 test examples.

Forests [webpage] This data set contains cartographic variables regarding forests including four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances. The original task was to predict the forest cover type, which has been removed during preprocessing. There are a total of **54** discrete variables, with **10** being the highest number of discrete values. This data set was randomly split into 464809 training examples, 58101 validation examples, and 58102 test examples.

Mushroom [webpage] This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. The original task was to predict whether each species is edible, which has been removed during preprocessing. There are a total of **21** discrete variables, with **12** being the highest number of discrete values. This data set was randomly split into 6499 training examples, 812 validation examples, and 813 test examples.

Nursery [webpage] This data set contains attributes of applicants to nursery schools, during a period when there was excessive enrollment to these schools in Ljubljana, Slovenia, and the rejected applications frequently needed an objective explanation. All data have been completely anonymized. The original task was to predict whether an applicant would be recommended for acceptance by hierarchical decision model, which has been removed during preprocessing. There are a total of **8** discrete variables, with **5** being the highest number of discrete values. This data set was randomly split into 10367 training examples, 1296 validation examples, and 1297 test examples.

PokerHands [webpage] This data set contains poker hands consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one Class attribute that describes the "poker hand". The original task was to predict the poker hand class (pairs, full house, royal flush, etc.), which has been removed during preprocessing. There are a total of **10** discrete variables, with **13** being the highest number of discrete values. This data set was randomly split into 820008 training examples, 102501 validation examples, and 102501 test examples.

USCensus90 [webpage] This data set contains a portion of the data collected as part of the 1990 census in the United States, with the data completely anonymized. There are a total of **68** discrete variables, with **18** being the highest number of discrete values. This data set was randomly split into 2212456 training examples, 122914 validation examples, and 122915 test examples.

C.2 Itemset Data sets

These data sets were taken from the Frequent Itemset Mining Data set Repository [webpage]. Each row is interpreted as a set of items with no emphasis on the ordering of items.

Retail [3] This data set contains anonymized retail market basket data from an anonymous Belgian retail store. We first removed rows with less than 4 items, then randomly sampled a subset of 4 items for every row. Items that appear in less 300 rows were dropped from the data set. The final preprocessed data set contains **765** distinct items. This data set was randomly split into 24280 training examples, 3035 validation examples, and 3036 test examples.

Accidents [11] This data set contains contains anonymized traffic accident data. Data on traffic accidents are obtained from the National Institute of Statistics (NIS) for the region of Flanders (Belgium) for the period 1991-2000. We first removed rows with less than 4 items, then randomly sampled a subset of 4 items for every row. This subsampling occurred 10 times if a row has 16 or more items, 5 times if the row has 8 to 15 items, and once if the row has 4-7 items. Items that appear in less 300 rows were dropped from the data set. The final preprocessed data set contains **213** distinct items. This data set was randomly split into 270129 training examples, 33766 validation examples, and 33767 test examples.

D Experimental Details

All experiments were run on a single NVIDIA V100 GPU. Detailed hyperparameter sweeps are below. We used the validation set to choose hyperparameters as well as to perform early stopping.

2D synthetic data sets Continuous data sets were quantized into 91 bins for each coordinate. For Voronoi dequantization, we dequantized each coordinate into an embedding space of 2 dimensions, with 91 Voronoi cells. The dequantization model is parameterized by 4 layers of coupling blocks, each with a 2 hidden layer MLP with 256 hidden units each, where the Swish activation function was used [39]. The flow model is similarly parameterized but with 16 layers of coupling blocks. Each block alternated between 4 different partitioning schemes: maksing out the first half, masking out the second half, masking out the odd indices, and masking out the even indices. We trained with the `Adam` optimizer [22] with a learning rate of `1e-3`.

Discrete-valued UCI data sets For Voronoi dequantization, we dequantized each coordinate into an embedding space of 4 or 6 dimensions, with the number of Voronoi cells set to the highest number of discrete values over all discrete variables. The dequantization model is parameterized by 4 layers of coupling blocks, each with a 2 hidden layer MLP with 256, 512, or 1024 hidden units each, where the Swish activation function was used [39]. The flow model is similarly parameterized but with 16 or 32 layers of coupling blocks. Each block alternated between 4 different partitioning schemes: maksing out the first half, masking out the second half, masking out the odd indices, and masking out the even indices. We trained with the `Adam` optimizer [22] with a learning rate sweep over {`1e-3`, `5e-4`, `1e-4`}.

Itemset data sets For Voronoi dequantization, we dequantized each coordinate into an embedding space of 6 dimensions, with the number of Voronoi cells set to the number of items in the data set. We used a continuous normalizing flow (CNF) with the ordinary differential equation (ODE) defined using a Transformer archiecture and a L_2-distance based multihead attention layer [21] and the GeLU activation function [15]. No positional embeddings were provided to the model to ensure the model is equivariant to permutations. We composed 12 CNF layers, each defined using a Transformer model that has 2 or 3 layers of alternating multihead attention and fully connected residual connections. To solve the ODE and train our model, we used the `dopri5` solver from the torchdiffeq library [5] with `atol=rtol=1e-5`. We trained with the `AdamW` optimizer [30, 49] with a learning rate of `1e-3` and weight decay of `1e-6`. For the Voronoi dequantization, we set D=6 for both data sets, though it may be possible to improve performance by tuning D.

Character-level language modeling We used the provided hyperparameters from the open source repository [URL]. Some parts of code had to be adapted for our usage, but model architecture and optimizer remained largely the same.

Continuous-valued UCI data sets Disjoint mixture modeling is the inverse of a dequantization method. We use a flow model which maps the data nonlinearly, $z = f(x)$. Then a Voronoi mixture model partitions the space and assigns each z an index through the set identification function $k = g(z)$. The probabilities $p(k)$ are parameterized through a softmax and learned. We then apply the inverse transformation from Section 3 to map from V_k to $\mathbb{R}^D$. A conditional normalizing flow is then used to model each mixture model $p(z|k)$. We set the base distribution to be a Gaussian with standard deviation 0.2, as this helps concentrate the density around the anchor points at initialization. We tuned the number of flow layers before and after the mixture layer, with the total number of layers in $\{16, 32, 48, 64\}$. We also tuned the number of mixtures in $\{8, 16, 32, 64, 128\}$. Each coupling block

uses a neural network with 3 hidden layers of dimension 64 with either the GeLU or Swish [39] activation function. We trained with a batch size of 1048 and the `Adam` optimizer [22] with a learning rate sweep over {`1e-3`, `5e-3`}.