Gaussianization Flows

Chenlin Meng* **Yang Song*** **Jiaming Song** **Stefano Ermon**
Computer Science Department, Stanford University

Abstract

Iterative Gaussianization is a fixed-point iteration procedure that can transform any continuous random vector into a Gaussian one. Based on iterative Gaussianization, we propose a new type of normalizing flow model that enables both efficient computation of likelihoods and efficient inversion for sample generation. We demonstrate that these models, named *Gaussianization flows*, are universal approximators for continuous probability distributions under some regularity conditions. Because of this guaranteed expressivity, they can capture multimodal target distributions without compromising the efficiency of sample generation. Experimentally, we show that Gaussianization flows achieve better or comparable performance on several tabular datasets compared to other efficiently invertible flow models such as Real NVP, Glow and FFJORD. In particular, Gaussianization flows are easier to initialize, demonstrate better robustness with respect to different transformations of the training data, and generalize better on small training sets.

1 INTRODUCTION

Maximum likelihood is a widely adopted approach for density estimation. However, for very expressive probabilistic models, *e.g.*, those parameterized by deep neural networks, evaluating likelihood can be intractable. Several special architectures have been proposed to build probabilistic models with tractable likelihoods. One such family of models is *normalizing flows* (Rezende and Mohamed, 2015; Dinh et al., 2014, 2015). These models learn a bijective mapping $\mathbf{T}$ that pushes forward the data distribution to a simple target distribution (typically Gaussian or uniform) such that the log determinant of the transformation's Jacobian ($\log|\det J_{\mathbf{T}}|$) is efficient to compute. The corresponding likelihood can then be efficiently computed via the change of variables formula, enabling efficient training via maximum likelihood.

Given a density model, it is often desirable to generate samples from it in an efficient way. This requires an additional property for normalizing flow models: the inverse of $\mathbf{T}$ must also be easy to compute. Unfortunately, even though flow models are invertible by construction, they are not always efficiently invertible in practice. For example, models like MAF (Papamakarios et al., 2017), NAF (Huang et al., 2018), Block-NAF (De Cao et al., 2019) all need D times more computation for inversion than for likelihood evaluation, where D is the data dimension. Continuous flow models, such as Neural ODE (Chen et al., 2018) and FFJORD (Grathwohl et al., 2018), take roughly the same time for inversion and likelihood evaluation, but both directions involve slow numerical integration procedures. Models based on coupling layers, *e.g.*, Real NVP (Dinh et al., 2016) and Glow (Kingma and Dhariwal, 2018), have efficient procedures for both inversion and likelihood computation, yet it is unclear whether their architectures are sufficiently expressive to capture all distributions.

To explore different flow architectures that are expressive and permit efficient sampling, we draw inspiration from iterative Gaussianization. First proposed in Chen and Gopinath (2000), it is an iterative approach to transform the data distribution to a standard (multivariate) Gaussian distribution. Specifically, we first transform each data point with a linear mapping (typically an orthogonal matrix computed by ICA or PCA), and then individually "Gaussianize" the marginal distributions of each data dimension. This is achieved by estimating each univariate CDF, mapping each data dimension to a uniform random variable, and then transforming it to a Gaussian by CDF inversion. Intuitively, the linear mapping in Gaussianization amounts to finding a specific direction where the marginals of the data distribution are as "non-Gaussian" as possible; this "non-Gaussianity" is reduced by the subsequent Gaussianization step performed for each marginal dis-

*Joint first authors. Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

tribution. As proved in Chen and Gopinath (2000), the transformed data distribution converges to a standard normal if this procedure is repeated a sufficiently large number of times (under some conditions). Though theoretically satisfying, this method has many limitations in practice. First, Gaussianizing marginal distributions is practically difficult, even in the univariate case, because non-parametric methods for CDF estimation (such as kernel density estimation) can be inaccurate and hard to tune. Second, finding optimal linear mappings such that the marginal distributions are "non-Gaussian" is challenging and traditional methods such as linear ICA do not have closed-form solutions and can be very slow to run for large scale datasets.

To mitigate these limitations while preserving theoretical guarantees, we propose to parameterize the Gaussianization procedure to make it jointly trainable, in lieu of following the original iterative refining approach. This results in a new family of flow models named *Gaussianization flows*. More specifically, we parameterize the linear mapping by stacking several Householder transformations with learnable parameters. After this linear mapping, we parameterize an element-wise non-linear transformation by composing the inverse Gaussian CDF with the CDF of a trainable mixture of logistic distributions. Combining the linear mapping and element-wise non-linear transformation, we get a differentiable *Gaussianization module* whose Jacobian determinant is available in closed-form, and inversion is easy to compute. We can stack several Gaussianization modules to form a Gaussianization flow model which is also easy to invert.

We can show that Gaussianization flows are universal approximators when the model is sufficiently wide and deep, meaning that the model architecture is theoretically expressive enough to transform any data distribution with strictly positive density to a Gaussian distribution (under some regularity conditions). Due to the connection between Gaussianization flows and iterative Gaussianization, the layers of Gaussianization flows have a natural interpretation. For example, the mixture of logistics in a Gaussianization flow should ideally capture the marginal distribution obtained after applying the Householder layer. We can therefore initialize the parameters of the mixture of logistic used for Gaussianization using a kernel density estimator with logistic kernels for better training. Because of the non-parametric nature of kernel density estimation, this intialization is more adaptive, providing some robustness with respect to re-parameterizations of the data.

In our experiments, we demonstrate that Gaussianization flows achieve better or comparable performance on density estimation for tabular data, compared to some efficient invertible baselines such as Real NVP, Glow and FFJORD. In particular, we achieve better performance when the number of training data points is limited, and our models show more robustness to reparameterizations of the data.

2 BACKGROUND

2.1 Density Estimation with Flow Models

Let $\mathcal{D} = \{\mathbf{x}_j \in \mathbb{R}^D\}_{j=1}^N$ be a dataset of continuous observations which are i.i.d. samples from an unknown continuous data distribution (denoted as p_{data}). Given this dataset $\mathcal{D}$, the goal of density estimation is to approximate p_{data} with a probabilistic model parameterized by $\boldsymbol{\theta}$ (denoted as $p_{\boldsymbol{\theta}}$). Specifically, we learn an invertible model $\mathbf{T}_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^D$, which performs a bijective, differentiable transformation of $\mathbf{x}$ to $\mathbf{z} = \mathbf{T}_{\boldsymbol{\theta}}(\mathbf{x})$. Using the change of variables formula,

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = p_z(\mathbf{T}_{\boldsymbol{\theta}}(\mathbf{x})) \left| \det \frac{\partial \mathbf{T}_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \mathbf{x}} \right| = p_z(\mathbf{z}) |\det J_{\mathbf{T}_{\boldsymbol{\theta}}}(\mathbf{x})|,$$

where $\det J_{\mathbf{T}_{\boldsymbol{\theta}}}(\mathbf{x})$ denotes the determinant of the Jacobian matrix evaluated at $\mathbf{x}$, and $p_z(\mathbf{z})$ is a simple fixed distribution with tractable density (*e.g.* the multivariate standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$). Note that in order to evaluate the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$, the determinant of Jacobian $\det J_{\mathbf{T}_{\boldsymbol{\theta}}}(\mathbf{x})$ must be easy to compute. Models with this property are named *normalizing flow models* (Rezende and Mohamed, 2015).

Multiple flow models $\mathbf{T}_1, \mathbf{T}_2, \cdots, \mathbf{T}_L$ can be stacked together to yield a deeper and more expressive model $\mathbf{T} = \mathbf{T}_1 \circ \mathbf{T}_2 \circ \cdots \circ \mathbf{T}_L$. Since $\mathbf{T}^{-1} = \mathbf{T}_L^{-1} \circ \mathbf{T}_{L-1}^{-1} \circ \cdots \circ \mathbf{T}_1^{-1}$, and $\det J_{\mathbf{T}} = \det J_{\mathbf{T}_1} \det J_{\mathbf{T}_2} \cdots \det J_{\mathbf{T}_L}$, as long as each component $\mathbf{T}_i$ is invertible and has tractable determinant of Jacobian, the combined model $\mathbf{T}$ also shares such properties.

2.2 Iterative Gaussianization

Training a flow model with maximum likelihood amounts to solving

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[-\log p_{\boldsymbol{\theta}}(\mathbf{x})] \\ &\qquad = \min_{\boldsymbol{\theta}} D_{\text{KL}}\left(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})\right) + \text{const.} \quad (1) \end{aligned}$$

When $p_{\boldsymbol{\theta}}(\mathbf{x})$ is the likelihood of a flow model $\mathbf{T}_{\boldsymbol{\theta}}(\mathbf{x})$ given by Eq. (1), we can transform the above objective using the fact that KL divergence is invariant to bijective mappings of random variables, which gives us

$$\begin{aligned} &\min_{\boldsymbol{\theta}} D_{\text{KL}}\left(p_{\text{data}}(\mathbf{x}) \parallel p_{\boldsymbol{\theta}}(\mathbf{x})\right) + \text{const} \\ &\qquad = \min_{\boldsymbol{\theta}} D_{\text{KL}}\left(p_{\mathbf{T}_{\boldsymbol{\theta}}}(\mathbf{z}) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})\right) + \text{const}, \quad (2) \end{aligned}$$

where $p_{\mathbf{T}_{\boldsymbol{\theta}}}$ denotes the distribution of $\mathbf{z} = \mathbf{T}_{\boldsymbol{\theta}}(\mathbf{x})$, when $\mathbf{x}$ is sampled from $p_{\boldsymbol{\theta}}(\mathbf{x})$. Intuitively, Eq. (2) means that training a flow model with maximum likelihood is equivalent to finding an invertible transformation to warp the data distribution to a multivariate standard normal distribution. This task is well-known as *Gaussianization* (Chen and Gopinath, 2000).

For one-dimensional (univariate) data $x \sim p_{\text{data}}(x)$, one could perform Gaussianzation by estimating its cumulative density function (CDF, *e.g.* using kernel density estimation) and applying the inverse Gaussian CDF. To see this, let Φ be the CDF of the standard normal distribution, and F_{data} be the CDF of the data distribution, we can transform any random variable $x \sim p_{\text{data}}$ to a Gaussian random variable z by $z = \Phi^{-1} \circ F_{\text{data}}(x)$.

For high dimensional data, one key observation is that the KL divergence between a distribution $p(\mathbf{x})$ and a multivariate standard Gaussian distribution can be decomposed as follows (Chen and Gopinath, 2000):

$$D_{\text{KL}}\left(p(\mathbf{x}) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})\right) \triangleq J(\mathbf{x}) = I(\mathbf{x}) + J_m(\mathbf{x}) \quad (3)$$

where $I(\mathbf{x})$ is the multi-information that measures the statistical dependence among components of $\mathbf{x}$:

$$I(\mathbf{x}) = D_{\text{KL}}\left(p(\mathbf{x}) \,\middle\|\, \prod_i^D p_i(x^{(i)})\right), \quad (4)$$

and $J_m(\mathbf{x})$ is the sum of KL divergences between the marginal distributions and univariate standard normal distributions:

$$J_m(\mathbf{x}) = \sum_{i=1}^{D} D_{\text{KL}}\left(p_i(x^{(i)}) \,\middle\|\, \mathcal{N}(0,1)\right). \quad (5)$$

Here we represent $\mathbf{x} = (x^{(1)}, x^{(2)}, \cdots, x^{(D)})^\intercal$, and let $p_i(x^{(i)})$ be the marginal distribution of $p(\mathbf{x})$. Intuitively, to transform the data distribution into a multivariate unit Gaussian, we need to make each dimension independent ($I(\mathbf{x}) = 0$), and each marginal distribution univariate standard normal ($J_m(\mathbf{x}) = 0$).

Based on the decomposition Eq. (3), a particular iterative Gaussianization (Chen and Gopinath, 2000) approach—Rotation-Based Iterative Gaussianization (RBIG, Laparra et al. (2011))—alternates between applying one-dimensional Gaussianization and rotations to the data. Specifically, RBIG estimates the marginal distribution corresponding to each dimension of the data distribution, and performs one-dimensional Gaussianization of all marginal distributions. Then, RBIG applies a rotation matrix to the transformed data.

The rationale behind RBIG is that dimension-wise Gaussianization will decrease $J_m(\mathbf{x})$ and leave $I(\mathbf{x})$ invariant, due to the fact $I(\mathbf{x})$ is invariant under dimension-wise invertible transformations (Laparra et al., 2011), whereas applying rotation to $p(\mathbf{x})$ will not modify the overall KL divergence objective $I(\mathbf{x})+J_m(\mathbf{x})$ since KL is invariant under bijective transformations (rotation in particular) and $\mathcal{N}(\mathbf{0}, \mathbf{I})$ is rotationally invariant. Therefore, $D_{\text{KL}}\left(p(\mathbf{x}) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})\right)$ will not increase (typically decreases) at each RBIG iteration. To improve the performance of RBIG, one could consider rotation operators that make $J_m(\mathbf{x})$ as large as possible, so that the subsequent marginal Gaussianization step removes $J_m(\mathbf{x})$ and results in a large decrease in $D_{\text{KL}}\left(p(\mathbf{x}) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})\right)$. Popular choices of rotation matrices include random matrices and those computed by independent component analysis (ICA) and principal component analysis (PCA). However, all three candidates are less than desirable. For random rotations and PCA, the procedure could require many RBIG steps to converge (Laparra et al., 2011). ICA, on the other hand, is optimal yet does not have closed-form solutions and is expensive to compute in practice.

3 METHOD

While iterative Gaussianization possesses the ability to transform a complex distribution to standard normal, density estimation with iterative Gaussianization is still difficult, because of the following challenges:

- One-dimensional (1D) Gaussianization is challenging for certain data distributions;
- Finding optimal rotation matrices is challenging (as in the case of ICA rotation matrices, which have no closed form solution).

In this section, we address these challenges with a new type of invertible flow model based on the iterative Gaussianization (RBIG) method, named *Gaussianization Flows* (GF). Specifically, GF improves the two components of RBIG where we replace 1D Gaussianization with a *trainable kernel layer* and a fixed rotation matrix with a *trainable orthogonal matrix layer*.

3.1 Building Trainable Kernel Layers

Marginal Gaussianization plays a crucial role in RBIG since it reduces the objective value $J_m(\mathbf{x})$ in Eq. (5) and is the only procedure that decreases the KL objective in Eq. (3) (rotation does not change the KL divergence because KL is invariant to bijective mappings, however, it enables progress in the next iteration). For a set of 1D scalars $\{x_j\}_{j=1}^M$, one could perform Gaussianization by first estimating a CDF (denoted as $F_{\text{data}}(x)$), and then applying the transformation $\phi : x \mapsto \Phi^{-1} \circ F_{\text{data}}(x)$ where Φ is the CDF for a 1D standard Gaussian.

One approach to estimate the CDF is via 1D density estimation, where the CDF can be computed from the PDF by taking the integral. As we are assuming the underlying data distribution is continuous, we can naturally employ kernel density estimation (KDE) methods to fit the data PDF, and then obtain the CDF by integrating out the kernels in closed-form. However, there are two shortcomings of KDE for large-scale density estimation. Firstly, the complexity of computing the KDE for each sample scales quadratically with the number of samples, making it prohibitive for larger batches/datasets; secondly, the performance of KDE largely depends on the sample size (Parzen, 1962; Devroye and Wagner, 1979) and bandwidth selection (Sheather, 2004), yet optimal bandwidths are difficult to obtain even with good bandwidth selection heuristics (Scott and Sheather, 1985).

To alleviate the limitations of existing non-parametric KDE approaches, we propose to learn a "parameterized KDE" for each data dimension, leading to *trainable kernel layers*. For each data dimension (indexed by $d = 1, 2, \cdots, D$), we learn a set of anchor points $\{\mu_j^{(d)}\}_{j=1}^K$ and bandwidth parameters $\{h_j^{(d)}\}_{j=1}^K$. This leads to a total of $2KD$ parameters for a trainable kernel layer. Mathematically, we parameterize a CDF with the following

$$F_{\boldsymbol{\theta}}^{(d)}(x) \triangleq \frac{1}{K}\sum_{j=1}^{K}\sigma\left(\frac{x^{(d)}-\mu_j^{(d)}}{h_j^{(d)}}\right), d = 1, \cdots, D, \quad (6)$$

where $\sigma(\cdot)$ denotes the sigmoid function throughout the paper, and $\boldsymbol{\theta}$ denotes the collection of all trainable parameters ($\{\mu_j^{(d)}\}_{j=1}^K$ and $\{h_j^{(d)}\}_{j=1}^K$). Learning this CDF amounts to performing KDE with a logistic kernel when $\sigma(\cdot)$ is the sigmoid function. Then, the Gaussianization procedure for dimension d can be parameterized as

$$\Psi_{\boldsymbol{\theta}}^{(d)}(x) \triangleq \Phi^{-1} \circ F_{\boldsymbol{\theta}}^{(d)}(x), \quad d = 1, \cdots, D, \quad (7)$$

and we denote $\Psi_{\boldsymbol{\theta}} = (\Psi_{\boldsymbol{\theta}}^{(1)}, \Psi_{\boldsymbol{\theta}}^{(2)}, \cdots, \Psi_{\boldsymbol{\theta}}^{(D)})^\intercal$.

By making anchor points and bandwidths trainable, our parametric trainable kernel layer can be more sample efficient compared to the traditional non-parametric KDE approach (when trained, for example, with maximum likelihood). We find that 20 to 100 anchor points work well in practice. In stark contrast, naïve KDE needs thousands of sample points to get comparable results, which is particularly inefficient given that the computational complexity scales quadratically with respect to K.

We note that Ψ is a transformation with a Jacobian whose determinant is tractable. Additionally, Ψ can be efficiently inverted:

- Φ, Φ', Φ^{-1} are not computable by elementary functions, yet they can be efficiently evaluated via numerical methods.
- As both Φ^{-1} and $F_{\boldsymbol{\theta}}^{(d)}$ are monotonic, $\Psi_{\boldsymbol{\theta}}^{(d)} = \Phi^{-1} \circ F_{\boldsymbol{\theta}}^{(d)}$ is also monotonic. We can therefore efficiently invert $\Psi_{\boldsymbol{\theta}}$ by inverting all of its dimensions with the *bisection method in parallel*.
- The Jacobian of Ψ is a diagonal matrix. The log-determinant is therefore the sum of the log-derivatives of $\Phi^{-1} \circ F_{\boldsymbol{\theta}}^{(d)}(x)$ over all dimensions.

3.2 Building Trainable Rotation Matrix Layers

In iterative Gaussianization, we transform the data using a rotation matrix after the marginal Gaussianization step. As mentioned in Section 2.2, finding a good rotation matrix is challenging using methods like ICA or PCA. Here, we discuss our approach to finding rotations by optimizing trainable rotation matrices.

3.2.1 Householder Reflections

We can parameterize the rotation matrix using Householder reflections, defined for any vector $\mathbf{v} \in \mathbb{R}^D$:

$$H = I - \frac{2\mathbf{v}\mathbf{v}^\intercal}{\|\mathbf{v}\|_2^2}. \quad (8)$$

Any $D \times D$ orthogonal matrix R can be represented as the product of at most D Householder reflections (Tomczak and Welling, 2016), *i.e.*, $R = H_1 H_2 \cdots H_D$.

By parameterizing the rotation matrix with multiple trainable Householder reflections, we define a *trainable orthogonal matrix layer*. Since the inverse of a rotation matrix is the transpose of itself, one can efficiently obtain the inverse by multiplying the transpose of the orthogonal matrix. Moreover, because the Jacobian determinant of an orthogonal transformation is always one, we can easily compute the Jacobian determinant of this layer, which is also equal to one.

One caveat is that each Householder reflection requires D parameters, and thus fully parameterizing a rotation matrix will require $O(D^2)$ parameters. This is reasonable when the data dimension is small. However, this may no longer be feasible in cases where D is large. For example, CIFAR-10 (Krizhevsky et al., 2009) images have $D = 3072$, and ImageNet (Deng et al., 2009) images can have D as large as 10^6. In such cases, one may need to trade off model flexibility for computational efficiency by using a smaller number ($< D$) of Householder reflections. Below, we explore one such approach that exploits the structure of images and utilizes a patch-based parameterization of

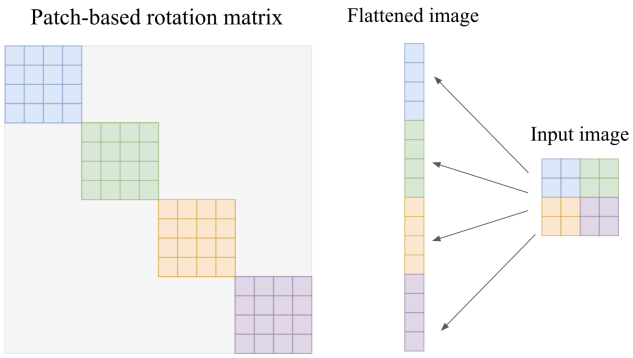

Figure 1: A patch-based rotation matrix where $L = 4, p = 2$ and $k = 2$. All entries with the grey color are zeros. Each 4×4 block on the diagonal corresponds to a new subspace of neighboring pixels, where we perform Householder reflections.

rotation matrices to significantly reduce the number of parameters.

3.2.2 Patch-Based Rotation Matrices

Intuitively, a pixel in an image is more correlated to its neighboring pixels than far away ones. Based on this intuition, we propose "patch-based" Householder reflections for parameterizing rotation matrices for images. Recalling that the role of the rotation matrix in RBIG is to render the components as independent as possible, patch-based Householder reflections are designed to focus on the components where we expect to get the biggest gains, i.e., the ones that are farthest from being independent.

For an image with dimension $L \times L$, the rotation matrix will have size $L^2 \times L^2$. Assuming p is a divisor of L and $L = p \times k$, we can partition the matrix into $k^2 \times k^2$ smaller blocks each with size $p^2 \times p^2$. Instead of directly parameterizing the $L^2 \times L^2$ rotation matrix using L^2 Householder reflections, we parameterize a block-diagonal rotation matrix with k^2 blocks. Each block on the diagonal is a $p^2 \times p^2$ rotation matrix, which requires p^2 Householder reflections to parameterize. Since rotation is now only performed in each $p \times p$-dimensional subspace, we leverage a "shift" operation on the input vectors to introduce dependency across different rotational subspaces. We call this block-diagonal rotation matrix a "patch-based rotation matrix" (see Fig. 1), and relegate extra details to Appendix B.

3.3 Deep Gaussianization Flows

Our proposed model, *Gaussianization flow*, is constructed by stacking trainable kernel layers (Section 3.1) and orthogonal matrix layers (Section 3.2) alternatively. Formally, we define an Gaussianization flow with L trainable kernel layers and orthogonal layers as:

$$T_{\boldsymbol{\theta}}(\mathbf{x}) = \Psi_{\boldsymbol{\theta}_L} \circ R_L \circ \Psi_{\boldsymbol{\theta}_{L-1}} \circ \cdots \circ \Psi_{\boldsymbol{\theta}_1} \circ R_1 \mathbf{x} \tag{9}$$

where $\boldsymbol{\theta}$ denotes the collection of all parameters.

Note that both forward and backward computations of the Gaussianization flow are efficient, and the log determinant of its Jacobian can be computed in closed-form. Consequently, we can *train* Gaussianization flows jointly with maximum likelihood, as well as producing samples efficiently. This is to the contrary of RBIG, which is a non-trainable iterative procedure.

3.4 Gaussianization Flows are Universal Approximators

We hereby prove that Gaussianization flows can transform any continuous distribution with a compact support to a standard normal, given that the number of layers and the number of parameters in each layer are sufficiently large. Ours is the first universal approximation result we are aware of for efficiently invertible normalizing flows.

Our results closely follow that of Chen and Gopinath (2000). However, we note that their results are weaker than what we need: they assume the marginal Gaussianization step can be done perfectly, whereas we use the learnable kernel layers for doing marginal Gaussianization. We defer all proofs to Appendix A.

Our proof starts by showing that mixtures of logistic distributions (as used in our learnable kernel layers) are universal approximators for continuous densities (see Lemma 2 in Appendix). Therefore, our learnable kernel layers will be able to do arbitrarily good marginal Gaussianization when sufficiently many anchor points are used. Based on this, we show that Gaussianization flow is a universal approximator given a sufficient number of layers:

Theorem 1. *Let p be any continuous distribution supported on a compact set $\mathcal{X} \subset \mathbb{R}^D$, and $\inf_{x \in \mathcal{X}} p(x) \geq \delta$ for some constant $\delta > 0$. Then, there exists a sequence of marginal Gaussianization layers $\{\Psi_{\boldsymbol{\theta}_1}, \Psi_{\boldsymbol{\theta}_2}, \cdots, \Psi_{\boldsymbol{\theta}_k}, \cdots\}$ and rotation matrices $\{R_1, R_2, \cdots, R_k, \cdots\}$ such that the transformed random variable*

$$\Psi_{\boldsymbol{\theta}_k} \circ R_k \circ \Psi_{\boldsymbol{\theta}_{k-1}} \circ R_{k-1} \circ \cdots \circ \Psi_{\boldsymbol{\theta}_1} \circ R_1 \mathbf{X} \xrightarrow{d} \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

where $\mathbf{X} \sim p$.

3.5 Building Invertible Networks with Proper Initializations

Since our Gaussianization flow is a trainble extension of RBIG, we propose to provide good initializations

for Gaussianization flows using RBIG. In the *trainable rotation matrix layers*, we randomly initialize each Householder reflection vector with samples from an isotropic Gaussian. This amounts to using random rotation matrices in RBIG. We abstain from using ICA/PCA layers for providing initialization both for the aforementioned computational issues, and for the fact that they provide similar results in practice.

In the *trainable kernel layer*, we consider a data-dependent initialization approach, using N random samples from the dataset. To initialize KDE anchor points in the first layer, we randomly draw N samples from the dataset. More generally, we initialize the KDE anchor points at layer $l+1$ using the outputs of the l-th trainable rotation matrix layer.

In fact, the initial state of our model corresponds to an iterative Gaussianization method, which, as shown previously, is capable of capturing distributions to a certain level. This allows our GF to outperform other normalizing flows at initial iterations. Because of the good initialization, our model also exhibit better robustness with respect to re-parameterizations of the data.

4 EXPERIMENTS

We evaluate our Gaussianization Flow (GF) on several datasets; these include synthetic 2D toy datasets, benchmark tabular UCI datasets (Papamakarios et al., 2017) (Power, Gas, Hepmass, MiniBoone, BSDS300) and two image datasets (MNIST and Fashion-MNIST). We compare with several popular invertible models for density estimation, including RealNVP (Dinh et al., 2015), Glow (Kingma and Dhariwal, 2018), FFJORD (Grathwohl et al., 2018), MAF (Papamakarios et al., 2017), TAN (Oliva et al., 2018) and NAF (Huang et al., 2018); we also compare directly with RBIG (Laparra et al., 2011) for reference.

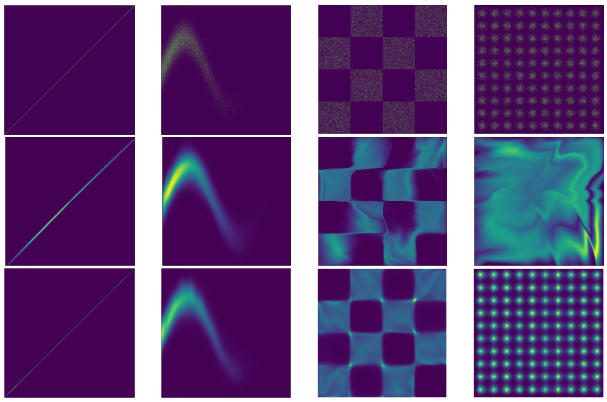

Figure 2: 2D density estimation results. **Top:** Ground truth samples. **Middle:** Glow. **Bottom:** GF.

Our experiments aim to answer the following questions:

- Is GF competitive against other methods in terms of density estimation (4.1, 4.2)?
- Does GF have better initialization than other normalizing flow models?
- Is GF robust against re-parameterization of the data with simple transformations (4.4)?
- Does GF achieve good performance when the training set is small (4.5)?

4.1 2D Toy Datasets

We first perform density estimation on four synthetic datasets drawn from complex two-dimensional distributions with various shapes and number of modes. We train the model by warping the predicted probability distribution to an isotropic Gaussian distribution. In Fig. 2, we visualize the estimated density of our Gaussianization Flow and Glow. The results show that our model is capable of fitting both continuous and discontinuous, connected and disconnected multi-modal distributions. Glow, on the other hand, has trouble modeling disconnected distributions.

4.2 Tabular and Image Datasets

We perform density estimation on five tabular datasets which are preprocessed using the method in Papamakarios et al. (2017). We compare our results directly with RealNVP, Glow and FFJORD as these are also efficiently invertible models which can be used for sample generation and inference; we list MAF, MADE, TAN and NAF results as reference as they have higher computational costs in sampling but are competitive in density estimation. From Tab. 1, we observe that GF achieves the top negative log-likelihood results in 3 out of 5 tabular datasets, and obtain comparable results on the remaining two. As expected, Gaussianization flow outperforms RBIG on all tasks by a large margin, which demonstrates the strong advantages of joint training by maximum likelihood.

For tabular datasets, we use D Householder reflections for each *trainable rotation matrix layer* where D equals the data dimension, so that the model possesses the ability to parameterize all possible rotation matrices. See Appendix D for more training details.

We also consider two image datasets, MNIST and Fashion-MNIST, and perform density estimation on the continuous distribution of uniformly dequantized images (see Tab. 1). For image data, we use patch-based rotation matrices as *trainable rotation matrix layers*.

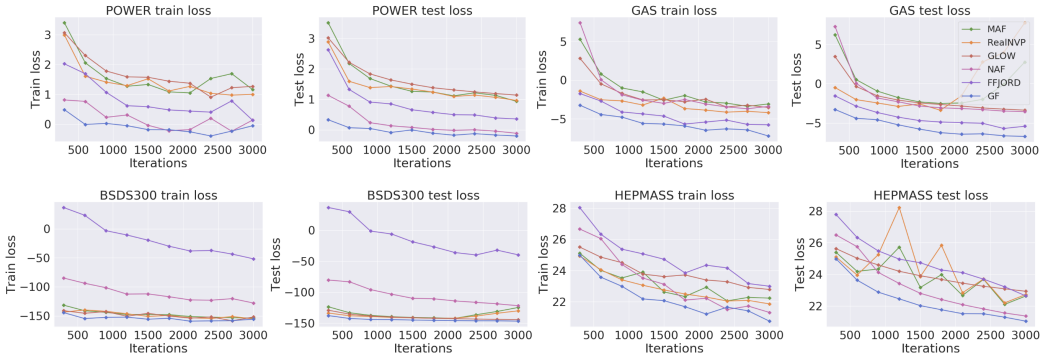

Figure 3: Negative log-likelihood (loss in nats) on training and test sets over initial training iterations.

Table 1: Negative log-likelihood for tabular datasets measured in nats, and image datasets measured in bpd. Smaller values are better.

Method	POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	FMNIST
Real NVP	-0.17	-8.33	18.71	13.55	-153.28	1.06	**2.85**
Glow	-0.17	-8.15	18.92	11.35	-155.07	1.05	2.95
FFJORD	-0.46	-8.59	**14.92**	10.43	**-157.40**	**0.99**	-
RBIG	1.02	0.05	24.59	25.41	-115.96	1.71	4.46
GF(ours)	**-0.57**	**-10.13**	17.59	**10.32**	-152.82	1.29	3.35
MADE	3.08	-3.56	20.98	15.59	-148.85	2.04	4.18
MAF	-0.24	-10.08	17.70	11.75	-155.69	1.89	-
TAN	-0.48	-11.19	15.12	11.01	-157.03	-	-
MAF-DDSF	-0.62	-11.96	15.09	8.86	-157.73	-	-

Specifically, we set the patch size to 4 and randomly pick the shifting constant c at each layer. We provide more training details in Appendix D. From the results in Tab. 1 we see that Gaussianization flow outperforms all other non-convolutional models on image datasets, including those that cannot be inverted efficiently, such as MAF and MADE (Germain et al., 2015).

4.3 Initial Performance

The data-dependent initialization of our model allows the training process to converge faster. To illustrate this, we choose four tabular datasets (pre-processed as described in Papamakarios et al. (2017)), where we set the batch size to be 500 and perform training for 3000 iterations using the default settings and model architectures. From the results in Fig. 3, Gaussianization flow achieves better training and validation performance across most iterations on the four datasets compared with other models such as RealNVP, Glow, FFJORD, MAF and NAF.

4.4 Stretched Tabular Datasets

In density estimation applications (such as anomaly detection), one could receive a stream of data that might not be sampled i.i.d. from a fixed distribution. In these cases, it would be difficult to find suitable pre-processing techniques to normalize the data, so it is desirable if our models can be robust under distributions that are not normalized. To evaluate whether the flow models are robust against certain distribution shifts (that could make normalization difficult), we consider density estimation on datasets that are not normalized. In particular, we select three pre-processed UCI datasets and transform the data using some simple invertible transformations before training. Here we keep the transformations invertible and differentiable so that we can use the change of variables formula to compute the likelihoods defined in the original data space. We consider two transformations: *cubic*, where $f(x) = x^3$; and *affine* where $f(x) = 1000x + 51$.

From the results in Tab. 2, we observe that Gaussianization flows have stable and consistent performance for both transformations and on all three datasets. In contrast, all other methods can fail in some settings. MAF-DDSF has numerical issues that lead to NaNs on datasets processed with the *affine* transformation; RealNVP, Glow, and MAF all have cases where test loss does not go down when training loss goes down; FFJORD has convergence issues when training on GAS

Table 2: Negative log-likelihood in nats for tabular datasets after simple transformations. "$*$" stands for loss larger than 1000. "$**$" implies loss does not converge and varies largely on different batches. "-" implies loss explosion on validation and test sets. "NaN" implies numerical issues encountered during training. Numbers in parentheses for GF denote the corresponding likelihood value under the original normalized transformation.

Transformation	$f(x) = x^3$			$f(x) = 1000x + 51$		
Method	POWER	MINIBOONE	GAS	POWER	MINIBOONE	GAS
Real NVP	17.47 (21.53)	93.98 (109.96)	32.27 (32.85)	-	-	-
Glow	1.67 (5.73)	91.86 (107.84)	-	41.64 (0.19)	315.30 (18.27)	49.26 (-6.00)
FFJORD	$*$	88.29 (104.27)	$**$	$*$	329.97 (32.94)	$*$
GF(ours)	**-4.41 (-0.35)**	**4.62 (20.60)**	**-6.91 (-6.33)**	**41.00(-0.45)**	325.72 (28.69)	**47.69 (-7.57)**
MAF	19.37 (23.43)	381.32 (397.3)	19.76 (20.34)	-	-	-
MAF-DDSF	-4.12 (-0.06)	7.88 (23.86)	-4.52 (-3.94)	NaN	NaN	NaN

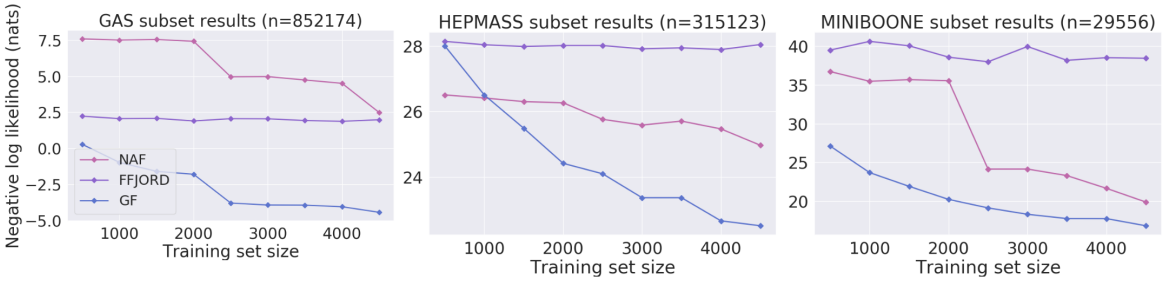

Figure 4: Negative log-likelihood results (measured in nats) over small subsets of the original training set. The subsets, with size ranging from 500 to 4500, are much smaller than the original training set size n as shown in the parentheses. We exclude MAF, RealNVP and Glow in the figures as validation error does not decrease.

with the *cubic* transformation, and on POWER and GAS with the *affine* transformation. Moreover, even with the added transformation, we are still able to obtain comparable likelihood when transformed back to the original space (see Tab. 2 results in parentheses).

4.5 Small Training Sets

The ability to quickly adapt to new distributions with relatively few samples (*e.g.* in a stream of data with continuous covariate shifts) can also be helpful. To this end, we further evaluate the generalization abilities of the models when trained on small subsets of the tabular datasets. We consider using the normalized tabular datasets, where we mix the training, validation and test datasets, shuffle them randomly and select 10,000 samples as validation/test sets respectively. We consider training various model on small training subsets with sizes ranging from 500 to 4500, where we perform validation and testing on the new validation/test sets.

We compare GF with Glow, RealNVP, MAF, FFJORD and NAF, using the same model architecture for the original tabular experiments and explore the learning rate to make the training process more stable. We show the results in Fig. 4. We note that MAF, Glow and RealNVP have trouble evaluating density on validation/test set when the training set is small enough, as the validation/test loss goes up as train loss goes down, which is the reason why we exclude them in the plots. GF significantly outperforms FFJORD and NAF in all settings except when subset size is 500 for HEPMASS, which suggests that our learnable KDE layers generalize well on test sets even when training data is scarce.

5 CONCLUSION

We introduce Gaussianization flows (GF), a new family of trainable flow models that builds upon rotation-based iterative Gaussianization. GFs exhibit fast likelihood evaluation and fast sample generation, and are expressive enough to be universal approximators for most continuous probability distributions. Empirical results demonstrate that GFs achieve better or comparable performance against existing state-of-the-art flow models that are efficiently invertible. Compared to other efficiently invertible models, GFs have better initializations, are more robust to distribution shifts in training data, and have superior generalization when training data are scarce. Combining the advantages of GFs with other efficiently invertible flow models would be an interesting direction for future research.

Acknowledgements

This research was supported by Amazon AWS, TRI, NSF (#1651565, #1522054, #1733686), ONR (N00014-19-1-2145), AFOSR (FA9550-19-1-0024).

References

Chen, S. S. and Gopinath, R. A. (2000). Gaussianization.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc.

De Cao, N., Titov, I., and Aziz, W. (2019). Block neural autoregressive flow. *arXiv preprint arXiv:1904.04676*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Devroye, L. P. and Wagner, T. J. (1979). The l_1 convergence of kernel density estimates. *Ann. Statist.*, 7(5):1136–1139.

Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: non-linear independent components estimation. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. *International Conference on Machine Learning*, 37:881–889.

Grathwohl, W., Ricky T. Q. Chen, Jesse Bettencourt, I. S., and Duvenaud, D. (2018). Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. (2018). Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2083–2092.

Huber, P. J. (1985). Projection pursuit. *The annals of Statistics*, pages 435–475.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*.

Krizhevsky, A. et al. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Laparra, V., Camps-Valls, G., and Malo, J. (2011). Iterative gaussianization: from ica to random rotations. *IEEE transactions on neural networks*, 22(4):537–549.

Oliva, J. B., Dubey, A., Zaheer, M., Póczos, B., Salakhutdinov, R., Xing, E. P., and Schneider, J. (2018). Transformation autoregressive networks.

Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347.

Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France. PMLR.

Scott, D. W. and Sheather, S. J. (1985). Kernel density estimation with binned data. *Communications in Statistics-Theory and Methods*, 14(6):1353–1359.

Sheather, S. J. (2004). Density estimation. *Statistical science*, pages 588–597.

Tomczak, J. M. and Welling, M. (2016). Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*.