On the Convergence Rate of Gaussianization with Random Rotations

Felix Draxler [1] **Lars Kühmichel** [1] **Armand Rousselot** [1] **Jens Müller** [1] **Christoph Schnörr** [1] **Ullrich Köthe** [1]

Abstract

Gaussianization (Chen & Gopinath, 2000) is a simple generative model that can be trained without backpropagation. It has shown compelling performance on low dimensional data. As the dimension increases, however, it has been observed that the convergence speed slows down. We show analytically that the number of required layers scales linearly with the dimension for Gaussian input. We argue that this is because the model is unable to capture dependencies between dimensions. Empirically, we find the same linear increase in cost for arbitrary input $p(x)$, but observe favorable scaling for some distributions. We explore potential speed-ups and formulate challenges for further research.

1. Introduction

Generative modeling is one of the most active areas of research in Machine Learning. A plethora of different architectures based on neural networks have been proposed, including: generative adversarial networks (Goodfellow et al., 2020), variational auto-encoders (Kingma et al., 2019), normalizing flows (Rezende & Mohamed, 2015), and most recently denoising diffusion models (Ho et al., 2020).

Diffusion models currently take the lead in efficient training and high-quality sampling. However, evidence so far is mostly empirical and subject to change. This calls for a rigorous comparison of the different approaches.

For all models, universal approximation theorems guarantee that all reasonable distributions can be represented by the different models (e.g. Teshima et al. (2020a;b); Koehler et al. (2021)). This is a strong statement in the sense that with enough resources, anything can be represented. However, these theorems are not useful for model selection, mainly as no statements about the required resources (model complex-

ity, training speed, sample complexity) to achieve a specified performance are made. Also, existing results often limit themselves to relatively weak measures of convergence.

In this work, we consider a variant of normalizing flows called Gaussianization (Chen & Gopinath, 2000) and its variants rotation-based iterative Gaussianization (RBIG) (Laparra et al., 2011), sliced iterative normalizing flows (SINF) (Dai & Seljak, 2021) and Gaussianization Flow (GF) (Meng et al., 2020). For the first time, we provide an explicit convergence rate for Gaussianization.

In particular, we contribute:

- We analytically derive that the number of Gaussianization layers required to achieve the same improvement in loss grows linearly with the dimensionality of the problem for Gaussian input and random rotations (see Figure 1 and Section 4.1).

- We demonstrate limits of determining better-than-random rotations from finite training data (see Section 4.2).

- We argue that this is due to the model being unable to capture dependencies between dimensions, which dominates in high dimensions (see Section 4.3).

- We empirically determine the scaling behavior for real world datasets, where we find a similar linear increase in complexity with dimension, and favorable scaling for some distributions (see Section 5.2).[1]

Broadly speaking, we analyze the scaling behaviour of Gaussianization with dimension analytically and empirically. This draws a general picture over the convergence behavior of different invertible layers, see Section 6.
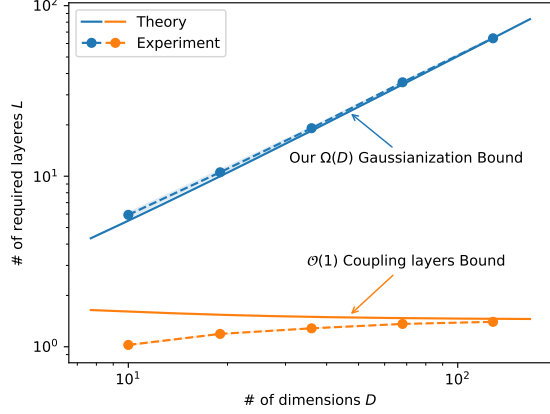
2. Related Work

There is a large collection of different methods which iteratively transform between distributions via rotations and single-dimensional transforms. Originally, the idea of iteratively transporting input data to standard normal latent codes

---

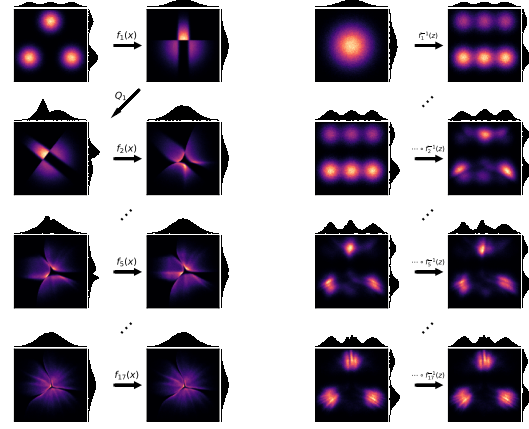[1]Code available at: https://github.com/vislearn/Gaussianization-Bound

Figure 1. **Empirical scaling of learning Gaussian distributions as a function of dimension D in the limit of low loss.** Gaussianization requires at least $\Omega(D)$ layers (Section 4.1), while only at most constant (i.e. $\mathcal{O}(1)$) number of coupling layers are needed (Section 4.3, based on Draxler et al. (2022)). The solid lines are the exact values predicted by the theories, the dots indicate experimental measurements. The shades show the inter-quartile range.

Figure 2. **Gaussianization learns a Gaussian mixture with three modes.** *(Left)* Gaussianization makes marginals normal and rotates randomly. Iterating makes the latent distribution gradually Gaussian. *(Right)* The first layer approximates $p(x)$ via the product of its marginals, see first row: $q_1(x) \approx p(x_1)p(x_2)$. Subsequent rows show the effect of additional later layers. The rows show the effect of layer $1, 2, 5,$ and 17.

has been proposed by Chen & Gopinath (2000). Laparra et al. (2011) extended the idea with rotation-based iterative Gaussianization (RBIG), by also considering the reverse transport from latent codes to the data distribution. Other variants like Iterative Distribution Transfer (IDT) replace the standard normal by an arbitrary other distribution (Pitié et al., 2007). Meng et al. (2020) leave the iterative scheme and train the flow end-to-end. They also demonstrate that the resulting method has a low sample complexity.

An important part in all these works is to find meaningful non-Gaussian projections of the data. Originally, random orthogonal matrices, ICA and PCA were suggested (Chen & Gopinath, 2000). Meng et al. (2020) learn rotations jointly with the dimension-wise transforms. SINF propose to use max K-SWD, which finds the K most non-Gaussian directions in Sliced Wasserstein Distance (SWD) (Dai & Seljak, 2021). Sliced Wasserstein Flows (SWF) had previously suggested utilizing SWD, but used this to iteratively solve a PDE (Liutkus et al., 2019).

There has been significant work on showing convergence guarantees for normalizing flows. Most work considers convergence under weak convergence or convergence under Wasserstein distance, both for Gaussianization (Chen & Gopinath, 2000; Meng et al., 2020) and for coupling normalizing flows (Teshima et al., 2020b; Koehler et al., 2021). While these measures of convergence ensure accurate samples "$x \sim q(x) \to x \sim p(x)$", they do not show convergence of the corresponding densities "$q \to p$" (Gibbs & Su, 2002). Like Draxler et al. (2022) for coupling flows, we fill this gap by considering the KL divergence,

a stronger notion of convergence. We also give explicit convergence rates instead of asymptotic guarantees. Our theoretical derivations are limited to Gaussian distributions close to convergence, however.

3. Gaussianization Fundamentals

3.1. Model

Gaussianization is a variant of Normalizing Flows. Here, we follow the presentation in our previous work Draxler et al. (2022). Flows represent a probability distribution via an invertible function $f_\theta(x)$ that maps samples x from the unknown data distribution $p(x)$ to *latent variables* $z = f_\theta(x)$ so that z follow a simple distribution, typically the standard normal. The function f_θ then yields an estimate $q(x)$ for the true data distribution $p(x)$ via the change of variables formula (e.g. Rezende & Mohamed (2015)):

$$q(x) = \mathcal{N}(f_\theta(x); 0, I)|\det J|, \tag{1}$$

where $J = \nabla f_\theta(x)$ is the Jacobian of $f_\theta(x)$. This is visualized in Figure 2.

A normalizing flow is often trained by maximizing the likelihood loss, which is equivalent to minimizing the Kullback-Leibler divergence between the distribution of the latent code $q(z)$, as given by $z = f_\theta(x)$ when $x \sim p(x)$, and the standard normal:

$$\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(q(z)\|\mathcal{N}(0, I)) \tag{2a}$$

$$= \mathbb{E}_{x \sim p(x)}\left[\tfrac{1}{2}\|f_\theta(x)\|^2 - \log|\det J|\right] + C. \tag{2b}$$

The additional term C consists of the log partition sum of a standard normal distribution, and the unknown entropy of the data: $C = (D/2)\log(2\pi) - H[p(x)]$. It is independent of the model parameters and can therefore be dropped for optimization.

For sampling from $q(x)$, draw a latent code $z \in \mathcal{N}(0, I)$ and apply the inverse: $x = f_\theta^{-1}(z)$.

A useful invertible architecture f_θ has to (i) be computationally easy to invert, (ii) be able to represent complex transformations, and (iii) have a tractable Jacobian determinant $|\det J|$ (Ardizzone et al., 2018). A plethora of such architectures have been suggested, e.g. (Kobyzev et al., 2021).

In this work, we focus on Gaussianization, first presented by (Chen & Gopinath, 2000), and its variants rotation-based iterative Gaussianization (RBIG) (Laparra et al., 2011), sliced iterative normalizing flows (SINF) (Dai & Seljak, 2021) and Gaussianization Flow (GF) (Meng et al., 2020). It is a deep architecture that consists of several blocks, each containing a rotation, and a *dimension-wise* nonlinear transform:

$$f_{\text{block}}(x) = (f_{\text{dim}} \circ f_{\text{rot}})(x). \tag{3}$$

The dimension-wise nonlinear transform f_{dim} treats each dimension $x_i, i = 1, \ldots, D$ independently:

$$x_i' = f_{\text{dim},\theta_i}(x_i), \tag{4}$$

where θ_i is a parameter vector defining the function for each dimension $i = 1, \ldots, D$. Crucially, x_i does not depend on x_j, for $i \neq j$.

Treating each dimension separately is motivated by an exact decomposition of the loss defined earlier in Equation (2):

$$\mathcal{L} = \mathcal{D}_{\text{KL}}(q(z) \| \mathcal{N}(0, I)) \tag{5a}$$
$$= \underbrace{\mathcal{D}_{\text{KL}}(q(z) \| q(z_1) \cdots q(z_D))}_{\text{Dependence } \mathcal{D}} \tag{5b}$$
$$+ \sum_i \underbrace{\mathcal{D}_{\text{KL}}(q(z_i) \| \mathcal{N}(0, 1))}_{\text{marginal loss } \mathcal{J}_i}. \tag{5c}$$

This representation makes clear that the loss is composed of two parts: The *dependence* $\mathcal{D}$, which measures how far $q(z)$ deviates from the product of its marginal $q(z_1) \cdots q(z_D)$, and the *marginal loss* for each dimension, which measure the deviation of each marginal $q(z_i)$ from the univariate standard normal. Such a result is known as a generalized Pythagorean Theorem in information geometry (Amari & Nagaoka, 2007). Equation (5) is due to (Cardoso, 2003).

A single layer can reduce the marginal losses J_i in all dimensions (close) to zero if each f_{dim,θ_i} is sufficiently rich. This leaves only the dependence $\mathcal{D}$ as the loss. Note that fundamentally, the dependence $\mathcal{D}$ cannot be changed by a single-dimensional transformation.

If all layers worked with the same set of dimensions, the dependency $\mathcal{D}$ part of the loss would never change. Here, the rotation f_{rot} layer in each block comes into play (see Equation (3)). It is parameterized by an orthogonal matrix $Q \in O(D)$ that rotates the data, varying the directions of action of the dimension-wise transforms:

$$f_{\text{rot, Q}}(x) = Qx. \tag{6}$$

Rotating the data does not affect the sum of the loss contributions $\mathcal{L} = \mathcal{D} + \sum_i \mathcal{J}_i$, but it redistributes it between the dependence $\mathcal{D}$ and the marginal losses $\mathcal{J}_i$. This makes choosing rotations Q crucial. We elaborate on this issue in Section 3.3.

Together, this model is sufficiently rich so that with random rotations and enough expressive layers, Gaussianization can approximate arbitrary distributions, i.e. map any input dataset $p(x)$ to normally distributed codes (Chen & Gopinath, 2000; Meng et al., 2020). Note that the existing guarantees only consider weak convergence, whereas no convergence guarantee on the KL divergence $\mathcal{L}$ is known. In this work, we consider the KL divergence, since this stronger notion of convergence ensures that both samples and density estimates converge.

3.2. Training

Gaussianization can be trained layer-by-layer (*iterative*) or by training all blocks jointly (*end-to-end*).

In iterative training, one adds layers one by one: The data set is used to train the first block f_{block} to maximally reduce the loss in Equation (2). The second block is then trained with the data transformed by the first layer.

In end-to-end training (Meng et al., 2020), we would concatenate a pre-specified number of blocks at initialization. The parameters of each block are then trained jointly using Equation (2). The advantage of end-to-end training is that blocks can collaborate as the training signal is the gradient of the entire pipeline and not just of a single block.

Iterative training has the advantage that there are fast approaches for learning the one-dimensional transforms. The cumulative distribution function (CDF) may be estimated on each 1D slice using the quantiles of the data, which by-passes gradient descent on the loss. We use rational-quadratic splines to fit the CDF due to their flexibility and analytical invertibility (Dai & Seljak, 2021).

In practice, end-to-end training requires fewer layers than iterative training when learning a given distribution in low dimensions (Meng et al., 2020). As the dimension increases, however, the convergence of the training saturates. State-of-the-art Gaussianization results in high dimensions are currently held by iterative training with SINF (Dai & Seljak, 2021).

3.3. Choosing rotations

Looking at the loss in Equation (2), it becomes clear that the rotation layers have a strong impact on the performance. The KL divergence $\mathcal{D}_{\text{KL}}(q(z)\|\mathcal{N}(0, I))$ itself is symmetric under rotations, and so the total loss $\mathcal{L}$ in Equation (2) does not change with the rotation. Instead, it distributes the loss between the dependence $\mathcal{D}$ and the marginal losses $\mathcal{J}_i$. To illustrate this, consider two special examples:

First, we take a distribution $p(x) \neq \mathcal{N}(0, I)$ which can be written as the product of marginals in some rotation of the data Q^*:

$$p(Q^*x) = p((Q^*x)_1)\ldots p((Q^*x)_D). \qquad (7)$$

If we evaluate the loss in this orientation, the dependence $\mathcal{D}$ becomes zero and all loss is contained in the marginals $\mathcal{J}_i$. This allows Gaussianization to fit $p(x)$ in one layer: The rotation layer chooses $Q = Q^*$ and the element-wise transforms fit $p((Q^*)x_i)$.

On the other end, consider a Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ where $\operatorname{tr}\Sigma = D$. Then, there exists a rotation Q^+ for which the standard deviations along the axis are one, i.e. $(\Sigma')_{ii} = ((Q^+)^{\text{T}}\Sigma Q^+)_{ii} = 1$. Then, $\mathcal{J}_i = 0$ and all the loss is contained in the dependence $\mathcal{D}$. The element-wise transformation layer cannot make any progress in this situation.

For the two examples given, it is in principle known how to obtain the optimal choice for Q: In the case of the first distribution, ICA aims to find Q^* such that the axes are independent. For the case of Gaussian distributions, PCA yields the optimal Q^* such that Σ' is diagonal and can be solved using one block.

For most real-world distributions, an orientation Q^* does not exist where the data dimensions become independent.

The Cramér-Wold theorem (Cramér & Wold, 1936) guarantees that the learned latent codes are exactly Gaussian if and only if there is no orientation Q with a non-Gaussian marginal. To visualize this result, imagine projecting the data set along a unit vector and looking at the histogram:

$$z_{\text{proj}} = w^{\text{T}}z. \qquad (8)$$

If the codes z are distributed like a multivariate standard normal distribution, then each projection z_{proj} will be distributed like a univariate standard normal distribution. If however, z is not normally distributed, then there must be some projection for which the data is also not distributed like a normal distribution.

This implies that for a single layer we want to choose Q such that the marginal projections are as non-Gaussian as possible. Then, as much loss as possible is contained in

J_i, which can then be removed by the single-dimensional transformations f_{dim,θ_i}.

There is a rich history in identifying interesting marginal directions in high-dimensional data. These are the most common choices for computing Q from data:

- Random rotations are randomly sampled as $Q \in O(D)$. We give several guarantees in Section 4 for this case.

- Principal Component Analysis (PCA) transforms any distribution with nontrivial covariance Σ such that its principal axes coincide with the coordinate axes, i.e. such that the resulting covariance matrix is diagonal $Q^{\text{T}}\Sigma Q = \operatorname{Diag}(S)$ with eigenvalues S.

- Independent Component Analysis (ICA) identifies the space in which a $p(u) = p(u_1)\cdots p(u_D)$, if $x = Au$ and u can be factorized in this form.

- max K-SWD identifies the directions in which the sliced Wasserstein distance can be maximally reduced. The sliced Wasserstein distances can be a proxy for the marginal KL divergences, as both measure a divergence between distributions.

We consider limitations of learned rotations in Section 4.2.

4. Analytic scaling behavior with dimension

In this section, we derive the scaling behavior of Gaussianization with the dimensionality of $p(x)$: We show that the number of required layers grows at least linearly with the dimensionality of the problem in the case of random rotations. We adopt the notation for asymptotic convergence behavior with dimension D from (Knuth, 1997), replacing $O \rightarrow \mathcal{O}$ to avoid name collision with the orthogonal group $O(D)$:

$$\mathcal{O}(f(D)) := \{g(D) \leq Cf(D) \text{ for } D > D^*\}, \qquad (9a)$$
$$\Omega(f(D)) := \{g(D) \geq Cf(D) \text{ for } D > D^*\}, \qquad (9b)$$
$$\Theta(f(D)) := \mathcal{O}(f(D)) \cap \Omega(f(D)). \qquad (9c)$$

4.1. Random rotations

We compute the convergence scaling for the special case where the data distribution to be learned is a multivariate Gaussian $p(x) = \mathcal{N}(0, \Sigma)$. This may seem like a large restriction compared to the general distributions Gaussianization promises to learn. However, the goal of Gaussianization is to map the data to a standard normal, so after enough Gaussianization blocks the latent estimate will be close to a normal distribution $\mathcal{N}(0, \Sigma)$ with $\Sigma \approx I$. As we only give lower bounds on the number of required layers, our bounds will be correct even if coming close to this case is hard. Together, we expect that difficulties in learning
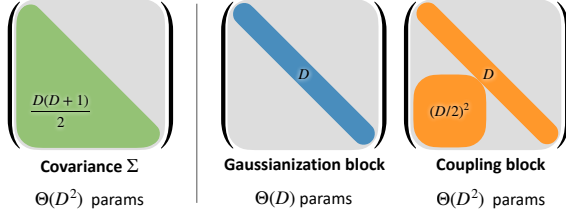
Figure 3. **Parameter counting argument**: The goal is to transform the covariance matrix Σ to the unit matrix I. The covariance has $D(D+1)/2$ degrees of freedom, of which Gaussianization can learn D per layer, and couplings $D^2/4 + D$ per layer.

simple distributions like Gaussians will also occur when learning complicated distributions. We provide experiments in Section 5 that show how the central insights transfer to arbitrary distributions.

Assumption 1. *The covariance is normalized:* $\operatorname{tr}\Sigma = D$.

Assumption 2. *The eigenvalues of the covariance matrix Σ are distinct:* $\lambda_i \neq \lambda_j$ for $i \neq j$.

Both assumptions are weak: Assumption 1 is a typical data preprocessing, and it is also achieved after a single Gaussianization layer. Assumption 2 is typically satisfied when working with real data that are in 'general position'.

For Gaussian input, we find the following result:

Theorem 1. *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under Assumptions 1 and 2. To exactly represent $p(x)$ with* random *rotations Q, at least*

$$L \geq \frac{1}{2}(D+1) \tag{10}$$

Gaussianization layers are required almost surely.

This is a lower bound on the required number of layers for both the end-to-end and iterative training approaches: **The number of Gaussianization blocks required grows (at least) linearly with the number of dimensions.** The proof is a simple parameter-counting argument: Σ has $D(D+1)/2$ degrees of freedom, but linear Gaussianization with random rotations only has D parameters per layer. Dividing $D(D+1)/2$ by D yields the result. This is visualized in Figure 3. In the proof in Appendix A.1 we give more details on the role of rotations.

The above argument assumes that we want exactly represent the target distribution $p(x) = \mathcal{N}(0, \Sigma)$. This is an unrealistic requirement for machine learning models: Imagine training a neural network with ReLU activations to fit a parabola $f(x) = x^2$. Being piecewise linear, you can never hope for the ReLU network to exactly represent the parabola. Instead, one requires that some error metric between the learned and the true function can be made arbitrarily small.

The existence of arbitrarily close models is called *universal approximation*.

In this work, we consider the loss that is used in practice: the Kullback-Leibler divergence in Equation (2), which is equivalent to the maximum likelihood criterion. It is a strong metric in the sense that it ensures convergence of both samples ("$x \sim q \to x \sim p$") and the corresponding densities ("$q \to p$"). For the Gaussian case, the KL divergence between some $p(x) = \mathcal{N}(0, \Sigma)$ and the latent $\mathcal{N}(0, I)$ reads:

$$\mathcal{L}^{(0)} = \mathcal{D}_{\text{KL}}(\mathcal{N}(0,\Sigma)\|\mathcal{N}(0,I)) \tag{11a}$$
$$= \tfrac{1}{2}(\operatorname{tr}\Sigma - D - \log\det\Sigma) \tag{11b}$$
$$\overset{(A1)}{=} -\tfrac{1}{2}\log\det\Sigma. \tag{11c}$$

This holds under the weak Assumption 1 that $\operatorname{tr}\Sigma = D$.

Universal approximation results are often pure existence theorems: They only state that a suitable network can be built, but often do not tell anything about the required model complexity. The existing universal approximation guarantees for Gaussianization (Chen & Gopinath, 2000; Meng et al., 2020) also do not provide an explicit guarantee for the required model capacity. In addition, they do not consider the KL divergence, but weak convergence, which does not imply convergence of densities.

For the first time, we give a rigorous lower bound on the KL divergence assuming a given number of layers for random rotations $Q \in O(D)$.

Theorem 2. *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under Assumption 1. Then, in the case $\mathcal{L} \ll 1$, the expected loss after L iterative Gaussianization blocks with random rotations is approximately:*

$$\mathbb{E}_{Q_{1\ldots L} \in O(D)}[\mathcal{L}^{(L)}] \leq \left(1 - \frac{2}{D+2}\right)^L \mathcal{L}. \tag{12}$$

Theorem 2 sharpens Theorem 1 to a statement on the KL divergence. Rewriting Equation (12) to derive how many layers L are required at least to reduce some initial loss $\mathcal{L}$ to $\mathcal{L}'$, we find the same scaling behavior with the dimension D as in Theorem 1:

$$L \geq \frac{\log(\mathcal{L}'/\mathcal{L})}{\log\left(1 - \frac{2}{D+2}\right)} \approx \log(\mathcal{L}/\mathcal{L}')\frac{D+1}{2} = \Omega(D). \tag{13}$$

In words: The number of layers L required to reduce the loss by a fixed amount scales at least linearly with the dimension D. For the proof, see Appendix A.3.

Figure 1 empirically confirms this theoretical bound on a variety of normal distributions with different initial Σ. It confirms the scaling with $\Omega(D)$ and shows that the actually required numbers to reduce the loss by a ratio is higher

than Equation (13). In order to be consistent with $\mathcal{L} \ll 1$, we measure the number of required layers for each input by extrapolating the convergence rate of the last two of $10D$ layers. If $\mathcal{L} \gg 0$, the performance deviates from the exact prediction in Equation (13) in both directions, but the scaling with dimension is preserved. Details are given in Appendix B.2.

While Theorem 2 only holds for the iterative case, end-to-end training may perform better. We address this in Corollary 1 in the following section.

To the best of our knowledge, Theorems 1 and 2 are the first explicit convergence results for Gaussianization.

4.2. Limitations of learned rotations

In the previous section, we showed the scaling behavior of Gaussianization for random rotations. In this section, we show that end-to-end training has the same scaling behavior with dimension. For iterative training, we point at a fundamental challenge in learning high-dimensional rotations from data.

End-to-end training with *learned* rotations may outperform Theorem 1 in terms of the number of required layers. In fact, rotations exist such that arbitrary Σ can be fit with a single layer. However, state-of-the-art rotation learning typically does not train on the full orthogonal group $O(D)$, as current methods become prohibitively expensive with increasing D. Instead, one considers subsets typically spanned by $k \cdot D$ independent parameters, such as spanned by Householder transforms $I - vv^T$ ($k = 1$) or block-diagonal orthogonal matrices (with $k = (b-1)/2$ for block size b) (Meng et al., 2020). Even with these parameterized rotations, however, the number of required layers scales with the dimension D. We then find:

Corollary 1. *Given a multivariate non-degenerate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ under Assumptions 1 and 2. To exactly represent $p(x)$ with* learned *rotations Q with $k \cdot D$ parameters each, at least*

$$L \geq \frac{1}{2(k+1)} D \qquad (14)$$

Gaussianization layers are required almost surely.

The scaling of L with D can only be avoided when $k \in \Omega(D)$, which does not hold for the parameterizations mentioned above.

For iterative training, the bulk of the literature focuses on rotating the data such that the marginals deviate as much as possible from the target normal distribution $\mathcal{N}(0, 1)$, see Section 3.3. Alternative methods differ mainly by their measure of marginal non-Gaussianity.

Unfortunately, this approach has an inherent tendency to overfit on finite training sets: In the non-asymptotic regime, there is a high probability that some marginal projections exhibit considerable spurious non-Gaussianity even when the data are sampled from a perfect standard normal, $x \sim \mathcal{N}(0, I)$. In other words, although training has converged, the iterative training algorithm will identify a rotation Q that appears to improve the loss but actually worsens it. In sufficiently high dimensions D, this still happens for large datasets with $N \gg D$, a typical situation in computer vision. Inspired by (Bickel et al., 2018), we illustrate the phenomenon at $D = 3072 = 32 \times 32 \times 3$ and $N = 60000$, the dimension and size of the CIFAR10 dataset. In Figure 4, we show the histogram of a projection

$$x_{\text{proj}} = w^T x, \qquad |w| = 1, x \sim \mathcal{N}(0, I) \qquad (15)$$

of N fixed samples from a D-dimensional standard normal. To demonstrate overfitting, we search for w such that the histogram is as close as possible to the bimodal distribution shown in the plot. Although asymptotically no such w exists, the optimization readily finds a solution in the finite dataset. Details on the experiment can be found in Appendix B.3.

A similar experiment is reported in Appendix D.1 of (Dai & Seljak, 2021). They show that max K-SWD can identify spurious non-Gaussian projections. We extend their experiment by demonstrating the effect of such projections on the learned distribution.

The theoretical analysis in chapter 8 of (Wainwright, 2019) shows that phenomena like this are fundamental in finite datasets. Specifically, they prove that *no* method can reliably estimate the eigenvectors of the *empirical* covariance in high dimensions if the ratio D/N is bounded away from zero, unless additional assumptions (e.g. sparsity of the covariance) are made. Consequently, even the straightforward idea of defining the optimal Q via PCA can fail and must be used with caution.

In practice, we expect Gaussianization with learned rotations to work well in the initial blocks, where the intermediate latent distributions are clearly non-Gaussian. Deeper in the network, however, intermediate distributions are already close to standard normal, and spurious projections will appear. The resulting overfit of the rotation Q will fool the subsequent marginal transformation, making the data less rather than more Gaussian. This can only be fully avoided by random orthogonal matrices Q, which almost surely do not result in spurious non-Gaussian projections (Bickel et al., 2018). This is also illustrated by the nearly perfectly Gaussian histogram resulting from an (uncurated) random projection in Figure 4. Random rotations are exactly the regime of our theoretical results.
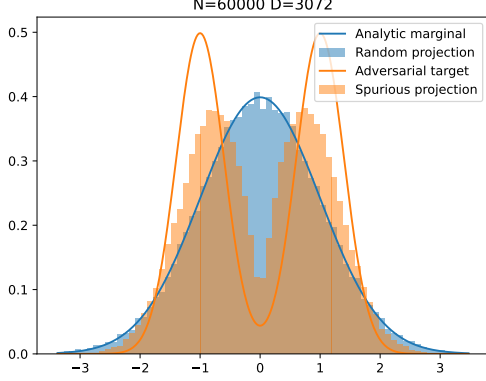
Figure 4. **Spurious projection of standard normal data**. The plot shows $N = 60,000$ samples from a $D = 3072$-dimensional standard normal distribution projected to a single dimension. The blue projection is selected randomly, and the resulting histogram is close to standard normal. The orange projection is optimized so that the dataset has a spurious bimodal histogram. The histograms coincide with the marginal distribution Gaussianization would learn, producing a bimodal distribution from Gaussian data in the second case.

4.3. Relation to coupling-based normalizing flows

We now compare the above scaling results to guarantees for another normalizing flows architecture, namely all variants of coupling-based normalizing flows (Dinh et al., 2015). See Draxler et al. (2022) for a list of architectures the following results apply to.

Coupling-based normalizing flows and Gaussianization (see Section 3.1) both represent invertible functions as a composition of invertible blocks, and each block consists of a rotation followed by a reshaping of the distribution. Each coupling-based reshaping splits the data dimensions into a D'-dimensional active and $(D - D')$-dimensional passive subspace, and reshapes the active subspace *conditional* on the passive subspace. Thus, couplings explicitly reduce the *dependency* between the active and passive dimensions, in contrast to Gaussianisation blocks, which reshape each dimension independently. The downside of this improvement is that only D' dimensions are modified in each block, with $D' = D/2$ being the most common choice.

Using exactly the same argument as in Theorem 1, we provide a first glance on how many coupling blocks are required to fit $p(x) = \mathcal{N}(0, \Sigma)$:

Corollary 2. *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$. To exactly represent $p(x)$, at least*

$$L_{\text{cpl}} \geq 2 \in \Omega(1) \tag{16}$$

coupling blocks with random rotations $Q \in O(D)$ are required almost surely.

This result says that the number of coupling blocks required to represent $\mathcal{N}(0, \Sigma)$ exactly is independent of dimension in the best case. For the proof, see Appendix A.4.

Just like Corollary 2 gives a lower bound on the number of blocks required, Theorem 2 of (Koehler et al., 2021) provides an upper bound, which we condense for simplicity:

Theorem 3 (Koehler et al. (2021))**.** *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$. To exactly represent $p(x)$, at most $L_{\text{cpl}} \leq 48 \in \mathcal{O}(1)$ coupling blocks with block permutations are required.*

This ensures that at most 48 blocks or $\mathcal{O}(1)$ are required to exactly fit a Gaussian $p(x) = \mathcal{N}(0, \Sigma)$. While this may seem like a large number, the result crucially ensures that the number of required blocks is independent of the dimension. In the statement, block permutations means that all rotations switch all active and passive dimensions:

$$Q = \begin{bmatrix} 0 & I_{D/2} \\ I_{D/2} & 0 \end{bmatrix}. \tag{17}$$

This means that in odd and even layers each modify one half of the dimensions respectively.

The above arguments considered exact representation of $p(x)$, and statements about the convergence rate are stronger as argued in Section 4.1. We therefore condense (see Appendix A.5) the main theorem in (Draxler et al., 2022) to have the same format as Theorem 2:

Theorem 4 (Draxler et al. (2022))**.** *Given a multivariate Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$. The initial loss $\mathcal{L}$ is given by Equation (11). Then, in the case $\mathcal{L} \ll 1, 1 \ll D$, the loss after L_{cpl} iterative coupling blocks with random rotations is at most:*

$$\mathbb{E}_{Q_{1\ldots L} \in O(D)}[\mathcal{L}^{(L_{\text{cpl}})}] \lesssim \left(\frac{1}{2}\right)^L \mathcal{L}. \tag{18}$$
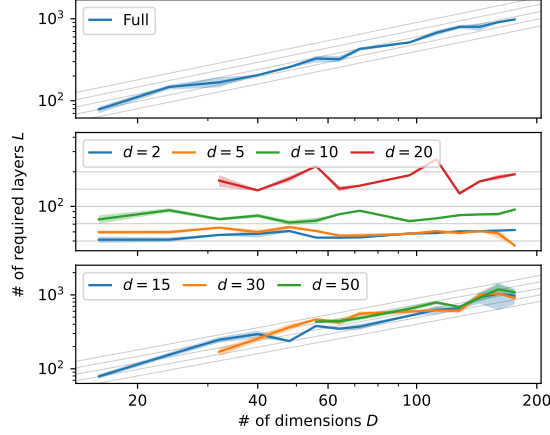
In parallel to the previous calculations, we derive how many coupling blocks are required to reduce the loss from $\mathcal{L}$ to $\mathcal{L}'$:

$$L_{\text{cpl}} \lesssim \frac{\log(\mathcal{L}/\mathcal{L}')}{\log(2)} = \mathcal{O}(1). \tag{19}$$

Together, Corollary 2 and Theorems 3 and 4 show that in contrast to marginal Gaussianization, the number of coupling blocks is independent of the problem dimension:

$$L_{\text{cpl}} = \Theta(1). \tag{20}$$

This means that on Gaussian data, **Gaussianization requires more layers because the blocks do not model dependencies between dimensions.**

Figure 5. **Required layers of Gaussianization on toy data.**
(Top) If all dimensions depend on one another, the number of required layers increases linearly with dimension. *(Middle)* Removing dependencies between trailing dimensions $i > d$ improves scaling, where a constant number of layers is sufficient. *(Bottom)* If the trailing dimensions are independent Gaussian noise, the number of layers increases linearly with dimension again. Shaded regions indicate 90% of the data. Gray lines show $\Theta(D)$ resp. $\Theta(1)$.

5. Experimental scaling behavior

In Section 4, we gave rigorous predictions for the convergence of Gaussianization on Gaussian input $p(x) = \mathcal{N}(0, \Sigma)$. The core result is that the number of blocks increases linearly with the dimension. Comparing this result to similar results on coupling-based normalizing flows, we identified that modeling dependencies between dimensions is the major bottleneck of Gaussianization.

We now lift the restriction to Gaussian input and consider $p(x) \neq \mathcal{N}(0, \Sigma)$. We find that as the dimension increases, the number of required layers L increases with dimension D, but favorable scaling can be achieved depending on the properties of the data. We give implementation details in Appendix B.4. We base our implementation of $f_{\dim}(x)$ on the code provided by SINF (Dai & Seljak, 2021).

5.1. Toy scaling experiment

For determining the scaling behavior of Gaussianization we require a family of distributions that are related to one another while the dimension D varies. We propose to build such a toy distribution via an autoregressive model:

$$p(x) = p(x_1) \prod_{i=2}^{D} p(x_i|A_i), \tag{21}$$

where the set $A_i \subseteq \{x_1, \ldots, x_{i-1}\}$ collects the random variables that x_i depends upon. This allows adding new dimensions by making them dependent on the previous

dimension.

We consider the following three variants: (1) Let every variable depend on all previous variables: $A_i^{(1)} = \{x_1, \ldots, x_{i-1}\}$). (2) We only make a subset of d variables depend on all previous, and let the remaining dimensions depend on this fixed subset of dimensions: $A_{i \leq d}^{(2)} = \{x_1, \ldots, x_{i-1}\}$ (core) and $A_{i > d}^{(2)} = \{x_1, \ldots, x_d\}$ (remainder). (3) Like the second case, but the remaining dimensions $> d$ are independent Gaussian noise: $A_{i \leq d}^{(2)} = A_{i \leq d}^{(3)}$ (core) and $A_{i > d}^{(3)} = \emptyset$ (noise).

In particular, we choose the continuous mixture of Gaussians

$$p(x_1) = \mathcal{N}(m_1, \sigma_1^2), \tag{22a}$$

$$p(x_i|A_i) = \mathcal{N}(m_i(x_j \in A_i), \sigma_2^2) \tag{22b}$$

where the dependencies are introduced through m_i:

$$m_i(x_j \in A_i) = m_0 + 5\tanh\left(\frac{1}{10}\sum_{x_j \in A_i} s_{ij} x_j^2\right). \tag{23}$$

The values $m_1, m_0 \in \mathbb{R}; \sigma_1, \sigma_2 \in \mathbb{R}_+, s_{ij} \in \{-1, 1\}$ are parameters to the distribution.

Figure 5 shows how many layers L are needed to reduce the loss by a fixed ratio $\gamma = \mathcal{L}'/\mathcal{L} < 1$ for each case as a function of dimension D.[2] We find that for cases (1) and (3), the number of required Gaussianization layers increases linearly with dimension, which is consistent with our theoretical result on Gaussian data in Equation (13). In case (2) however, the number of required layers remains roughly constant with dimension (but it does depend on the number of dependent dimensions). We show in Figure 10 in Appendix B.5 that random projections are less Gaussian in this case, as additional variables carry information about the core dimensions. This makes it easier for Gaussianization to fit the data, efficiently removing loss by fitting the non-Gaussian marginals.

This toy experiment indicates that linear increase in required layers holds for some distributions, and a favorable scaling behavior may be obtained for certain input.

5.2. Real dataset experiment

We now consider the scaling behavior of Gaussianization on a real dataset, the EMNIST digits (Cohen et al., 2017). To measure the scaling with dimensions, we construct variants of the data with different dimensions. We therefore rescale the images to scales between 2×2 and the original 28×28, see Figure 6.

---

[2]In a preliminary version of this paper, we conjectured that *alone the number* of dependencies causes the scaling behavior
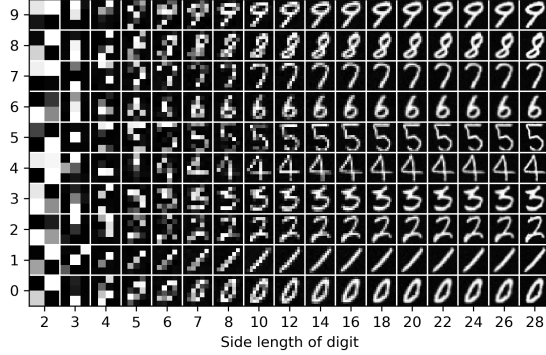
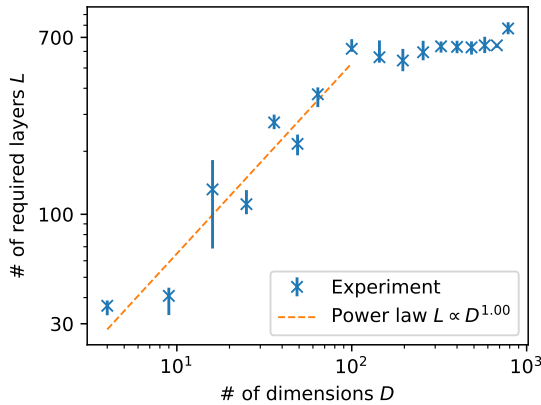*Figure 6.* Our multi-scale EMNIST digits dataset.

Figure 7. **Gaussianization requires more layers for higher resolution datasets.** From 2×2 to 10×10, fitting a power law yields a linear function. For larger images, the difficulty does not increase further. We think that this is due to an increasing number pixels being largely determined from the other dimensions. Data points show the median and error bars cover 90% of the training runs.

Figure 7 shows how many layers are required as a function of dimension, extrapolated from training 64 Gaussianization layers. We find that up to a scale of $D = 10 \times 10$, the number of required layers roughly increases as $\Theta(D)$, like in Equation (13), and then remains about constant.

Consistently, around a side length $l = 10$, the main characteristics of each digit become clear. Afterwards, only local details are filled in. We identify the corresponding scaling of Gaussianization with case (2) in Section 5.1, where additional dimensions are highly correlated with others, so that random marginal distributions become less Gaussian.

Note that for computing the absolute KL divergence as defined in Equation (2), we need to evaluate the entropy of the data $H[p(x)]$. This is an unknown value in general and

with dimension. After more detailed experiments, this hypothesis turned out to be false.

for rescaled EMNIST digits in particular. We therefore replace the ground truth dataset by generative models trained on each respective scale. We use a Normalizing Flow for our ground truth distribution, which achieves better density estimates in general than Gaussianization. This removes bias from our convergence rate estimates: If we were to use the trained model itself as ground truth, we would find spuriously fast convergence even if the model does not converge – as we compute the convergence rate to a suboptimal optimum. See Appendix B.6 for all experimental details.

6. Conclusion

Gaussianization is a simple generative model, that has shown advantages in situations with little data (Meng et al., 2020). Differently from other approaches, it can be trained without loss and neural networks, yet provides useful density estimates and samples in low and moderate dimensions.

Scaling Gaussianization to high dimensions remains a major challenge, and we confirm this rigorously. We show analytically for the Gaussian distribution $p(x) = \mathcal{N}(0, \Sigma)$ (see Section 4) that the number of required layers typically scales with the dimensionality of the problem if the rotations Q are chosen at random. On non-Gaussian distributions, we show that convergence can be favorable for some distributions.

Generally speaking, our work complements the theory on networks that are constrained in order to be invertible: Should we have few unconstrained yet expensive layers (i.e. autoregressive, with all dependencies modeled in a single, highly expressive layer (Knothe, 1957; Rosenblatt, 1952)) – or should we constrain each layer, but have more of them (i.e. coupling or Gaussianization layers)? Our theory indicates that Gaussianization, limited in learning dependencies between dimensions, scales slower than alternatives.

An important point remains open, however: Improving Gaussianization to high dimensions may be possible by constructing better rotations Q. Our work points out fundamental limits: There exist spurious non-Gaussian directions that may be spuriously identified (see Section 4.2), and we can never expect that a single Gaussianization block can fit real-world distributions (see Section 3.3).

Acknowledgements

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Amari, S.-i. and Nagaoka, H. *Methods of Information Geometry*. American Mathematical Society, 2007.

Ardizzone, L., Kruse, J., Rother, C., and Köthe, U. Analyzing Inverse Problems with Invertible Neural Networks. In *International Conference on Learning Representations*, 2018.

Bickel, P. J., Kur, G., and Nadler, B. Projection Pursuit in High Dimensions. *PNAS*, 115(37):9151–9156, 2018.

Cardoso, J.-F. Dependence, Correlation and Gaussianity in Independent Component Analysis. *Journal of Machine Learning Research*, 4:1177–1203, 2003.

Cartwright, D. I. and Field, M. J. A refinement of the arithmetic mean-geometric mean inequality. *Proceedings of the American Mathematical Society*, 71(1):36–38, 1978.

Chen, S. and Gopinath, R. Gaussianization. In Leen, T., Dietterich, T., and Tresp, V. (eds.), *Advances in Neural Information Processing Systems*, volume 13, 2000.

Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Costarelli, D. and Spigler, R. How sharp is the Jensen inequality? *Journal of Inequalities and Applications*, 2015(1):69, December 2015.

Cramér, H. and Wold, H. Some Theorems on Distribution Functions. *Journal of the London Mathematical Society*, s1-11(4):290–294, October 1936.

Dai, B. and Seljak, U. Sliced iterative normalizing flows. In *International Conference on Machine Learning*, 2021.

Dinh, L., Krueger, D., and Bengio, Y. NICE: Non-linear Independent Components Estimation. In *International Conference on Learning Representations, Workshop Track*, 2015.

Draxler, F., Schnörr, C., and Köthe, U. Whitening Convergence Rate of Coupling-based Normalizing Flows. In *Advances in Neural Information Processing Systems*, 2022.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. In *Advances in Neural Information Processing Systems*, 2019.

Falcon, W. and The PyTorch Lightning team. PyTorch Lightning, March 2019.

Gibbs, A. L. and Su, F. E. On Choosing and Bounding Probability Metrics. *International Statistical Review / Revue Internationale de Statistique*, 70(3):419–435, 2002.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33, 2020.

Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

Kingma, D. P., Welling, M., et al. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.

Knothe, H. Contributions to the theory of convex bodies. *Michigan Mathematical Journal*, 4(1):39–52, 1957.

Knuth, D. E. *The art of computer programming*, volume 3. 1997.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021.

Koehler, F., Mehta, V., and Risteski, A. Representational aspects of depth and conditioning in normalizing flows. In *International Conference on Machine Learning*, 2021.

Kühmichel, L. E. and Draxler, F. Lightning trainable, 2023. Software available from https://github.com/larskue/lightning-trainable.

Laparra, V., Camps-Valls, G., and Malo, J. Iterative Gaussianization: From ICA to Random Rotations. *IEEE Transactions on Neural Networks*, 22(4):537–549, April 2011.

Liutkus, A., Simsekli, U., Majewski, S., Durmus, A., and Stöter, F.-R. Sliced-Wasserstein flows: Nonparametric generative modeling via optimal transport and diffusions. In *International Conference on Machine Learning*, volume 97 of *Proceedings of machine learning research*, 2019.

Meng, C., Song, Y., Song, J., and Ermon, S. Gaussianization Flows. In *International Conference on Artificial Intelligence and Statistics*, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Pitié, F., Kokaram, A. C., and Dahyot, R. Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding*, 107(1-2):123–137, July 2007.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning*, 2015.

Rosenblatt, M. Remarks on a multivariate transformation. *The annals of mathematical statistics*, 23(3):470–472, 1952.

Teshima, T., Ishikawa, I., Tojo, K., Oono, K., Ikeda, M., and Sugiyama, M. Coupling-based Invertible Neural Networks Are Universal Diffeomorphism Approximators. In *Advances in Neural Information Processing Systems*, 2020a.

Teshima, T., Tojo, K., Ikeda, M., Ishikawa, I., and Oono, K. Universal Approximation Property of Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems, Workshop Track*, 2020b.

The pandas development team. pandas-dev/pandas: Pandas, February 2020.

Wainwright, M. *High-dimensional statistics: A non-asymptotic viewpoint*. Cambridge University Press, 2019.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman (eds.), *9th Python in Science Conference*, 2010.

A. Proofs

A.1. Proof of Theorem 1

Proof. We need to exclude the special case that the accumulated rotation $Q^{(l)} = Q_l \cdots Q_1$ (partially) aligns with the eigenspace of Σ for some $l = 1, \ldots, L$. Then, the Σ is (partially) diagonal in the input to this block l and f_{dim} is able to map $\Sigma \to I$ already earlier. However, this alignment has probability mass zero under random Q.

To perfectly map the input Gaussian $\mathcal{N}(0, \Sigma)$ to the latent distribution $\mathcal{N}(0, I)$, we need to learn a linear function A such that $A^T \Sigma A = I$. As a lower bound on how many layers we need to represent a suitable A, our learned function needs to have at least as many degrees of freedom as the covariance matrix $\Sigma \in \mathbb{R}^{D \times D}$: As it is symmetric Σ has $D(D+1)/2$ independent degrees of freedom. The D linear single-dimensional transforms in each block $f_{\text{dim},i}$ have a total of D degrees of freedom. Thus we need more than $D(D+1)/(2D) = (D+1)/2 \geq D/2$ layers to represent Σ. □

A.2. Proof of Corollary 1

Proof. The proof follows by replacing the number of parameters per layer in Appendix A.1 by $(k+1)D$ instead of D. We find at least $D(D+1)/(2(k+1)D) = (D+1)/(2(k+1)) \geq D/(2(k+1))$ layers to represent Σ. □

A.3. Proof of Theorem 2

Proof. We start with a covariance matrix Σ with $\operatorname{tr} \Sigma = D$ by Assumption 1. Given a fixed rotation Q, the output of the rotation layer is:

$$p(Qx) = \mathcal{N}(0, Q\Sigma Q^T). \tag{24}$$

The marginals of this distribution read:

$$p((Qx)_i) = \mathcal{N}(0, (Q\Sigma Q^T)_{ii}). \tag{25}$$

Each marginal can be transported immediately to a univariate standard normal via:

$$f_{\text{dim},\theta_i}(x_i) = \frac{x_i}{\sqrt{(Q\Sigma Q^T)_{ii}}}. \tag{26}$$

This makes the collected action of the element-wise layer:

$$f_{\text{dim}}(x) = \operatorname{Diag}(Q\Sigma Q^T)^{-1/2} x =: S^{-1/2} x. \tag{27}$$

Here, $S = \operatorname{Diag}(Q\Sigma Q^T)$ collects the diagonal of the rotated covariance.

The output of the layer is again a Gaussian distribution:

$$p(x') = \mathcal{N}(0, \Sigma'), \quad \Sigma' = S^{-1/2} Q\Sigma Q^T S^{-1/2}. \tag{28}$$

Inserting this into the loss in Equation (11), we find:

$$\mathcal{L}' = -\tfrac{1}{2} \log \det \Sigma' \tag{29a}$$
$$= -\tfrac{1}{2} \log \det(S^{-1/2} Q\Sigma Q^T S^{-1/2}) \tag{29b}$$
$$= -\tfrac{1}{2}(\log \det S + \log \det \Sigma) \tag{29c}$$
$$= \mathcal{L} - \tfrac{1}{2} \log \det S. \tag{29d}$$

As we consider random rotations $Q \in O(D)$, we compute the expected loss over rotations:

$$\mathbb{E}_{Q \in O(D)}[\mathcal{L}'] = \mathcal{L} - \tfrac{1}{2} \mathbb{E}_{Q \in O(D)}[\log \det S] \tag{30a}$$
$$= \mathcal{L} - \tfrac{1}{2} \sum_{i=1}^{D} \mathbb{E}_{Q \in O(D)}[\log(Q\Sigma Q^T)_{ii}] \tag{30b}$$
$$\leq \mathcal{L}. \tag{30c}$$

Here, we have used Jensen's inequality and that $\mathbb{E}_{Q \in O(D)}[(Q\Sigma Q^{\mathrm{T}})_{ii}] = \operatorname{tr}\Sigma/D = 1$. (This is a vacuous bound: The expected loss after the layer is at least as good as before the layer.)

We can estimate the error of Jensen's inequality to get an estimate for a lower bound on $\mathbb{E}_{Q \in O(D)}[\mathcal{L}']$. Here, we make use of the result in (Costarelli & Spigler, 2015) applied to $\log$:

$$\mathbb{E}[\varphi(x)] - \varphi(\mathbb{E}[x]) \le \frac{1}{2} \max_{x \in I} \varphi''(x) \operatorname{Var}[x]. \tag{31}$$

In our case, $\varphi(x) = -\log x$, and $x \in [\lambda_{\min}, \lambda_{\max}]$, the extremal eigenvalues of Σ (which are invariant under rotation).

In (Draxler et al., 2022), the authors show that:

$$\operatorname{Var}[(Q\Sigma Q^{\mathrm{T}})_{ii}] = \frac{2}{(D+2)} \operatorname{Var}[\lambda], \tag{32}$$

for the variance of the eigenvalues of Σ, given by $\operatorname{Var}_i[\lambda_i]$.

For us, this means:

$$\mathcal{L} - \mathcal{L}' \le \frac{1}{2\lambda_{\min}^2} \frac{2}{(D+2)} \operatorname{Var}[\lambda]. \tag{33}$$

We make use of following arithmetic mean-geometric mean (AM-GM) inequality by (Cartwright & Field, 1978):

$$\frac{\operatorname{Var}[\lambda]}{2\lambda_{\max}} \le \bar{\lambda} - g \le \frac{\operatorname{Var}[\lambda]}{2\lambda_{\min}}, \tag{34}$$

where g is the geometric mean of the eigenvalues:

$$g := \left(\prod_{i=1}^{D} \lambda_i \right)^{1/D}, \tag{35}$$

and find:

$$\mathcal{L} - \mathcal{L}' \le \frac{1}{2\lambda_{\min}^2} \frac{2}{(D+2)} 2\lambda_{\max}(1-g). \tag{36}$$

As $\lambda_{\max} > 1$, we can multiply the right hand side by $\lambda_{\max}$:

$$\mathcal{L} - \mathcal{L}' \le \frac{1}{2\lambda_{\min}^2} \frac{2}{(D+2)} 2\lambda_{\max}^2(1-g). \tag{37}$$

Then, rewrite using the conditioning number $\kappa = \lambda_{\max}/\lambda_{\min}$:

$$\mathcal{L} - \mathcal{L}' \le \frac{2}{(D+2)} \kappa^2(1-g). \tag{38}$$

Note that one can write the loss $\mathcal{L}$ directly via g and vice versa:

$$\mathcal{L} = -\frac{1}{2} \log g^D = -\frac{D}{2} \log g, \tag{39a}$$
$$g = \exp(-2L/D). \tag{39b}$$

As we want a bound that merely depends on the loss, we upper bound κ using a function of the loss:

$$\max_{\substack{\lambda_1, \dots, \lambda_D \\ \sum_i \lambda_i = D \\ \prod_i \lambda_i^{1/D} = g}} \kappa = \frac{1 + \sqrt{1 - g^D}}{1 - \sqrt{1 - g^D}}. \tag{40}$$

13

Then,

$$\mathcal{L} - \mathcal{L}' \leq \frac{2}{(D+2)} \frac{1 + \sqrt{1 - g^D}}{1 - \sqrt{1 - g^D}}^2 (1 - g). \tag{41}$$

By assumption, we have the limit $\mathcal{L} \ll 0$, which corresponds to $g \to 1$. We find:

$$\mathcal{L}' \geq \left(1 - \frac{2}{D+2}\right)\mathcal{L} - |\mathcal{O}(\mathcal{L})| \tag{42}$$

$\square$

A.4. Proof of Corollary 2

Proof. The proof follows by replacing the number of parameters per layer in Appendix A.1 by $(D/2)^2 + D/2$ instead of D. We find at least $2 > D(D+1)/(2((D/2)^2 + D/2)) = 2(D+1)/(D+2) > 1$ layers to represent Σ. $\square$

A.5. Proof of Theorem 4

We use Theorem 4 in (Draxler et al., 2022). Like Section 4, it concerns centers data:

Assumption 3. *The data $p(x)$ is centered: $\mathbb{E}_{x \sim p(x)}[x] = 0$.*

Theorem 5. *Given D-dimensional data fulfilling Assumptions 1 and 3 with covariance Σ. Then, after L coupling blocks, the expected loss is smaller than:*

$$\mathbb{E}_{Q_1,\dots,Q_L \in O(D)}[\mathcal{S}(\Sigma_L)] \leq \gamma(\mathcal{S}(\Sigma))^L \mathcal{S}(\Sigma), \tag{43}$$

where the convergence rate depends on the non-Standardness before training:

$$\gamma(\mathcal{S}) = 1 + \frac{1}{4\mathcal{S}/D} \log\left(1 - \frac{D^2}{(D-1)(D+2)} \frac{1 - \sqrt{1 - g(\mathcal{S})^D}}{1 + \sqrt{1 - g(\mathcal{S})^D}}(1 - g(\mathcal{S}))\right) < 1. \tag{44}$$

Here, the non-Standardness $\mathcal{S}$ is the loss $\mathcal{L}$ in the Gaussian case, see Equation (11). We write $\mathcal{L}$ in the remainder of this section:

Proof. We start from Theorem 5. We first take the limit of $\gamma(\mathcal{L})$ for $\mathcal{L} \ll 1$ and then $D \gg 1$. We use the computer algebra system sympy to take the limits:

$$\gamma(\mathcal{L}) \xrightarrow{\mathcal{L} \to 0} \frac{D(D+2) - 4}{2(D-1)(D+2)} + \mathcal{O}(\mathcal{L}) \xrightarrow{D \to \infty} \frac{1}{2} + \mathcal{O}(\mathcal{L}) + \mathcal{O}(D^{-1}). \tag{45}$$

$\square$

To further justify the usage of

$$\gamma(\mathcal{L}) \xrightarrow{\mathcal{L} \to 0, D \to \infty} \frac{1}{2}, \tag{46}$$

note that

$$\gamma(\mathcal{L}) \xrightarrow{\mathcal{L} \to 0} \frac{D(D+2) - 4}{2(D-1)(D+2)} \in [1/2, 5/9] \leq 0.555\dots \tag{47}$$

B. Experimental details

B.1. Measuring number of required layers

We proceed as follows to predict the number of required layers to reduce the loss by a factor $\mathcal{L}'/\mathcal{L}$ from training L_{train} layers in all experiments:

1. Determine the entropy $H[p(x)]$ of the data distribution $p(x)$. This is required for computing the KL divergence in Equation (2), and restricts us to distributions $p(x)$ which we can sample from (for training) and where we can evaluate the entropy via:

$$H[p(x)] = -\mathbb{E}_{x \sim p(x)}[\log p(x)] \tag{48}$$

2. Train a fixed number of layers L_{train} iteratively.

3. In general, we observe the loss as a function of depth L to decrease in a geometric series:

$$\mathcal{L}_{L_{\text{train}}} := \mathcal{L}_0 \gamma^{L_{\text{train}}}. \tag{49}$$

We determine γ from the initial loss $\mathcal{L}_0$ and the loss after L_{train} layers:

$$\gamma := \left[\mathcal{L}_{L_{\text{train}}} / \mathcal{L}_0 \right]^{1/L_{\text{train}}} \tag{50}$$

4. We then extrapolate Equation (49) to predict the number of layers required for an arbitrary loss ratio $\mathcal{L}'/\mathcal{L}$:

$$L = \frac{\log(\mathcal{L}'/\mathcal{L})}{\log \gamma} \tag{51}$$

We evaluate Equation (51) throughout the paper for a loss ratio of $\log(\mathcal{L}'/\mathcal{L}) = 1$, that is $\mathcal{L}' = e^{-1}\mathcal{L} = 36.8\%\mathcal{L}$, but this is arbitrary as we are only interested in the scaling with dimension which is independent of $\mathcal{L}'/\mathcal{L}$.

B.2. Gaussian data

As our dataset, we construct the same dataset of Gaussians as Draxler et al. (2022). They construct covariances given a dimension D by choosing the eigenvalues of the covariance. We then normalize for fulfilling Assumption 1 and obtain:

$$\Sigma = \frac{1}{\sum_i \lambda_i} \text{Diag}(\lambda_1, \dots, \lambda_D). \tag{52}$$

We consider the following eigenvalue spectra:

1. Single eigenvalue varying: $\lambda_1 = \alpha, \lambda_{>1} = 1$ for $\alpha \in (\lambda_{\min}, 1/\lambda_{\min}) \setminus \{1\}$.

2. All eigenvalues varying but one: $\lambda_1 = 1, \lambda_{>1} = \alpha$ for $\alpha \in (\lambda_{\min}, 1/\lambda_{\min}) \setminus \{1\}$.

3. Half small, half big: $\lambda_{\leq D/2} = \alpha, \lambda_{>D/2} = 1/\alpha$ for $\alpha \in (\lambda_{\min}, 1) \setminus \{1\}$.

4. Randomly sample $\lambda_i \sim [0, 2]$ uniformly.

5. Randomly sample $\lambda_i \sim [\lambda_{\min}, 1/\lambda_{\min}]$ log-uniformly.

We choose D in 10 geometrically spaced values from 10 to 128. We choose $\lambda_{\min} = 10^{-3}$. The case where all eigenvalues are equal to 1, $\lambda_i = 1$, is excluded, as Gaussianization has converged at this point.

For each Σ_i in the resulting data set, we create N_{rot} differently rotated variants $\Sigma_i^r = Q_r^\top \Sigma_i Q_r$, where we choose $N_{\text{rot}} = 8$, whatever is larger. This corresponds to the initial unknown rotation of the data.

We then apply the analytic solution of a Gaussianization block on the covariance, given by Equation (28). Then, another N_{rot} rotations are drawn, which rotate each resulting covariance. This procedure is repeated until the specified number of layers is reached. We use $L_{\text{train}} = 8D$ layers as we expect the number of required layers to increase with $\Omega(D)$.

To evaluate Theorem 2, we compare Equation (13) with the empirical result. To estimate the required number of layers for a fixed loss ratio, we proceed as in Appendix B.1 so that we end up in the regime where $\mathcal{L} \ll 1$. For Figure 1, we fit γ from the loss ratio of the last two layers: $\gamma = \sqrt{\mathcal{L}_{L_{\text{train}}}/\mathcal{L}_{L_{\text{train}}-2}}$ instead of Equation (50). The maximum loss in this case is 10^{-2}, so we are in the regime of $\mathcal{L} \ll 1$.

In Figure 8, we observe the same linear scaling behavior of required layers with dimension for $\mathcal{L} \gg 0$, i.e. at the beginning of training. However, in this case, the bound in Equation (13) is violated in few scenarios: They require less layers than predicted. However, these cases eventually reach the case of $\mathcal{L} \ll 1$ and also the linear scaling with dimensions is correct.
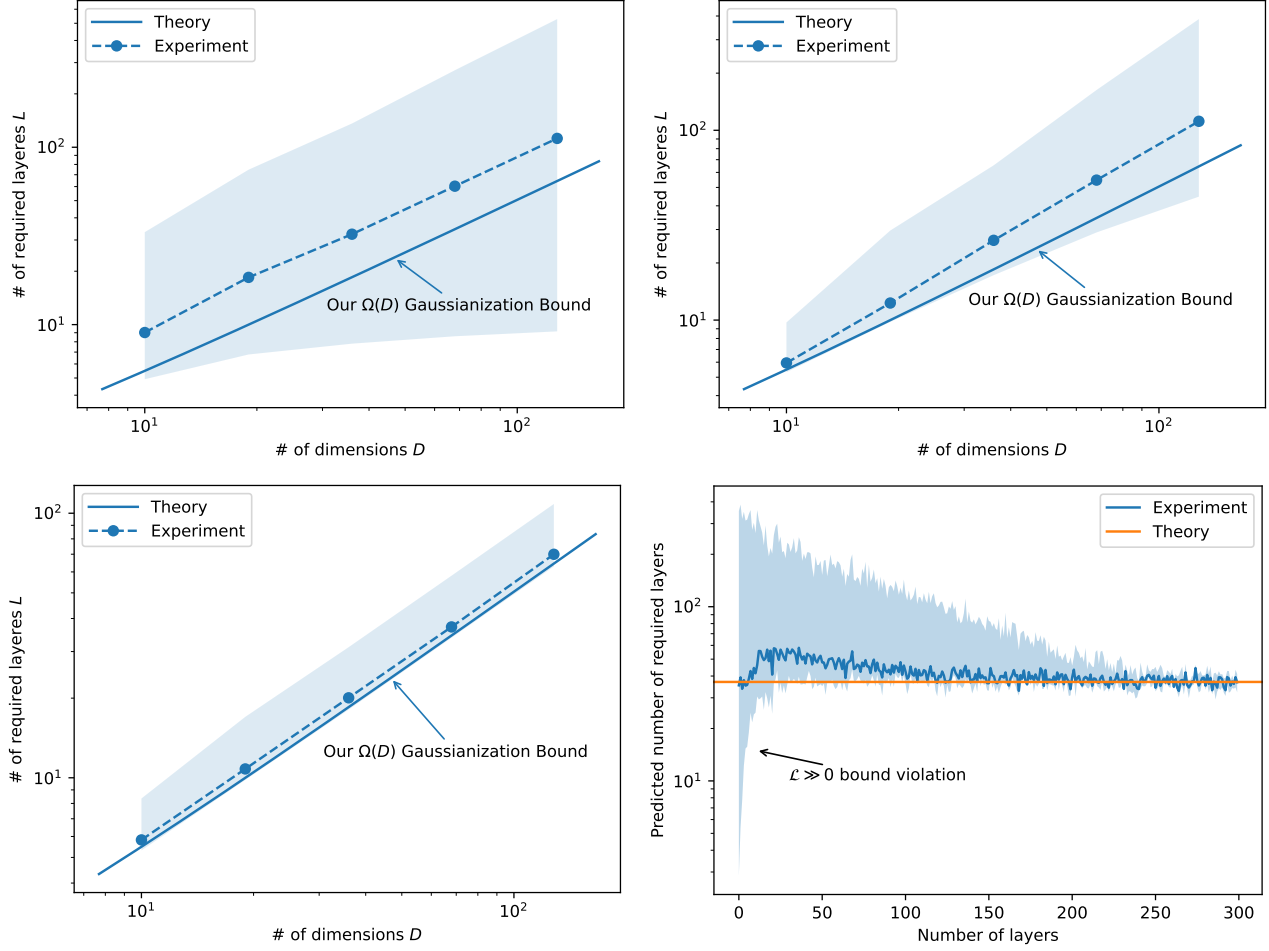
Figure 8. **For larger loss $\mathcal{L} \gg 0$, the median number of required layers in the data set is larger than predicted. Some cases require less layers, but the scaling behavior is still right.** *(Top left)* If we predict the number of required layers from the first $L_{\text{train}} = D$ layers, the majoritoy of cases show slower convergence than predicted. Some cases show faster convergence, but show the same scaling behavior. *(Top right)* The same holds if we fit the number of required layers from the first $L_{\text{train}} = 3D$ layers. *(Bottom left)* The bound is valid for at least 75% of the training data for $L_{\text{train}} = 10D$ layers. *(Right)* We identify the cases that are faster than predicted as originating from "All eigenvalues varying but one", where for the first about 15 layers there are cases converging faster than Equation (13). All averaging is performed via the median and shaded regions cover 90% of the cases.

B.3. Finding spurious dimensions in standard normal data

We randomly sample $N = 60,000$ normal samples of dimension $D = 3072$, which corresponds to the size of the CIFAR10 dataset. We optimize $w \in \mathbb{R}^D, \|w\| = 1$ to minimize the 1-dimensional sample-based 2-Wasserstein distance to a predefined bimodal distribution $p_{\text{adv}} = \left(\mathcal{N}(-d/2, \sigma) + \mathcal{N}(d/2, \sigma)\right)/2$:

$$W_2^2 = \sum_{i=1}^{N} \left((w^\top x)_{\pi_w(i)} - y_i\right)^2. \tag{53}$$

Here, y_i are sorted samples from p_{adv}. The permutation $\pi_w : [N] \to [N]$ sorts the projected values $(w^\top x)$.

We choose the spread of the bimodal distribution to be $d = 2$ and the standard deviation of each mode as $\sigma = 0.4$.

We optimize w for 64 steps using SGD with a learning rate of 10 and momentum .8. After each update, we rescale $\|w\| = 1$. The final Wasserstein distance we obtain reads 0.03, down from 0.1 for a random w.

For the visualization in Figure 4, we project the data once with a random $w_{\text{rand}}^\top x$ and once with $w^\top x$ into 70 bins.

B.4. Gaussianization implementation

Following Equation (3), we construct f_{rot} as a random rotation $f_{rot}(x) = Qx$ where $Q \sim O(D)$ for data dimensionality D. For f_{dim}, we choose rational-quadratic splines, which allow for approximation of arbitrary functions by separating their domain into b bins. Given bin edges (or knots) $x^{(k)}, y^{(k)}, x^{(k+1)}, y^{(k+1)}$ and derivatives $\delta^{(k)}, \delta^{(k+1)}$ at those edges for bin $k = 1, ..., b$, they can be interpolated with a rational-quadratic polynomial as described in (Durkan et al., 2019). Beyond the outermost bin edges, the spline is extrapolated with linear tails.
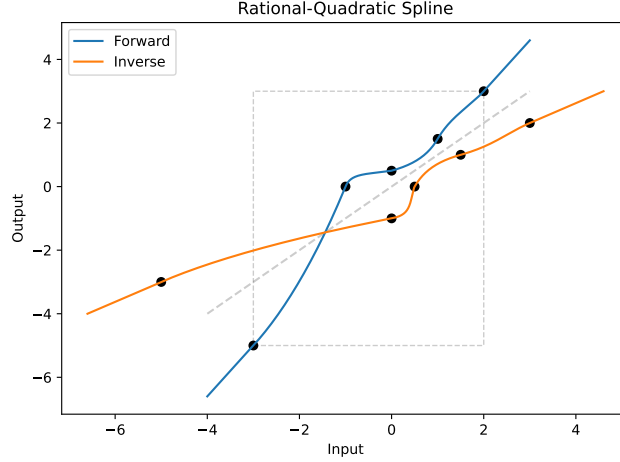

Figure 9. **Rational-Quadratic Spline.** Each pair of knots (black) is interpolated with a rational-quadratic polynomial. The outside is extrapolated with linear tails.

We use an implementation of RQ splines based on (Dai & Seljak, 2021), where $\psi(x, \alpha) = (1 - \alpha)RQ(x) + \alpha x$ with a scalar regularization parameter α. We choose $b = 128$ bins, as well as $\alpha_1 = 0.9$ for the spline and $\alpha_2 = 0.99$ for the linear extrapolation, such that

$$f_{dim, \theta_i}(x_i) = \begin{cases} \psi(x_i, \alpha_1) & x_i^{(1)} \leq x_i \leq x_i^{(b+1)} \\ \psi(x_i, \alpha_2) & \text{otherwise} \end{cases}. \tag{54}$$

The $\alpha_{1,2}$ significantly slow down training, but increase performance (Dai & Seljak, 2021). It should not alter the scaling behavior of the number of required layers with dimension, up to a constant factor independent of dimension.

Splines are fit to the CDF of the data by evenly distributing bin knots on the quantiles of the data and applying the inverse Gaussian CDF:

$$x_i^{(k)} = q\left(\frac{k}{b+2}\right), \tag{55a}$$

$$y_i^{(k)} = G^{-1}\left(x_i^{(k)}\right), \tag{55b}$$

where $q(p) = \sqrt{2}\,\mathrm{erf}^{-1}(2p - 1)$ is the quantile function and $G(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$ is the standard normal CDF. The inner derivatives $\delta^{(k)}$ can then be estimated by finite differences, following (Durkan et al., 2019):

$$h_i^{(k)} = x_i^{(k+1)} - x_i^{(k)}, \tag{56a}$$

$$s_i^{(k)} = \frac{y_i^{(k+1)} - y_i^{(k)}}{x_i^{(k+1)} - x_i^{(k)}}, \tag{56b}$$

$$\delta_i^{(k+1)} = \frac{s_i^{(k)}h^{(k+1)} + s^{(k+1)}h^{(k)}}{h^{(k+1)} + h^{(k)}}, \quad k = 1, ..., b - 1. \tag{56c}$$

We choose identity tails, i.e. $\delta^{(1)} = \delta^{(b+1)} = 1$.
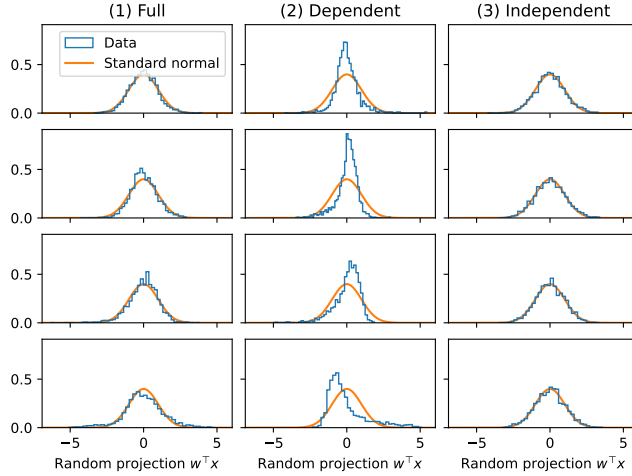
17

Figure 10. Random projections of data from the different versions of toy data in $D = 128$. Columns show the different cases, and each row depicts a different random projection $w \in \mathbb{R}^D, \|w\| = 1$. In case (2, *center*), most dimensions depend on the same dimensions, making random projections deviate strongly from a standard normal. For case (1) and (3), random projections are very close to a Gaussian, preventing a fast fit using Gaussianization.

B.5. Toy scaling experiment

Toy data distribution Our goal is to create a family of distribution $p(x)$ which naturally extend over different dimensions D by the continous mixture of Gaussians Equation (22) where the mean of each dimension is conditioned on the previous via $m_i(x_j \in A_i)$. In particular, we choose:

$$m_1 = \frac{1}{2}, \quad m_0 = 0, \quad \sigma_1^2 = 0.8, \quad \sigma_2^2 = 0.2. \tag{57}$$

The $s_{ij}(D)$ are drawn randomly from $\{-1, +1\}$ for each dimension D and for each seed. This is the main source of noise between runs.

Training We employ the architecture described in Appendix B.4 for 64 layers. We use $N = 60,000$ training samples. We resample them for some runs after training to see if overfitting is a problem. We find that it is not and run the remaining runs without resampling, yielding a total of four runs per case (except for case 1, for which we average over eight runs). Before training, the data is normalized to zero mean and unit standard deviation.

Evaluation We compute the number of required layers for each run via the procedure in Appendix B.1. This number is then averaged over all runs for each case and dimension.

B.6. Multi-scale EMNIST experiment

Data distribution As described in Section 5.2, we make use of a normalizing flow as our data density $p(x)$. Each flow architecture consists of a fixed normalization layer, followed by 20 affine coupling blocks. We use purely full-connected networks for odd scales, and convolutional networks for even scales. We use wavelet downsampling before the first. If the image side length scale is a power of 4, we add a second wavelet downsampling after the eigth affine coupling layer. Each convolutional subnetwork uses two hidden layers with 16 channels respectively 32 channels after the second downsampling each, and a kernel size of 3. The final 4 coupling blocks are fully connected. Each fully connected subnetwork has two hidden layers with hidden width equal to the total number of dimensions D. The details for each architecture are given in Table 1.

We train the normalizing flows for 30 epochs for $D = 28 \times 28$, and 20 for the other scales using negative log-likelihood, see Equation (2). We use Adam with a learning rate of 10^{-3} and a batch size of 256.

After training, we replace the latent distribution with a normal distribution $\mathcal{N}(0, \hat{\sigma}^2 I_D)$ with $\hat{\sigma} = 0.8$ to achieve a better
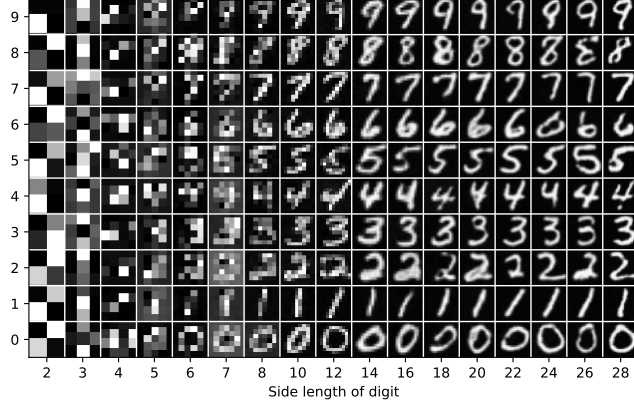
Figure 11. Samples from our down-scaled EMNIST normalized models which we use as ground truth distributions $p(x)$. Normalizing Flows are sampled with reduced temperature 0.8.

sample quality. Note that this does not influence the ability to compute density estimates $p(x)$ from our model, but it does reduce the entropy of the data.

Scale	Network type	# of downsamplings	# of parameters
$4 = 2 \times 2$	conv	1	149k
$9 = 3 \times 3$	fc	0	7k
$16 = 4 \times 4$	conv	2	194k
$25 = 5 \times 5$	fc	0	38k
$36 = 6 \times 6$	conv	1	164k
$49 = 7 \times 7$	fc	0	134k
$64 = 8 \times 8$	conv	2	235k
$100 = 10 \times 10$	conv	1	254k
$144 = 12 \times 12$	conv	2	406k
$196 = 14 \times 14$	conv	1	544k
$256 = 16 \times 16$	conv	2	861k
$324 = 18 \times 18$	conv	1	1M
$400 = 20 \times 20$	conv	2	2M
$484 = 22 \times 22$	conv	1	3M
$576 = 24 \times 24$	conv	2	4M
$676 = 26 \times 26$	conv	1	5M
$784 = 28 \times 28$	conv	2	6M

Table 1. Normalizing Flow architecture as a function of image size. A purely fully-connected network is labeled by "fc", "conv" networks are partially convolutional.

Traininig Like in the toy experiment, we use the implementation from Appendix B.4. We again choose $N = 60,000$ training samples without resampling. We average each case over 10 runs.

Evaluation We use the same procedure as in Appendix B.5.

B.7. Compute and libraries

Experiments were performed on three workstations, each with a single high-end consumer GPU and CPU each. We build our code upon the following python libraries: PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019), Lightning Trainable (Kühmichel & Draxler, 2023), Tensorflow (Abadi et al., 2015) for FID score evaluation, Numpy (Harris et al., 2020), Matplotlib (Hunter, 2007) for plotting and Pandas (Wes McKinney, 2010; The pandas development team, 2020) for data evaluation.