

---

# Resampling Base Distributions of Normalizing Flows

---

Vincent Stumper<sup>1,2</sup>

<sup>1</sup>Max Planck Institute for  
Intelligent Systems, Germany

Bernhard Schölkopf<sup>1</sup>

José Miguel Hernández-Lobato<sup>2</sup>  
<sup>2</sup>University of Cambridge,  
United Kingdom

## Abstract

Normalizing flows are a popular class of models for approximating probability distributions. However, their invertible nature limits their ability to model target distributions whose support have a complex topological structure, such as Boltzmann distributions. Several procedures have been proposed to solve this problem but many of them sacrifice invertibility and, thereby, tractability of the log-likelihood as well as other desirable properties. To address these limitations, we introduce a base distribution for normalizing flows based on learned rejection sampling, allowing the resulting normalizing flow to model complicated distributions without giving up bijectivity. Furthermore, we develop suitable learning algorithms using both maximizing the log-likelihood and the optimization of the Kullback-Leibler divergence, and apply them to various sample problems, i.e. approximating 2D densities, density estimation of tabular data, image generation, and modeling Boltzmann distributions. In these experiments our method is competitive with or outperforms the baselines.

## 1 INTRODUCTION

Inferring and approximating probability distributions is a central problem of unsupervised machine learning. A popular class of models for this task are normalizing flows (Tabak and Vanden-Eijnden, 2010; Tabak and Turner, 2013; Rezende and Mohamed, 2015), which are given by an invertible map transforming a simple base distribution such as a Gaussian to obtain a

complex distribution matching our target. Normalizing flows have been applied successfully to a variety of problems, such as image generation (Dinh et al., 2015, 2017; Kingma and Dhariwal, 2018; Ho et al., 2019; Grcić et al., 2021), audio synthesis (van den Oord et al., 2018), variational inference (Rezende and Mohamed, 2015), semi-supervised learning (Izmailov et al., 2020) and approximating Boltzmann distributions (Noé et al., 2019; Wu et al., 2020; Wirnsberger et al., 2020) among others (Papamakarios et al., 2021). However, with respect to some performance measures they are still outperformed by autoregressive models (Chen et al., 2018; Parmar et al., 2018; Child et al., 2019), generative adversarial networks (GANs) (Gulrajani et al., 2017; Karras et al., 2019, 2020a,b), and diffusion based models (Sohl-Dickstein et al., 2015; Kingma et al., 2021). One reason for this is an architectural limitation. Due to their bijective nature the normalizing flow transformation leaves the topological structure of the support of the base distribution unchanged and, since it is usually simple, there is a topological mismatch with the often complex target distribution (Cornish et al., 2020), thereby diminishing the modeling performance and even causing exploding inverses (Behrmann et al., 2021). Several solutions have been proposed, e.g. augmenting the space the model operates on (Huang et al., 2020), continuously indexing the flow layers (Cornish et al., 2020), and adding stochastic (Wu et al., 2020) or surjective layers (Nielsen et al., 2020). However, these approaches sacrifice the bijectivity of the flow transformation, which means in most cases that the model is no longer tractable, memory savings during training are no longer possible (Gomez et al., 2017), and the model is no longer a perfect encoder-decoder pair. Some work has been done on using multimodal base distributions (Izmailov et al., 2020; Ardizzone et al., 2020; Hagemann and Neumayer, 2021), but the intention was to do classification or solve inverse problems with flow-based models and not to capture the inherent multimodal nature of the target distribution. Papamakarios et al. (2017) took a mixture of Gaussians as base distribution and showed that this can improve

the performance.

In this work, we develop a method to obtain a more expressive base distribution through learned accept/reject sampling (LARS) (Bauer and Mnih, 2019). It can be estimated jointly with the flow map by either maximum likelihood (ML) learning or minimizing the Kullback-Leibler (KL) divergence, matching the topological structure of the target’s support. Moreover, we propose how the method can be scaled up to high dimensional datasets and demonstrate the effectiveness of our procedure on the tasks of learning 2D densities, estimating the density of tabular data, generating images, and the approximation of a 22 atom molecule’s Boltzmann distribution.

## 2 BACKGROUND

### 2.1 Normalizing Flows

Let  $z$  be a random variable taking values in  $\mathbb{R}^d$ , having the density  $p_\phi(z)$  parameterized by  $\phi$ . Furthermore, let  $F_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$  be a bijective map parameterized by  $\theta$ . We can compute the tractable density of the new random variable  $x := F_\theta(z)$  with the change of variables formula

$$p(x) = p_\phi(z) |\det(J_{F_\theta}(z))|^{-1}, \quad (1)$$

where  $J_{F_\theta}$  is the Jacobian matrix of  $F_\theta$ . This way of constructing a complex probability distribution  $p(x)$  from a simple *base distribution*  $p(z)$  is called a *normalizing flow*. We can use them to approximate a target density  $p^*(x)$ , which is done by optimizing a training objective. If the target density is unknown but samples from the corresponding distribution are available, we maximize the expected log-likelihood (LL) of the model

$$\text{LL}(\theta, \phi) = \mathbb{E}_{p^*(x)} [\log(p(x))]. \quad (2)$$

Conversely, if the target density is given, we minimize the (reverse) KL divergence<sup>1</sup> (Papamakarios et al., 2021)

$$\text{KLD}(\theta, \phi) := \mathbb{E}_{p(x)} [\log p(x)] - \mathbb{E}_{p(x)} [\log p^*(x)], \quad (3)$$

or another difference measure for probability distributions such as the  $\alpha$ -divergence (Hernández-Lobato et al., 2016).

To deal with high dimensional data, such as images, the multiscale architecture was introduced by Dinh et al. (2017). As sketched in Figure 5, at the first level, the entire input  $x$  is transformed by several flow layers  $F_1$ . The result is split up into two parts,  $h_1^{(1)}$  and  $h_1^{(2)}$ .

<sup>1</sup>For simplicity, we will call it just *KL divergence* from now on.

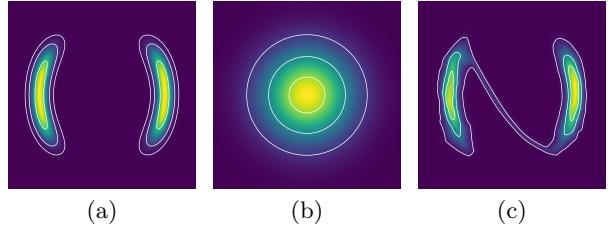


Figure 1: Illustration of the architectural limitation of normalizing flows. (a) depicts the multimodal target distribution, (b) the Gaussian base distribution used, and (c) the learned real NVP model. The model’s support has one connected component with a density filament between the modes.

For images, this is typically done by first squeezing the image, i.e. reducing its height and width by a factor 2 and adding the surplus pixels as additional channels, and then splitting the resulting tensor along the channel dimension.  $h_1^{(1)}$  is immediately factored out in the density, while  $h_2^{(2)}$  is further transformed by the next set of flow layers  $F_2$ . The process is then repeated until a desired depth is reached. The full density for a multiscale architecture with  $n$  levels is given by

$$p(x) = \prod_{i=1}^n |\det(J_{F_i}(h_{i-1}))| p(z_i), \quad (4)$$

where we set  $h_0 = x$ .

Normalizing flows can compete with other machine learning models on many benchmarks (Papamakarios et al., 2021). However, their performance is still impaired by an architectural weakness. The transformations defining a normalizing flow are invertible and such maps leave the topology of the sets they map unchanged (Runde, 2005). Consequently, the topological structure of the support of  $p(z)$  is the same as that of  $p(x)$ . Usually, the base distribution is a Gaussian, which has only one mode, so its support consists of one connected component but the target distribution might be multimodal with the density between the modes being close to zero or even numerically zero due to finite precision so that the support consists of multiple disconnected components. As an exemplification we fit a real-valued non-volume preserving (real NVP) flow model with 8 coupling layers to a multimodal target distribution, see Figure 1. The density of the trained model consists of one connected component covering the modes of the target, but connecting them via a density filament. Certain flow-based models, such as the residual flow (Behrmann et al., 2019; Chen et al., 2019), can only converge to the target if they become non-invertible due to the topological mismatch (Cornish et al., 2020), thereby causing unstable

training behaviour (Behrmann et al., 2021). Proposed solutions include increasing the model size significantly (Chen et al., 2019), but this increases the computational cost and memory demand while the stability issues persist. Training can be stabilized via a suitable regularization, but this reduces the performance (Behrmann et al., 2021). Other approaches are discussed in Section 5.

## 2.2 Learned accept/reject sampling

Learned accept/reject sampling (LARS) is a method to approximate a  $d$ -dimensional distribution  $q(z)$  by reweighting a proposal distribution  $\pi(z)$  through a learned acceptance function  $a_\phi : \mathbb{R}^d \rightarrow [0, 1]$ , where  $\phi$  are the learned parameters (Bauer and Mnih, 2019). Given a sample  $z_i$  from  $\pi$ , we will accept it with a probability  $a_\phi(z_i)$ , otherwise we reject it and draw a new sample until we accept one of the proposed samples. The resulting distribution is given by

$$p_\infty(z) = \frac{\pi(z)a_\phi(z)}{Z}; \quad Z := \int \pi(z)a_\phi(z)dz. \quad (5)$$

In order to limit the computational cost caused by high rejection rates, Bauer and Mnih (2019) introduced a truncation parameter  $T \in \mathbb{N}$ . If the first  $T - 1$  samples from the proposal get rejected, we accept the  $T^{\text{th}}$  sample no matter the value of the learned acceptance probability. Through this intervention, we alter the final sampling distribution to become

$$p_T(z) = (1 - \alpha_T) \frac{a_\phi(z)\pi(z)}{Z} + \alpha_T\pi(z), \quad (6)$$

where  $\alpha_T := (1 - Z)^{T-1}$ , which reduces to (5) for  $T \rightarrow \infty$ . The integral (5) defining  $Z$  is not tractable, so we cannot compute it directly. Instead, it is estimated via Monte Carlo sampling, i.e.

$$Z \approx \frac{1}{S} \sum_{s=1}^S a_\phi(z_s), \quad (7)$$

where  $z_s \sim \pi(z)$ , which needs to be recomputed in every training iteration, as parameter changes in  $a_\phi$  cause a change in  $Z$ .

LARS was first used to create a more expressive prior for variational autoencoders (VAEs) (Kingma and Welling, 2014), making it closer to the aggregate posterior distribution, thereby bringing the approximate posterior distribution closer to the ground truth. The resampled priors are trained jointly with the likelihood and the approximate posterior via maximization of the evidence lower bound. Since this only requires to evaluate the density of the prior at the data points, it is not even required to perform rejection sampling during training; therefore, the computational cost of training the whole model is only increased slightly.

## 3 METHOD

### 3.1 Resampled base distributions

In Section 2.1, we argued that the topological structure of the support of the base distribution equals that of the overall flow distribution. To avoid artefacts resulting from mismatches between them, we aim to make the latter closer to the former. Therefore, we resample the base distribution with LARS, i.e. use it as our proposal so that its density becomes (6). Since there are no restrictions on the acceptance function  $a_\phi$ , we can use an arbitrarily complex neural network to model any desired topological structure. The resulting log-probability of the model is given by

$$\log p(x) = \log \pi(z) + \log \left( \alpha_T + (1 - \alpha_T) \frac{a_\phi(z)}{Z} \right) - \log |\det J_{F_\theta}(z)|, \quad (8)$$

where  $F_\theta$  is the flow transformation, i.e. the composition of all flow layers, and  $z = F_\theta^{-1}(x)$ . In our case, the proposal is a Gaussian but it could be any other distribution or a more complicated model, such as a mixture of Gaussians or an autoregressive model. Depending on the application,  $a_\phi$  will be a fully connected or a convolutional neural network, and details about how the architecture can be chosen are given in Appendix C.1. Since the evaluation of  $a_\phi$  can be parallelized over the number of dimensions of the data  $d$ , we only add a constant computational overhead to our model. In contrast, autoregressive models scale linearly with  $d$ . We can sample from the model by performing LARS and propagating the accepted values through the flow map. The rejection rate, and hence the sampling speed, can be controlled via the truncation parameter  $T$ , which we set to 100 in our experiments unless otherwise stated, but also through adding  $Z$  to our loss function, which is discussed in Appendix C.2.

Usually, the base distribution of normalizing flows has mean and variance parameters being trained with the flow layer parameters. Our proposal is simply a standard normal distribution, i.e. a diagonal Gaussian with mean zero and variance one. Thereby, we ensure that the samples from the proposal, which are the input for the neural network representing the learned acceptance probability  $a_\phi$ , come from a distribution which does not change during training. Instead, the mean and variance of the distribution can be altered after the resampling process by applying an affine flow layer with scale and shift being learnable parameters.

Note that while we retain the invertibility of the flow, the probability distribution (8) cannot be evaluated exactly since  $Z$  needs to be estimated via (7). However, for large  $T$  the base distribution reduces to (5)

and, hence, we are only off by a constant meaning there would not be a bias when doing importance sampling, which is crucial for applications such as Boltzmann generators, see Section 4.4.

### 3.2 Learning algorithms

The resampled base distribution can be trained jointly with the flow layers of our model. Both, the expected LL and the KL divergence, can be used as objectives. The former corresponds to maximizing (2), which is done via stochastic gradient decent. As done by Bauer and Mnih (2019), we sample from the proposal in each iteration to estimate the gradient of  $Z$  with respect to the parameters, see (7). To stabilize training, we estimate the value of the normalization constant by an exponential moving average, see Appendix A.1 for more details.

When the unnormalized target density  $\hat{p}^*(x)$  is known, we can use the KL divergence (3) as our objective. However, because sampling from the base distribution includes an acceptance/rejection step, we cannot apply the reparameterization trick (Kingma and Welling, 2014) to obtain the gradients with respect to the model parameters. Instead, we derive an expression of the gradients of the KL divergence similar to that introduced by Grover et al. (2018).

**Theorem 1.** *Let  $p_\phi(z)$  be the base distribution of a normalizing flow, having parameters  $\phi$ , and  $F_\theta$  be the respective invertible mapping, depending on its parameters  $\theta$ , such that the density of the model is*

$$\log(p(x)) = \log(p_\phi(z)) - \log|\det J_{F_\theta}(z)|, \quad (9)$$

with  $x = F_\theta(z)$ . Then, the gradients of the KL divergence with respect to the parameters are given by

$$\nabla_\phi \text{KLD}(\theta, \phi) = \text{Cov}_{p_\phi(z)} \left\{ \nabla_\phi \log p_\phi(z), \log(p_\phi(z)) - \log|\det J_{F_\theta}(z)| - \log \hat{p}^*(F_\theta(z)) \right\} \quad (10)$$

$$\nabla_\theta \text{KLD}(\theta, \phi) = -\mathbb{E}_{p_\phi(z)} [\nabla_\theta (\log \hat{p}^*(F_\theta(z)) + \log|\det J_{F_\theta}(z)|)] \quad (11)$$

The proof is given in Appendix A.2. We will use (10) and (11) to compute the gradients of the KL divergence in our experiments and, thereby, demonstrate its effectiveness.

### 3.3 Application to multiscale architecture

LARS cannot be applied to very high dimensional distributions because we have to estimate  $Z$  and its gradients via Monte Carlo sampling and the number of samples needed grows exponentially with the number of dimensions (Bauer and Mnih, 2019). Although the

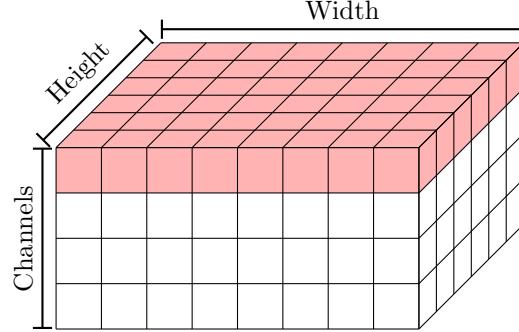


Figure 2: Visualization of a feature map when processing an image in a machine learning model. The unit which is used for factorization in our resampled base distribution is shown in red.

base distribution of a normalizing flow must have the same number of dimensions as the target, we can reduce the number of dimensions significantly by factorization. Therefore, we extend the multiscale architecture, see Section 2.1 and (Dinh et al., 2017), by further subdividing the base distribution at each level into factors with less than 100 dimensions. First, we squeeze the feature map until the product of height and width is smaller than 100. Then, each channel is treated as a separate factor, see Figure 2. To reduce the complexity of the model, we use parameter sharing to express the distribution of factors, i.e. there is one neural network per level with multiple outputs, each representing the acceptance probability  $a_\phi$  for one channel. This also has the advantage that we can estimate the normalization constant and its gradient of all factors of one level in parallel by sampling from a Gaussian, passing the samples through the neural network and computing the average for each output dimension separately. As mentioned in Section 3.1, the mean and variance is added via a constant coupling layer. Furthermore, the base distribution can be made class-conditional by making the mean and variance and/or  $a_\phi$  dependent on the class. The latter can be efficiently achieved by adding more outputs to the neural network to have one value for  $a_\phi$  per class and distribution if needed.

## 4 EXPERIMENTS

### 4.1 2D distributions

In this section, we aim to demonstrate that our method is indeed capable of modeling complicated distributions. Our code for all experiments is publicly available on GitHub at <https://github.com/VincentStimper/resampled-base-flows>.

We start with simple 2D distributions having supports with various topological structures, i.e. a distribution

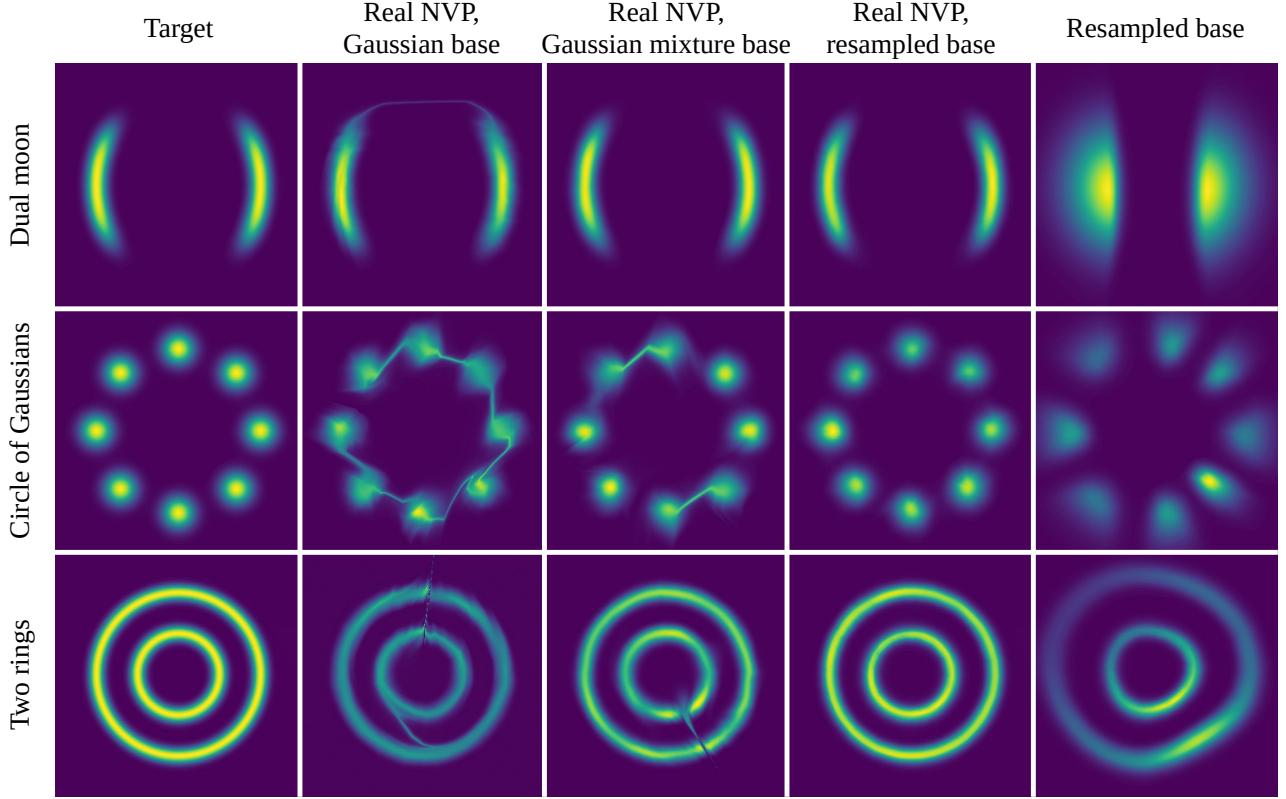


Figure 3: Visualization of the real NVP densities as well as the learned resampled base distribution when approximating three 2D distributions with complex topological structure. The models are trained via ML learning.

Table 1: KL divergences of the target distribution and the flow models which are trained to approximate the three 2D distributions, shown in Figure 3, with ML learning. For each target distribution and flow architecture, the model with the lowest KL divergence is marked in **bold**.

Flow architecture Base distribution	Real NVP Gaussian	Real NVP Mixture	Real NVP Resampled	Residual Gaussian	Residual Mixture	Residual Resampled
Dual moon	1.83	1.80	<b>1.77</b>	1.82	1.80	<b>1.76</b>
Circle of Gaussians	0.090	0.060	<b>0.043</b>	0.045	0.042	<b>0.039</b>
Two rings	10.7	10.6	<b>10.4</b>	11.7	10.8	<b>10.4</b>

with two modes, one with eight modes, and one with two rings, see Figure 3 and Table 8. We use both learning algorithms discussed in Section 3.2. To train our flows via ML, we draw samples from our distributions via rejection sampling. As flow architectures, we choose real NVP (Dinh et al., 2017) and residual flow (Behrmann et al., 2019; Chen et al., 2019) with 16 layers each. For each flow architecture, we train models with a Gaussian, a mixture of 10 Gaussians, and a resampled base distribution, having a Gaussian proposal and a neural network with 2 hidden layers with 256 hidden units each as well as a sigmoid output function as acceptance probability.

The densities of the trained real NVP and residual flow models are show in the Figures 3 and 9, respectively. With a Gaussian base distribution, the flows struggle to model the complex topological structure. For the trained real NVP models it is especially visible in Figure 3 that the density essentially consists of one connected component since there are density filaments between the modes and rings are not closed. The multimodal distributions can be fitted much better when using a mixture of Gaussians as base distribution, but especially the ring distribution can still not be represented properly. With a resampled base distribution the flow models the target distributions accurately without any artefacts. The base distribu-

Table 2: LL on the test sets of the respective datasets of NSF, its CIF variant, and a NSF with a resampled base distribution (RBD). The values are averaged over 3 runs each and the standard error is given as a measure of uncertainty. The highest values within the confidence interval are marked in **bold**.

Method	Power	Gas	Hepmass	Miniboone
NSF	<b><math>0.69 \pm 0.00</math></b>	$13.01 \pm 0.02$	$-14.30 \pm 0.05$	$-10.68 \pm 0.06$
CIF-NSF	<b><math>0.68 \pm 0.01</math></b>	$13.08 \pm 0.00$	<b><math>-13.83 \pm 0.10</math></b>	$-9.93 \pm 0.06$
RBD-NSF (ours)	<b><math>0.69 \pm 0.01</math></b>	<b><math>13.29 \pm 0.05</math></b>	<b><math>-14.02 \pm 0.12</math></b>	<b><math>-9.45 \pm 0.03</math></b>

tions assume the respective topological structure of the target while the flow transformation does the fine adjustment of the density. We also estimate the KL divergences of the target and the model distributions which are listed in Table 1. In all cases the flow model with the resampled base distribution outperforms the respective baselines.

Moreover, we train real NVP models with Gaussian and resampled base distributions with the KL divergence using the gradient estimators derived in Theorem 1. The same architecture as the models trained with ML learning are used and their resulting densities are shown in Figure 7. In addition, we also computed the KL divergences listed in Table 3. As for the previous experiments, the flow with the resampled base distribution clearly outperforms its baseline visually and quantitatively for all the three targets.

Table 3: KL divergences of the target distribution and the models which were trained using the KL divergence, shown in Figure 7. For each target distribution, the real NVP model with the lower KL divergences is marked in **bold**.

Base distribution	Gaussian	Resampled
Dual moon	1.844	<b>1.839</b>
Circle of Gaussians	0.167	<b>0.122</b>
Two rings	11.5	<b>10.3</b>

## 4.2 Tabular data

Next, we estimate the density of four tabular datasets from the UCI Machine Learning Repository (Dheeru and Taniskidou, 2022). We use the same preprocessing and training, validation, and test splits as Papamakarios et al. (2017), which have been adopted by others in the field (Durkan et al., 2019; Cornish et al., 2020). For each dataset, we train a Neural Spline Flow (NSF) (Durkan et al., 2019), its continuously indexed (CIF) variant (Cornish et al., 2020), and one with a resampled base distribution. The LL of the models are shown in Table 2. More details about the setup and the architecture as well as results for real NVP flows on the same datasets are given in Appendix E.

There is no significant performance difference of the three methods on the power dataset. On Hepmass, the resampled base distributions achieves similar performance to CIF, while both are better than the vanilla NSF. For the Gas and Miniboone dataset, the flow with the resampled base distribution clearly outperforms its baselines. When using real NVP, the difference is even larger on all datasets but Miniboone, as can be seen in Table 9.

## 4.3 Image generation

To model images with our method, we train Glow (Kingma and Dhariwal, 2018) on the CIFAR-10 dataset (Krizhevsky, 2009). We use the multiscale architecture introduced in Section 3.3, where we compare a Gaussian with a respective resampled base distribution. As done by Kingma and Dhariwal (2018), we use 3 levels, but train models with 8, 16, and 32 layers per level with each base distribution, with more details provided in Appendix F. For each model architecture, we do three seeded training runs and report bits per dimension on the test set in Table 4.

Table 4: Bits per dimension on the test set of the Glow models with Gaussian and resampled base distribution trained on CIFAR-10. For each architecture, three seeded training runs were done, the reported bits per dimension values are averages over these runs and the standard error is given as an uncertainty estimate. For each number of layers, the lowest values within the confidence interval is marked in **bold**.

Base distribution	Gaussian	Resampled
8 layers per level	$3.403 \pm 0.002$	<b><math>3.399 \pm 0.001</math></b>
16 layers per level	$3.339 \pm 0.001$	<b><math>3.332 \pm 0.001</math></b>
32 layers per level	<b><math>3.283 \pm 0.002</math></b>	<b><math>3.282 \pm 0.001</math></b>

The flow with the resampled base distribution outperforms the baseline when using 8 or 16 layers per level, while performing about equal with 32 layers. The difference is larger for smaller models, i.e. those where fewer layers are used, since models with many layers are already rather expressive. Using a more expressive base distribution also increases the model size and the

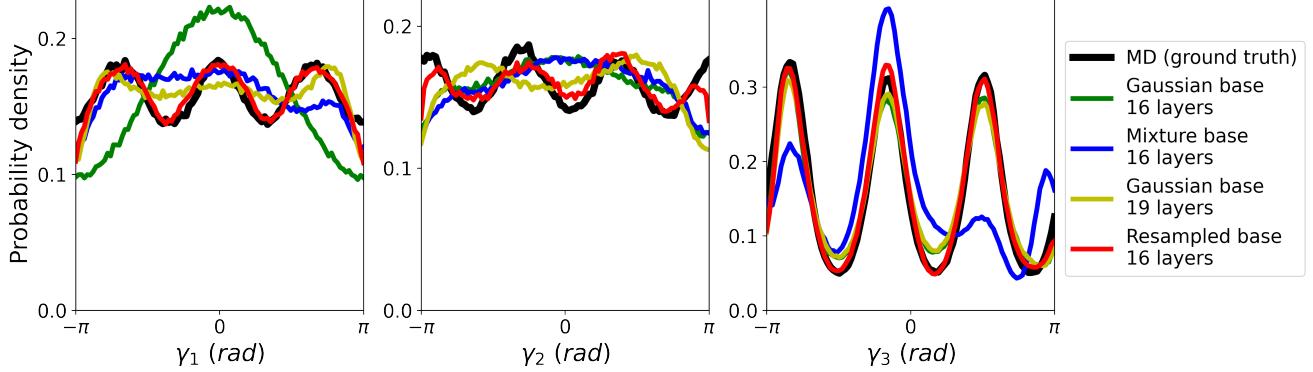


Figure 4: Marginal distribution of three dihedral angles of Alanine dipeptide. The ground truth was determined with a MD simulation. The flow models are based on real NVP and were trained via ML.

Table 5: Quantitative comparison of the real NVP models approximating the Boltzmann distribution of Alanine dipeptide trained via ML learning. The LL is evaluated on a test set obtained with a MD simulation. The KL divergences of the 60 marginals were computed and the mean and median of them are reported. All results are averages over 10 runs, the standard error is given, and higher LL as well as lowest KL divergences are marked in **bold**.

Base distribution	Gaussian	Mixture	Gaussian	Resampled
Number of layers	16	16	19	16
LL ( $\times 10^2$ )	$1.8096 \pm 0.0002$	$1.8106 \pm 0.0002$	$1.8109 \pm 0.0001$	<b><math>1.8118 \pm 0.0001</math></b>
Mean KLD ( $\times 10^{-3}$ )	$1.76 \pm 0.08$	$8.23 \pm 0.82$	$1.35 \pm 0.03$	<b><math>1.12 \pm 0.02</math></b>
Median KLD ( $\times 10^{-4}$ )	$5.20 \pm 0.10$	$43.5 \pm 6.0$	$4.63 \pm 0.08$	<b><math>4.36 \pm 0.05</math></b>

training time, but this amounts only to 0.4–1.5% and 5–15%, respectively, versus a roughly linear increase with the number of layers. Hence, this can be a desirable trade-off, depending on the use case.

#### 4.4 Boltzmann generators

An important application of normalizing flows is the approximation of Boltzmann distributions. Given the atom coordinates  $x$  of a molecule, the likelihood of finding it in this state, i.e. the Boltzmann distribution, is proportional to  $e^{-u(x)}$ , where  $u$  denotes the energy of the system, which can be obtained through physical modeling. Usually, samples are drawn from this distribution through molecular dynamics (MD) simulations (Leimkuhler and Matthews, 2015). However, the sampling process can be greatly accelerated by approximating the Boltzmann distribution with a normalizing flow, called a Boltzmann generator, and then sampling from the flow model (Noé et al., 2019).

Here, we approximate the Boltzmann distribution of the 22 atom Alanine dipeptide, which has been used as a benchmark system in the machine learning literature (Wu et al., 2020; Campbell et al., 2021; Köhler et al., 2021). We use the coordinate transformation introduced by Noé et al. (2019), see also Appendix

G.1, which incorporates the translational and rotational symmetry and reduces the number of dimensions from 66 to 60. Both ML learning and training using the KL divergence are used. For the former we generate a training dataset through a MD simulation over  $10^7$  steps each and keep every 10<sup>th</sup> sample, resulting in datasets with  $10^6$  samples. With the same procedure we generate a test set to evaluate all trained models.

With ML learning we train real NVP models having 16 layers with a Gaussian, a mixture of 10 Gaussians, and a resampled base distribution. Furthermore, we train another real NVP model with a Gaussian base, but having 19 layers, which has roughly the same number of parameters as the real NVP model with the resampled base. More details of the architecture and the training procedure are listed in Appendix G.2. The marginal distribution of three dihedral angles are shown in Figure 4.

Although we tried various methods of initializing the mixture of Gaussians, training it jointly with the flow turns out to be unstable leading to a poor fit of the marginals, which is especially visible for  $\gamma_3$ . Moreover, for two of the three angles, the 16-layered model with the Gaussian base distribution cannot represent the multimodal nature of the distribution accurately.

Increasing the number of layers to 19 improves the result, but even this model is clearly outperformed by the real NVP with a resampled base distribution. To compare the performance quantitatively, we computed the LL on the test set and estimated the KL divergence between the MD samples and the models of the marginals through histograms for all 60 dimensions and report the mean and median. All performance measures were averaged over 10 seeded runs and are shown in Table 5. The real NVP model with the resampled base distribution outperforms all the baselines. The improved performance comes at the cost of increased training time, i.e. 49% and 26%, and sampling time, i.e. by a factor of 4 and 1.8, for the real NVP and the residual flow models with the resampled base distribution, when compared to their Gaussian counterparts. A further analysis of the Ramachandran plots of the models is done in Appendix G.3. There, we also do a comparison to stochastic normalizing flows (Wu et al., 2020) and show the results of training residual flows with ML whereby the model with the resampled base distribution outperforms the baselines as well.

Moreover, we used the KL divergence to train real NVP models with Gaussian, mixture of Gaussians, and resampled base distributions as well, having the same architecture as the real NVP models with 16 layers in the experiments above. This is a challenging task since if samples from the model are too far away from the modes of the Boltzmann distribution, their gradients can be very high making training unstable. However, it is important for the application of Boltzmann generators since the necessity of creating a dataset through other expensive sampling procedures diminishes their ability to reduce the overall computational time needed for sampling. Details of the model architectures and the training procedure are given in Appendix G.2. Although it involves rejection sampling, training the flow models with the resampled base distribution only took 15% longer than the baseline models. As can be seen in Table 15, the real NVP model with a resampled base outperforms those with a Gaussian and a mixture of Gaussians; however, they are still inferior to flows trained via ML.

## 5 DISCUSSION AND RELATED WORK

The main challenge we tackle in this work, i.e. that normalizing flows struggle to model distributions with supports having a complicated topological structure due to their invertible nature, has been addressed in several articles. Cornish et al. (2020) introduced a new set of variables for each flow layer, called contin-

uous indices, which they used as additional input to the flow maps. Thereby, they relaxed the bijectivity of the transformation leading to a better model performance. Huang et al. (2020) augmented the dataset by auxiliary dimensions before applying their normalizing flow model. Although the topological constraints are still present in the augmented space, the marginal distribution of interest can be arbitrary complex. Wu et al. (2020); Nielsen et al. (2020) suggested adding sampling layers to the model. Hence, the topology of the support can be changed through the sampling process. Nielsen et al. (2020) also introduced surjective layers, which do not suffer from topological constraints and essentially combine VAEs with flow-based models. These approaches sacrifice the invertibility of the flow map, which has several disadvantages. First of all, the model is no longer a perfect autoencoder, i.e. the original datapoint cannot be fully recovered from its latent representation. Second, if the layers of the flow-based model are bijective, significant memory savings are possible (Gomez et al., 2017). Usually, when training layered models such as neural networks the activations of each layer need to be stored in the forward pass because they are needed for gradient computation in the backward pass. However, if the layers are invertible, the activations of the forward pass can be recomputed in the backward pass by applying the inverse of the layer to the activations of the previous layers. Thereby, models can be made basically infinitely deep with a fixed memory budget. Thirdly, exact evaluation of the likelihood is no longer possible. To train the models, a bound needs to be derived which is optimized instead of the actual likelihood. Our model does not make this sacrifice since only the base distribution is altered, but the transformation of the normalizing flow model is still invertible. On the other side, the base distribution itself cannot be evaluated exactly because its normalization constant is unknown. It can be estimated via Monte Carlo sampling, but its logarithm, appearing in the LL of the model, is biased. However, as discussed in Section 3.1 for large truncation parameter  $T$  we are only off by a constant so e.g. importance sampling could be done without a bias. Moreover, drawing samples from our model is less efficient as many samples from the proposal might get rejected before finally one is accepted and propagated through the flow.

An autoregressive base distribution was introduced by Bhattacharyya et al. (2020). While they only considered image generation, their entire model, i.e. including the base distribution, is tractable in contrast to ours. However, the computational cost of their models scales with the square root of the number of pixels, while ours is constant. Izmailov et al. (2020); Ardizzone et al. (2020); Hagemann and Neumayer (2021)

explored normalizing flows with a multimodal base distribution, in their case a mixture of Gaussians. However, their intention was to model data with multiple classes, thereby performing classification and solving inverse problems. Our model allows to describe data with multiple classes as well through a conditional distribution, similar to the work of Dinh et al. (2017); Kingma and Dhariwal (2018), but is also able to describe the complicated topological structure of the distribution of each class.

Bauer and Mnih (2019) used LARS successfully to create more expressive priors for VAEs, thereby boosting their performance. They demonstrated that the resampled prior can be learned jointly with the encoder and decoder by maximizing the evidence lower bound. In contrast, we showed that a resampled base distribution can be jointly trained with a normalizing flow transformation using both the LL and the KL divergence as an objective. For the latter we derived an expression of the gradient with reduced variance inspired by the work of Grover et al. (2018). Furthermore, Bauer and Mnih (2019) reported that they tried to fully factorize their resampled prior, which would allow them to scale to higher dimensional problems, but they were not able not beat the baseline of a VAE with a factorized Gaussian prior. We were successful by not fully factorizing our resampled base distribution, but defining factors for groups of variables. Moreover, combining LARS with the multiscale architecture of Dinh et al. (2017) and using a factorization similar to (Ma et al., 2019) allowed us to scale up our base distribution even further. The largest base distribution in our work, used in Glow to model the CIFAR10 dataset, has 3072 dimensions, while the largest prior of Bauer and Mnih (2019) only had 100.

## 6 CONCLUSION

In this work, we introduced a base distribution for normalizing flows based on learned rejection sampling. We derived how it can be trained jointly with the flow layers maximizing the expected LL or minimizing the KL divergence. This base distribution can assimilate the complex topological structure of a target and, thereby, overcome a structural weakness of normalizing flows. By applying our procedure to 2D distributions, tabular data, images, and Boltzmann distributions we demonstrated that resampling the base distribution can improve their performance qualitatively and quantitatively.

## Acknowledgements

We thank Matthias Bauer, Richard Turner, Andrew Campbell, Austin Tripp, and David Liu for the help-

ful discussions. José Miguel Hernández-Lobato acknowledges support from a Turing AI Fellowship under grant EP/V023756/1. This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039B; and by the Machine Learning Cluster of Excellence, EXC number 2064/1 - Project number 390727645.

## References

- Ardizzone, L., Mackowiak, R., Rother, C., and Köthe, U. (2020). Training Normalizing Flows with the Information Bottleneck for Competitive Generative Classification. In *Advances in Neural Information Processing Systems 33*.
- Bauer, M. and Mnih, A. (2019). Resampled Priors for Variational Autoencoders. *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 66–75.
- Behrman, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J.-H. (2019). Invertible Residual Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 573–582.
- Behrman, J., Vicol, P., Wang, K.-C., Grosse, R., and Jacobsen, J.-H. (2021). Understanding and Mitigating Exploding Inverses in Invertible Neural Networks. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 1792–1800. PMLR.
- Bhattacharyya, A., Mahajan, S., Fritz, M., Schiele, B., and Roth, S. (2020). Normalizing Flows With Multi-Scale Autoregressive Priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8415–8424.
- Campbell, A., Chen, W., Stimper, V., Hernandez-Lobato, J. M., and Zhang, Y. (2021). A Gradient Based Strategy for Hamiltonian Monte Carlo Hyperparameter Optimization. In *Proceedings of the 38th International Conference on Machine Learning*, pages 1238–1248. PMLR.
- Chen, R. T. Q., Behrman, J., Duvenaud, D., and Jacobsen, J.-H. (2019). Residual Flows for Invertible Generative Modeling. In *Advances in Neural Information Processing Systems*, volume 32.
- Chen, X., Mishra, N., Rohaninejad, M., and Abbeel, P. (2018). PixelSNAIL: An Improved Autoregressive Generative Model. In *Proceedings of the 35th International Conference on Machine Learning*.
- Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating Long Sequences with Sparse Transformers. *arXiv:1904.10509 [cs, stat]*.
- Cornish, R., Caterini, A. L., Deligiannidis, G., and Doucet, A. (2020). Relaxing Bijectivity Constraints

- with Continuously Indexed Normalising Flows. In *Proceedings of the 37th International Conference on Machine Learning*.
- Dheeru, D. and Taniskidou, E. K. (2022). UCI machine learning repository. <http://archive.ics.uci.edu/ml>.
- Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: Non-linear Independent Components Estimation. In *3rd International Conference on Learning Representations, Workshop Track Proceedings*.
- Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using Real NVP. *International Conference on Learning Representations*.
- Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Neural Spline Flows. In *Advances in Neural Information Processing Systems*, volume 32.
- Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. (2017). The Reversible Residual Network: Back-propagation Without Storing Activations. In *Advances in Neural Information Processing Systems*, volume 32.
- Grcić, M., Grubišić, I., and Šegvić, S. (2021). Densely connected normalizing flows. In *Advances in Neural Information Processing Systems 34*.
- Grover, A., Gummadi, R., Lazaro-Gredilla, M., Schuurmans, D., and Ermon, S. (2018). Variational Rejection Sampling. In *International Conference on Artificial Intelligence and Statistics*, volume 84, pages 823–832.
- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, volume 30, pages 5767–5777.
- Hagemann, P. and Neumayer, S. (2021). Stabilizing invertible neural networks using mixture models. *Inverse Problems*, 37(8):085002.
- Hernández-Lobato, J. M., Li, Y., Rowland, M., Hernández-Lobato, D., Bui, T., and Turner, R. E. (2016). Black-box alpha-divergence Minimization. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 1511–1520.
- Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2722–2730.
- Huang, C.-W., Dinh, L., and Courville, A. (2020). Augmented Normalizing Flows: Bridging the Gap Between Generative Flows and Latent Variable Models. *arXiv:2002.07101 [cs, stat]*.
- Izmailov, P., Kirichenko, P., Finzi, M., and Wilson, A. G. (2020). Semi-Supervised Learning with Normalizing Flows. In *Proceedings of the 37th International Conference on Machine Learning*, pages 4615–4630.
- Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2018). Averaging Weights Leads to Wider Optima and Better Generalization. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 876–885.
- Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., and Aila, T. (2020a). Training Generative Adversarial Networks with Limited Data. *Advances in Neural Information Processing Systems*, 33:12104–12114.
- Karras, T., Laine, S., and Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410.
- Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020b). Analyzing and Improving the Image Quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119.
- Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*.
- Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative Flow with Invertible 1x1 Convolutions. *Advances in Neural Information Processing Systems*.
- Kingma, D. P., Salimans, T., Poole, B., and Ho, J. (2021). Variational Diffusion Models. *arXiv:2107.00630 [cs, stat]*.
- Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations*.
- Köhler, J., Krämer, A., and Noé, F. (2021). Smooth Normalizing Flows. In *Advances in Neural Information Processing Systems 34*.
- Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. page 60.
- Leimkuhler, B. and Matthews, C. (2015). *Molecular Dynamics With Deterministic and Stochastic Numerical Methods*. Number 39 in *Interdisciplinary Applied Mathematics*. Springer.
- Ma, X., Kong, X., Zhang, S., and Hovy, E. (2019). Macow: Masked convolutional generative flow. In Walach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

- Nielsen, D., Jaini, P., Hoogeboom, E., Winther, O., and Welling, M. (2020). SurVAE Flows: Surjections to Bridge the Gap between VAEs and Flows. In *Advances in Neural Information Processing Systems*, volume 33.
- Noé, F., Olsson, S., Köhler, J., and Wu, H. (2019). Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457).
- Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2021). Normalizing Flows for Probabilistic Modeling and Inference. *Journal of Machine Learning Research*, 22(57):1–64.
- Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*, volume 30, pages 2338–2347.
- Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. (2018). Image Transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR.
- Polyak, B. (1990). New stochastic approximation type procedures. *Avtomatika i Telemekhanika*, 7:98–107.
- Rezende, D. J. and Mohamed, S. (2015). Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*.
- Runde, V. (2005). *A Taste of Topology*. Universitext. Springer-Verlag, New York.
- Ruppert, D. (1988). Efficient estimators from a slowly converging robbins-monro process.
- Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2256–2265. PMLR.
- Tabak, E. G. and Turner, C. V. (2013). A Family of Nonparametric Density Estimation Algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164.
- Tabak, E. G. and Vanden-Eijnden, E. (2010). Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233.
- van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., van den Driessche, G., Lockhart, E., Cobo, L. C., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. (2018). Parallel WaveNet: Fast High-Fidelity Speech Synthesis. In *Proceedings of the 35th International Conference on Machine Learning*.
- Wirnsberger, P., Ballard, A. J., Papamakarios, G., Abercrombie, S., Racanière, S., Pritzel, A., Jimenez Rezende, D., and Blundell, C. (2020). Targeted free energy estimation via learned mappings. *The Journal of Chemical Physics*, 153(14):144112.
- Wu, H., Köhler, J., and Noe, F. (2020). Stochastic normalizing flows. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5933–5944. Curran Associates, Inc.

---

## Supplementary Material: Resampling Base Distributions of Normalizing Flows

---

### A LEARNING ALGORITHMS

#### A.1 Estimating the normalization constant

To stabilize training, we use the exponential moving average to estimate the value of the normalization constant (Bauer and Mnih, 2019). In practice, this means that if  $Z_i$  is the current Monte Carlo estimate of the normalization constant, the exponential moving average  $\langle Z \rangle_i$  is computed by

$$\langle Z \rangle_1 = Z_1, \tag{12}$$

$$\langle Z \rangle_i = (1 - \epsilon) \langle Z \rangle_{i-1} + \epsilon Z_i \quad \text{for } i > 1, \tag{13}$$

where  $\epsilon$  is the decay parameter which we set to 0.05 throughout this article. However, the gradients are estimated only with the current Monte Carlo estimate  $Z_i$ , because otherwise backpropagation through the entire history of  $\langle Z \rangle_i$  would be necessary, which would be computationally expensive and memory demanding.

#### A.2 Gradient estimators of the Kullback-Leibler divergence

We repeat Theorem 1 as stated in the main text and supplement its proof.

**Theorem 1.** *Let  $p_\phi(z)$  be the base distribution of a normalizing flow, having parameters  $\phi$ , and  $F_\theta$  be the respective invertible mapping, depending on its parameters  $\theta$ , such that the density of the model is*

$$\log(p(x)) = \log(p_\phi(z)) - \log |\det J_{F_\theta}(z)|, \tag{14}$$

with  $x = F_\theta(z)$ . Then, the gradients of the KL divergence with respect to the parameters are given by

$$\nabla_\phi \text{KLD}(\theta, \phi) = \text{Cov}_{p_\phi(z)} \{ \log(p_\phi(z)) - \log |\det J_{F_\theta}(z)| - \log \hat{p}^*(F_\theta(z)), \nabla_\phi \log p_\phi(z) \} \tag{15}$$

$$\nabla_\theta \text{KLD}(\theta, \phi) = -\mathbb{E}_{p_\phi(z)} [\nabla_\theta (\log |\det J_{F_\theta}(z)| + \log \hat{p}^*(F_\theta(z)))] \tag{16}$$

*Proof.* The KL divergence is defined as

$$\text{KLD}(\theta, \phi) := \mathbb{E}_{p(x)} [\log p(x)] - \mathbb{E}_{p(x)} [\log p^*(x)]. \tag{17}$$

By plugging in (14) into (17) we obtain

$$\text{KLD}(\theta, \phi) = \mathbb{E}_{p_\phi(z)} [\log p_\phi(z) - \log |\det J_{F_\theta}(z)| - \log \hat{p}^*(F_\theta(z))]. \tag{18}$$

Computing the gradient of (18) with respect to  $\theta$  is straight forward.

$$\begin{aligned} \nabla_\theta \text{KLD}(\theta, \phi) &= \nabla_\theta \mathbb{E}_{p_\phi(z)} [\log p_\phi(z) - \log |\det J_{F_\theta}(z)| - \log \hat{p}^*(F_\theta(z))] \\ &= \mathbb{E}_{p_\phi(z)} [\nabla_\theta (\log p_\phi(z) - \log |\det J_{F_\theta}(z)| - \log \hat{p}^*(F_\theta(z)))] \\ &= -\mathbb{E}_{p_\phi(z)} [\nabla_\theta (\log |\det J_{F_\theta}(z)| + \log \hat{p}^*(F_\theta(z)))] \end{aligned} \tag{19}$$

To get the gradient with respect to  $\phi$ , we decompose (18) into two parts and consider their gradients separately.

$$\begin{aligned}
 \nabla_{\phi} \mathbb{E}_{p_{\phi}(z)} [\log p_{\phi}(z)] &= \nabla_{\phi} \int p_{\phi}(z) \log p_{\phi}(z) dz \\
 &= \int \nabla_{\phi} (p_{\phi}(z) \log p_{\phi}(z)) dz \\
 &= \int \nabla_{\phi} p_{\phi}(z) + \log p_{\phi}(z) \nabla_{\phi} p_{\phi}(z) dz \\
 &= \nabla_{\phi} \underbrace{\int p_{\phi}(z) dz}_{=1} + \int p_{\phi}(z) \log p_{\phi}(z) \nabla_{\phi} \log p_{\phi}(z) dz \\
 &= \mathbb{E}_{p_{\phi}(z)} [\log p_{\phi}(z) \nabla_{\phi} \log p_{\phi}(z)]
 \end{aligned} \tag{20}$$

$$\begin{aligned}
 \text{ld} &:= \log |\det J_{F_{\theta}}(z)| + \log \hat{p}^*(F_{\theta}(z)) \\
 \nabla_{\phi} \mathbb{E}_{p_{\phi}(z)} [\text{ld}] &= \nabla_{\phi} \int \text{ld} p_{\phi}(z) dz \\
 &= \int \text{ld} \nabla_{\phi} p_{\phi}(z) dz \\
 &= \int \text{ld} p_{\phi}(z) \nabla_{\phi} \log p_{\phi}(z) dz \\
 &= \mathbb{E}_{p_{\phi}(z)} [\text{ld} \nabla_{\phi} \log p_{\phi}(z)]
 \end{aligned} \tag{21}$$

Using these two expressions, we obtain

$$\nabla_{\phi} \text{KLD}(\theta, \phi) = \mathbb{E}_{p_{\phi}(z)} [(\log p_{\phi}(z) - \log |\det J_{F_{\theta}}(z)| - \log \hat{p}^*(F_{\theta}(z))) \nabla_{\phi} \log p_{\phi}(z)] \tag{22}$$

$$= \text{Cov}_{p_{\phi}(z)} \{p_{\phi}(z) - \log |\det J_{F_{\theta}}(z)| - \log \hat{p}^*(F_{\theta}(z)), \nabla_{\phi} \log p_{\phi}(z)\}. \tag{23}$$

When concluding (23) from (22) we used the well known identity

$$\begin{aligned}
 \mathbb{E}_{p_{\phi}(z)} [\nabla_{\phi} \log p_{\phi}(z)] &= \int p_{\phi}(z) \nabla_{\phi} \log p_{\phi}(z) dz \\
 &= \int \frac{p_{\phi}(z)}{p_{\phi}(z)} \nabla_{\phi} p_{\phi}(z) dz = \nabla_{\phi} \underbrace{\int p_{\phi}(z) dz}_{=1} = 0.
 \end{aligned} \tag{24}$$

□

## B MULTISCALE ARCHITECTURE

As already mentioned in the main paper, Dinh et al. (2017) introduced the multiscale architecture for normalizing flows to deal with high dimensional data such as images. As sketched in Figure 5, initially, the entire input  $x$  is transformed by several flow layers. The result is split up into two parts,  $h_1^{(1)}$  and  $h_1^{(2)}$ . Dinh et al. (2017) did this by first squeezing the image, i.e. reducing the height and width of the image by a factor 2 and adding the surplus pixels as additional channels, and then splitting the resulting tensor along the channel dimension.  $h_1^{(1)}$  is immediately factored out in the density, while  $h_1^{(2)}$  is further transformed by  $F_2$ . The process is the repeated until a desired depth is reached. The output of the last map, in Figure 5 it is  $F_4$ , is not split, but directly passed to its base distribution. The full density for a multiscale architecture with  $n$  levels is given by

$$p(x) = \prod_{i=1}^n |\det(J_{F_i}(h_{i-1}))| p(z_i),$$

where we set  $h_0 = x$ .

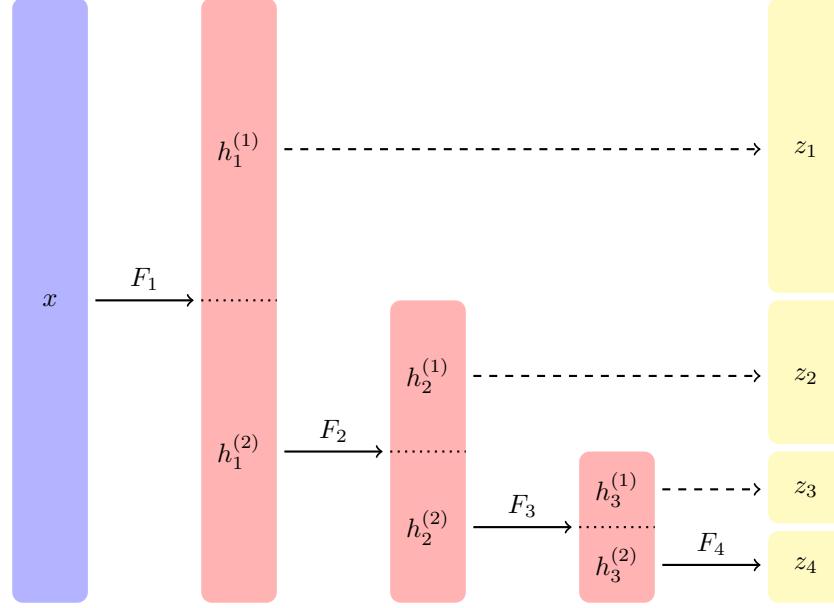


Figure 5: Multiscale architecture with four levels as introduced in (Dinh et al., 2017). First, the entire input  $x$  is transformed by  $F_1$ . The result is then split up into two parts of which one of them is factored out immediately and the other one is further processed by  $F_2$ . This process is repeated a few times until the desired depth is reached. The input is drawn in blue, intermediate results are red, and the components of the final variable  $z$  are yellow.

## C LEARNED ACCEPTANCE PROBABILITY

### C.1 Choosing the architecture

In order to get an impression of what the architecture of the neural network defining the acceptance probability  $a$  for LARS, we did an ablation experiment on the Power UCI dataset. We left the flow architecture of a real NVP model constant but changed the number of hidden layers and units of the neural network representing  $a$ . The baseline model with a Gaussian base distribution achieved  $0.330 \pm 0.003$  on the test set. When changing the number of hidden layers we used 512 hidden units and 3 hidden layers when changing the number hidden units.

Table 6: LL of the test set for different number of hidden layers for  $a$  while leaving the number of hidden units constant at 512.

Hidden layers	1	3	5	7	9
LL	0.37	0.53	0.58	0.62	0.63

Table 7: LL of the test set for different number of hidden layers for  $a$  while leaving the number of hidden layers constant at 3.

Hidden units	32	128	512	2048	8192
LL	0.39	0.45	0.53	0.61	0.61

We see that both the number of hidden layers and units is important. The LL increases as we are adding more with diminishing returns. However, note that especially increasing the number of hidden units increases the parameter count as well as the computational cost; hence, an application specific trade-off needs to be found.

### C.2 Tuning the rejection rate

As discussed in the main text, the rejection rate of LARS can be controlled through the truncation parameter  $T$ . It sets a limit on how often subsequent proposals can be rejection in order to generate one sample. However,

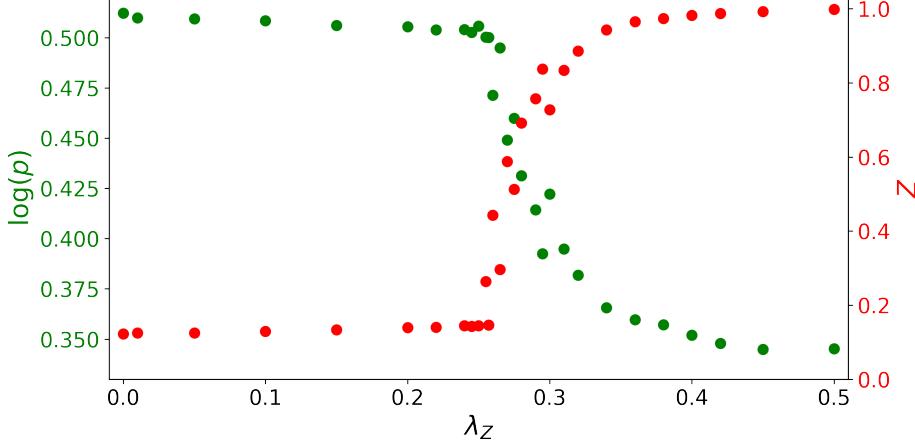


Figure 6: LL on the test set and  $Z$  with respect to the hyperparameter  $\lambda_Z$  introduced in (26).

it does not tell us something about the actual rejection rate determining the sampling speed, which might be lower. The number of expected samples per sample from the proposal  $\pi$  is given by

$$\mathbb{E}_\pi(z)[a(z)] = \int a(z)\pi(z)dz = Z, \quad (25)$$

which is equivalent to the normalization constant  $Z$ . Hence, if we increase  $Z$  we can decrease the rejection rate. We can simply do so by including it in our optimization, e.g. when doing ML we can instead minimize the loss

$$\mathcal{L} = -\mathbb{E}_{p^*(x)}[\log p(x)] - \lambda_Z Z, \quad (26)$$

where  $\lambda_Z \in \mathbb{R}_+$  is a positive hyperparameter.

In order to test this procedure, we trained 30 real NVP models with a resampled base distribution with different values of  $\lambda_Z$  on the UCI Power dataset. The neural network representing the acceptance probability  $a$  had 3 hidden layers with 512 hidden units and we set  $T = 20$ . In Figure 6 we show the LL of the models on the test set as well as  $Z$  depending on the hyperparameter  $\lambda_Z$ . We see that by increasing  $\lambda_Z$  we can trade off performance in terms of LL with the expected number of LARS samples per sample from the proposal. When  $Z$  approaches one, i.e. nearly all samples from the proposal get accepted, the LL drops to the value achieved by the flow with a Gaussian base distribution being  $0.330 \pm 0.003$ , see Table 9.

## D 2D DISTRIBUTIONS

The densities of the distributions used as sample targets in Section 4.1 are given in Table 8.

Table 8: Logarithm of the unnormalized densities of the target distributions used in Section 4.1.

	Unnormalized log density
Dual Moon	$-\frac{(\ z\  - 1)^2}{0.08} - \frac{( z_1  - 2)^2}{0.18} + \log \left( 1 + e^{-\frac{4z_1}{0.09}} \right)$
Circle of Gaussians	$\log \left[ \sum_{i=1}^8 \left( \frac{9}{2\pi(2 - \sqrt{2})} e^{-\frac{9((z_1 - 2 \sin(\frac{2\pi}{8i}))^2 + (z_2 - 2 \cos(\frac{2\pi}{8i}))^2}{4 - 2\sqrt{2}}} \right) \right]$
Two Rings	$\log \left[ \sum_{i=1}^2 \left( \frac{32}{\pi} e^{-32(\ z\  - i - 1)^2} \right) \right]$

All models approximating a 2D distribution uses for each layer a fully connected network having 2 hidden layers with 32 hidden units each as parameter map or residual learning block, respectively. The mixture of Gaussian

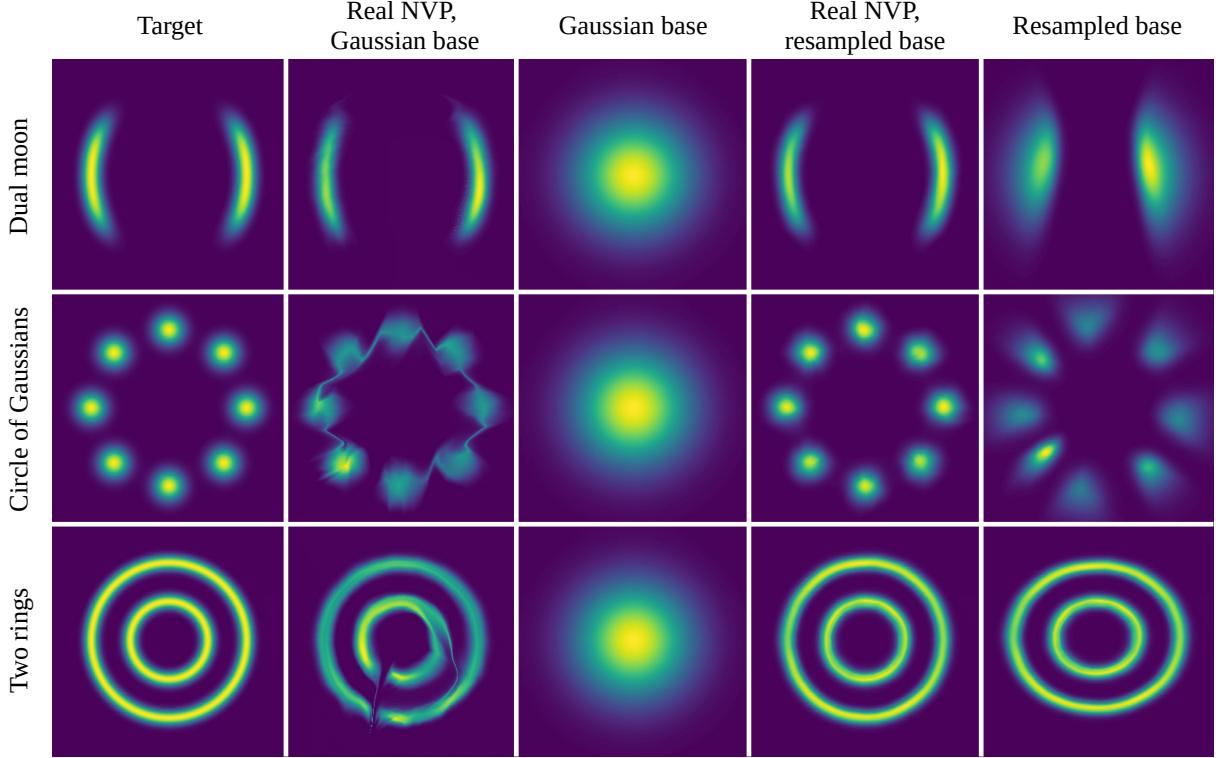


Figure 7: Visualization of the densities when approximating three 2D distributions with complex topological structure. Real NVP models with Gaussian and a resampled base distributions were trained using the KL divergence.

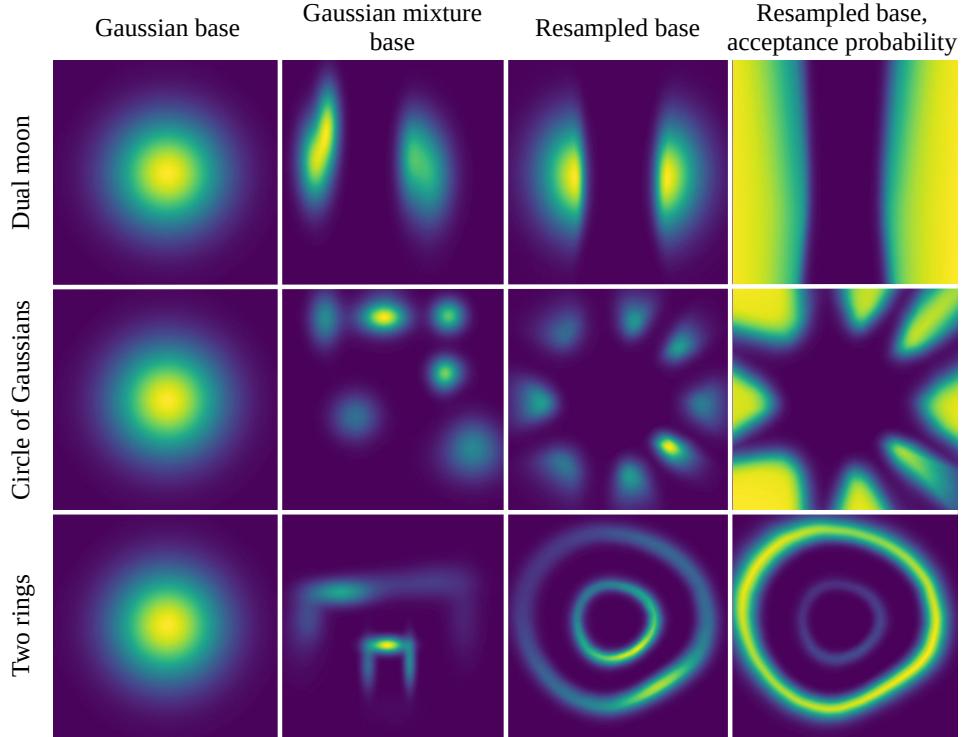


Figure 8: Visualization of the learned base distributions of the real NVP flow models shown in Figure 3.

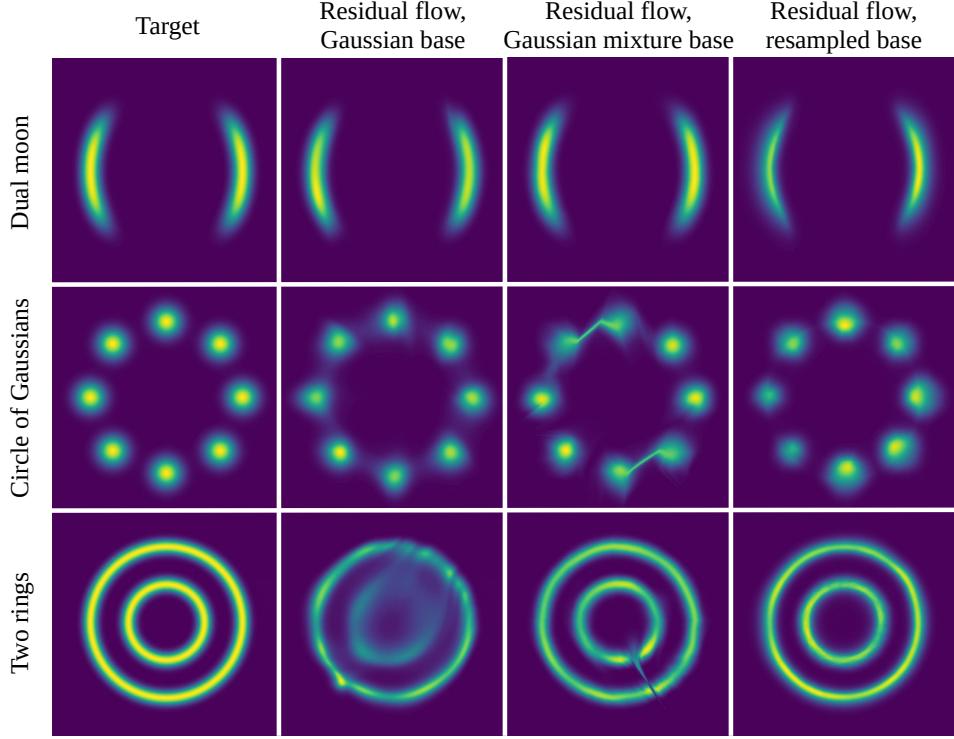


Figure 9: Visualization of the residual flow densities when approximating three 2D distributions with complex topological structure. The models were trained using ML learning and the corresponding base distributions are shown in Figure 10.

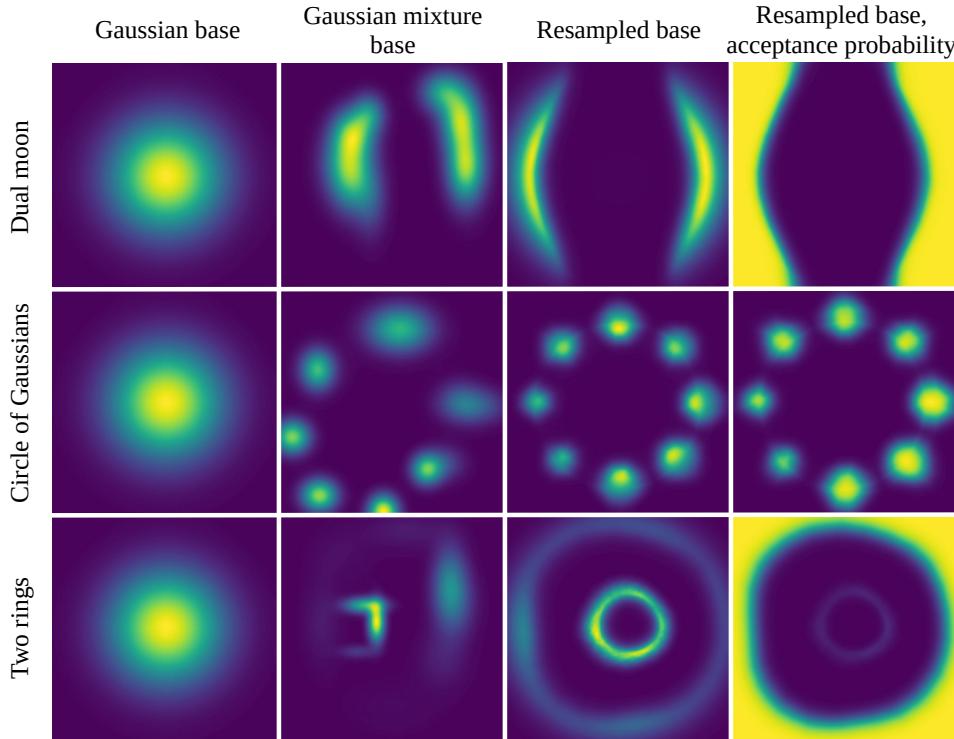


Figure 10: Visualization of the learned base distributions of the residual flow models shown in Figure 9.

base distributions are initialized by uniformly sampling the mean in the hypercube  $[-2.5, 2.5]^D$  and setting the variances to  $0.5 \cdot \mathbf{1}_D$ , where  $\mathbf{1}_D$  is the  $D$ -dimensional identity matrix.

The models are trained on a computer with 6 Intel i5-9400F CPUs and a Nvidia GeForce RTX 2070 graphics card. The Adam optimizer with a learning rate of  $10^{-3}$  is used. Training is done for  $2 \cdot 10^4$  iterations with a batch size of 1024.

## E TABULAR DATA

In addition to the NSF models, we also trained real NVP models with a Gaussian, a mixture of Gaussians, and a resampled base distribution to the four UCI datasets. The results are shown in Table 9.

Table 9: LL of real NVP models with different base distributions on the test sets of the respective datasets. The values are averaged over 3 runs each and the standard error is given as a measure of uncertainty. The highest values within the confidence interval are marked in **bold**.

Base distribution	Power	Gas	Hepmass	Miniboone
Gaussian	$0.330 \pm 0.003$	$10.1 \pm 0.1$	$-19.5 \pm 0.1$	$-11.65 \pm 0.05$
Mixture	$0.341 \pm 0.001$	$9.9 \pm 0.2$	$-19.5 \pm 0.1$	$-11.49 \pm 0.04$
Resampled	<b><math>0.560 \pm 0.006</math></b>	<b><math>12.8 \pm 0.1</math></b>	<b><math>-18.4 \pm 0.1</math></b>	<b><math>-11.48 \pm 0.01</math></b>

Table 10: Details about datasets from the UCI machine learning repository, the architecture of the NSF models as well as the resampled base distribution, and the training procedure.

	Power	Gas	Hepmass	Miniboone
Dimension	6	8	21	43
Train data points	$1.6 \cdot 10^6$	$8.5 \cdot 10^5$	$3.2 \cdot 10^5$	$3.0 \cdot 10^4$
Flow layers	10	10	10	10
Hidden layers flow maps	2	2	2	1
Hidden units flow maps	256	128	256	64
Hidden layers $a$	7	9	4	2
Hidden units $a$	512	512	512	128
Truncation parameter $T$	100	50	40	40
Dropout rate	0	0.1	0.3	0.3
Batch size	512	512	256	64
Learning rate	$3 \cdot 10^{-4}$	$4 \cdot 10^{-4}$	$4 \cdot 10^{-4}$	$3 \cdot 10^{-4}$

Table 11: Details about the architecture of the real NVP models used as well as the resampled base distribution, and the training procedure.

	Power	Gas	Hepmass	Miniboone
Flow layers	16	16	16	16
Hidden layers flow maps	2	2	2	2
Hidden units flow maps	128	128	64	32
Hidden layers $a$	3	3	3	3
Hidden units $a$	512	512	512	256
Truncation parameter $T$	100	100	100	100
Dropout rate	0	0.1	0.2	0.2
Batch size	512	512	256	128
Learning rate	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$

In all experiments regarding the UCI datasets, we use dropout both in the neural networks defining the flow map and the acceptance probability function  $a$  of the resampled base distribution during training. Adamax is used as

an optimizer (Kingma and Ba, 2015). The experiments are run on machines with 36 Intel Xeon Platinum 9242 CPUs and 128 GB RAM. Further details on the datasets, the flow architecture, and the training procedure are given in Table 10 and Table 11.

## F IMAGE GENERATION

The parameter maps of the Glow models are convolutional neural networks (CNNs) with 3 layers, the first and the last having a kernel size of  $3 \times 3$  and the middle layer of  $1 \times 1$ . The number of channels of the middle layer is 512 and those of the other layers is determined by the respective input and output. This is the same architecture as used in (Kingma and Dhariwal, 2018).

To ensure that each factor of the base distribution has not more than 100 dimensions, we apply a squeeze operation to the feature map of the first level before passing it to the base distribution. Therefore, each channel has a maximum size of  $8 \times 8 = 64$ . A CNN with 4 layers, having 32 channels and a kernel size of  $3 \times 3$  each and a fully connected output layer, is used as acceptance function at each level. The convolutions of this CNN are strided with a stride of 2 until the image size is  $4 \times 4$ . The normalization constants are updated with 2048 samples per iteration and before evaluating our models we estimated them with  $10^{10}$  samples.

Each model is trained for  $10^6$  iterations with the Adam optimizer having a learning rate of  $10^{-3}$ . The learning rate is warmed up linearly over  $10^3$  iterations and the batch size is 512. The models with 8, 16, and 32 layers per level are trained in a distributed fashion on 1, 2, and 4 Nvidia Quadro RTX 5000 graphics cards. We apply Polyak-Ruppert weight averaging (Polyak, 1990; Ruppert, 1988) with an update rate of  $10^{-3}$ , where the exponential moving average of the model weights is computed in order to improve the generalization performance on the test set (Izmailov et al., 2018).

Table 12: Percentage increase in training time and model size when using a resampled instead of a Gaussian base distribution for the models trained in Section 4.3.

Layers per level	Training time	Model size
8	4.7%	1.5%
16	15%	0.75%
32	9.1%	0.38%

## G BOLTZMANN GENERATORS

### G.1 Coordinate transformation

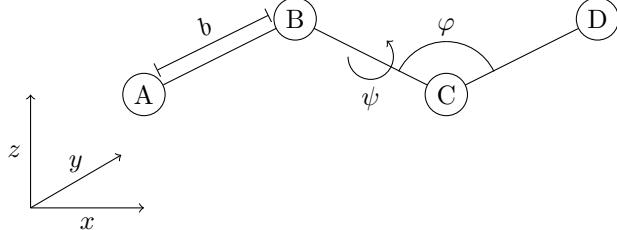


Figure 11: Illustration of molecular coordinates. The state of the molecule can be described through the Cartesian coordinates, i.e.  $x$ ,  $y$ , and  $z$ , of each of the four atoms A, B, C, and D. Alternatively, internal coordinates, i.e. bond lengths, bond angles, and dihedral angles, can be used. Here, the bond length  $b$  is the distance between atom A and B, the bond angle  $\varphi$  being the angle between the bonds between B and C as well as C and D, and the dihedral angle  $\psi$  is the angle between the planes spanned by A, B, and C as well as B, C, and D. We use a combination of Cartesian and internal coordinates.

To simplify the approximation Boltzmann distributions of complex molecules, a coordinate transformation was introduced (Noé et al., 2019). Some of the Cartesian coordinates are mapped to their respective internal coordinates, i.e. bond lengths, bond angles, and dihedral angles, which are illustrated in Figure 11. The internal

coordinates are normalized, with mean and standard deviation calculated on the training dataset generated through MD, but a suitable experimental dataset could be used as well. To the remaining Cartesian coordinates principal component analysis is applied. Subsequently, the weights of all but the last six principal components are used as coordinates. Thereby, six degrees of freedom are eliminated, corresponding to the three translational and free rotational coordinates which leave the Boltzmann distribution invariant.

## G.2 Setup of the experiments

All real NVP models trained via ML have a neural network with 2 hidden layers and 64 hidden units as a parameter map at each coupling layer. Between the coupling layers, we apply a invertible linear transformation which is learned with the other parameters of the flow, similar to the invertible 1x1 convolutions introduced in (Kingma and Dhariwal, 2018). The acceptance function of the resampled base distribution is a fully connected neural network with 2 hidden layers having 256 hidden units each. At each iteration, the normalization constant  $Z$  is updated with 512 samples from the Gaussian proposal during training. Before evaluating our models, we estimated  $Z$  with  $10^{10}$  samples. The residual flow models have 8 layers each with each layer having 2 layer fully connected neural network with 64 hidden units and the resampled base distribution has 3 hidden layers with 512 hidden units. All models are trained for  $5 \cdot 10^5$  iterations with the Adam optimizer (Kingma and Ba, 2015) and a batch size of 512. The learning rate is set to  $10^{-3}$  and decreased to  $10^{-4}$  after  $2.5 \cdot 10^5$  iterations. We also do Polyak-Ruppert weight averaging (Polyak, 1990; Ruppert, 1988) with an update rate of  $10^{-2}$ . Each model is trained and evaluated on a server with 16 Intel Xeon E5-2698 CPUs and a Nvidia GTX980 GPU.

The real NVP models trained by minimizing the KL divergence have the same architecture as those in the previous experiment. However, to improve the stability of the training process, the models are trained with double precision numbers on 32 Intel Xeon E5-2698 CPUs each.  $10^5$  iterations are done with the Adam optimizer with a learning rate of  $10^{-4}$ , which is exponentially decayed every  $2.5 \cdot 10^4$  iterations by a factor of 0.5.

The KL divergences were computed by drawing  $10^6$  samples from the model and estimating the respective integrals with histograms.

## G.3 Further results

As additional performance metric to compare the models, we compute the Ramachandran plots, i.e. a 2D histogram of two dihedral angles. These plots are frequently used to analyse how proteins fold locally and are hence of high importance for many applications. Some Ramachandran plots are show in Figure 13. We also estimate the KL divergences of the ground truth Ramachandran plot obtained from the MD test set and the plots of the models by performing numerical integration with the histograms. The results are given in Table 13, Table 14, and Table 15.

We also evaluated the stochastic normalizing flow model trained by Wu et al. (2020) through ML on our metrics. The median KL divergences of the marginals is  $2.3 \cdot 10^{-3}$  while the mean is  $2.6 \cdot 10^{-2}$ , which is almost an order of magnitude higher than the results of the models with a resampled base distribution. However, the stochastic normalizing flow models the Ramachandran plot very well, where the KL divergence is only  $2.4 \cdot 10^{-1}$ . Note that these results have to be taken with a grain of salt, since Wu et al. (2020) used an augmented normalizing flow with less layers than we did. We tried to include their stochastic layers into our models but found training to be very unstable in this setting.

Table 13: KL divergence of the Ramachandran plots of the MD simulation, serving as a ground truth, and real NVP models trained via ML learning. It was estimated based on a histogram computed from  $10^6$  samples.

Base distribution	Gaussian	Mixture	Gaussian	Resampled
Number of layers	16	16	19	16
KL divergence	$4.79 \pm 0.73$	$10.8 \pm 7.3$	<b><math>2.26 \pm 0.27</math></b>	$3.00 \pm 0.36$

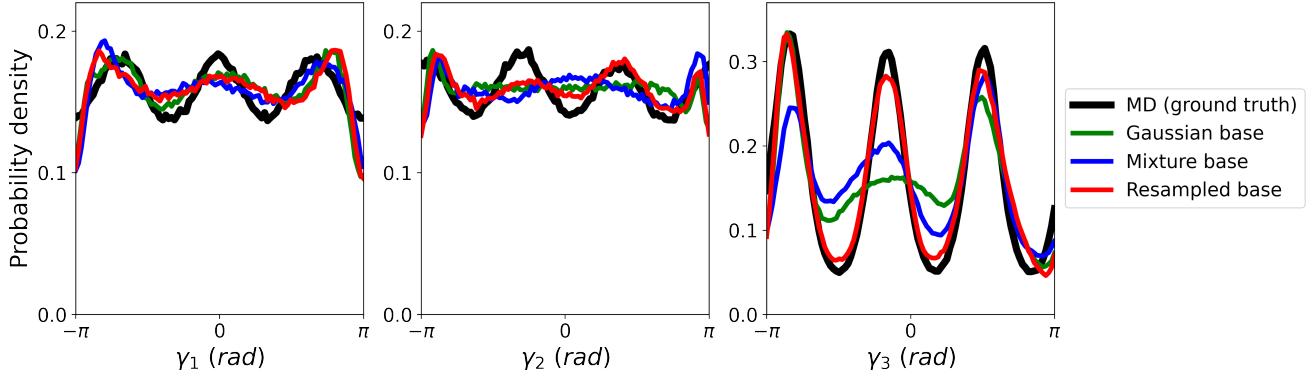


Figure 12: Marginal distribution of three dihedral angles of Alanine dipeptide. The ground truth was determined with a MD simulation. The flow models are based on the residual flow architecture and were trained via ML learning.

Table 14: Quantitative comparison of the residual flow models approximating the Boltzmann distribution of Alanine dipeptide trained via ML learning. The LL is evaluated on a test set obtained with a MD simulation. The KL divergences of the 60 marginals were computed and the mean and median of them are reported. Moreover, the KL divergences of the Ramachandran plots are listed. All results are averages over 10 runs, the standard error is given, and higher LL as well as lowest KL divergences are marked in **bold**.

Base distribution	Gaussian	Mixture	Resampled
LL ( $\times 10^2$ )	$1.8048 \pm 0.0002$	$1.8061 \pm 0.0002$	<b><math>1.8144 \pm 0.0002</math></b>
Mean KLD marginals ( $\times 10^{-3}$ )	$6.16 \pm 0.17$	$31.5 \pm 1.8$	<b><math>3.49 \pm 0.15</math></b>
Median KLD marginals ( $\times 10^{-4}$ )	$5.21 \pm 0.12$	$14.2 \pm 5.2$	<b><math>4.67 \pm 0.05</math></b>
KLD Ramachandran plot	$8.1 \pm 2.2$	$25.4 \pm 10.2$	<b><math>4.4 \pm 0.9</math></b>

Table 15: Quantitative comparison of the real NVP models approximating the Boltzmann distribution of Alanine dipeptide trained via the KL divergence. The LL is evaluated on a test set obtained with a MD simulation. The KL divergences of the 60 marginals were computed and the mean and median of them are reported. Moreover, the KL divergences of the Ramachandran plots are listed. All results are averages over 10 runs, the standard error is given, and higher LL as well as lowest KL divergences are marked in **bold**.

Base distribution	Gaussian	Mixture	Resampled
LL ( $\times 10^2$ )	$-2.78 \pm 0.07$	$-2.70 \pm 0.04$	<b><math>-1.84 \pm 0.13</math></b>
Mean KLD marginals ( $\times 10^{-1}$ )	$2.91 \pm 0.05$	$2.98 \pm 0.02$	<b><math>2.84 \pm 0.07</math></b>
Median KLD marginals ( $\times 10^{-3}$ )	$4.75 \pm 0.04$	$4.77 \pm 0.03$	<b><math>4.66 \pm 0.05</math></b>
KLD Ramachandran plot	$7.63 \pm 0.18$	$16.6 \pm 8.4$	<b><math>6.92 \pm 0.37</math></b>

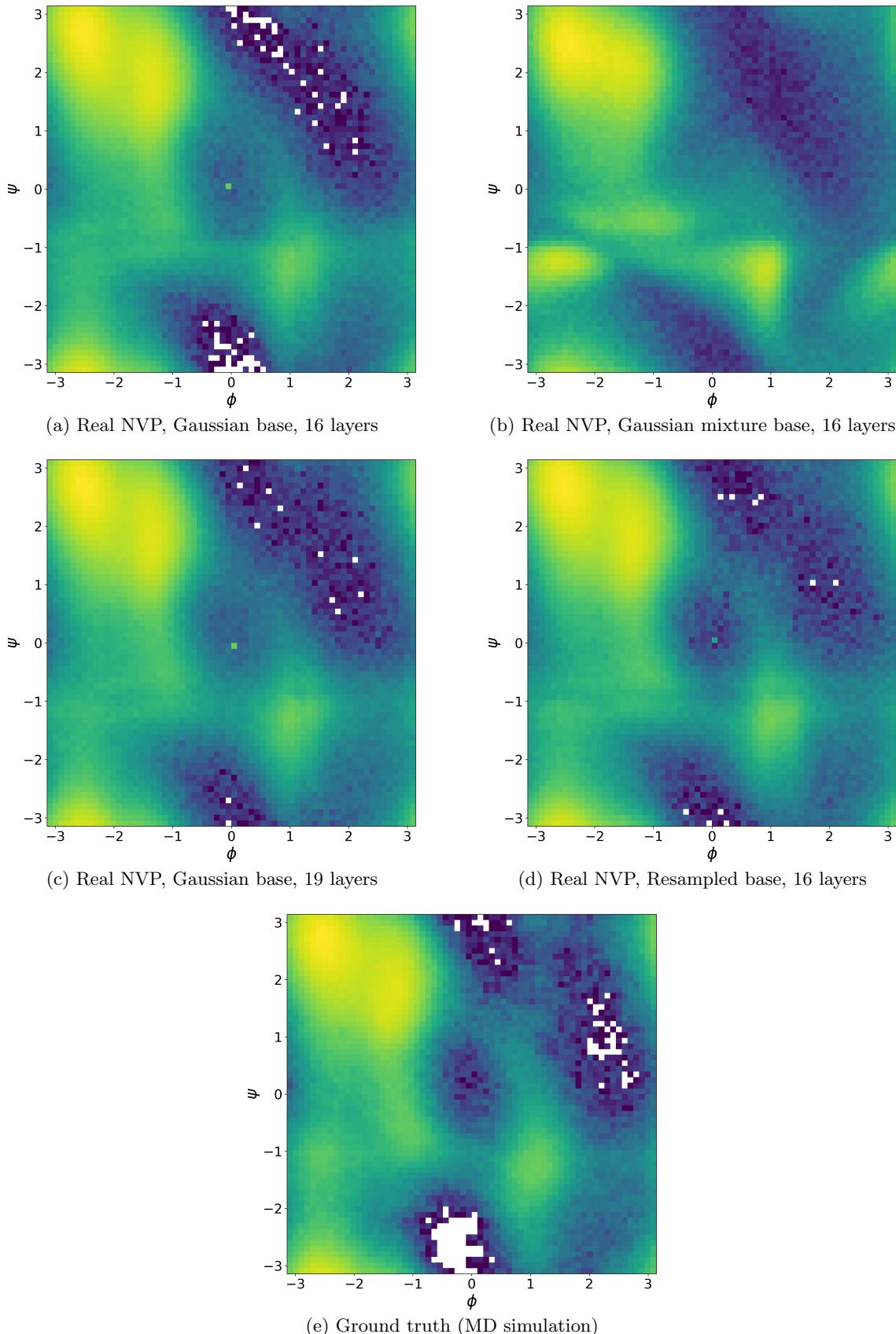


Figure 13: Ramachandran plots of Alanine dipeptide. The flow models were trained via ML learning